Concept: The classes $\mathcal{P}$ and $\mathcal{NP}$

1. A problem can be in $\mathcal{P}$ and not in $\mathcal{NP}$.
   (A) True
   (B) Not known
   (C) False

2. A problem can be in $\mathcal{NP}$ and not in $\mathcal{P}$.
   (A) True
   (B) Not known
   (C) False

3. All problems are in $\mathcal{P}$.
   (A) Not known
   (B) False
   (C) True

4. All problems are in $\mathcal{NP}$.
   (A) True
   (B) Not known
   (C) False

5. $\mathcal{NP}$ stands for:
   (A) Non-intractable Program.
   (B) Non-deterministic Polynomial.
   (C) Non-Polynomial.
   (D) Non-exponential Program.

6. $\text{T or F}$: A constant time algorithm is in $\mathcal{P}$.

7. $\text{T or F}$: A linear time algorithm is in $\mathcal{P}$.

8. $\text{T or F}$: A constant time algorithm is in $\mathcal{NP}$.

9. $\text{T or F}$: A linear time algorithm is in $\mathcal{NP}$.

10. Someone shows you a correct algorithm for problem A whose solution can be verified in polynomial time. You can conclude:
    (A) problem A is in $\mathcal{NP}$
    (B) nothing about whether problem A is in $\mathcal{NP}$ or not.
    (C) problem A is not in $\mathcal{NP}$

11. Someone proves that for a correct algorithm for problem A, solutions must be verified in at least exponential time. You can conclude:
    (A) nothing about whether problem A is in $\mathcal{NP}$ or not.
    (B) problem A is in $\mathcal{NP}$
    (C) problem A is not in $\mathcal{NP}$

12. Someone shows you a correct polynomial time algorithm for problem A. You can conclude:
    (A) problem A is not in $\mathcal{P}$
    (B) nothing about whether problem A is in $\mathcal{P}$ or not.
    (C) problem A is in $\mathcal{P}$
13. Someone shows you a correct exponential time algorithm for problem A whose solution can be verified in polynomial time. You can conclude:

(A) problem A is in \( \mathcal{P} \) 
(B) problem A is not in \( \mathcal{P} \) 
(C) nothing about whether problem A is in \( \mathcal{P} \) or not.

14. Someone shows you a correct polynomial time algorithm for problem A. You can conclude:

(A) nothing about whether problem A is in \( \mathcal{NP} \) or not. 
(B) problem A is not in \( \mathcal{NP} \) 
(C) problem A is in \( \mathcal{NP} \)

15. Which one of the following is not a valid way to prove a problem is in \( \mathcal{NP} \):

(A) show that a solution can be found in polynomial time on a deterministic computer.
(B) show that a solution can be verified in polynomial time on a non-deterministic computer.
(C) show that a solution can be found in polynomial time on a non-deterministic computer.
(D) show that a solution can be verified in polynomial time on a deterministic computer.

**Concept: \( \mathcal{NP} \)-completeness**

16. To show that a problem A is \( \mathcal{NP} \)-complete, one task is to:

(A) show A is not in \( \mathcal{P} \).
(B) show A is in \( \mathcal{NP} \).
(C) show A is not in \( \mathcal{NP} \).
(D) show A is in \( \mathcal{P} \).

17. Suppose B is an \( \mathcal{NP} \)-complete problem. To show that a problem A is \( \mathcal{NP} \)-complete, one task could be:

(A) show a polynomial time/space reduction from B to A.
(B) show an exponential time/space reduction from B to A.
(C) show a polynomial time/space reduction from A to B.
(D) show an exponential time/space reduction from B to A.

18. Another way of stating “a reduction from A to B” is:

(A) solve A-type problems with an algorithm for B
(B) convert an algorithm for A to an algorithm for B
(C) convert an algorithm for B to an algorithm for A
(D) solve B-type problems with an algorithm for A

**Concept: If \( \mathcal{P} = \mathcal{NP} ? \)**

19. If \( \mathcal{P} = \mathcal{NP} \), then all problems in \( \mathcal{P} \) are in \( \mathcal{NP} \).

(A) False
(B) Not known
(C) True

20. If \( \mathcal{P} = \mathcal{NP} \), then all problems in \( \mathcal{NP} \) are in \( \mathcal{P} \).

(A) False
(B) True
(C) Not known

21. If \( \mathcal{P} \neq \mathcal{NP} \), then there exist problems in \( \mathcal{P} \) that are not in \( \mathcal{NP} \).

(A) True
(B) False
(C) Not known

22. If \( \mathcal{P} \neq \mathcal{NP} \), then there exist problems in \( \mathcal{NP} \) that are not in \( \mathcal{P} \).

(A) Not known
(B) True
(C) False
Concept: Proving $\mathcal{P} = \mathcal{NP}$.

23. *Factoring* is in $\mathcal{NP}$. Currently, the best known algorithm on a conventional computer takes exponential time. If *factoring* is proved to take at least exponential time, what is the effect on the question $\mathcal{P} = \mathcal{NP}$?
   
   (A) the question is still unanswered  
   (B) $\mathcal{P} = \mathcal{NP}$  
   (C) $\mathcal{P} \neq \mathcal{NP}$

24. *Factoring* is in $\mathcal{NP}$. Currently, the best known algorithm on a conventional computer takes exponential time. If *factoring* is shown to take polynomial time, what is the effect on the question $\mathcal{P} = \mathcal{NP}$?
   
   (A) $\mathcal{P} \neq \mathcal{NP}$  
   (B) $\mathcal{P} = \mathcal{NP}$  
   (C) the question is still unanswered

25. *Factoring* is in $\mathcal{NP}$ and the best known algorithm takes exponential time. In the past, a linear time algorithm was discovered for quantum computers. What is the effect on the question $\mathcal{P} = \mathcal{NP}$?
   
   (A) $\mathcal{P} = \mathcal{NP}$? is still unanswered.  
   (B) $\mathcal{P} = \mathcal{NP}$, but just for quantum computers  
   (C) $\mathcal{P} = \mathcal{NP}$ for all types of computers.

26. *Subset Sum* is $\mathcal{NP}$-complete. Currently, the best known algorithm on a conventional computer takes exponential time. If *Subset Sum* is proved to take at least exponential time, what is the effect on the question $\mathcal{P} = \mathcal{NP}$?
   
   (A) $\mathcal{P} \neq \mathcal{NP}$  
   (B) the question is still unanswered  
   (C) $\mathcal{P} = \mathcal{NP}$

27. *Subset Sum* is $\mathcal{NP}$-complete. Currently, the best known algorithm on a conventional computer takes exponential time. If solving *Subset Sum* can be shown to take polynomial time, what is the effect on the question $\mathcal{P} = \mathcal{NP}$?
   
   (A) the question is still unanswered  
   (B) $\mathcal{P} = \mathcal{NP}$  
   (C) $\mathcal{P} \neq \mathcal{NP}$

28. In the past, it was shown how to solve Hamiltonian Path (an $\mathcal{NP}$-complete problem) in linear time, using a DNA-based computer. However, the algorithm takes a factorial number of DNA strands, which need to be created each time. This means:
   
   (A) $\mathcal{P} \neq \mathcal{NP}$ for all types of computers.  
   (B) $\mathcal{P} \neq \mathcal{NP}$? is still unanswered.  
   (C) $\mathcal{P} = \mathcal{NP}$, but just for DNA-based computers

29. **T** or **F**: $\mathcal{P} = \mathcal{NP}$ is just another way of saying, for problems in $\mathcal{NP}$, finding a solution is no harder than verifying a solution.