One hundred (or so) problems covering CS1 Concepts

Concept Overview

- Algorithm analysis with o, O, Θ, Ω, ω
- Iteration
- Recursion and mutual recursion
- Arrays and Lists
- $O(n)$ searching
- $O(n^2)$ and $O(n \log n)$ sorting
- Stacks, Queues, and Binary Search Trees
- Hash Tables

Concept: *mathematics*

1. $\log_2 n$ is:
   (a) $o(\log_{10} n)$
   (b) $\Theta(\log_{10} n)$
   (c) $\omega(\log_{10} n)$
2. $\log_2 n$ is equal to:
   (a) $\log_{10} n$ / $\log_2 10$
   (b) $\log_2 n$ / $\log_{10} 2$
   (c) $\log_{10} n$ / $\log_{10} 2$
   (d) $\log_2 n$ / $\log_2 10$
3. log (nm) is equal to:
   (a) $(\log\ n)^m$
   (b) $m \log n$
   (c) $\log n + \log m$
   (d) $n \log m$
4. log (n^m) is equal to:
   (a) $\log n + \log m$
   (b) $m \log n$
   (c) $(\log\ n)^m$
   (d) $n \log m$
5. $\log_2 2$ can be simplified to:
   (a) 4
   (b) 1
   (c) 2
   (d) $\log_2 2$ cannot be simplified any further
6. $2^{\log_2 n}$ is equal to:

(a) $\log_2 n$
(b) n^2
(c) 2^n
(d) n

Concept: Order notation

7. What does big Omicron roughly mean?
   (a) always better than
   (b) better than or equal
   (c) the same as
   (d) always worse than
   (e) worse than or equal
8. What does little omicron roughly mean?
   (a) always better than
   (b) better than or equal
   (c) the same as
   (d) always worse than
   (e) worse than or equal
9. What does θ roughly mean?
   (a) the same as
   (b) always worse than
   (c) better than or equal
   (d) worse than or equal
   (e) always better than
10. All algorithms are $\omega(1)$.
    (a) False
    (b) True
11. All algorithms are $\theta(1)$.
    (a) True
    (b) False
12. All algorithms are $\Omega(1)$.
    (a) True
    (b) False
13. There exist algorithms that are $\omega(1)$.
    (a) True
    (b) False
14. All algorithms are $O(n^n)$.
    (a) True
    (b) False

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

    (a) n log n, because mergesort takes n log n time
    (b) constant, because n is fixed
    (c) n^2, because mergesort takes quadratic time

16. Which of the following has the correct order?

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

17. Suppose algorithm $f = \omega(g)$. Algorithm f is guaranteed to always run slower than g, regardless of input size.

    (a) true
    (b) false

18. Suppose algorithm $f = \omega(g)$. There exists a problem size above which f is always slower than g, in the worst case.

    (a) false
    (b) true

19. 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

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

    (a) true
    (b) false

21. 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

22. Suppose algorithm $f = \Omega(g)$. There exists a problem size above which f is always equal to or slower than g, in the worst case.

    (a) false
    (b) true

23. 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) false
    (b) true

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

    (a) true
    (b) false

25. Suppose algorithm $f = \Theta(g)$ and that a stopwatch is used to time implementations of f and g on the same input. It is guaranteed that f takes equal time as g according to the stopwatch.

    (a) false
    (b) true

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

    (a) true
    (b) false

Concept: *bounds*

27. Θ is:

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

Concept: iteration

28. Is this factorial function correct?

```
def factorial(n):
    total = 0
    while (n > 1):
        total = total * n
        n = n - 1
    return total
```

    (a) no
    (b) yes
    (c) yes, except for the `factorial(1)`

29. Is this factorial function correct?

```
def factorial(n):
    total = 1
    while (n > 0):
        n = n - 1
        total = total * n
    return total
```

    (a) yes
    (b) yes, except for `factorial(1)`
    (c) no

30. Is this factorial function correct?

```
def factorial(n):
    total = 1
    while (n > 0):
        total = total * n
        n = n - 1
    return total
```

