Concept: mathematics notation

1. \( \log_2 n \) is:
   (A) \( o(\log_{10} n) \)  
   (B) \( \Theta(\log_{10} n) \)  
   (C) \( \omega(\log_{10} n) \)

2. \( \log_2 n \) is equal to:
   (A) \( \frac{\log_2 n}{\log_{10} 2} \)  
   (B) \( \frac{\log_{10} n}{\log_2 10} \)  
   (C) \( \frac{\log_2 n}{\log_{10} m} \)  
   (D) \( \frac{\log_{10} n}{\log_{10} 2} \)

3. \( \log(nm) \) is equal to:
   (A) \( n \log m \)  
   (B) \( \log n + \log m \)  
   (C) \( (\log n)^m \)  
   (D) \( m \log n \)

4. \( \log(n^m) \) is equal to:
   (A) \( m \log n \)  
   (B) \( \log n + \log m \)  
   (C) \( (\log n)^m \)  
   (D) \( n \log m \)

5. \( \log_2 2 \) can be simplified to:
   (A) \( 1 \)  
   (B) \( 2 \)  
   (C) \( 4 \)  
   (D) \( \log_2 2 \) cannot be simplified any further

6. \( 2^{\log_2 n} \) is equal to:
   (A) \( n \)  
   (B) \( 2^n \)  
   (C) \( \log_2 n \)  
   (D) \( n^2 \)

7. \( n^2 \) is \( o(n^3) \). Therefore, \( \log n^2 \) is \(?(\log n^3) \). Choose the tightest bound.
   (A) \( \theta(\log n) \)  
   (B) \( \omega(\log n) \)  
   (C) \( \Theta(\log n) \)  
   (D) \( \omega(\log n) \)  
   (E) \( \Omega(\log n) \)

8. \( \log n^m \) is \( \Theta(?) \).
   (A) \( n \log m \)  
   (B) \( \log n \)  
   (C) \( \log n \)  
   (D) \( \log n^{\log n} \)

9. \( \log 2^n \) is \( \Theta(?) \).
   (A) \( \log n \)  
   (B) \( 2^n \)  
   (C) \( n \)  
   (D) \( n \log n \)

10. The number of permutations of a list of \( n \) items is:
    (A) \( 2^n \)  
    (B) \( n \log n \)  
    (C) \( \log n \)  
    (D) \( n! \)  
    (E) \( n \)
Concept: relative growth rates

11. Which of the following has the correct order in terms of growth rate?

(A) \( 1 < \sqrt{n} < \log n < n < n \log n < n^2 < n^3 < n! < 2^n < n^n \)
(B) \( 1 < \sqrt{n} < \log n < n < n \log n < n^2 < n^3 < n! < 2^n < n^n \)
(C) \( 1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < 2^n < n^n \)

12. What is the correct ordering of growth rates for the following functions:

- \( f(n) = n^{0.9} \log n \)
- \( g(n) = 1.1^n \)
- \( h(n) = 9.9n \)

(A) \( f < g < h \)
(B) \( h < g < f \)
(C) \( g < h < f \)

(D) \( h < f < g \)
(E) \( f < h < g \)
(F) \( g < f < h \)

13. What is the correct ordering of growth rates for the following functions:

- \( f(n) = n(\log n)^2 \)
- \( g(n) = n \log 2^n \)
- \( h(n) = n \log(\log n) \)

(A) \( f > h > g \)
(B) \( f > g > h \)
(C) \( h > g > f \)

(D) \( g > h > f \)
(E) \( h > f > g \)

(F) \( g > f > h \)

Concept: order notation

14. What does big Omicron roughly mean?

(A) worse than
(B) better than or the same as
(C) worse than or the same as

(D) better than
(E) the same as

15. What does \( \omega \) roughly mean?

(A) the same as
(B) worse than
(C) better than or the same as

(D) worse than or the same as
(E) better than

16. What does \( \theta \) roughly mean?

(A) worse than or the same as
(B) better than
(C) the same as

(D) better than or the same as
(E) worse than

17. T or F: All algorithms are \( \omega(1) \).

18. T or F: All algorithms are \( \theta(1) \).

19. T or F: All algorithms are \( \Omega(1) \).

20. T or F: There exist algorithms that are \( \omega(1) \).

21. T or F: There exist algorithms that are \( O(1) \).
22. T or F: All algorithms are \( O(n^3) \).

23. Consider sorting 1,000,000 numbers with mergesort. What is the time complexity of this operation? [THINK]
   (A) \( n \log n \), because mergesort takes \( n \log n \) time
   (B) \( n^2 \), because mergesort takes quadratic time
   (C) constant, because \( n \) is fixed

**Concept: comparing algorithms using order notation**

Assume the worst case and sufficiently large input size unless otherwise indicated. The phrase *the same time as* means *equal within a constant factor (or lower order term)* unless otherwise indicated. The phrase *by a stopwatch* means the actual amount of time needed for the algorithm to run to completion, as measured by a stopwatch.

24. T or F: If \( f = \omega(g) \), then algorithm \( f \) always runs faster than \( g \).

25. T or F: If \( f = \omega(g) \), then algorithm \( f \) always runs faster than \( g \), in all cases.

26. T or F: If \( f = \omega(g) \), then algorithm \( f \) always runs faster than \( g \), regardless of input size.

27. T or F: If \( f = \omega(g) \), then algorithm \( f \) always runs faster than or takes the same time as \( g \).

28. T or F: If \( f = \omega(g) \), then algorithm \( f \) always runs faster than or takes the same time as \( g \), in all cases.

29. T or F: If \( f = \omega(g) \), then algorithm \( f \) always runs faster than or takes the same time as \( g \), regardless of input size.

30. T or F: If \( f = \omega(g) \), then it is possible that algorithm \( f \) can run faster than \( g \), in some cases.

31. T or F: If \( f = \omega(g) \), then it is possible that algorithm \( f \) can run faster than \( g \), in all cases.

32. T or F: If \( f = \omega(g) \), then algorithm \( f \) always runs slower than \( g \).

33. T or F: If \( f = \omega(g) \), then algorithm \( f \) always runs slower than \( g \), in all cases.

34. T or F: If \( f = \omega(g) \), then algorithm \( f \) always runs slower than \( g \), regardless of input size.

35. T or F: If \( f = \omega(g) \), then algorithm \( f \) always runs slower than or takes the same time as \( g \).

36. T or F: If \( f = \omega(g) \), then algorithm \( f \) always runs slower than or takes the same time as \( g \), in all cases.

37. T or F: If \( f = \omega(g) \), then algorithm \( f \) always runs slower than or takes the same time as \( g \), regardless of input size.

38. T or F: If \( f = \omega(g) \), then algorithm \( f \) can take the same time as \( g \), in some cases.

39. T or F: If \( f = \omega(g) \), then algorithm \( f \) can run in the same time as \( g \).

40. T or F: If \( f = \Omega(g) \), then algorithm \( f \) always runs faster than \( g \).

41. T or F: If \( f = \Omega(g) \), then algorithm \( f \) always runs faster than \( g \), in all cases.

42. T or F: If \( f = \Omega(g) \), then algorithm \( f \) always runs faster than \( g \), regardless of input size.

43. T or F: If \( f = \Omega(g) \), then algorithm \( f \) always runs faster than or takes the same time as \( g \).

44. T or F: If \( f = \Omega(g) \), then algorithm \( f \) always runs faster than or takes the same time as \( g \), in all cases.

45. T or F: If \( f = \Omega(g) \), then algorithm \( f \) always runs faster than or takes the same time as \( g \), regardless of input size.

46. T or F: If \( f = \Omega(g) \), then it is possible that algorithm \( f \) can run faster than \( g \), in some cases.

47. T or F: If \( f = \Omega(g) \), then it is possible that algorithm \( f \) can run faster than \( g \), in some cases.

48. T or F: If \( f = \Omega(g) \), then algorithm \( f \) always runs slower than \( g \).
49. T or F: If \( f = \Omega(g) \), then algorithm \( f \) always runs slower than \( g \), in all cases.

50. T or F: If \( f = \Omega(g) \), then algorithm \( f \) always runs slower than \( g \), regardless of input size.

51. T or F: If \( f = \Omega(g) \), then algorithm \( f \) always runs slower than or takes the same time as \( g \).

52. T or F: If \( f = \Omega(g) \), then algorithm \( f \) always runs slower than or takes the same time as \( g \), in all cases.

53. T or F: If \( f = \Omega(g) \), then algorithm \( f \) always runs slower than or takes the same time as \( g \), regardless of input size.

54. T or F: If \( f = o(g) \), then algorithm \( f \) always runs slower than \( g \).

55. T or F: If \( f = o(g) \), then algorithm \( f \) always runs slower than \( g \), in all cases.

56. T or F: If \( f = o(g) \), then algorithm \( f \) always runs slower than \( g \), regardless of input size.

57. T or F: If \( f = o(g) \), then algorithm \( f \) always runs slower than or takes the same time as \( g \).

58. T or F: If \( f = o(g) \), then algorithm \( f \) always runs slower than or takes the same time as \( g \), in all cases.

59. T or F: If \( f = o(g) \), then algorithm \( f \) always runs slower than or takes the same time as \( g \), regardless of input size.

60. T or F: If \( f = o(g) \), then it is possible that algorithm \( f \) can run slower than \( g \), in some cases.

61. T or F: If \( f = o(g) \), then it is possible that algorithm \( f \) can run slower than \( g \).

62. T or F: If \( f = o(g) \), then algorithm \( f \) always runs faster than \( g \).

63. T or F: If \( f = o(g) \), then algorithm \( f \) always runs faster than \( g \), in all cases.

64. T or F: If \( f = o(g) \), then algorithm \( f \) always runs faster than \( g \), regardless of input size.

65. T or F: If \( f = o(g) \), then algorithm \( f \) always runs faster than or takes the same time as \( g \).

66. T or F: If \( f = o(g) \), then algorithm \( f \) always runs faster than or takes the same time as \( g \), in all cases.

