Prerequisite Concepts for Analysis of Algorithms

Basic Data Structures (Version 7)

Name: ____________________________
Email: ____________________________

Code: 80530

Concept: mathematics notation

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

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

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

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

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

6. \( 2^{\log_2 n} \) is equal to:
   (A) \( 2^n \)
   (B) \( n \)
   (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) little omega
   (B) big omega
   (C) theta
   (D) big omicron
   (E) little omicron

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

9. \( \log 2^n \) is \( \Theta(?) \).
   (A) \( \log n \)
   (B) \( n \)
   (C) \( 2^n \)
   (D) \( n \log n \)
10. The number of permutations of a list of \( n \) items is:

(A) \( n \)  
(B) \( n \log n \)  
(C) \( \log n \)  
(D) \( n! \)  
(E) \( 2^n \)

**Concept: relative growth rates**

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

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

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) \( f < h < g \)  
(B) \( f < g < h \)  
(C) \( h < g < f \)  
(D) \( h < f < g \)  
(E) \( g < h < f \)  
(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) \( f > g > h \)  
(B) \( g > f > h \)  
(C) \( f > h > g \)  
(D) \( h > g > f \)  
(E) \( g > h > f \)  
(F) \( h > f > g \)

**Concept: order notation**

14. What does big Omicron roughly mean?

(A) always better than  
(B) worse than or equal  
(C) better than or equal  
(D) always worse than  
(E) the same as

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

(A) worse than or equal  
(B) always worse than  
(C) the same as  
(D) always better than  
(E) better than or equal

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

(A) the same as  
(B) always better than  
(C) always worse than  
(D) worse than or equal  
(E) better than or equal

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 $O(n^\ast)$.

23. Consider sorting 1,000,000 numbers with mergesort. What is the time complexity of this operation? [THINK!]

(A) constant, because $n$ is fixed

(B) $n \log n$, because mergesort takes $n \log n$ time

(C) $n^2$, because mergesort takes quadratic time

Concept: comparing algorithms using order notation

Consider the worst case behavior 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 in a practical sense 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 $f$ runs slower than $g$.

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

26. T or F: If $f = \Theta(g)$, then algorithm $f$ runs in time equal to $g$.

27. T or F: If $f = o(g)$, then algorithm $f$ runs faster than as $g$, regardless of input size.

28. T or F: If $f = \Omega(g)$, then algorithm $f$ runs faster than or the same time to $g$, in all cases.

29. T or F: If $f = O(g)$, then there exists a problem size above which $f$ is always runs slower than or the same time as $g$, even in the best of cases.

30. T or F: If $f = O(g)$, then there exists a problem size above which $f$ always runs faster than or the same time as $g$.

31. T or F: If $f = o(g)$, then there exists a problem size above which $f$ always runs faster than $g$.

32. T or F: If $f = \Omega(g)$, then there exists a problem size above which $f$ always runs faster than or the same time as $g$.

33. T or F: If $f = o(g)$, then there exists a problem size above which $f$ always runs faster than or equal to $g$.

34. T or F: If $f = o(g)$, then there exists a problem size above which $f$ is always runs slower than $g$, even in the best of cases.

35. T or F: If $f = \Omega(g)$, then $f$ and $g$ can be the same algorithm.

36. T or F: If $f = o(g)$, then $f$ and $g$ can be the same algorithm.

37. T or F: Suppose algorithm $f = \theta(g)$. Algorithm $f$ is never slower than $g$, in a practical sense.

38. T or F: Suppose algorithm $f = \theta(g)$. $f$ and $g$ can be the same algorithm.

Consider proving that $3n^2 - 2n + 7 = \omega(n)$ using L’Hôpital’s rule.