(a) yes
(b) yes, except for `factorial(1)`
(c) no

31. Is this factorial function correct?

```
def factorial(n):
    total = n
    while (n > 1):
        n = n - 1
        total = total * n
    return total
```

(a) no
(b) yes
(c) yes, except for `factorial(1)`

32. Is this Fibonacci function correct?

```
def fib(n):
    prev = 0
    current = 1
    while (n > 0):
        temp = prev
        prev = current
        current = current + prev
        n = n - 1
    return prev
```

(a) yes
(b) no
(c) yes, except for `fib(0)`
(d) yes, except for `fib(1)`

33. Is this Fibonacci function correct?

```
def fib(n):
    prev = 0
    current = 1
    while (n > 0):
        temp = prev
        prev = current
        current = current + temp
        n = n - 1
    return prev
```

(a) yes, except for `fib(1)`
(b) yes
(c) no
(d) yes, except for `fib(0)`

34. Is this Fibonacci function correct?

```
def fib(n):
    prev = 0
    current = 1
    while (n > 0):
        temp = prev
        prev = current
        current = prev + temp
        n = n - 1
    return prev
```

(a) yes, except for `fib(0)`
(b) yes, except for `fib(1)`
(c) no
(d) yes

35. Is this Fibonacci function correct?

```
def fib(n):
    prev = 0
    current = 0
    while (n > 0):
        temp = prev
        prev = current
        current = current + temp
        n = n - 1
    return prev
```

(a) yes
(b) yes, except for `fib(1)`
(c) no
(d) yes, except for `fib(0)`

36. Are the following two functions equivalent, in terms of input and output?

```
def gcd(a,b):                  def gcd(a,b):
    while (b != 0):                  if (b == 0):
        a = b                          return a
        b = a % b                  else:
    return a                           return gcd(b,a % b)
```

(a) yes
(b) no

37. Are the following two functions equivalent, in terms of input and output?

```
def gcd(a,b):                  def gcd(a,b):
    while (b != 0):                if (b == 0):
        temp = a                       return a
        a = b                      else:
        b = temp % b                   return gcd(b,a % b)
    return a
```

(a) yes
(b) no

38. Are the following two functions equivalent, in terms of input and output?

```
def f(x,y)                                  def f(x,y):
    if (x == 0):                                  while (x != 0):
        return y                                    y = y + 1
    else:                                           x = x - 1
        return f(x - 1,y + 1)                   return y
```

(a) yes
(b) no

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

```
for i in range(0,n,1):
    print(i)
```

Simplify your answer.

(a) $\theta(n)$
(b) $\theta(n^2)$
(c) $\theta(n^{\sqrt{n}})$
(d) $\theta(n\sqrt{n})$
(e) $\theta(\log^2 n)$
(f) $\theta(n \log n)$

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

```
i = 1
while (i < n):
    print(i)
    i = i * 2
```

Simplify your answer.

(a) $\theta(n)$
(b) $\theta(n^2)$
(c) $\theta(n \log n)$
(d) $\theta(n\sqrt{n})$
(e) $\theta(n^{\sqrt{n}})$
(f) $\theta(\log^2 n)$

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

```
step = sqrt(n)
for i in range(0,n,step):
    print(i)
```

Simplify your answer.

(a) $\theta(n - \sqrt{n})$
(b) $\theta(\frac{n}{\sqrt{n}})$
(c) $\theta(n)$
(d) $\theta(\sqrt{n})$

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

```
i = 1
while (i < n):
    print(i)
    i = i * sqrt(n)
```

Simplify your answer.

(a) $\theta(n)$
(b) $\theta(1)$
(c) $\theta(\frac{n}{\sqrt{n}})$
(d) $\theta(\sqrt{n})$
(e) $\theta(n - \sqrt{n})$

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

```
for i in range(0,n,1):
    for j in range(0,n,1):
        print(i,j)
```

Simplify your answer.

(a) $\theta(n)$
(b) $\theta(n\sqrt{n})$
(c) $\theta(n^2)$
(d) $\theta(n^{\sqrt{n}})$
(e) $\theta(n \log n)$
(f) $\theta(\log^2 n)$

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

```
for i in range(0,n,1):
    for j in range(i,n,1):
        print(i,j)
```

Simplify your answer.