67. T or F: If \( f = o(g) \), then algorithm \( f \) always runs faster than or takes the same time as \( g \), regardless of input size.

68. T or F: If \( f = o(g) \), then algorithm \( f \) can run slower than \( g \), in some cases.

69. T or F: If \( f = o(g) \), then algorithm \( f \) can take the same time as \( g \), in some cases.

70. T or F: If \( f = O(g) \), then algorithm \( f \) always runs slower than \( g \).

71. T or F: If \( f = O(g) \), then algorithm \( f \) always runs slower than \( g \), in all cases.

72. T or F: If \( f = O(g) \), then algorithm \( f \) always runs slower than \( g \), regardless of input size.

73. T or F: If \( f = O(g) \), then algorithm \( f \) always runs slower than or takes the same time as \( g \).

74. T or F: If \( f = O(g) \), then algorithm \( f \) always runs slower than or takes the same time as \( g \), in all cases.

75. T or F: If \( f = O(g) \), then algorithm \( f \) always runs slower than or takes the same time as \( g \), regardless of input size.

76. T or F: If \( f = O(g) \), then it is possible that algorithm \( f \) can run slower than \( g \), in some cases.

77. T or F: If \( f = O(g) \), then it is possible that algorithm \( f \) can run slower than \( g \).

78. T or F: If \( f = O(g) \), then algorithm \( f \) always runs faster than \( g \).

79. T or F: If \( f = O(g) \), then algorithm \( f \) always runs faster than \( g \), in all cases.

80. T or F: If \( f = O(g) \), then algorithm \( f \) always runs faster than \( g \), regardless of input size.
81. T or F: If \( f = O(g) \), then algorithm \( f \) always runs faster than or takes the same time as \( g \).

82. T or F: If \( f = O(g) \), then algorithm \( f \) always runs faster than or takes the same time as \( g \), in all cases.

83. T or F: If \( f = O(g) \), then algorithm \( f \) always runs faster than or takes the same time as \( g \), regardless of input size.

84. T or F: If \( f = \Theta(g) \), then algorithm \( f \) takes the same time as \( g \).

85. T or F: If \( f = \Theta(g) \), then algorithm \( f \) takes the same time as \( g \), in all cases.

86. T or F: If \( f = \Theta(g) \), then algorithm \( f \) takes the same time as \( g \), regardless of input size.

87. T or F: If \( f = \Theta(g) \), then it is possible that algorithm \( f \) can run slower than \( g \), in some cases.

88. T or F: If \( f = O(g) \), then it is possible that algorithm \( f \) can run slower than \( g \).

89. T or F: If \( f = O(g) \), then it is possible that algorithm \( f \) can run slower than \( g \), for some problem sizes.

90. T or F: If \( f = \Theta(g) \), then it is possible that algorithm \( f \) can run faster than \( g \), in some cases.

91. T or F: If \( f = \Theta(g) \), then it is possible that algorithm \( f \) can run faster than \( g \), regardless of input size.

92. T or F: If \( f = \Theta(g) \), then it is possible that algorithm \( f \) can run true than \( g \), for some problem sizes.

93. T or F: Suppose algorithm \( f = \omega(g) \), then \( f \) and \( g \) can be the same algorithm.

94. T or F: Suppose algorithm \( f = \omega(g) \), then \( f \) and \( g \) can be the same algorithm.

95. T or F: Suppose algorithm \( f = \omega(g) \), then \( f \) and \( g \) can be the same algorithm.

96. T or F: Suppose algorithm \( f = \omega(g) \), then \( f \) and \( g \) can be the same algorithm.

97. T or F: Suppose algorithm \( f = \theta(g) \), then \( f \) and \( g \) can be the same algorithm.

98. T or F: If \( f = \Omega(g) \) and \( g = O(f) \), then \( f = \Theta(g) \).

99. T or F: If \( f = \Omega(g) \) and \( g = O(f) \), then \( f \) and \( g \) must be the same algorithm.

100. T or F: Suppose algorithm \( f = \theta(g) \). Algorithm \( f \) is never slower than \( g \), by a stopwatch.

101. T or F: Suppose algorithm \( f = \theta(g) \). Algorithm \( f \) always takes the same time as \( g \), by a stopwatch.

**Concept: analyzing code**

In the pseudocode, the lower limit of a \texttt{for} loop is inclusive, while the upper limit is exclusive. The step, if not specified, is one.

102. What is the time complexity of this function? Assume the initial value of \( i \) is zero.

```plaintext
function f(i,n)
{
    if (i < n)
    {
        println(i);
        f(i+1,n);
    }
}
```

(A) \( \theta(n^2) \) \hspace{2cm} (D) \( \theta(\sqrt{n}) \)
(B) \( \theta(\log n) \) \hspace{2cm} (E) \( \theta(n) \)
(C) \( \theta(1) \) \hspace{2cm} (F) \( \theta(\log n) \)
103. What is the time complexity of this function? Assume the initial value of \( i \) is one.

\[
\text{function } f(i,n) \\
\text{ } \\
\{ \\
\quad \text{if } (i < n) \\
\quad \{ \\
\quad\quad \text{println}(i); \\
\quad\quad f(i*3,n); \\
\quad \} \\
\}
\]

(A) \( \theta(n \log n) \) \hspace{1cm} (D) \( \theta(n \sqrt{n}) \)  \\
(B) \( \theta(n) \) \hspace{1cm} (E) \( \theta((\log n)^3) \)  \\
(C) \( \theta(n^2) \) \hspace{1cm} (F) \( \theta(\log n) \)

104. What is the time complexity of this function? Assume the initial value of \( i \) is one.

\[
\text{function } f(i,n) \\
\text{ } \\
\{ \\
\quad \text{if } (i < n) \\
\quad \{ \\
\quad\quad f(i*\sqrt{n},n); \\
\quad\quad \text{println}(i); \\
\quad \} \\
\}
\]

(A) \( \theta(n^2) \) \hspace{1cm} (D) \( \theta(n) \)  \\
(B) \( \theta(\log n) \) \hspace{1cm} (E) \( \theta(1) \)  \\
(C) \( \theta(\sqrt{n}) \) \hspace{1cm} (F) \( \theta(n \sqrt{n}) \)

105. What is the time complexity of this function? Assume the initial value of \( i \) and \( j \) is zero.

\[
\text{function } f(i,j,n) \\
\text{ } \\
\{ \\
\quad \text{if } (i < n) \\
\quad \{ \\
\quad\quad \text{if } (j < n) \\
\quad\quad\quad f(i,j+1,n); \\
\quad\quad \text{else} \\
\quad\quad\quad f(i+1,i+1,n); \\
\quad\} \\
\quad \text{println}(i,j); \\
\}
\]

(A) \( \theta(n) \) \hspace{1cm} (D) \( \theta(\sqrt{n}) \)  \\
(B) \( \theta(n \log n) \) \hspace{1cm} (E) \( \theta(1) \)  \\
(C) \( \theta(n^2) \) \hspace{1cm} (F) \( \theta(\log^2 n) \)
106. What is the time complexity of this function? Assume the initial value of $i$ and $j$ is zero.

```java
function f(i, j, n)
{
  if (i < n)
  {
    if (j < i)
      f(i, j+1, n);
    else
      f(i, 0, i+1);
  }
  println(i, j);
}
```

(A) $\Theta(\log^2 n)$  
(B) $\Theta(\sqrt{n})$  
(C) $\Theta(n \log n)$  
(D) $\Theta(n^2)$  
(E) $\Theta(n)$  
(F) $\Theta(1)$

107. What is the time complexity of this function? Assume the initial value of $i$ is one and $j$ is zero.

```java
function f(i, j, n)
{
  if (i < n)
  {
    if (j < n)
      f(i, j+1, n);
    else
      f(i*2, i*2, n);
  }
  println(i, j);
}
```

(A) $\Theta(\log^2 n)$  
(B) $\Theta(1)$  
(C) $\Theta(n \sqrt{n})$  
(D) $\Theta(n^2)$  
(E) $\Theta(n \log n)$  
(F) $\Theta(n)$

108. What is the time complexity of this function? Assume the initial value of $i$ is one and $j$ is zero.

```java
function f(i, j, n)
{
  if (i < n)
  {
    if (j < n)
      f(i, j+1, n);
    else
      f(i*2, i*2, n);
  }
  println(i, j);
}
```

(A) $\Theta(n^2)$  
(B) $\Theta(n \sqrt{n})$  
(C) $\Theta(n \log n)$  
(D) $\Theta(n)$  
(E) $\Theta(1)$  
(F) $\Theta(\log^2 n)$
109. What is the time complexity of this function? Assume the initial value of \( i \) is zero and \( j \) is one.

```java
function f(i,j,n)
{
    if (i < n)
    {
        if (j < n)
            f(i,j*2,n);
        else
            f(i+1,1,n);
    }
    println(i,j);
}
```

(A) \( \theta(1) \)  
(B) \( \theta(n^2) \)  
(C) \( \theta(n \log n) \)  
(D) \( \theta(\log^2 n) \)  
(E) \( \theta(n) \)  
(F) \( \theta(n\sqrt{n}) \)

110. What is the time complexity of this function? Assume positive, integral input and integer division.

```java
function f(n)
{
    if (n > 0)
    {
        f(n/2);
        for (var i from 0 until n)
        {
            println(n);
        }
    }
}
```

(A) \( \theta(1) \)  
(B) \( \theta(\log n) \)  
(C) \( \theta(n \log n) \)  
(D) \( \theta(\log^2 n) \)  
(E) \( \theta(n) \)  
(F) \( \theta(n^2) \)

111. What is the time complexity of this code fragment?

```java
for (i from 0 until n by 1)
    for (j from 0 until i by 1)
        println(i,j);
```

(A) \( \theta(1) \)  
(B) \( \theta(\log^2 n) \)  
(C) \( \theta(n \log n) \)  
(D) \( \theta(n \sqrt{n}) \)  
(E) \( \theta(n^2) \)  
(F) \( \theta(n^2) \)

