One hundred (or so) problems for CS2

Concept Overview

- Mathematical Concepts
- O (n) Selection
- O (n) Sorting
- Inductive Proofs
- Lower Bounds on Comparison-Based Sorts
- Master Recurrence Theorem
- Self-balancing BSTs
- Amortized Analysis
- Binomial and Fibonacci Heaps
- Dynamic Programming / Results Caching
- Hash tables
- P, NP, NP-completeness

Concept: *mathematics*

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) $\log_{10} n$ / $\log_2 10$
   (b) $\log_2 n$ / $\log_2 10$
   (c) $\log_2 n$ / $\log_{10} 2$
   (d) $\log_{10} n$ / $\log_{10} 2$
3. $\log (n\, m)$ is equal to:
   (a) $(\log n)^m$
   (b) $\log n + \log m$
   (c) $n \log m$
   (d) $m \log n$
4. $\log (n^m)$ is equal to:
   (a) $\log n + \log m$
   (b) $(\log n)^m$
   (c) $m \log n$
   (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^2

(b) 2^n

(c) n

(d) $\log_2 n$

7. Which of the following has the correct order?

(a) $1 < log\, n < \sqrt{n} < n < n\, log\, n < n^2 < n^3 < 2^n < n^n < n!$

(b) $1 < log\, n < \sqrt{n} < n < n\, log\, n < n^2 < n^3 < 2^n < n! < n^n$

(c) $1 < \sqrt{n} < log\, n < n < n\, log\, n < n^2 < n^3 < n! < 2^n < n^n$

(d) $1 < \sqrt{n} < log\, n < n < n\, log\, n < n^2 < n^3 < 2^n < n! < n^n$

(e) $1 < \sqrt{n} < log\, n < n < n\, log\, n < n^2 < n^3 < n! < 2^n < n^n$

8. n^2 is $o(n^3)$. Therefore, log n^2 is ?(log n^3). Choose the tightest bound.

(a) theta

(b) little omicron

(c) big omega

(d) big omicron

(e) little omega

9. log n^n is Θ(?).

(a) n log n

(b) log n

(c) n

(d) log $n^{\log n}$

10. log 2^n is Θ(?).

(a) n log n

(b) n

(c) log n

(d) 2^n

11. What does big Omicron roughly mean?

(a) always better than

(b) the same as

(c) worse than or equal

(d) better than or equal

(e) always worse than

12. What does ω roughly mean?

(a) better than or equal

(b) always better than

(c) worse than or equal

(d) always worse than

(e) the same as

13. What does Θ roughly mean?

(a) better than or equal

(b) always better than

(c) worse than or equal

(d) the same as

(e) always worse than

14. All algorithms are $\omega(1)$.

(a) False

(b) True

15. All algorithms are $o(1)$.

(a) False

(b) True

16. All algorithms are $\Omega(1)$.

(a) False

(b) True

17. There exist algorithms that are $\omega(1)$.

(a) False

(b) True

18. All algorithms are $O(n^n)$.

(a) True

(b) False

19. 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^2, because mergesort takes quadratic time

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

20. Suppose algorithm $f = o(g)$. Algorithm f is guaranteed to always run faster than g, regardless of input size.

(a) false

(b) true

21. Suppose algorithm $f = o(g)$. There exists a problem size above which f is always faster than g, in the worst case.

(a) true

(b) false

22. Suppose algorithm $f = \omega(g)$. There exists a problem size above which f is always slower than g, even for best-case input.

(a) false

(b) true

23. Suppose algorithm $f = \Omega(g)$. f and g can be the same algorithm.

    (a) true
    (b) false

24. Suppose algorithm $f = \Omega(g)$. Algorithm f is guaranteed to always run equal to or slower than g, regardless of input size.

    (a) true
    (b) false

25. Suppose algorithm $f = O(g)$. There exists a problem size above which f is always equal to or faster than g, in the worst case.

    (a) false
    (b) true

26. Suppose algorithm $f = \Omega(g)$. There exists a problem size above which f is always equal to or slower than g, even for best-case input.

    (a) true
    (b) false

27. Suppose algorithm $f = O(g)$. f and g can be the same algorithm.

    (a) true
    (b) false

28. Suppose algorithm $f = \Theta(g)$ and that a stopwatch is used to time implementations of f and g on the same input. It is possible that f takes less time than g according to the stopwatch.