(a) $\theta(n^{\sqrt{n}})$
(b) $\theta(n^2)$
(c) $\theta(n\sqrt{n})$
(d) $\theta(n \log n)$
(e) $\theta(\log^2 n)$
(f) $\theta(n)$

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

```
i = 1
while (i < n):
    for j in range(0,n,1):
        print(i,j);
    i = i * 2
```

Simplify your answer.

(a) $\theta(n^{\sqrt{n}})$
(b) $\theta(n)$

(c) $\theta(n^2)$

(d) $\theta(n\sqrt{n})$

(e) $\theta(n \log n)$

(f) $\theta(\log^2 n)$

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

```
for i in range(0,n,1):
    j = 1
    while (j < n):
        print(i,j)
        j = j * 2
```

Simplify your answer.

(a) $\theta(n \log n)$

(b) $\theta(n)$

(c) $\theta(\log^2 n)$

(d) $\theta(\log n^2)$

(e) $\theta(n + \log n)$

(f) $\theta(n^2)$

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

```
for i in range(0,n,1):
    print(i)
for j in range(0,n,1):
    print(j)
```

Simplify your answer.

(a) $\theta(n^2)$

(b) $\theta(n)$

(c) $\theta(n\sqrt{n})$

(d) $\theta(\log^2 n)$

(e) $\theta(n^{\sqrt{n}})$

(f) $\theta(n \log n)$

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

```
i = 1
while (i < n):
    print(i)
    i = i * 2
for j in range(0,n,1):
    print(j);
```

Simplify your answer.

(a) $\theta(n^2)$

(b) $\theta(n)$

(c) $\theta(n^{\sqrt{n}})$

(d) $\theta(\log^2 n)$

(e) $\theta(n\sqrt{n})$

(f) $\theta(n \log n)$

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

```
for i in range(0,n,1):
    print(i)
j = 1
while (j < n):
    print(j)
    j = j * 2
```

Simplify your answer.

(a) $\theta(n^2)$
(b) $\theta(n + \log n)$
(c) $\theta(n)$
(d) $\theta(n \log n)$
(e) $\theta(\log^2 n)$
(f) $\theta(\log n^2)$

Concept: recursion

50. Which of the following describes recursive Fibonacci's time complexity?

(a) $\theta(1)$
(b) $\theta(n - \sqrt{n})$
(c) $\theta(\frac{n}{\sqrt{n}})$
(d) $\theta(\phi^n)$
(e) $\theta(\phi)$
(f) $\theta(\frac{\phi}{n})$
(g) $\theta(\sqrt{n})$

51. Which of the following describes recursive Fibonacci's space complexity? Simplify your answer.

(a) $\theta(n)$
(b) $\theta(1)$
(c) $\theta(\frac{\phi}{n})$
(d) $\theta(\frac{n}{\sqrt{n}})$
(e) $\theta(n - \sqrt{n})$
(f) $\theta(\sqrt{n})$

52. Which of the following describes iterative Fibonacci's time complexity?

(a) $\theta(n - \sqrt{n})$
(b) $\theta(n)$
(c) $\theta(\frac{\phi}{n})$
(d) $\theta(\frac{n}{\sqrt{n}})$
(e) $\theta(\sqrt{n})$
(f) $\theta(1)$

53. Which of the following describes iterative Fibonacci's space complexity?

(a) $\theta(\sqrt{n})$

(b) $\theta(n)$
(c) $\theta(1)$
(d) $\theta(\frac{\phi}{n})$
(e) $\theta(n - \sqrt{n})$
(f) $\theta(\frac{n}{\sqrt{n}})$

Concept: arrays and lists

54. What is a major characteristic of a plain vanilla array?
    (a) finding an element takes, at least, log n time
    (b) inserting an element can be done in constant time
    (c) deleting an element can be done in constant time
    (d) accessing an element can be done in constant time

55. What is *not* a major characteristic of a circular array?
    (a) appending an element can be done in constant time
    (b) inserting an element in the middle can be done in constant time
    (c) prepending an element can be done in constant time
    (d) elements are presumed to be contiguous
    (e) finding an element takes, at least, linear time