39. Finish this theorem: $f \equiv \omega(g)$ if and only if:

(A) $\lim_{n \to \infty} \frac{g(x)}{f(n)} = \infty$

(B) $\lim_{n \to \infty} \frac{f(n)}{g(x)} = 0$

(C) $\lim_{n \to \infty} \frac{f(x)}{g(n)} = C$

(D) $\lim_{n \to \infty} \frac{g(n)}{f(x)} = C$

(E) $\lim_{n \to \infty} \frac{f(x)}{g(n)} = 0$

(F) $\lim_{n \to \infty} \frac{g(n)}{f(x)} = \infty$
40. After the first application of L'Hôpital's rule, we have:

(A) \( \lim_{n \to \infty} \frac{n^3 - n^2 + 7n}{1} \)
(B) \( \lim_{n \to \infty} \frac{8n^2 - 2}{1} \)
(C) \( \lim_{n \to \infty} \frac{n^3 - n^2 + 7n}{1} \)
(D) \( \lim_{n \to \infty} \frac{1}{8n^2 - 2} \)

41. After the second application of L'Hôpital's rule, we have:

(A) \( \lim_{n \to \infty} \frac{1}{4n^2 - \frac{n^2}{2} + \frac{7n}{2}} \)
(B) 0
(C) negative infinity
(D) You can't apply L'Hôpital's rule in this case
(E) \( \lim_{n \to \infty} \frac{n^4 - 2n^2 + 7n}{2} \)
(F) infinity

42. How many applications of L'Hôpital's rule are needed to complete the proof of \( f = o(g) \), where \( f = 5n^3 + 2n^2 + 4 \) and \( g = 6n^4 + 8n^2 + 7n \)? You should apply L'Hôpital's rule as many times as possible.

(A) 2
(B) an infinite number
(C) 4
(D) 3
(E) 1
(F) 0

Concept: analyzing code

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.

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

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

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

44. 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) \( \theta(n^3) \)
(B) \( \theta(\log n) \)
(C) \( \theta(\log^2 n) \)
(D) \( \theta(n) \)
(E) \( \theta(n \sqrt{n}) \)
(F) \( \theta(n \log n) \)

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

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

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

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

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

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

```java
step = sqrt(n);
for (i from 0 until n by step)
    println(i);
```

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

48. 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+sqrt(n), n);
        println(i);
    }
}
```

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

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

```java
step = sqrt(n);
for (i from 0 until n by step)
    println(i);
```

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

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

(A) \( \theta(n^{\frac{2}{3}}) \)  (D) \( \theta(n\sqrt{n}) \)
(B) \( \theta(1) \)  (E) \( \theta(n) \)
(C) \( \theta(\sqrt{n}) \)  (F) \( \theta(n – \sqrt{n}) \)

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

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

(A) \( \theta(n – \sqrt{n}) \)  (D) \( \theta(n^{\frac{2}{3}}) \)
(B) \( \theta(1) \)  (E) \( \theta(\sqrt{n}) \)
(C) \( \theta(n\sqrt{n}) \)  (F) \( \theta(n) \)

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

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

(A) \( \theta(n) \)  (D) \( \theta(\sqrt{n}) \)
(B) \( \theta(1) \)  (E) \( \theta(n – \sqrt{n}) \)
(C) \( \theta(n\sqrt{n}) \)  (F) \( \theta(n^{\frac{2}{3}}) \)

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

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

(A) \( \theta(\sqrt{n}) \)  (D) \( \theta(n – \sqrt{n}) \)
(B) \( \theta(n\sqrt{n}) \)  (E) \( \theta(n) \)
(C) \( \theta(n^{\frac{2}{3}}) \)  (F) \( \theta(1) \)
54. What is the space complexity of this function? Assume the initial value of \( i \) is one.

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

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

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

\[
\text{for (} i \text{ from 0 until } n \text{ by 1) } \\
\quad \text{for (} j \text{ from 0 until } n \text{ by 1) } \\
\quad \text{println}(i, j); \\
\]

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

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

\[
\text{function } f(i, j, n) \\
\{ \\
\quad \text{println}(i, j); \\
\quad \text{if (} i < n \text{)} \\
\quad \{ \\
\quad \quad \text{if (} j < n \text{)} \\
\quad \quad \quad f(i, j+1, n); \\
\quad \quad \quad \text{else} \\
\quad \quad \quad f(i+1, 0, n); \\
\quad \quad \} \\
\quad \} \\
\]

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

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

\[
\text{for (} i \text{ from 0 until } n \text{ by 1) } \\
\quad \text{for (} j \text{ from 0 until } n \text{ by 1) } \\
\quad \text{println}(i, j); \\
\]

(A) \( \theta(1) \)  
(B) \( \theta(n^2) \)  
(C) \( \theta(n\log n) \)  
(D) \( \theta(\sqrt{n}) \)  
(E) \( \theta(\log^2 n) \)  
(F) \( \theta(n) \)
58. 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 < n)
                    f(i,j+1,n);
                else
                    f(i+1,0,n);
            }
        println(i,j);
    }
```