112. What is the time complexity of this code fragment?

```java
i = 1;
while (i < n)
{
    for (j from 0 until n by 1)
        println(i,j);
    i = i * 2;
}
```

(A) \( \theta(\log^2 n) \)  
(B) \( \theta(n^2) \)  
(C) \( \theta(n) \)  
(D) \( \theta(1) \)  
(E) \( \theta(n \log n) \)  
(F) \( \theta(n \sqrt{n}) \)
113. What is the time complexity of this code fragment?

```java
i = 1;
while (i < n)
{
    for (j from 0 until i by 1)
        println(i,j);
    i = i * 2;
}
```

(A) \(\theta(1)\) 
(B) \(\theta(n)\)  
(C) \(\theta(n \log n)\) 
(D) \(\theta(n^2)\)  
(E) \(\theta(\log^2 n)\)  
(F) \(\theta(n \sqrt{n})\)

114. What is the time complexity of this code fragment?

```java
for (i from 0 until n)
{
    j = 1;
    while (j < n)
    {
        println(i,j);
        j = j * 2;
    }
}
```

(A) \(\theta(n^2)\)  
(B) \(\theta(n \sqrt{n})\)  
(C) \(\theta(\log^2 n)\)  
(D) \(\theta(n)\)  
(E) \(\theta(n \log n)\)  
(F) \(\theta(1)\)

115. What is the time complexity of this code fragment?

```java
for (i from 0 until n by 1)
    println(i);
```

(A) \(\theta(\log^2 n)\)  
(B) \(\theta(n^2)\)  
(C) \(\theta(n \sqrt{n})\)  
(D) \(\theta(n)\)  
(E) \(\theta(n \log n)\)  
(F) \(\theta(n \log n)\)

116. What is the time complexity of this code fragment?

```java
i = 1;
while (i < n)
{
    println(i);
    i = i * 2;
}
```

(A) \(\theta(n \sqrt{n})\)  
(B) \(\theta(n \log n)\)  
(C) \(\theta(n^2)\)  
(D) \(\theta(\log n)\)  
(E) \(\theta((\log n)^2)\)  
(F) \(\theta(n)\)
117. What is the time complexity of this code fragment?

```java
i = 1;
while (i < n)
{
    println(i);
    i = i * sqrt(n);
}
```

(A) $\Theta(n\sqrt{n})$  
(B) $\Theta(n - \sqrt{n})$  
(C) $\Theta(\sqrt{n})$  
(D) $\Theta(1)$  
(E) $\Theta(n^{\frac{1}{2}})$  
(F) $\Theta(n)$

118. What is the time complexity of this code fragment?

```java
i = 1;
while (i < n)
{
    println(i);
    i = i * 2;
}
for (j from 0 until n by 2)
    println(j);
```

(A) $\Theta(\log^2 n)$  
(B) $\Theta(n^2)$  
(C) $\Theta(1)$  
(D) $\Theta(n \log n)$  
(E) $\Theta(n \sqrt{n})$  
(F) $\Theta(n)$

119. What is the time complexity of this code fragment?

```java
for (i from 0 until n by 1)
    println(i);
for (j from 0 until n by 1)
    println(j);
```

(A) $\Theta(\log^2 n)$  
(B) $\Theta(n)$  
(C) $\Theta(n \log n)$  
(D) $\Theta(1)$  
(E) $\Theta(n^2)$  
(F) $\Theta(n \sqrt{n})$

120. What is the time complexity of this code fragment?

```java
for (i from 0 until n by 2)
    println(i);
    j = 1;
    while (j < n)
    {
        println(j);
        j = j * 2;
    }
```

(A) $\Theta(\log^2 n)$  
(B) $\Theta(n \log n)$  
(C) $\Theta(n \sqrt{n})$  
(D) $\Theta(1)$  
(E) $\Theta(n^2)$  
(F) $\Theta(n)$

121. What is the space complexity of this code fragment?

```java
for (i from 0 until n by 1)
    println(i);
```

(A) $\Theta(n^2)$  
(B) $\Theta(\sqrt{n})$  
(C) $\Theta(1)$  
(D) $\Theta(n \log n)$  
(E) $\Theta(n)$  
(F) $\Theta(\log n)$
122. What is the space complexity of this code fragment?

```java
i = 1;
while (i < n)
{
    println(i);
    i = i * sqrt(n);
}
```

(A) \( \theta(1) \)  
(B) \( \theta(\sqrt{n}) \)  
(C) \( \theta(n^2) \)  
(D) \( \theta(\log n) \)  
(E) \( \theta(n \log n) \)  
(F) \( \theta(n) \) 

123. What is the space complexity of this code fragment?

```java
i = 1;
while (i < n)
{
    println(i);
    i = i * 2;
}
```

(A) \( \theta(n \log n) \)  
(B) \( \theta(n) \)  
(C) \( \theta(\sqrt{n}) \)  
(D) \( \theta(\log n) \)  
(E) \( \theta(n^2) \)  
(F) \( \theta(1) \) 

124. What is the space complexity of this code fragment?

```java
for (i from 0 until n by 1)
    println(i);
for (j from 0 until n by 1)
    println(j);
```

(A) \( \theta(1) \)  
(B) \( \theta(n^2) \)  
(C) \( \theta(\sqrt{n}) \)  
(D) \( \theta(\log n) \)  
(E) \( \theta(n \log n) \)  
(F) \( \theta(n) \) 

125. What is the space complexity of this code fragment?

```java
i = 1;
while (i < n)
{
    println(i);
    i = i * 2;
}
for (j from 0 until n by 2)
    println(j);
```

(A) \( \theta(n) \)  
(B) \( \theta(n \log n) \)  
(C) \( \theta(\log n) \)  
(D) \( \theta(\sqrt{n}) \)  
(E) \( \theta(n^2) \)  
(F) \( \theta(1) \)
126. What is the space complexity of this code fragment?

```java
for (i from 0 until n by 2)
    println(i);
j = 1;
while (j < n)
{
    println(j);
    j = j * 2;
}
```

(A) $\theta(n \log n)$  
(B) $\theta(1)$  
(C) $\theta(\sqrt{n})$  
(D) $\theta(\log n)$  
(E) $\theta(n)$  
(F) $\theta(n^2)$

127. What is the space complexity of this code fragment?

```java
for (i from 0 until n by 1)
    for (j from 0 until i by 1)
        println(i, j);
```

(A) $\theta(n)$  
(B) $\theta(n^2)$  
(C) $\theta(1)$  
(D) $\theta(\log n)$  
(E) $\theta(\sqrt{n})$  
(F) $\theta(n \log n)$

128. What is the space complexity of this code fragment?

```java
i = 1;
while (i < n)
{
    for (j from 0 until i by 1)
        println(i, j);
    i = i * 2;
}
```

(A) $\theta(n \log n)$  
(B) $\theta(1)$  
(C) $\theta(\log n)$  
(D) $\theta(\sqrt{n})$  
(E) $\theta(n)$  
(F) $\theta(n^2)$

129. What is the space complexity of this code fragment?

```java
for (i from 0 until n)
{
    j = 1;
    while (j < n)
    {
        println(i, j);
        j = j * 2;
    }
}
```

(A) $\theta(n^2)$  
(B) $\theta(\log n)$  
(C) $\theta(\sqrt{n})$  
(D) $\theta(n \log n)$  
(E) $\theta(n)$  
(F) $\theta(1)$
130. What is the space complexity of this code fragment?

```java
d = 1;
while (i < n)
{
    for (j from 0 until n by 1)
        println(i,j);
    i = i * 2;
}
```

(A) \( \theta(n) \)  
(B) \( \theta(\sqrt{n}) \)  
(C) \( \theta(n^2) \)  
(D) \( \theta(1) \)  
(E) \( \theta(n \log n) \)  
(F) \( \theta(\log n) \)

131. What is the space complexity of this function? Assume the initial value of \( i \) and \( j \) is zero.

```java
function f(i,j,n)
{
    if (i < n)
    {
        if (j < n)
            f(i,j+1,n);
        else
            f(i+1,i+1,n);
    }
    println(i,j);
}
```

(A) \( \theta(1) \)  
(B) \( \theta(n^2) \)  
(C) \( \theta(\sqrt{n}) \)  
(D) \( \theta(\log^2 n) \)  
(E) \( \theta(n \log n) \)  
(F) \( \theta(\log n) \)

132. What is the space complexity of this function? Assume the initial value of \( i \) and \( j \) is zero.

```java
function f(i,j,n)
{
    if (i < n)
    {
        if (j < i)
            f(i,j+1,n);
        else
            f(i+1,0,i+1);
    }
    println(i,j);
}
```

(A) \( \theta(\log^2 n) \)  
(B) \( \theta(n) \)  
(C) \( \theta(\sqrt{n}) \)  
(D) \( \theta(n \log n) \)  
(E) \( \theta(n^2) \)  
(F) \( \theta(1) \)
133. What is the space complexity of this function? Assume the initial value of \( i \) is one and \( j \) is zero.

```plaintext
function f(i,j,n)
{
    if (i < n)
    {
        if (j < n)
            f(i,j+1,n);
        else
            f(i*2,0,n);
        println(i,j);
    }
}
```

(A) \( \theta(\log^2 n) \)  
(B) \( \theta(n \log n) \)  
(C) \( \theta(n \sqrt{n}) \)  
(D) \( \theta(1) \)  
(E) \( \theta(n) \)  
(F) \( \theta(n^2) \)

134. What is the space complexity of this function? Assume the initial value of \( i \) is one and \( j \) is zero.

```plaintext
function f(i,j,n)
{
    if (i < n)
    {
        if (j < n)
            f(i,j*2,n);
        else
            f(i+1,1,n);
        println(i,j);
    }
}
```

(A) \( \theta(\log^2 n) \)  
(B) \( \theta(n) \)  
(C) \( \theta(n \log n) \)  
(D) \( \theta(n^2) \)  
(E) \( \theta(1) \)  
(F) \( \theta(n \sqrt{n}) \)