    (a) true
    (b) false

29. Suppose algorithm $f = \Theta(g)$. f and g can be the same algorithm.

    (a) false
    (b) true

Concept: *bounds*

30. Θ is:

    (a) a lower bound
    (b) both an upper and lower bound
    (c) an upper bound

31. Ω is:

    (a) an upper bound
    (b) a lower bound
    (c) both an upper and lower bound

Concept: *comparison sorting – basics*

32. In a decision tree for a comparison sort of n numbers, what is the smallest possible depth of a leaf? Assume the root is at depth 0.

(a) $\log n$

(b) n - 1

(c) $n \log n$

(d) n

(e) $n!$

33. In a decision tree for a comparison sort of n numbers, what is the smallest / largest number of leaves possible?

(a) 2^n / n^n

(b) $\log n!$ / $n!$

(c) $n \log n$ / $n!$

(d) $n!$ / no limit

34. In a decision tree for a comparison sort of n numbers, what does the shortest path from the root to a leaf represent?

(a) nothing, the longest path is what's important

(b) the average case behavior of the sort

(c) the best case behavior of the sort

(d) the worst case behavior of the sort

35. The lower time bound on a comparison sort is:

(a) $\Omega(\ n \log n\ )$

(b) $O(\ n \log n\ )$

(c) $\Omega(\ n\ )$

(d) $\omega(\ n\ )$

36. The upper time bound on a comparison sort is:

(a) $\Omega(\ n \log n\ )$

(b) $\Omega(\ n\ )$

(c) $\omega(\ n \log n\ )$

(d) $O(\ n\ )$

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

(a) log linear

(b) quadratic

(c) exponential

(d) linear

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

(a) exponential

(b) quadratic

(c) log linear

(d) linear

39. The best case behavior for insertion sort is:

(a) linear

(b) exponential

(c) quadratic

(d) log linear

40. Stability in a sort means:

(a) the asymptotic run time does not vary upon the input

(b) that numbers with the same value appear in the output in the same order as they appear in the input

(c) the sort can work on any kind of number (integers, reals, etc.)

41. A linear-time sort does not compare entire elements with one another.

(a) False

(b) True

42. A linear-time sort must always compare entire elements with one another.

(a) False

(b) True

43. Counting sort is:

(a) stable if lower indexed elements from the input array are transferred to the output array before higher indexed elements.

(b) always stable

(c) stable if higher indexed elements from the input array are transferred to the output array before lower indexed elements.

(d) always unstable

44. Consider using a counting sort to sort the input array `[2,5,0,3,2,0,3,3]`. After the first phase, when `C[i]` holds the number of elements equal to i, the array C looks like:

(a) `[2,0,2,3,0,1]`

(b) `[2,2,4,6,7,8]`

(c) `[2,2,4,7,7,8]`

(d) `[2,0,5,2,3,0,3,3]`

(e) `[0,0,2,2,3,3,3,5]`

45. Consider using a counting sort to sort the input array `[2,5,0,3,2,0,3,3]` with auxiliary array C. After the second phase, when `C[i]` holds the number of elements less than or equal to i, the array C looks like:

(a) `[0,0,2,2,3,3,3,5]`

(b) `[2,0,5,2,3,0,3,3]`

(c) `[2,0,2,3,0,1]`

(d) `[2,2,4,7,7,8]`

(e) `[2,2,4,6,7,8]`

46. Consider using a stable counting sort to sort the input array `[2,5,0,3,2,0,3,3]` with destination array B. At start of phase three, using a right to left placement, the third element to be placed in B is:

(a) 3

(b) 0

(c) 5

(d) 2

47. Consider using a stable counting sort to sort the input array `[2,5,0,3,2,0,3,3]` with destination array B. At start of phase three, using a right to left placement, the third element to be placed in B at index:

(a) 2

(b) 0

(c) 5

(d) 3

48. Let n be the count of numbers in a collection of non-negative, base$_{10}$ numbers. Let k be the maximum number in the collection. The time complexity of counting sort is:

    (a) $\Theta(n + k)$

    (b) $\Theta(n\ k)$

    (c) $\Theta(n^k)$

    (d) $\Theta(n \log k)$

49. Let n be the count of numbers in a collection of non-negative, base$_{10}$ numbers. Let k be the maximum number in the collection. If $k = O(\ n\ )$, then the time complexity of counting sort is:

    (a) $\Theta(n \log k)$

    (b) $\Theta(n\ k)$

    (c) $\Theta(n)$

    (d) $\Theta(n^2)$

50. Consider using radix sort for sorting the following numbers:

    ```
    549
    354
    318
    622
    ```

    After the first pass, the order of the numbers are:

    (a) 318, 354, 549, 622

    (b) 622, 354, 318, 549

    (c) 354, 318, 549, 622

    (d) 354, 549, 318, 622

51. Let n be the count of numbers in a collection of positive, base$_{10}$ numbers. Let m be the number of digits in the largest number in the collection. Suppose the auxiliary sort works in $\Theta(\ n\ )$ time. Then radix sorting takes time:

    (a) $\Theta(m \log n)$

    (b) $\Theta(n \log n)$

    (c) $\Theta(n \log m)$

    (d) $\Theta(n\ m)$

52. Let n be the count of numbers in a collection of positive, base$_{10}$ numbers. Let m be the number of digits in the largest number in the collection. Suppose the auxiliary sort works in $\Theta(\ n \log n\ )$ time. Then radix sorting takes time:

    (a) $\Theta(m\ n \log m)$

    (b) $\Theta(n \log(m + n))$

    (c) $\Theta(n \log m)$

    (d) $\Theta(n \log(m\ n))$

53. Suppose during one pass of radix sort, there is a tie between two numbers. Since they are considered the same number, it does not matter if those two numbers swap positions.

(a) False

(b) True

54. Consider the sort used for each pass of radix sort. Must the auxiliary sort be stable?

(a) No, because there can be no ties in radix sort.

(b) Yes, because swapping ties can undo the work of previous passes

(c) Yes, because swapping ties can undo the work of future passes

(d) No, because radix sort works even if the auxilliary sort is unstable.

55. Consider using bucket sort to sort n numbers evenly distributed over the range 0 to m. Roughly, how many buckets should you use?

(a) n

(b) m

(c) $m \ / \ n$

(d) $n \ / \ m$

56. Consider using bucket sort to sort n non-negative numbers evenly distributed over the range 0 to m. The range of the first bucket should be:

(a) 0 to $n \ / \ m$

(b) $n \ / \ m$ to $(2 \ n \ ) \ / \ m$

(c) $m \ / \ n$ to $(2 \ m \ ) \ / \ n$

(d) 0 to $m \ / \ n$

57. Consider using bucket sort to sort n non-negative numbers evenly distributed over the range 0 to m. The average count of numbers in a bucket is:

(a) $m \ / \ n$

(b) 0

(c) 1

(d) $n \ / \ m$

58. Consider a bucket sort that uses insertion sort to sort the individual buckets. Suppose you could bound the maximum count of numbers in a bucket to a constant C. Then the asymptotic running time to sort one bucket would be:

(a) $\Theta(n)$, because insertion sort is linear in the best case

(b) $\Theta(n \log n)$, because insertion sort is log-linear in the average case

(c) $\Theta(n^2)$, because insertion sort is quadratic in the worst case

(d) $\Theta(1)$

59. Consider using bucket sort to sort n non-negative numbers evenly distributed over the range 0 to m. The *expected* running time is:

(a) quadratic

(b) linear

(c) constant

(d) log-linear

60. Consider using bucket sort to sort n non-negative numbers in the range 0 to m with no a priori knowledge of the distribution of the numbers. The worst case running time is:

(a) cubic

(b) quadratic

(c) constant

(d) linear

Concept: *recurrence equations*

61. Stooge sort has the following algorithm. Recursively sort the lower two-thirds of an array, then recursively sort the upper two-thirds, then recursively sort the lower two-thirds again. The recursion stops when the array consists of two or fewer elements. If the array size is two, the elements are swapped if necessary. Which of the following recurrence equations describe stooge sort?

    (a) `T(n) = 2T(3n/2) + O(1)`

    (b) `T(n) = 2T(n/3) + O(1)`

    (c) `T(n) = 3T(n/2) + O(log n)`

    (d) `T(n) = 3T(2n/3) + O(1)`

    (e) `T(n) = 3T(n/2) + O(n)`

62. Merge sort has the following algorithm. Recursively sort the lower half of an array, then recursively sort the upper half, then merge the two halves. When merging, a single pass is made through each half. The recursion stops when the array consists of one element. Which of the following recurrence equations describe merge sort?

    (a) `T(n) = 2T(n/2) + O(1)`

    (b) `T(n) = 2T(n/2) + O(n)`

    (c) `T(n) = 4T(n/2) + O(1)`

    (d) `T(n) = T(n-1) + O(n)`

63. Binary search has the following algorithm. Look at the middle element in an array. If the element is the one begin looked for (the target), return FOUND. Otherwise, if the element is greater than the target, recursively search the lower half of the array. Otherwise, recursively search the upper half of the array. If the array consists of three or fewer elements, perform a linear search of the array, returning FOUND if found and MISSING otherwise. Which of the following recurrence equations describe binary search?