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

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

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

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

60. 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 < n)
                    f(i,j+1,n);
                else
                    f(i+1,i+1,n);
            }
        println(i,j);
    }
```

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

61. 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) $\theta(n \sqrt{n})$  \hspace{1cm} (D) $\theta(n^2)$
(B) $\theta(n \log n)$  \hspace{1cm} (E) $\theta(n)$
(C) $\theta(\log^2 n)$  \hspace{1cm} (F) $\theta(1)$
62. 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, 0, n);
    }
    println(i, j);
}
```

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

63. 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) \(\theta(n \log n)\)  
(C) \(\theta(n)\)  
(D) \(\theta(\log^2 n)\)  
(E) \(\theta(n \sqrt{n})\)  
(F) \(\theta(n^2)\)

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

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

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

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

```java
for (i from 0 until n by 2)
    println(i);
for (j from 0 until n by 1)
    println(j);
```

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

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

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

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

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

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

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

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

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

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

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

(A) $\theta(\log n)$  (D) $\theta(n^{\sqrt{n}})$
(B) $\theta(1)$  (E) $\theta(n^2)$
(C) $\theta(n \log n)$  (F) $\theta(n)$
Concept: analysis of classic, simple algorithms

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

(A) \( \theta\left(\frac{n}{\sqrt{n}}\right) \)  
(B) \( \theta(\Phi) \)  
(C) \( \theta(\sqrt{n}) \)  
(D) \( \theta(n - \sqrt{n}) \)  

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

(A) \( \theta(\Phi) \)  
(B) \( \theta(1) \)  
(C) \( \theta(\sqrt{n}) \)  
(D) \( \theta(n - \sqrt{n}) \)  

75. Which of the following describes iterative fibonacci’s time complexity?

(A) \( \theta(\sqrt{n}) \)  
(B) \( \theta(n - \sqrt{n}) \)  
(C) \( \theta\left(\frac{\Phi}{n}\right) \)  
(D) \( \theta(1) \)  

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

(A) \( \theta\left(\frac{n}{\sqrt{n}}\right) \)  
(B) \( \theta(n) \)  
(C) \( \theta(\sqrt{n}) \)  
(D) \( \theta(n - \sqrt{n}) \)  

Concept: searching

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

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

(A) yes  
(B) no

78. Does the following code set the variable \( max \) to the maximum value in an unsorted, non-empty array?

```javascript
max = array[0]
for (i from 0 to array.length)
    if (array[i] > max)
        max = a[i]
```

(A) yes  
(B) no

79. Does the following function always return \textbf{True} if the value of \texttt{item} is \textit{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;
}
```

(A) no  
(B) yes
80. Does the following function always return **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;
}
```

(A) yes  
(B) no

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

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

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

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

**Concept: sorting**

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

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

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

86. 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  
(B) bubble sort  
(C) mergesort  
(D) heapsort  
(E) insertion sort  
(F) selection sort

87. Which sort is an optimized version of bubble sort?

(A) heapsort  
(B) mergesort  
(C) insertion sort 
(D) quicksort  
(E) stooge sort  
(F) selection sort
Concept: *space and time complexity*

88. What is the best time case complexity for merge sort?
   - (A) quadratic
   - (B) $n \log n$
   - (C) $\log n$
   - (D) cubic
   - (E) linear

89. What is the worst case complexity for merge sort?
   - (A) $\log n$
   - (B) quadratic
   - (C) linear
   - (D) $n \log n$
   - (E) cubic

90. 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) linear
   - (B) log linear
   - (C) exponential
   - (D) quadratic