135. What is the space complexity of this function? Assume the initial value of \( i \) is zero and \( j \) is one.

```plaintext
function f(i,j,n)
{
    if (i < n)
    {
        if (j < n)
            f(i,j*2,n);
        else
            f(i+1,1,n);
        println(i,j);
    }
}
```

(A) \( \theta(1) \)  
(B) \( \theta(n \log n) \)  
(C) \( \theta(n \sqrt{n}) \)  
(D) \( \theta(n) \)  
(E) \( \theta(\log^2 n) \)  
(F) \( \theta(n^2) \)
136. What is the space complexity of this function? Assume positive, integral input and integer division.

```java
function f(x,n)
{
    if (x > 0)
    {
        f(x/2,n);
        for (var i from 0 until n)
        {
            println(n);
        }
    }
}
```

(A) \(\theta(n)\)  (D) \(\theta(1)\)
(B) \(\theta(\log n)\)  (E) \(\theta(n\sqrt{n})\)
(C) \(\theta(n\log n)\)  (F) \(\theta(n^2)\)

137. What is the space complexity of this function? Assume the initial value of \(i\) is zero.

```java
function f(i,n)
{
    if (i < n)
    {
        f(i+1,n);
        println(i);
    }
}
```

(A) \(\theta(1)\)  (D) \(\theta(n\log n)\)
(B) \(\theta(\log n)\)  (E) \(\theta(\sqrt{n})\)
(C) \(\theta(n^2)\)  (F) \(\theta(n)\)

138. What is the space complexity of this function? Assume the initial value of \(i\) is one.

```java
function f(i,n)
{
    if (i < n)
    {
        f(i*2,n);
        println(i);
    }
}
```

(A) \(\theta(\sqrt{n})\)  (D) \(\theta(\log n)\)
(B) \(\theta(n\log n)\)  (E) \(\theta(n)\)
(C) \(\theta(1)\)  (F) \(\theta(n^2)\)

139. What is the space complexity of this function? Assume the initial value of \(i\) is one.

```java
function f(i,n)
{
    if (i < n)
    {
        f(i*\sqrt{n},n);
        println(i);
    }
}
```

(A) \(\theta(n - \sqrt{n})\)  (D) \(\theta(\sqrt{n})\)
(B) \(\theta(n\sqrt{n})\)  (E) \(\theta(n)\)
(C) \(\theta(1)\)  (F) \(\theta(n\sqrt{n})\)
140. What is the space complexity of this function? Assume the initial value of \( i \) is one.

```java
function f(i,n)
{
    if (i < n)
    {
        f(i+sqrt(n),n);
        println(i);
    }
}
```

(A) \( \theta(n^{\frac{1}{2}}) \)  
(B) \( \theta(n^{\sqrt{n}}) \)  
(C) \( \theta(n) \)  
(D) \( \theta(1) \)  
(E) \( \theta(\sqrt{n}) \)  
(F) \( \theta(n - \sqrt{n}) \)

**Concept: analysis of classic, simple algorithms**

141. Which of the following describes the classic recursive fibonacci’s time complexity?

(A) \( \theta(\sqrt{n}) \)  
(B) \( \theta(\Phi^n) \)  
(C) \( \theta(\Phi) \)  
(D) \( \theta(\frac{\Phi^n}{\sqrt{n}}) \)  
(E) \( \theta(1) \)  
(F) \( \theta(\frac{\Phi^n}{n}) \)  
(G) \( \theta(\frac{\Phi}{\sqrt{n}}) \)

142. Which of the following describes the classic recursive fibonacci’s space complexity?

(A) \( \theta(1) \)  
(B) \( \theta(n - \sqrt{n}) \)  
(C) \( \theta(\frac{n}{\sqrt{n}}) \)  
(D) \( \theta(\sqrt{n}) \)  
(E) \( \theta(n) \)  
(F) \( \theta(\frac{\Phi^n}{n}) \)

143. Which of the following describes iterative factorial’s time complexity?

(A) \( \theta(\frac{\Phi^n}{n}) \)  
(B) \( \theta(n - \sqrt{n}) \)  
(C) \( \theta(\sqrt{n}) \)  
(D) \( \theta(1) \)  
(E) \( \theta(\frac{\Phi^n}{\sqrt{n}}) \)  
(F) \( \theta(\frac{\Phi^n}{n}) \)

144. Which of the following describes iterative fibonacci’s space complexity?

(A) \( \theta(n - \sqrt{n}) \)  
(B) \( \theta(\frac{\Phi^n}{n}) \)  
(C) \( \theta(\frac{n}{\sqrt{n}}) \)  
(D) \( \theta(1) \)  
(E) \( \theta(n) \)  
(F) \( \theta(\sqrt{n}) \)

**Concept: searching**

145. T or F: The following code reliably sets the variable \( min \) to the minimum value of an unsorted, non-empty array.

```java
min = 0;
for (i from 0 until array.length)
    if (array[i] < min)
        min = array[i];
```

146. T or F: The following code reliably sets the variable \( max \) to the maximum value in an unsorted, non-empty array.

```java
max = array[0]
for (i from 0 to array.length)
    if (array[i] > max)
        max = array[i]
```
147. T or F: The following function reliably returns True if the value of item is present in the unsorted, non-empty array.

```javascript
function find(array, item)
{
    found = False;
    for (i from 0 until array.length)
        if (array[i] == item)
            found = True;
    return found;
}
```

148. T or F: The following function reliably returns False if the value of item is missing in the unsorted, non-empty array.

```javascript
function find(array, item)
{
    found = True;
    for (i from 0 until array.length)
        if (array[i] != item)
            found = False;
    return found;
}
```

149. What is the average and worst case time complexity, respectively, for searching an unordered list?

(A) linear, log
(B) linear, linear
(C) log, linear
(D) log, log

150. What is the average and worst case time complexity, respectively, for searching an ordered list?

(A) log, log
(B) log, linear
(C) linear, log
(D) linear, linear

Concept: sorting

151. The following strategy is employed by which sort: find the most extreme value in the unsorted portion and place it at the boundary of the sorted and unsorted portions?

(A) quicksort
(B) selection sort
(C) heapsort
(D) bubble sort
(E) insertion sort
(F) mergesort

152. The following strategy is employed by which sort: sort the lower half of the items to be sorted, then sort the upper half, then arrange things such that the largest item in the lower half is less than or equal to the smallest item in the upper half?

(A) heapsort
(B) mergesort
(C) quicksort
(D) insertion sort
(E) bubble sort
(F) selection sort

153. The following strategy is employed by which sort: take the first value in the unsorted portion and place it where it belongs in the sorted portion?

(A) bubble sort
(B) selection sort
(C) mergesort
(D) quicksort
(E) insertion sort
(F) heapsort
154. The following strategy is employed by which sort: pick a value and arrange things such that the largest item in the lower portion is less than or equal to the value and that the smallest item in the upper portion is greater than or equal to the value, then sort the lower portion, then sort the upper?

(A) quicksort  (D) bubble sort
(B) selection sort  (E) heapsort
(C) insertion sort  (F) mergesort

155. Which sort optimizes the worst case behavior of bubble sort?

(A) insertion sort  (D) selection sort
(B) heapsort  (E) stooge sort
(C) quicksort  (F) mergesort

**Concept: space and time complexity**

156. What is the best time case complexity for classical mergesort?

(A) cubic  (D) linear
(B) \( n \log n \)  (E) \( \log n \)
(C) quadratic

157. What is the worst case complexity for classical mergesort?

(A) quadratic  (D) \( n \log n \)
(B) linear  (E) cubic
(C) \( \log n \)

158. If quicksort is implemented such that the pivot is chosen to be the first element in the array, the worst case behavior of the sort is:

(A) quadratic  (C) exponential
(B) \( \log n \)  (D) linear

159. If quicksort is implemented such that a random element is chosen to be the pivot, the worst case behavior of the sort is:

(A) linear  (C) quadratic
(B) exponential  (D) \( \log \) linear

160. What is the best case complexity for quicksort?

(A) quadratic  (D) linear
(B) \( n \log n \)  (E) \( \log n \)
(C) cubic

161. What is the best case complexity for classical selection sort?

(A) \( n \log n \)  (D) cubic
(B) \( \log n \)  (E) quadratic
(C) linear

162. What is the worst case complexity for classical selection sort?

(A) linear  (D) cubic
(B) \( \log n \)  (E) quadratic
(C) \( n \log n \)
163. What is the best case complexity for classical insertion sort?

- (A) cubic
- (B) log \( n \)
- (C) \( n \log n \)
- (D) linear
- (E) quadratic

164. What is the worst case complexity for classical insertion sort?

- (A) log \( n \)
- (B) linear
- (C) quadratic
- (D) \( n \log n \)
- (E) cubic

**Concept: simple arrays**

Assume zero-based indexing for all arrays.

In the pseudocode, the lower limit of a for loop is inclusive, while the upper limit is exclusive. The step, if not specified, is one.

For all types of fillable arrays, the size is the number of elements added to the array; the capacity is the maximum number of elements that can be added to the array.

165. Consider a small array \( a \) and large array \( b \). Accessing the element in the first slot of \( a \) takes more/less/the same amount of time as accessing the element in the first slot of \( b \).

- (A) it depends on how the arrays were allocated
- (B) more time
- (C) the same amount of time
- (D) less time

166. Consider a small array \( a \) and large array \( b \). Accessing the element in the last slot of \( a \) takes more than/less than/the same amount of time as accessing an element in the middle slot of \( a \). Both indices are supplied.