    (a) `T(n) = 2T(n/2) + O(n)`

    (b) `T(n) = 2T(n/2) + O(log n)`

    (c) `T(n) = T(n/2) + O(1)`

    (d) `T(n) = 2T(n/2) + O(1)`

    (e) `T(n) = T(n/2) + O(log n)`

    (f) `T(n) = T(n/2) + O(n)`

    (g) `T(n) = 2T(n-1) + O(1)`

64. The recurrence equation `T(n) = T(n - 1) + O(n)` describes which sort?

    (a) radix sort

    (b) heap sort

    (c) insertion sort

    (d) merge sort

65. Consider using insertion sort to sort a previously sorted array. Which of the following recurrence equations describes this situation?

    (a) `T(n) = T(n - 1) + O(n)`

    (b) `T(n) = T(n - 1) + O(1)`

    (c) `T(n) = 2T(n/2) + O(1)`

    (d) `T(n) = 2T(n - 1) + O(1)`

(e) `T(n) = T(n/2) + O(log n)`

Concept: *master recurrence theorem*

66. Under which case of the master recurrence theorem does the equation `T(n) = 4T(n/4) + O(n^2)` fall?
    (a) case 2
    (b) case 1
    (c) the master recurrence equation does not apply
    (d) case 3
67. Under which case of the master recurrence theorem does the equation `T(n) = 4T(n-4) + O(n)` fall?
    (a) the master recurrence equation does not apply
    (b) case 1
    (c) case 2
    (d) case 3
68. Under which case of the master recurrence theorem does the equation `T(n) = 3T(n/6) + O(log n)` fall?
    (a) case 1
    (b) case 2
    (c) the master recurrence equation does not apply
    (d) case 3

Concept: *inductive proofs*

69. (3 pts) Prove by strong induction that $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.

70. (3 pts) Prove by strong induction that $\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$.

71. (3 pts) Prove by strong induction that $\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}$.

72. (3 pts) Prove by strong induction that $\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$.

73. (3 pts) Prove by strong induction that the number of nodes in a perfect binary tree with n levels is $2^n - 1$.

74. (3 pts) Prove by strong induction that the number of nodes in a perfect quaternary tree with n levels is $\frac{4^n-1}{3}$.

75. (3 pts) Prove by strong induction that the number of leaves in a perfect binary tree with n levels is $2^n - 1$.

76. (3 pts) Prove by strong induction that the number of interior nodes in a perfect binary tree with n levels is $2^{n-1} - 1$.

77. (3 pts) Prove by strong induction that the number of child pointers in an arbitrary binary tree with n nodes is $n - 1$.

Concept: *rotations in a BST*

Assume that if a node n is rotated, it moves upward one level.

78. Which of the following is true for rotations in a BST:
    (a) the number of leaf nodes always increases
    (b) BST-ordering is always preserved
    (c) the tree always becomes more balanced
    (d) the number of leaf nodes always decreases

79. Consider a right-rotation of a node n in a BST. The right child of n, if it exists:
    (a) becomes the niece or nephew of n
    (b) becomes the left child of the parent of n
    (c) becomes the left child of n
    (d) becomes the sibling of n
    (e) becomes the right child of the parent of n
    (f) remains the right child of n

80. Consider a right-rotation of a node n in a BST. The left child of n, if it exists:
    (a) remains the left child of n
    (b) becomes the sibling of n
    (c) becomes the left child of the parent of n
    (d) becomes the niece or nephew of n
    (e) becomes the right child of the parent of n
    (f) becomes the right child of n

81. Consider a right-rotation of a node n in a BST. The parent of n, assuming it exists:
    (a) becomes the left child of n
    (b) becomes the sibling of n
    (c) becomes the right child of n
    (d) becomes the niece or nephew of n
    (e) remains the parent of n

82. Consider a right-rotation of a node n in a BST. The sibling of n, assuming it exists:
    (a) becomes the left child of n
    (b) becomes the right child of n
    (c) remains the parent of n
    (d) becomes the niece or nephew of n
    (e) becomes a grandchild of n

Concept: *red-black trees*

83. The maximum number of rotations that occur after an insertion into a red-black tree is:

    (a) $O(n)$
    (b) $O(\log n)$
    (c) $O(\log \log n)$
    (d) $O(n \log n)$
    (e) $O(1)$

84. Consider a node n in a red-black tree and all paths from n to a leaf. Which of the following is a constraint on red-black-trees?