91. If quicksort is implemented such that a random element is chosen to be the pivot, the worst case behavior of the sort is:
   - (A) quadratic
   - (B) linear
   - (C) exponential
   - (D) log linear

92. What is the best case complexity for quicksort?
   - (A) cubic
   - (B) $n \log n$
   - (C) quadratic
   - (D) linear
   - (E) $\log n$

93. What is the worst case complexity for naive quicksort?
   - (A) cubic
   - (B) $n \log n$
   - (C) linear
   - (D) quadratic
   - (E) $\log n$

94. What is the best case complexity for selection sort?
   - (A) log $n$
   - (B) cubic
   - (C) $n \log n$
   - (D) quadratic
   - (E) linear

95. What is the worst case complexity for selection sort?
   - (A) cubic
   - (B) linear
   - (C) $n \log n$
   - (D) quadratic
   - (E) $\log n$

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

97. What is the worst case complexity for insertion sort?
   - (A) $\log n$
   - (B) cubic
   - (C) linear
   - (D) $n \log n$
   - (E) quadratic
Concept: **simple arrays**

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

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

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

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

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

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

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

(A) getting the value at an index can be done in constant time  
(B) removing an element can be done in constant time  
(C) inserting an element can be done in constant time  
(D) finding an element can be done in constant time

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

(A) finding an element can be done in constant time  
(B) swapping two elements can be done in constant time  
(C) getting the value at an index can be done in constant time  
(D) setting the value at an index can be done in constant time

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

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

(A) yes, if all the elements are negative  
(B) yes, if all the elements are positive  
(C) only if the true minimum value is zero  
(D) always  
(E) only if all elements have the same value

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

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

(A) only if all elements have the same value  
(B) yes, if all the elements are negative  
(C) yes, if all the elements are positive  
(D) always  
(E) only if the true minimum value is at index 0

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

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

(A) only if all elements have the same value  
(B) yes, if all the elements are negative  
(C) only if the true minimum value is at index 0  
(D) yes, if all the elements are positive  
(E) always
106. Does the following code set the variable \( v \) to the minimum value in an unsorted, non-empty array?

\[
v = \text{array}[0];
\text{for (i from 0 until array.length)}
  \text{if (array[i] > v)}
    v = \text{array}[i];
\]

(A) yes, if all the elements are negative           (D) only if the true minimum value is at index 0
(B) always                                        (E) only if all elements have the same value
(C) yes, if all the elements are positive

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

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

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

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

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

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

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

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

(A) always
(B) only if the item is in the array
(C) never
(D) only if the item is not in the array
110. Does this \textit{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) only if the item is not in the array  (C) never
(B) always  (D) only if the item is in the array

\textbf{Concept: simple fillable arrays}

111. What is \textit{not} a property of a simple fillable array?

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

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

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

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

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

114. Removing an element at front of a simple fillable array can be done in:

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

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

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

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

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

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

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

(A) zero  (C) the length of the underlying simple array
(B) the capacity of the array  (D) one
119. Suppose a simple fillable array is full. The capacity of the array is:

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

120. 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.

\[
\text{for (i from j until s-2)}
\]
\[
\text{array[i] = array[i+1];}
\]
\[
\text{array[i] = newElement;}
\]
\[---\]
\[
\text{for (i from s-2 until j)}
\]
\[
\text{array[i+1] = array[i];}
\]
\[
\text{array[i] = newElement;}
\]

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

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

\[
\text{for (i from j until s-2)}
\]
\[
\text{array[i+1] = array[i];}
\]
\[
\text{array[i] = newElement;}
\]
\[---\]
\[
\text{for (i from s-2 until j)}
\]
\[
\text{array[i] = array[i+1];}
\]
\[
\text{array[i] = newElement;}
\]

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

**Concept: circular arrays**

In this section, assume \( f \) is the start index, \( s \) is the size, and \( c \) is the capacity of the array.