- (A) more time
- (B) it depends on how the arrays were allocated
- (C) the same amount of time
- (D) less time

167. Accessing the middle element of an array takes more/less/the same amount of time than accessing the last element.

- (A) more time
- (B) less time
- (C) it depends on how the array were allocated
- (D) the same amount of time

168. What is a major characteristic of a simple array?

- (A) inserting an element between indices \( i \) and \( i+1 \) can be done in constant time
- (B) getting the value at an index can be done in constant time
- (C) finding an element can be done in constant time

169. What is a not a major characteristic of a simple array?

- (A) setting the value at an index can be done in constant time
- (B) swapping two elements can be done in constant time
- (C) finding an element can be done in constant time
- (D) getting the value at an index can be done in constant time
170. Does the following code set the variable \( v \) to the minimum value in an unsorted array with at least two elements?

\[
\begin{align*}
v &= 0; \\
for \ (i \ from \ 0 \ until \ array.length) \\
&\quad \text{if} \ (array[i] < v) \\
&\quad \quad v = array[i]; \\
\end{align*}
\]

(A) yes, if all the elements are positive  \hspace{1cm} (D) yes, if all the elements are negative

(B) always  \hspace{1cm} (E) only if the true minimum value is zero

(C) only if all elements have the same value  \hspace{1cm} (F) never

171. Does the following code set the variable \( v \) to the minimum value in an unsorted array with at least two elements?

\[
\begin{align*}
v &= array[0]; \\
for \ (i \ from \ 0 \ until \ array.length) \\
&\quad \text{if} \ (array[i] < v) \\
&\quad \quad v = array[i]; \\
\end{align*}
\]

(A) only if all elements have the same value  \hspace{1cm} (D) never

(B) yes, if all the elements are positive  \hspace{1cm} (E) yes, if all the elements are negative

(C) only if the true minimum value is at index 0  \hspace{1cm} (F) always

172. Does the following code set the variable \( v \) to the minimum value in an unsorted array with at least two elements?

\[
\begin{align*}
v &= array[0]; \\
for \ (i \ from \ 0 \ until \ array.length) \\
&\quad \text{if} \ (array[i] > v) \\
&\quad \quad v = array[i]; \\
\end{align*}
\]

(A) never  \hspace{1cm} (D) only if the true minimum value is at index 0

(B) yes, if all the elements are positive  \hspace{1cm} (E) yes, if all the elements are negative

(C) only if all elements have the same value  \hspace{1cm} (F) always

173. Does the following code set the variable \( v \) to the minimum value in an unsorted, non-empty array?

\[
\begin{align*}
v &= array[0]; \\
for \ (i \ from \ 0 \ until \ array.length) \\
&\quad \text{if} \ (array[i] > v) \\
&\quad \quad v = array[i]; \\
\end{align*}
\]

(A) never  \hspace{1cm} (D) yes, if all the elements are negative

(B) yes, if all the elements are positive  \hspace{1cm} (E) only if all elements have the same value

(C) only if the true minimum value is at index 0  \hspace{1cm} (F) always

174. Does this \texttt{find} function return the expected result? Assume the array has at least two elements.

\[
\text{function find(array, item)} \\
\quad \{ \\
\quad \quad \text{var i; var found = False; } \\
\quad \quad \text{for (i from 0 until array.length)} \\
\quad \quad \quad \text{if (array[i] == item)} \\
\quad \quad \quad \quad \quad \text{found = True; } \\
\quad \quad \quad \text{return found; } \\
\quad \}
\]

(A) always  \hspace{1cm} (C) only if the item is not in the array

(B) only if the item is in the array  \hspace{1cm} (D) never
175. Does this \texttt{find} function return the expected result? Assume the array has at least two elements.

\begin{verbatim}
function find(array,item)
{
    var i;
    for (i from 0 until array.length)
        if (array[i] == item)
            return False;
    return True;
}
\end{verbatim}

(A) only if the item is in the array
(B) only if the item is not in the array
(C) never
(D) always

176. Is this \texttt{find} function correct? Assume the array has at least two elements.

\begin{verbatim}
function find(array,item)
{
    var i; var found = True;
    for (i from 0 until array.length)
        if (array[i] != item)
            found = False;
    return found;
}
\end{verbatim}

(A) only if the item is not in the array
(B) never
(C) only if the item is in the array
(D) always

177. Does this \texttt{find} function return the expected result? Assume the array has at least two elements.

\begin{verbatim}
function find(array,item)
{
    var i;
    for (i from 0 until array.length)
        if (array[i] == item)
            return True;
    return False;
}
\end{verbatim}

(A) never
(B) only if the item is in the array
(C) always
(D) only if the item is not in the array

\textbf{Concept: simple fillable arrays}

Assume the back index in a simple fillable array points to the first available slot.

178. What is not a property of a simple fillable array?

(A) elements can be added in constant time
(B) there exists an element that can be removed in constant time
(C) elements are presumed to be contiguous
(D) the underlying simple array can increase in size

179. What is a property of a simple fillable array?

(A) any element can be removed in constant time
(B) an element can be added anywhere in constant time
(C) elements are presumed to be contiguous
(D) more that one element can be next to an empty slot

180. Adding an element at back of a simple fillable array can be done in:

(A) logarithmic time
(B) quadratic time
(C) linear time
(D) constant time
181. Removing an element at front of a simple fillable array can be done in:

(A) linear time
(B) logarithmic time
(C) quadratic time
(D) constant time

182. Suppose a simple fillable array has size \( s \) and capacity \( c \). The next value to be added to the array will be placed at index:

(A) \( c - 1 \)
(B) \( c \)
(C) \( s - 1 \)
(D) \( s \)
(E) \( s + 1 \)
(F) \( c + 1 \)

183. Suppose for a simple fillable array, the size is one less than the capacity. How many values can still be added?

(A) two
(B) this situation cannot exist
(C) zero, the array is full
(D) one

184. Suppose for a simple fillable array, the capacity is one less than the size. How many values can still be added?

(A) zero, the array is full
(B) one
(C) two
(D) this situation cannot exist

185. Suppose a simple fillable array is empty. The size of the array is:

(A) the capacity of the array
(B) one
(C) the length of the underlying simple array
(D) zero

186. Suppose a simple fillable array is full. The capacity of the array is:

(A) one
(B) its size minus one
(C) zero
(D) the length of the underlying simple array

187. Which code fragment correctly inserts a new element into index \( j \) of a simple fillable array with size \( s \)? Assume there is room for the new element.

```java
for (i from j until s-2)
    array[i] = array[i+1];
array[i] = newElement;
---
for (i from s-2 until j)
    array[i+1] = array[i];
array[i] = newElement;
```

(A) neither are correct
(B) the second fragment
(C) both are correct
(D) the first fragment

188. Which code fragment correctly inserts a new element into index \( j \) of an array with size \( s \)?

```java
for (i from j until s-2)
    array[i+1] = array[i];
array[i] = newElement;
---
for (i from s-2 until j)
    array[i] = array[i+1];
array[i] = newElement;
```

(A) neither are correct
(B) both are correct
(C) the first fragment
(D) the second fragment
Concept: circular arrays

For circular arrays, assume \( f \) is the start index, \( e \) is the end index, \( s \) is the size, and \( c \) is the capacity of the array. Both \( f \) and \( e \) point to the first available slots.

189. What is a property of a theoretical (not practical) circular array?

(A) an element can be added anywhere in constant time  
(B) there are two places an element can be added  
(C) any element can be removed in constant time  
(D) elements do not have to be contiguous

190. What is not a property of a theoretical (not practical) circular array?

(A) elements are presumed to be contiguous  
(B) prepending an element can be done in constant time  
(C) inserting an element in the middle can be done in constant time  
(D) appending an element can be done in constant time

191. The next value to be added to the front of a circular array will be placed at index:

(A) \( c - f \)  
(B) \( c - 1 \)  
(C) \( f \)  
(D) \( s - f \)  
(E) \( f - 1 \)  
(F) \( s + f \)

192. Suppose for a circular array, the size is equal to the capacity. Can a value be added?

(A) Yes, there is room for one more value  
(B) No, the array is completely full

193. Suppose a circular array is empty. The size of the array is:

(A) zero  
(B) the length of the array  
(C) one  
(D) the capacity of the array

194. In a circular array, which is not a proper way to correct the start index \( f \) after an element is added to the front of the array?

(A) \( f = (f - 1 + c) \% c; \)  
(B) \( f = f == 0? c - 1 : f - 1; \)  
(C) \( \text{if } (f == 0) f = c - 1; \text{ else } f = f - 1; \)

195. T or F: In a circular array, the start index (after correction) can never equal the size of the array.

196. T or F: In a circular array, the start index (after correction) can never equal the capacity of the array.

197. Is a separate end index \( e \) needed in a circular array?

(A) no, it can be computed from \( s, c, \) and \( f \).  
(B) no, it can be computed from \( s \) and \( f \).  
(C) yes  
(D) no, it can be computed from \( c \) and \( f \).  
(E) no, it can be computed from \( s \) and \( c \).

Concept: dynamic arrays

198. What is not a major characteristic of a dynamic array?

(A) elements are presumed to be contiguous  
(B) the only allowed way to grow is doubling the size  
(C) the array can grow to accommodate more elements  
(D) inserting an element in the middle takes linear time  
(E) finding an element takes at most linear time

199. Suppose a dynamic array has size \( s \) and capacity \( c \), with \( s \) equal to \( c \). Is the array required to grow on the next addition?