    (a) the number of red nodes on each path is constant
    (b) the number of black nodes on each path is constant
    (c) each path must start with a black node
    (d) the number of nodes (red or black) on each path is constant

85. Consider a node n in a red-black tree and all paths from n to a leaf. Which of the following is a constraint on red-black-trees?

    (a) no red node can have a red parent
    (b) no red node can have a black parent
    (c) no black node can have a red parent
    (d) no black node can have a black parent

86. Consider a red node n in a red-black tree and any path from n to a leaf. Which of the following is a constraint on red-black-trees?

    (a) the number of red nodes must equal the number of black nodes
    (b) the number of red nodes must not exceed the number of black nodes
    (c) the number of black nodes must not exceed the number of red nodes
    (d) the number of red nodes may exceed the number of black nodes by at most 1

87. Consider a node n in a red-black tree and the length of the shortest path from n to a leaf, S and the length of the longest path from n to a leaf, L. Which of the following is a constraint on red-black-trees?

    (a) $L \leq 2S$
    (b) $L \leq S/2$
    (c) $L \leq S + 1$
    (d) $L = S$

88. Inserting a value in a red-black tree and a regular BST, respectively, takes time:

    (a) $O(\log n)$ and $O(n)$
    (b) $O(1)$ and $O(\log n)$
    (c) $O(n)$ and $O(n)$
    (d) $O(\log n)$ and $O(\log n)$

89. Suppose one wished to allow more red nodes in a red-black tree, but still wished this new tree to have the same asymptotic behavior as before. One could allow more red nodes on any path to a leaf as long as:

    (a) the number of black nodes between any two red nodes is bounded by a constant.
    (b) the number of red nodes between any two black nodes is bounded by a constant.
    (c) no red node could have a red sibling.

(d) no black node could have a red parent.

Concept: *AVL trees*

90. The maximum number of rotations that occur after an insertion into a AVL tree is:

    (a) $O(n)$
    (b) $O(1)$
    (c) $O(\log n)$
    (d) $O(\log \log n)$
    (e) $O(n \log n)$

Concept: *amortized analysis*

91. Suppose a dynamic array was implemented so that growing the array increased the capacity by 1000 elements. What is the worst-case cost of the append operation?

    (a) quadratic
    (b) linear
    (c) log linear
    (d) constant

92. Suppose a dynamic array was implemented so that growing the array increased the capacity by 2%. What is the amortized cost of the append operation?

    (a) constant
    (b) quadratic
    (c) linear
    (d) log linear

93. Consider implementing a queue with two stacks. Enqueues are translated to pushes onto the first stack. For a dequeue, each element on the first stack is popped, then pushed onto the second stack, in turn. When the transfer is finished, the item popped from the second stack is returned.
    The worst-case times for enqueue and dequeue are:

    (a) $\Theta(1)$ and $\Theta(n)$
    (b) $\Theta(n)$ and $\Theta(1)$
    (c) $\Theta(n)$ and $\Theta(n)$
    (d) $\Theta(1)$ and $\Theta(1)$

94. Consider implementing a queue with two stacks. Enqueues are translated to pushes onto the first stack (*s1*). For a dequeue, each element on the first stack is popped, then pushed onto the second stack (*s2*), in turn. When the transfer is finished, the item popped from the second stack is returned.

    Which of the following potential functions can be used to show an amortized bound of $\Theta(1)$ for operations on this kind of queue?

    (a) $\Phi = \text{size}\ (s1) + \text{size}\ (s2)$
    (b) $\Phi = 2 * \text{size}\ (s1)$
    (c) $\Phi = \text{size}\ (s1) + 2 * \text{size}\ (s2)$
    (d) $\Phi = \text{size}\ (s1)$

95. Suppose a data structure has operation A with a real cost of 1 and operation B with a real cost of $n/2$. After an A operation, n increases by 2 while after a B operation, n decreases by half.

    Which of the following potential functions can be used to show an amortized bound of $\Theta(1)$ for operations A and B on this data structure?

    (a) $\Phi = 2n$
    (b) $\Phi = 3n$
    (c) $\Phi = n$

96. Suppose a data structure has operation A with a real cost of $3n$ and operation B with a real cost of $\frac{3}{2}n$. After an A operation, n doubles while after a B operation, n decreases by half.