122. What is a property of a circular array?

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

123. What is **not** a property of a circular array?

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

124. Before correction, the next value to be added to the front of the array will be placed at index:

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

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

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

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

(A) the capacity of the array  
(B) zero  
(C) the length of the array  
(D) one
127. 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) $f -= 1; f = f < 0 ? c - 1 : f;$
- (B) $f = (f - 1 + c) \% c;$
- (C) if $(f == 0) f = c - 1; else f = f - 1;$
- (D) $f = f == 0 ? c - 1 : f - 1;$
- (E) $f -= 1; f == 0 ? c - 1 : f;$

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

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

130. Is a separate *end* index needed in a circular array?

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

**Concept: dynamic arrays**

131. What is *not* a major characteristic of a dynamic array?

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

132. 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, but only if the dynamic array is not circular
- (B) yes
- (C) no

133. Suppose array capacity grows by 50. If the only events are insertions, the growing events:

- (A) occur less and less frequently
- (B) cannot be characterized in terms of frequency
- (C) occur periodically
- (D) occur more and more frequently

134. Suppose array capacity doubles every time the array fills, If the only events are insertions, the average cost of an insertion, in the limit, is:

- (A) the log of the size
- (B) constant
- (C) the log of the capacity
- (D) linear

135. Suppose array capacity grows by 10 every time the array fills, If the only events are insertions, the average cost of an insertion, in the limit, is:

- (A) quadratic
- (B) the log of the capacity
- (C) constant
- (D) linear
- (E) the log of the size

136. If array capacity grows by 10 every time the array grows, the average cost of an insertion in the limit is:

- (A) constant
- (B) linear
- (C) the log of the capacity
- (D) the log of the size

137.Appending to a singly-linked list without a tail pointer takes:

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

139. 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 to:
   (A) log time
   (B) \( n \log n \) time
   (C) linear time
   (D) constant time

140. 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) \( n \log n \) time
   (B) constant time
   (C) log time
   (D) linear time

141. 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) 3
   (B) 2
   (C) 5
   (D) 1
   (E) 4

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

142. Removing the first item from a singly-linked list without a tail pointer takes:
   (A) \( n \log n \) time
   (B) linear time
   (C) log time
   (D) constant time

143. Removing the last item from a singly-linked list with a tail pointer takes:
   (A) log time
   (B) \( n \log n \) time
   (C) constant time
   (D) linear time

144. Removing the last item from a singly-linked list without a tail pointer takes:
   (A) \( n \log n \) time
   (B) constant time
   (C) log time
   (D) linear time

145. Removing the first item from a singly-linked list with a tail pointer takes:
   (A) constant time
   (B) log time
   (C) \( n \log n \) time
   (D) linear time

146. In a singly-linked list, you can move the tail pointer back one node in:
   (A) linear time
   (B) constant time
   (C) \( n \log n \) time
   (D) log time

147. 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) log time
   (C) linear time
   (D) \( n \log n \) time
148. Suppose you have a pointer to a node in a singly-linked list. You can then delete that node with as few pointer assignments as:

(A) 2  
(B) 1  
(C) 3

(D) 4  
(E) 5

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

149. Appending to a doubly-linked list without a tail pointer takes:

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

150. Appending to a doubly-linked list with a tail pointer takes:

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

151. Removing the first item from a doubly-linked list without a tail pointer takes:

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

152. Suppose you have a pointer to a node in a doubly-linked list. You can then insert a new node just prior in:

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

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

(A) 5  
(B) 3  
(C) 2

(D) 1  
(E) 4

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

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

155. T : F: Making a doubly-linked list circular removes the need for a separate tail pointer.

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

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

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

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

(A) \( \log \) time  
(B) linear time  
(C) constant time  
(D) \( n \log n \) time
158. In a doubly-linked list, what does a tail-pointer gain you?

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

Concept: input-output order

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

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

160. LIFO ordering is the same as:

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

161. FIFO ordering is the same as:

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

Concept: time and space complexity

162. Consider a stack based upon a fillable array with pushes onto the front of the array. What is the lower bound of worst case behavior for push and pop, respectively? You may assume there is sufficient space for each operation.

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

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

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

164. Consider a stack based upon a dynamic array with pushes onto the back of the array. What is the lower bound of worst case behavior for push and pop, respectively? You may assume the array does shrink.