(A) yes  
(B) yes, but only if the dynamic array is not circular  
(C) no
200. Suppose array capacity grows by 50% every time a dynamic array fills. If the only events are insertions, the growing events:
   (A) cannot be characterized in terms of frequency   (C) occur less and less frequently
   (B) occur periodically                              (D) occur more and more frequently

201. Suppose array capacity doubles every time a dynamic array fills. If the only events are insertions, the average cost of an insertion, in the limit, is:
   (A) the log of the capacity                        (C) the log of the size
   (B) constant                                      (D) linear

202. Suppose array capacity grows by 10 every time a dynamic array fills. If the only events are insertions, the average cost of an insertion, in the limit, is:
   (A) constant                                      (D) the log of the size
   (B) quadratic                                     (E) the log of the capacity
   (C) linear

203. Suppose array capacity grows by 10 every time a dynamic array fills. If the only events are insertions, the growing events:
   (A) occur less and less frequently                (C) cannot be characterized in terms of frequency
   (B) occur periodically                            (D) occur more and more frequently

204. If array capacity grows by 10 every time a dynamic array fills, the average cost of an insertion in the limit is:
   (A) constant                                      (C) the log of the size
   (B) linear                                        (D) the log of the capacity

Concept: **singly-linked lists (insertions)**

205. Appending to a singly-linked list without a tail pointer takes:
   (A) constant time                                 (C) log time
   (B) \( n \log n \) time                             (D) linear time

206. Appending to a singly-linked list with a tail pointer takes:
   (A) log time                                      (C) linear time
   (B) \( n \log n \) time                             (D) constant time

207. Suppose you have a pointer to a node near the end of a long singly-linked list. You can then insert a new node just prior in:
   (A) linear time                                   (C) log time
   (B) \( n \log n \) time                             (D) constant time

208. Suppose you have a pointer to a node near the end of a long singly-linked list. You can then insert a new node just after in:
   (A) constant time                                 (C) \( n \log n \) time
   (B) linear time                                   (D) log time

209. Suppose you have a pointer to a node near the end of a long singly-linked list. You can then insert a new node just after with as few pointer assignments as:
   (A) 2                                             (D) 1
   (B) 3                                             (E) 5
   (C) 4
Concept: *singly-linked lists (deletions)*

210. Removing the first item from a singly-linked list without a tail pointer takes:

(A) constant time  
(B) linear time  
(C) log time  
(D) \( n \log n \) time

211. Removing the last item from a singly-linked list with a tail pointer takes:

(A) log time  
(B) linear time  
(C) \( n \log n \) time  
(D) constant time

212. Removing the last item from a singly-linked list without a tail pointer takes:

(A) log time  
(B) \( n \log n \) time  
(C) linear time  
(D) constant time

213. Removing the first item from a singly-linked list with a tail pointer takes:

(A) \( n \log n \) time  
(B) linear time  
(C) constant time  
(D) log time

214. In a singly-linked list, you can move the tail pointer back one node in:

(A) linear time  
(B) \( n \log n \) time  
(C) constant time  
(D) log time

215. Suppose you have a pointer to a node in the middle of a singly-linked list. You can then delete that node in:

(A) constant time  
(B) linear time  
(C) \( n \log n \) time  
(D) log time

Concept: *doubly-linked lists (insertions)*

216. Appending to a non-circular, doubly-linked list without a tail pointer takes:

(A) log time  
(B) \( n \log n \) time  
(C) constant time  
(D) linear time

217. Appending to a non-circular, doubly-linked list with a tail pointer takes:

(A) constant time  
(B) linear time  
(C) log time  
(D) \( n \log n \) time

218. Removing the first item from a non-circular, doubly-linked list without a tail pointer takes:

(A) log time  
(B) constant time  
(C) \( n \log n \) time  
(D) linear time

219. Suppose you have a pointer to a node in the middle of a doubly-linked list. You can then insert a new node just after in:

(A) linear time  
(B) constant time  
(C) log time  
(D) \( n \log n \) time

220. Suppose you have a pointer to a node in the middle of a doubly-linked list. You can then insert a new node just prior with as few pointer assignments as:

(A) 2  
(B) 3  
(C) 5  
(D) 1  
(E) 4
221. **T : F**: Making a doubly-linked list circular removes the need for a separate tail pointer.

**Concept: doubly-linked lists (deletions)**

222. Removing the first item from a doubly-linked list with a tail pointer takes:

- (A) constant time
- (B) \( n \log n \) time
- (C) \log\, time
- (D) linear time

223. In a doubly-linked list, you can move the tail pointer back one node in:

- (A) linear time
- (B) constant time
- (C) \log\, time
- (D) \( n \log n \) time

224. In a doubly-linked list, what does a tail-pointer gain you?

- (A) the ability to remove the first element of list in constant time
- (B) the ability to prepend the list in constant time
- (C) the ability to remove the last element of list in constant time
- (D) the ability to append the list in constant time
- (E) the ability to both prepend and remove the first element of list in constant time
- (F) the ability to both append and remove the last element of list in constant time

**Concept: input-output order**

225. These values are pushed onto a stack in the order given: 1 5 9. A **pop** operation would return which value?

- (A) 5
- (B) 9
- (C) 1

226. LIFO ordering is the same as:

- (A) FIFO
- (B) FILO
- (C) LILO

**Concept: time and space complexity**

227. Consider a stack based upon a fillable array with pushes onto the back of the array. What is the time complexity of the worst case behavior for **push** and **pop**, respectively? You may assume there is sufficient space for the **push** operation.

- (A) linear and constant
- (B) constant and linear
- (C) constant and constant
- (D) linear and linear

228. Consider a stack based upon a circular array with pushes onto the front of the array. What is the time complexity of the worst case behavior for **push** and **pop**, respectively? You may assume there is sufficient space for the **push** operation.

- (A) linear and constant
- (B) constant and constant
- (C) linear and linear
- (D) constant and linear

229. Consider a stack based upon a dynamic array with pushes onto the back of the array. What is the time complexity of the worst case behavior for **push** and **pop**, respectively? You may assume there is sufficient space for the **push** operation and that the array never shrinks.

- (A) constant and constant
- (B) constant and linear
- (C) linear and linear
- (D) linear and constant
230. Consider a stack based upon a dynamic array with pushes onto the back of the array. What is the time complexity of the worst case behavior for push and pop, respectively? You may assume there is sufficient space for the push operation and that the array may shrink.

(A) linear and linear  
(B) constant and constant  
(C) constant and linear  
(D) linear and constant

231. Consider a stack based upon a dynamic circular array with pushes onto the front of the array. What is the time complexity of the worst case behavior for push and pop, respectively? You may assume the array may grow or shrink.

(A) constant and linear  
(B) linear and linear  
(C) linear and constant  
(D) constant and constant

232. Consider a stack based upon a singly-linked list without a tail pointer with pushes onto the front of the list. What is the time complexity of the worst case behavior for push and pop, respectively?

(A) linear and linear  
(B) linear and constant  
(C) constant and constant  
(D) constant and linear

233. Consider a stack based upon a singly-linked list with a tail pointer with pushes onto the front of the list. What is the time complexity of the worst case behavior for push and pop, respectively?

(A) linear and linear  
(B) constant and linear  
(C) constant and constant  
(D) linear and constant

234. Consider a stack based upon a non-circular, doubly-linked list without a tail pointer with pushes onto the front of the list. What is the time complexity of the worst case behavior for push and pop, respectively?

(A) linear and constant  
(B) constant and linear  
(C) constant and constant  
(D) linear and linear

235. Consider a stack based upon a doubly-linked list with a tail pointer with pushes onto the front of the list. What is the time complexity of the worst case behavior for push and pop, respectively?

(A) constant and constant  
(B) linear and linear  
(C) linear and constant  
(D) constant and linear

236. Suppose a simple fillable array with capacity c is used to implement two stacks, one growing from each end. The stack sizes at any given time are stored in i and j, respectively. If maximum space efficiency is desired, a reliable condition for the stacks being full is:

(A) i == c/2 || j == c/2  
(B) i == c/2 && j == c/2  
(C) i == c/2-1 || j == c/2-1  
(D) i + j == c-2  
(E) i + j == c  
(F) i == c/2-1 && j == c/2-1