    Which of the following potential functions can be used to show an amortized bound of $\Theta(1)$ for operations A and B on this data structure?

    (a) $\Phi = 2n$
    (b) $\Phi = 3n$
    (c) $\Phi = \frac{3}{2}n$

97. Suppose a data structure has operation A with a real cost of 1 and operation B with a real cost of $k+n$. After an A operation, n increases by 1. After a B operation, n decreases to $k+1$.

    Which of the following potential functions can be used to show an amortized bound of $\Theta(1)$ for A operations and $\Theta(k)$ for B operations?

    (a) $\Phi = n$
    (b) $\Phi = 2n$
    (c) $\Phi = 3n$

Concept: *binomial and Fibonacci heaps*

98. One can delete a value in a binomial heap in time that is:

    (a) constant
    (b) log log
    (c) log
    (d) linear
    (e) log-linear

99. One can insert a value into a Fibonacci heap in amortized time that is:

    (a) log
    (b) constant
    (c) log log
    (d) linear
    (e) log-linear

Concept: *NP-Completeness*

100. A problem can be in P and not in NP.

(a) False
(b) Not known
(c) True

101. A problem can be in NP and not in P.

(a) True
(b) False
(c) Not known

102. All problems are in P.

(a) Not known
(b) False
(c) True

103. All problems are in NP.

(a) Not known
(b) True
(c) False

104. If P = NP, then all problems in P are in NP.

(a) False
(b) True
(c) Not known

105. If P = NP, then all problems in NP are in P.

(a) False
(b) Not known
(c) True

106. If P != NP, then there exist problems in P that are not in NP.

(a) Not known
(b) False
(c) True

107. If P != NP, then there exist problems in NP that are not in P.

(a) Not known
(b) False
(c) True

108. NP stands for:

(a) *Non-exponential Program.*
(b) *Non-deterministic Polynomial.*
(c) *Non-Polynomial.*
(d) *Non-intractable Program.*

109. A constant time algorithm is in P.

(a) False
(b) True

110. A linear time algorithm is in P.

(a) False

(b) True

111. A constant time algorithm is in NP.

(a) False

(b) True

112. A linear time algorithm is in NP.

(a) False

(b) True

113. *Factoring* is in 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 P = NP?

(a) P != NP

(b) P = NP

(c) the question is still unanswered

114. *Factoring* is in NP. Currently, the best known algorithm on a conventional computer takes exponential time. If *factoring* is shown to take take polynomial time, what is the effect on the question P = NP?

(a) the question is still unanswered

(b) P = NP

(c) P != NP

115. *Factoring* is in NP. Recently a linear time algorithm was discovered for quantum computers. What is the effect on the question P = NP?

(a) P != NP, but just for quantum computers

(b) the question is still unanswered

(c) P = NP, but just for quantum computers

116. *Subset Sum* is 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 P = NP?

(a) P = NP

(b) P != NP

(c) the question is still unanswered

117. *Subset Sum* is NP-complete. Currently, the best known algorithm on a conventional computer takes exponential time. If *Subset Sum* is shown to take polynomial time, what is the effect on the question P = NP?

(a) P = NP

(b) the question is still unanswered

(c) P != NP

118. Recently, it was shown how to solve Hamiltonian Cycle (an NP-complete problem) in linear time, using a DNA-based computer. However, the algorithm takes exponential space. This means:

(a) P != NP, but just for DNA-based computers

(b) the question is still unanswered

(c) P = NP, but just for DNA-based computers

119. In general, you can trade off time for space and vice versa.

(a) False

(b) True

120. P = NP is just another way of saying, for problems in NP, finding a solution is no harder than verifying a solution.

(a) False

(b) True

121. Someone shows you a correct exponential time algorithm for Problem A. You can conclude:

(a) Problem A is in NP

(b) Problem A is not in NP

(c) nothing about whether Problem A is in NP or not.

122. Someone shows you a correct polynomial time algorithm for Problem B. You can conclude:

(a) nothing about whether Problem B is in NP or not.

(b) Problem B is not in NP

(c) Problem B is in NP

123. Which one of the following is not a valid way to prove a problem is in NP:

(a) show that a solution can be found in polynomial time on a non-deterministic computer.

(b) show that a solution can be found in polynomial time on a deterministic computer.

(c) show that a solution can be verified in polynomial time on a deterministic computer.

(d) show that a solution can be verified in polynomial time on a non-deterministic computer.