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

165. Consider a stack based upon a dynamic circular array with pushes onto the front of the array. What is the lower bound of worst case behavior for push and pop, respectively? You may assume the array does shrink.

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

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

(A) linear and constant
(B) linear and linear
(C) constant and linear
(D) constant and constant
167. Consider a stack based upon a singly-linked list with a tail pointer with pushes onto the front of the list. What is the lower bound of worst case behavior for push and pop, respectively?

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

168. Consider a stack based upon a doubly-linked list without a tail pointer with pushes onto the front of the list. What is the lower bound of worst case behavior for push and pop, respectively?

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

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

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

170. 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) $i + j = c$  
(B) $i + j = c-2$  
(C) $i == c/2-1 \&& j = c/2-1$  
(D) $i == c/2 \|| j == c/2$  
(E) $i == c/2-1 \|| j == c/2-1$  
(F) $i == c/2 \&& j = c/2$

**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);
  }
}
```

171. 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) 5 6  
(C) 3 3  
(D) 1 2

**Concept: input-output order**

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

(A) 4  
(B) 9  
(C) 1  
(D) 5
173. FIFO ordering is the same as:
(A) FILO  
(B) LIFO  
(C) LILO

174. LIFO ordering is the same as:
(A) FIFO  
(B) LILO  
(C) FILO

**Concept: complexity**

175. Consider a queue based upon a simple fillable array with enqueues onto the front of the array. What is the lower bound of worst case behavior for enqueue and dequeue, respectively?

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

176. Consider a queue based upon a circular array with enqueues onto the front of the array. What is the lower bound of worst case behavior for enqueue and dequeue, respectively?

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

177. Consider a queue based upon a singly-linked list without a tail pointer with enqueues onto the front of the list. What is the lower bound of worst case behavior for enqueue and dequeue, respectively?

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

178. Consider a queue based upon a singly-linked list with a tail pointer with enqueues onto the front of the list. What is the lower bound of worst case behavior for enqueue and dequeue, respectively?

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

179. Consider a queue based upon a doubly-linked list with a tail pointer with enqueues onto the front of the list. What is the lower bound of worst case behavior for enqueue and dequeue, respectively?

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

180. Consider a queue based upon a doubly-linked list without a tail pointer with enqueues onto the front of the list. What is the lower bound of worst case behavior for enqueue and dequeue, respectively?

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

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

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

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

(A) \( \sqrt{n} \)  
(B) constant  
(C) \( \log n \)  
(D) \( n \log n \)  
(E) linear  
(F) quadratic
183. Consider a binary search tree with \( n \) nodes. What is the minimum and maximum height (using order notation)?

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

Concept: balance

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

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

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

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

186. What is the best definition of a perfectly balanced binary tree?

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

Concept: tree shapes

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

(A) there is no limit on the size of the tree 
(B) 10 
(C) 8 
(D) 9 
(E) 7 

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

(A) 9 
(B) 8 
(C) 10 
(D) there is no limit on the size of the tree 
(E) 7 

189. Let \( P_0, P_1, \) and \( P_2 \) 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 nodes are \( P_0 \) or \( P_2 \) 
(C) all interior nodes are \( P_1 \) 
(D) all interior nodes \( P_1 \), except the root 
(E) all interior nodes are \( P_2 \); all leaves are equidistant from the root 
(F) all interior nodes are \( P_2 \)