Concept: stack applications

For the following questions, assume the tokens in a post-fix equation are processed with the following code, with all functions having their obvious meanings and integer division.

```java
s.push(readEquationToken());
s.push(readEquationToken());
while (moreEquationTokens())
{
    t = readEquationToken();
    if (isNumber(t))
        s.push(t);
    else /* t must be an operator */
    {
        operandB = s.pop();
        operandA = s.pop();
        result = performOperation(t, operandA, operandB);
    }
}```
s.push(result);
}
}

237. If the tokens of the postfix equation \[8 \ 2 \ 3 \ - \ / \ 2 \ 3 \ * \ + \ 5 \ 1 \ * \ -\] are read in the order given, what are the top two values in \(s\) immediately after the result of the first multiplication is pushed?

(A) 1 6  
(B) 3 3  
(C) 5 6  
(D) 1 2

Concept: \textit{input-output order}

238. These values are enqueued onto a queue in the order given: 1 5 9 4. A dequeue operation would return which value?

(A) 9  
(B) 4  
(C) 5  
(D) 1

239. FIFO ordering is the same as:

(A) FILO  
(B) LIFO  
(C) LILO  
(D) LIFO

Concept: \textit{complexity}

240. Consider a queue based upon a simple fillable array with enqueues onto the front of the array. What is the time complexity of the worst case behavior for \textit{enqueue} and \textit{dequeue}, respectively? Assume there is room for the operations.

(A) linear and linear  
(B) linear and constant  
(C) constant and constant  
(D) constant and linear

241. Consider a queue based upon a circular array with enqueues onto the front of the array. What is the time complexity of the worst case behavior for \textit{enqueue} and \textit{dequeue}, respectively? Assume there is room for the operations.

(A) constant and constant  
(B) constant and linear  
(C) linear and linear  
(D) linear and constant

242. Consider a queue based upon a singly-linked list without a tail pointer with enqueues onto the front of the list. What is the time complexity of the worst case behavior for \textit{enqueue} and \textit{dequeue}, respectively?

(A) linear and linear  
(B) linear and constant  
(C) constant and linear  
(D) constant and linear

243. Consider a queue based upon a singly-linked list with a tail pointer with enqueues onto the front of the list. What is the time complexity of the worst case behavior for \textit{enqueue} and \textit{dequeue}, respectively?

(A) constant and constant  
(B) linear and linear  
(C) linear and constant  
(D) constant and linear

244. Consider a queue based upon a doubly-linked list with a tail pointer with enqueues onto the front of the list. What is the time complexity of the worst case behavior for \textit{enqueue} and \textit{dequeue}, respectively?

(A) linear and linear  
(B) constant and linear  
(C) constant and constant  
(D) linear and constant

245. Consider a queue based upon a non-circular, doubly-linked list without a tail pointer with enqueues onto the front of the list. What is the time complexity of the worst case behavior for \textit{enqueue} and \textit{dequeue}, respectively?

(A) linear and constant  
(B) linear and linear  
(C) constant and constant  
(D) constant and linear
Concept: **complexity**

246. Consider a worst-case binary search tree with \( n \) nodes. What is the average case time complexity for finding a value at a leaf?

(A) \( n \log n \)  
(B) constant  
(C) linear  
(D) \( \log n \)  
(E) \( \sqrt{n} \)  
(F) quadratic

247. Consider a binary search tree with \( n \) nodes. What is the worst case time complexity for finding a value at a leaf?

(A) linear  
(B) \( \log n \)  
(C) \( \sqrt{n} \)  
(D) quadratic  
(E) \( n \log n \)  
(F) constant

248. Consider a binary search tree with \( n \) nodes. What is the minimum and maximum height (using order notation)?

(A) \( \log n \) and \( \log n \)  
(B) linear and linear  
(C) constant and \( \log n \)  
(D) \( \log n \) and linear  
(E) constant and linear

Concept: **balance**

249. Which ordering of input values builds the most unbalanced BST? Assume values are inserted from left to right.

(A) 1 2 3 4 5 7 6  
(B) 1 7 2 6 3 5 4  
(C) 4 3 1 6 2 8 7

250. Which ordering of input values builds the most balanced BST? Assume values are inserted from left to right.

(A) 1 2 7 6 0 3 8  
(B) 4 3 1 6 2 8 7  
(C) 1 4 3 2 5 7 6

Concept: **tree shapes**

251. What is the best definition of a perfect binary tree?

(A) all null children are equidistant from the root  
(B) all leaves are equidistant from the root  
(C) all leaves have zero children  
(D) all nodes have zero or two children

252. Suppose a binary tree has 10 leaves. How many nodes in the tree must have two children?

(A) 9  
(B) no limit  
(C) 10  
(D) 7  
(E) 8

253. Suppose a binary tree has 10 nodes. How many nodes are children of some other node in the tree?

(A) 9  
(B) 7  
(C) 10  
(D) 8  
(E) no limit

254. Let P0, P1, and P2 refer to nodes that have zero, one or two children, respectively. Using the generally accepted definition, what is a **full** binary tree?

(A) all leaves are equidistant from the root  
(B) all interior nodes P1, except the root  
(C) all nodes are P2  
(D) all interior nodes are P2; all leaves are equidistant from the root  
(E) all interior nodes are P2  
(F) all interior nodes are P1
255. Let P0, P1, and P2 refer to nodes that have zero, one or two children, respectively. Using the generally accepted definition, what is a perfect binary tree?

(A) all interior nodes P1, except the root
(B) all interior nodes are P1
(C) all leaves are equidistant from the root
(D) all interior nodes are P2; all leaves are equidistant from the root
(E) all interior nodes are P2
(F) all nodes are P0 or P2

256. Let P0, P1, and P2 refer to nodes that have zero, one or two children, respectively. Using the generally accepted definition, what is a degenerate binary tree?

(A) all interior nodes are P2
(B) all leaves are equidistant from the root
(C) all interior nodes are P2; all leaves are equidistant from the root
(D) all nodes are P0 or P2
(E) all interior nodes P1, except the root
(F) all interior nodes are P1

257. Let P0, P1, and P2 refer to nodes that have zero, one or two children, respectively. Using the generally accepted definition of a complete binary tree, which of the following actions can be used to make any complete tree? Assume the leftmost and rightmost sets may be empty.

(A) another name for a perfect tree
(B) making the leftmost leaf of a perfect tree P1
(C) making the leftmost leaves of a perfect tree P2
(D) making the leftmost leaves of a perfect tree P1 or P2
(E) removing the rightmost leaves from a perfect tree

258. T or F: All perfect trees are full trees.

259. T or F: All full trees are complete trees.

260. T or F: All complete trees are perfect trees.

261. How many distinct binary trees can be formed from exactly two nodes with values 1, 2, or 3 respectively (hint: think about how many permutations of values there are for each tree shape)?

(A) 5
(B) 1
(C) 4

(D) 3
(E) 2

262. How many distinct binary tree shapes can be formed from exactly two nodes?

(A) 5
(B) 4
(C) 2

(D) 3
(E) 1

263. Let k be the the number of steps from the root to a leaf in a perfect tree. What are the number of nodes in the tree?

(A) \(2^{k+1}\)
(B) \(2^k - 1\)
(C) \(2^{k-1} - 1\)

(D) \(2^{k+1} - 1\)

264. Let k be the the number of steps from the root to the furthest leaf in a binary tree. What would be the minimum number of nodes in such a tree? Assume k is a power of two.

(A) \(2^{k+1} - 1\)
(B) \(k\)
(C) \((\log k) + 1\)

(D) \(\log k\)
(E) \(k + 1\)
(F) \(2^{k+1}\)
265. Let \( k \) be the number of steps from the root to the furthest leaf in a binary tree. What would be the maximum number of nodes in such a tree? Assume \( k \) is a power of two.

(A) \( 2^{k+1} - 1 \) 
(B) \( k \) 
(C) \( 2^{k+1} \) 
(D) \( k + 1 \) 
(E) \( \log k \) 
(F) \( (\log k) + 1 \)

Concept: ordering in a BST

266. For all child nodes in a BST, what relationship holds between the value of a left child node and the value of its parent? Assume unique values.

(A) there is no relationship 
(B) less than 
(C) greater than

267. For all sibling nodes in a BST, what relationship holds between the value of a left child node and the value of its sibling? Assume unique values.

(A) there is no relationship 
(B) greater than 
(C) less than

268. Which statement is true about the successor of a node in a BST, if it exists?

(A) it is always an interior node 
(B) it may be an ancestor 
(C) it is always a leaf node 
(D) has no right child 
(E) has no left child

269. Consider a node which holds neither the smallest or the largest value in a BST. Which statement is true about the node which holds the next higher value of a node in a BST, if it exists?

(A) it may be an ancestor 
(B) it is always an interior node 
(C) has no left child 
(D) it is always a leaf node 
(E) has no right child

Concept: traversals

270. Consider printing out the node values of a binary tree with 25 nodes to the left of the root and 38 nodes to the right. How many nodes are processed before the root’s value is printed in a pre-order traversal?

(A) 38 
(B) 25 
(C) 53 
(D) none of the other answers are correct 
(E) 0 
(F) 54

271. Consider printing out the node values of a binary tree with 25 nodes to the left of the root and 38 nodes to the right. How many nodes are processed before the root’s value is printed in an in-order traversal?

(A) none of the other answers are correct 
(B) 53 
(C) 0 
(D) 38 
(E) 25 
(F) 54

272. Consider printing out the node values of a binary tree with 25 nodes to the left of the root and 38 nodes to the right. How many nodes are processed before the root’s value is printed in a post-order traversal?

(A) 38 
(B) none of the other answers are correct 
(C) 0 
(D) 54 
(E) 25 
(F) 63
273. Consider a perfect BST with even values 0 through 12, to which the value 7 is then added. Which of the following is an in-order traversal of the resulting tree?

(A) 7 0 2 4 6 8 10 12 
(B) 12 10 8 7 6 4 2 0 
(C) 0 2 4 6 8 10 12 7 
(D) 0 4 2 7 8 10 12 6 
(E) 0 2 4 6 7 8 10 12 
(F) 6 2 10 0 4 8 12 7

274. Consider a perfect BST with even values 0 through 12, to which the value 7 is then added. Which of the following is a level-order traversal of the resulting tree?

(A) 0 2 4 6 8 10 12 7 
(B) 0 2 4 6 7 8 10 12 
(C) 12 10 8 7 6 4 2 0 
(D) 0 4 2 7 8 10 12 6 
(E) 7 0 2 4 6 8 10 12 
(F) 6 2 10 0 4 8 12 7

275. Consider a level-order traversal of C B A D F E and an in-order traversal of B C A F D E. Do these traversals generate a unique tree and, if so, what is that tree’s pre-order traversal?

(A) yes, but the correct answer is not listed 
(B) yes, C B A F D E 
(C) no 
(D) yes, C B A D F E 
(E) yes, C A D B E F

276. Consider an in-order traversal of B C A F D E and a pre-order traversal of C B A D F E. Do these traversals generate a unique tree and, if so, what is that tree’s post-order traversal?

(A) yes, but the correct answer is not listed 
(B) yes, F A D B E C 
(C) yes, B F A E D C 
(D) no 
(E) yes, B F E D A C

277. Consider an in-order traversal of B C A F D E and a post-order traversal of C B A D F E. Do these traversals generate a unique tree and, if so, what is that tree’s level-order traversal?

(A) yes, E A C F B D 
(B) yes, E F C D B A 
(C) no 
(D) yes, but the correct answer is not listed 
(E) yes, E F A D B C

278. Consider a level-order traversal of C F D E B A and an pre-order traversal of C F E A D B. Do these traversals generate a unique tree and, if so, what is that tree’s in-order traversal?

(A) no 
(B) yes, F A E C B D 
(C) yes, but the correct answer is not listed 
(D) yes, F E A C D B 
(E) yes, F A E C B D

Concept: *insertion and deletion*

279. T or F: Suppose you are given an in-order traversal of an unbalanced BST. If you were to insert those values into an empty BST in the order given, would the result be a balanced tree?

280. T or F: Suppose you are given a pre-order traversal of an unbalanced BST. If you were to insert those values into an empty BST in the order given, would the result be a balanced tree?

281. T or F: Suppose you are given an in-order traversal of a balanced BST. If you were to insert those values into an empty BST in the order given, would the result be a balanced tree?

282. T or F: Suppose you are given a pre-order traversal of a balanced BST. If you were to insert those values into an empty BST in the order given, would the result be a balanced tree?
283. Suppose 10 values are inserted into an empty BST. What is the minimum and maximum resulting heights of the tree? The height is the number of steps from the root to the furthest leaf.

(A) 5 and 10  
(B) 3 and 9  
(C) 4 and 9  
(D) 4 and 10  
(E) 5 and 9  
(F) 3 and 10

284. Which, if any, of these deletion strategies for non-leaf nodes reliably preserve BST ordering?

(i) Swap the values of the node to be deleted and the smallest leaf node with a larger value, then remove the leaf.

(ii) Swap the values of the node to be deleted with its predecessor or successor. If the predecessor or successor is a leaf, remove it. Otherwise, repeat the process.

(iii) If the node to be deleted does not have two children, simply connect the parent’s child pointer to the node to the node’s child pointer, otherwise, use a correct deletion strategy for nodes with two children.

(A) ii  
(B) all  
(C) iii  
(D) i and ii  
(E) i and iii  
(F) none  
(G) ii and iii  
(H) i

Concept: heap shapes

285. In a heap, the upper bound on the number of leaves is:

(A) $O(n \log n)$  
(B) $O(\log n)$  
(C) $O(n)$  
(D) $O(1)$

286. In a heap, the distance from the root to the furthest leaf is:

(A) $\theta(n \log n)$  
(B) $\theta(1)$  
(C) $\theta(\log n)$  
(D) $\theta(n)$

287. In a heap, let $d_f$ be the distance of the furthest leaf from the root and let $d_c$ be the analogous distance of the closest leaf. What is $d_f - d_c$, at most?

(A) $\theta(\log n)$  
(B) 0  
(C) 1  
(D) 2

288. What is the most number of nodes in a heap with a single child?

(A) 1  
(B) $\Theta(n)$  
(C) 0  
(D) 2  
(E) $\Theta(\log n)$

289. What is the fewest number of nodes in a heap with a single child?

(A) one per level  
(B) 2  
(C) 0  
(D) 1

290. T or F: There can be two or more nodes in a heap with exactly one child.

291. T or F: A heap can have no nodes with exactly one child.

292. T or F: All heaps are perfect trees.

293. T or F: No heaps are perfect trees.

294. T or F: All heaps are complete trees.

295. T or F: No heaps are complete trees.
296. **T** or **F**: A binary tree with one node must be a heap.

297. **T** or **F**: A binary tree with two nodes and with the root having the smallest value must be a min-heap.

298. **T** or **F**: If a node in a heap is a right child and has two children, then its sibling must also have two children.

299. **T** or **F**: If a node in a heap is a right child and has one child, then its sibling must also have one child.

**Concept: heap ordering**

300. In a min-heap, what is the relationship between a parent and its left child?
   (A) the parent has a larger value  
   (B) the parent has the same value  
   (C) the parent has a smaller value  
   (D) there is no relationship between their values

301. In a min-heap, what is the relationship between a left child and its sibling?
   (A) there is no relationship between their values  
   (B) the left child has a smaller value  
   (C) the right child has a larger value  
   (D) both children cannot have the same value

302. **T** or **F**: A binary tree with three nodes and with the root having the smallest value and two children must be a min heap.

303. **T** or **F**: The largest value in a max-heap can be found at the root.

304. **T** or **F**: The largest value in a min-heap can be found at the root.

305. **T** or **F**: The largest value in a min-heap can be found at a leaf.

**Concept: heaps stored in arrays**

306. How would this heap be stored in an array?