190. Let \( P_0, P_1, \) and \( P_2 \) 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 are \( P_1 \) 
(B) all interior nodes \( P_1 \), except the root 
(C) all nodes are \( P_0 \) or \( P_2 \) 
(D) all interior nodes are \( P_2 \); all leaves are equidistant from the root 
(E) all leaves are equidistant from the root 
(F) all interior nodes are \( P_2 \)

191. Let \( P_0, P_1, \) and \( P_2 \) 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 \( P_1 \) 
(B) all interior nodes are \( P_2 \) 
(C) all interior nodes \( P_1 \), except the root 
(D) all leaves are equidistant from the root 
(E) all interior nodes are \( P_2 \); all leaves are equidistant from the root 
(F) all nodes are \( P_0 \) or \( P_2 \)
192. 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 P1 or P2
- (D) removing the rightmost leaves from a perfect tree
- (E) making the leftmost leaves of a perfect tree P2

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

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

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

196. How many distinct binary trees can be formed from two nodes with unique values?

- (A) 4
- (B) 5
- (C) 3
- (D) 2
- (E) 1

197. 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) \( 2^{k-1} + 1 \)
- (C) \( 2^k - 1 \)
- (D) \( 2^{k+1} - 1 \)
- (E) \( 2^{k-1} - 1 \)

198. 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) \( 2^{k+1} \)
- (B) \( k \)
- (C) \( 2^{k+1} - 1 \)
- (D) \( k + 1 \)
- (E) \((\log k) + 1\)
- (F) \(\log k\)

199. Let \( k \) be the 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) \( k + 1 \)
- (B) \( k \)
- (C) \( \log k \)
- (D) \( 2^{k+1} - 1 \)
- (E) \( 2^{k+1} \)
- (F) \((\log k) + 1\)

**Concept: ordering in a BST**

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

- (A) always less than
- (B) always greater than
- (C) less than, if the child is a left child
- (D) greater than, if the child is a left child
- (E) there is no relationship

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

- (A) always greater than
- (B) always less than
- (C) there is no relationship
- (D) greater than, if the sibling is a right child
- (E) less than, if the sibling is a right child

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

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

**Concept: traversals**

204. Consider 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 in a pre-order traversal?

(A) 53  (D) 25
(B) 0  (E) 54
(C) 38

205. Consider 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 in an in-order traversal?

(A) 38  (D) 53
(B) 25  (E) 0
(C) 54

206. Consider 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 in a post-order traversal?

(A) 25  (D) 0
(B) 63  (E) 54
(C) 38

207. 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) 0 7 4 2 8 12 10 6  (E) 0 4 2 7 8 12 10 6
(B) 12 10 8 7 6 4 2 0  (F) 6 2 0 4 10 8 7 12
(C) 6 0 2 4 8 10 12 7  (G) 6 0 2 4 7 10 12 8
(D) 6 2 0 4 7 10 8 12  (H) 0 2 4 6 7 8 10 12

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

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

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

(A) 12 10 8 7 6 4 2 0  (E) 6 0 2 4 8 10 12 7
(B) 6 2 0 4 7 10 8 12  (F) 0 4 2 7 8 12 10 6
(C) 6 0 2 4 7 10 12 8  (G) 0 2 4 6 7 8 10 12
(D) 6 2 0 4 10 8 7 12  (H) 0 7 4 2 8 12 10 6
210. Consider a perfect BST with even values 0 through 12, to which the value 7 is then added. Which of the following is a post-order traversal of the resulting tree?

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

211. If a six-node binary tree has a level-order traversal of C B A D F E and an in-order traversal of B C A F D E does it have a unique pre-order traversal and, if so, what is it?

(A) yes, C A D B E F
(B) yes, C B A D F E
(C) yes, C B A F D E
(D) no

212. If a six-node binary tree has an in-order traversal of B C A F D E has a pre-order traversal of C B A D F E does it have a unique post-order traversal and, if so, what is it?

(A) yes, B F E D A C
(B) yes, F A D B E C
(C) yes, B F A E D C
(D) no

213. If a six-node binary tree has an in-order traversal of B C A F D E has a post-order traversal of C B A D F E does it have a unique level-order traversal and, if so, what is it?

(A) yes, E A C F B D
(B) yes, E F C D B A
(C) no
(D) yes, E F A D C B

214. If a six-node binary tree has a level-order traversal of C F D E B A and a pre-order traversal of C F E A D B does it have a unique in-order traversal and, if so, what is it?

(A) no
(B) yes, F A E C B D
(C) yes, F A E C B D
(D) yes, F E A C D B

**Concept: insertion and deletion**

215. 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?

216. 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?

217. 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?

218. 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?

219. 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) 4 and 9
(B) 5 and 9
(C) 4 and 10
(D) 3 and 10
(E) 3 and 9
(F) 5 and 10
220. 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) i (E) all
(B) none (F) i and ii
(C) ii (G) iii
(D) ii and iii (H) i and iii

Concept: heap shapes

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

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

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

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

223. 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) 2
(B) 0
(C) $\theta(\log n)$
(D) 1

224. T or F: If a node in a heap has no sibling, it must be a left child.

225. T or F: The maximum number of nodes in a heap with exactly one child must be greater than zero.

226. T or F: The maximum number of nodes in a heap with exactly one child must be no greater than one.

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

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

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

230. T or F: No heaps are complete trees.

231. T or F: A binary tree with one node must be a heap.

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

233. 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.

234. 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

235. 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
236. In a min-heap, what is the relationship between a left child and its sibling?

(A) the right child has a larger value  (C) there is no relationship between their values
(B) both children cannot have the same value  (D) the left child has a smaller value

237. 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.

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

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

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

Concept: heaps stored in arrays

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

![Heap Diagram]

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

242. Printing out the values in the array yield what kind of traversal of the heap?

(A) post-order  (C) in-order
(B) pre-order  (D) level-order

243. Suppose the heap has \( n \) values. The root of the heap can be found at which index?

(A) \( n \)  (C) 1
(B) \( n-1 \)  (D) 0

244. Suppose the heap has \( n \) values. The left child of the root can be found at which index?

(A) 0  (D) \( n-1 \)
(B) \( n \)  (E) \( n-2 \)
(C) 1  (F) 2

245. Left children are stored at what kind of indices?

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

246. Right children are stored at what kind of indices?

(A) a roughly equal mix of odd and even  (D) all odd but one
(B) all even  (E) all odd
(C) all even but one
247. The formula for finding the left child of a node stored at index $i$ is:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(A) $(i - 2)/3$
(B) $i/3 - 1$
(C) $(i + 1)/3$
(D) $(i + 2)/3$
(E) $(i - 1)/3$
(F) $i/3 + 1$
257. In a max-heap, the minimum value can be found in time:
   (A) $\theta(1)$   (C) $\theta(n)$
   (B) $\theta(\log n)$   (D) $\theta(n \log n)$

258. Suppose a min-heap with $n$ values is stored in an array $a$. In the $\text{extractMin}$ operation, which element immediately replaces the root element (prior to this new root being sifted down).
   (A) $a[2]$   (C) $a[1]$
   (B) $a[n-1]$   (D) the minimum of $a[1]$ and $a[2]$

259. The $\text{findMin}$ operation takes how much time?
   (A) $\Theta(n \log n)$   (C) $\Theta(\log n)$
   (B) $\Theta(1)$   (D) $\Theta(n)$

260. The $\text{extractMin}$ operation takes how much time?
   (A) $\Theta(1)$   (C) $\Theta(n \log n)$
   (B) $\Theta(\log n)$   (D) $\Theta(n)$

261. Merging two heaps of size $n$ and $m$, $m < n$ takes how much time?
   (A) $\Theta(n \log m)$   (D) $\Theta(n)$
   (B) $\Theta(n \cdot m)$   (E) $\Theta(\log n \cdot \log m)$
   (C) $\Theta(\log n + \log m)$   (F) $\Theta(m \log n)$

262. The $\text{insert}$ operation takes how much time?
   (A) $\Theta(n)$   (C) $\Theta(n \log n)$
   (B) $\Theta(\log n)$   (D) $\Theta(1)$

263. Turning an unordered array into a heap takes how much time?
   (A) $\Theta(n \log n)$   (C) $\Theta(\log n)$
   (B) $\Theta(n)$   (D) $\Theta(1)$

264. Suppose the values 21, 15, 14, 10, 8, 5, and 2 are inserted, one after the other, into an empty $\text{min}$-heap. What does the resulting heap look like?
   (A) $w$   (C) $z$
   (B) $x$   (D) $y$

265. Using the standard $\text{heapify}$ operation to turn an unordered array into a $\text{max}$-heap, how many parent-child swaps are made if the initial unordered array is $[5,21,8,15,25,3,9]$?
   (A) 3   (D) 7
   (B) 4   (E) 2
   (C) 6   (F) 5