```
     2
  10   4
 11 13  5 15
21 12
```

   (A) [2, 10, 4, 11, 13, 5, 15, 21, 12]  
   (B) [2, 10, 11, 21, 12, 13, 4, 5, 15]  
   (C) [21, 11, 12, 10, 13, 2, 5, 4, 15]  
   (D) [2, 4, 5, 10, 11, 12, 13, 15, 21]

307. Printing out the values in the array yield what kind of traversal of the heap?
   (A) post-order  
   (B) in-order  
   (C) level-order  
   (D) pre-order

308. Suppose the heap has $n$ values. The root of the heap can be found at which index?
   (A) $n-1$  
   (B) $n$  
   (C) 0  
   (D) 1

309. Suppose the heap has $n$ values. The left child of the root can be found at which index?
   (A) 1  
   (B) 2  
   (C) 0  
   (D) $n-2$  
   (E) $n$  
   (F) $n-1$
310. Left children in a heap are stored at what kind of indices?

(A) a roughly equal mix of odd and even          (D) all even but one
(B) all odd                                      (E) all even
(C) all odd but one                              

311. Right children in a heap are stored at what kind of indices?

(A) all odd                                      (D) a roughly equal mix of odd and even
(B) all even                                     (E) all even but one
(C) all odd but one                              

312. The formula for finding the left child of a node stored at index $i$ is:

(A) $i \times 2 + 1$                            (C) $i \times 2$
(B) $i \times 2 - 1$                            (D) $i \times 2 + 2$

313. The formula for finding the right child of a node stored at index $i$ is:

(A) $i \times 2$                                (C) $i \times 2 - 1$
(B) $i \times 2 + 1$                            (D) $i \times 2 + 2$

314. The formula for finding the parent of a node stored at index $i$ is:

(A) $(i + 2)/2$                                 (C) $(i - 1)/2$
(B) $i/2$                                      (D) $(i + 1)/2$

315. If the array uses one-based indexing, the formula for finding the left child of a node stored at index $i$ is:

(A) $i \times 2 - 1$                            (C) $i \times 2 + 2$
(B) $i \times 2 + 1$                            (D) $i \times 2$

316. If the array uses one-based indexing, the formula for finding the right child of a node stored at index $i$ is:

(A) $i \times 2 - 1$                            (C) $i \times 2$
(B) $i \times 2 + 1$                            (D) $i \times 2 + 2$

317. If the array uses one-based indexing, the formula for finding the parent of a node stored at index $i$ is:

(A) $(i + 2)/2$                                 (C) $(i - 1)/2$
(B) $(i + 1)/2$                                 (D) $i/2$

318. Consider a trinary heap stored in an array. The formula for finding the left child of a node stored at index $i$ is:

(A) $i \times 3 - 2$                            (D) $i \times 3 + 2$
(B) $i \times 3 + 1$                            (E) $i \times 3 - 1$
(C) $i \times 3 + 3$                            (F) $i \times 3$

319. Consider a trinary heap stored in an array. The formula for finding the middle child of a node stored at index $i$ is:

(A) $i \times 3$                                (D) $i \times 3 - 2$
(B) $i \times 3 - 1$                            (E) $i \times 3 + 3$
(C) $i \times 3 + 1$                            (F) $i \times 3 + 2$

320. Consider a trinary heap stored in an array. The formula for finding the right child of a node stored at index $i$ is:

(A) $i \times 3$                                (D) $i \times 3 - 1$
(B) $i \times 3 + 3$                            (E) $i \times 3 + 1$
(C) $i \times 3 - 2$                            (F) $i \times 3 + 2$
321. Consider a trinary heap stored in an array. The formula for finding the parent of a node stored at index \( i \) is:

(A) \( (i-2)/3 \)  \hspace{1cm} (D) \( i/3 + 1 \)
(B) \( (i+2)/3 \)  \hspace{1cm} (E) \( i/3 - 1 \)
(C) \( (i+1)/3 \)  \hspace{1cm} (F) \( (i-1)/3 \)

Concept: heap operations

322. In a max-heap with no knowledge of the minimum value, the minimum value can be found in time:

(A) \( \theta(n) \)  \hspace{1cm} (C) \( \theta(\log n) \)
(B) \( \theta(1) \)  \hspace{1cm} (D) \( \theta(n \log n) \)

323. Suppose a min-heap with \( n \) values is stored in an array \( a \). In the \textit{extractMin} operation, which element immediately replaces the root element (prior to this new root being sifted down).

(A) \( a[n-1] \)  \hspace{1cm} (C) \( a[1] \)
(B) \( a[2] \)  \hspace{1cm} (D) the minimum of \( a[1] \) and \( a[2] \)

324. The \textit{findMin} operation in a min-heap takes how much time?

(A) \( \Theta(n \log n) \)  \hspace{1cm} (C) \( \Theta(\log n) \)
(B) \( \Theta(n) \)  \hspace{1cm} (D) \( \Theta(1) \)

325. The \textit{extractMin} operation in a min-heap takes how much time?

(A) \( \Theta(1) \)  \hspace{1cm} (C) \( \Theta(\log n) \)
(B) \( \Theta(n) \)  \hspace{1cm} (D) \( \Theta(n \log n) \)

326. Merging two heaps of size \( n \) and \( m \), \( m < n \) takes how much time?

(A) \( \Theta(\log n \log m) \)  \hspace{1cm} (D) \( \Theta(n \times m) \)
(B) \( \Theta(n + m) \)  \hspace{1cm} (E) \( \Theta(\log n \log m) \)
(C) \( \Theta(m \log n) \)  \hspace{1cm} (F) \( \Theta(n \log m) \)

327. The \textit{insert} operation takes how much time?

(A) \( \Theta(1) \)  \hspace{1cm} (C) \( \Theta(\log n) \)
(B) \( \Theta(n \log n) \)  \hspace{1cm} (D) \( \Theta(n) \)

328. Turning an unordered array into a heap takes how much time?

(A) \( \Theta(1) \)  \hspace{1cm} (C) \( \Theta(n \log n) \)
(B) \( \Theta(n \log n) \)  \hspace{1cm} (D) \( \Theta(\log n) \)

329. Suppose the values 21, 15, 14, 10, 8, 5, and 2 are inserted, one after the other, into an empty min-heap. What does the resulting heap look like? Heap properties are maintained after every insertion.

(A) \( w \)  \hspace{1cm} (C) \( x \)
(B) \( z \)  \hspace{1cm} (D) \( y \)

36
330. Using the standard \textit{buildHeap} operation to turn an unordered array into a \textit{max}-heap, how many parent-child swaps are made if the initial unordered array is $[5, 21, 8, 15, 25, 3, 9]$?

(A) 6  
(B) 3  
(C) 2  
(D) 4  
(E) 7  
(F) 5