56. What is *not* a major characteristic of a dynamic array?
    (a) finding an element takes linear time
    (b) elements are presumed to be contiguous
    (c) the only allowed way to grow is doubling the size
    (d) inserting an element in the middle takes linear time
    (e) the array can grow to accommodate more elements

57. Appending to a singly-linked list without a tail pointer takes:
    (a) log time
    (b) linear time
    (c) constant time
    (d) n log n time

58. Appending to a singly-linked list with a tail pointer takes:
    (a) log time
    (b) constant time
    (c) n log n time
    (d) linear time

59. Removing the last 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

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

(a) log time
(b) linear time
(c) $n \log n$ time
(d) constant time

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

(a) linear time
(b) log time
(c) $n \log n$ time
(d) constant time

62. 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

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

(a) log time
(b) linear time
(c) constant time
(d) $n \log n$ time

64. Making a doubly-linked list circular removes the need for a separate tail pointer.

(a) False
(b) True

65. Suppose you have a pointer to a node in a singly linked list. You can then insert a new node just prior in:

(a) log time
(b) constant time
(c) linear time
(d) $n \log n$ time

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

(a) log time
(b) $n \log n$ time
(c) linear time
(d) constant time

67. Suppose you have a pointer to a node in a singly linked list. You can then insert a new node just after in:

(a) linear time
(b) log time
(c) constant time
(d) $n \log n$ time

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

(a) linear time
(b) constant time

(c) log time
(d) $n \log n$ time

69. In a singly-linked list, you can move a tail pointer backwards one node in:

    (a) constant time
    (b) log time
    (c) linear time
    (d) $n \log n$ time

70. In a doubly-linked list, you can move a tail pointer backwards one node in:

    (a) constant time
    (b) linear time
    (c) log time
    (d) $n \log n$ time

71. Given you have a pointer to the first of two nodes in a singly-linked list and a pointer to a new node, you can insert the new node between the two existing nodes with as few pointer assignments as:

    (a) 1
    (b) 2
    (c) 4
    (d) 3
    (e) 5

72. Given you have a pointer to the first of two nodes in a doubly-linked list and a pointer to a new node, you can insert the new node between the two existing nodes with as few pointer assignments as:

    (a) 3
    (b) 4
    (c) 1
    (d) 2
    (e) 5

73. Which code fragment correctly inserts a new element into the middle of an array?

```
for (i = insertionPoint; i < size - 2; i += 1)
    {
    array[i] = array[i + 1];
    }
array[i] = newElement;

---

for (i = size - 2; i > insertionPoint; i -= 1)
    {
    array[i+1] = array[i];
    }
array[i] = newElement;
```

    (a) both are correct
    (b) the second fragment
    (c) the first fragment

74. In a singly linked list, what does a tail-pointer gain you?

    (a) the ability to prepend the 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 append and remove the last element of list in constant time
    (e) the ability to remove the last element of list in constant time
    (f) the ability to both prepend and remove the first element of list in constant time

Concept: searching

75. Does the following code set the variable *min* to the minimum value in an unsorted, non-empty array?

```
min = 0
for i in range(0,len(array)):
    if (array[i] < min):
        min = array[i]
```

    (a) yes
    (b) no

76. Does the following code set the variable *max* to the maximum value in an unsorted, non-empty array?

```
max = array[0]
for i in range(0,len(array)):
    if (array[i] > max):
        max = array[i]
```

    (a) no
    (b) yes

77. Does the following function always return `True` if the value of item is *present* in the unsorted, non-empty array and `False` otherwise?

```
def find(array,item):
    found = False
    for i in range(0,len(array)):
        if array[i] == item:
            found = True
    return found
```

    (a) no
    (b) yes

78. Does the following function always return `False` if the value of item is *missing* in the unsorted, non-empty array and `True` otherwise?

```
def find(array,item):
    found = True
    for i = range(0,len(array)):
        if array[i] != item:
            found = False
    return found
```

    (a) no
    (b) yes

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

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

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

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

Concept: sorting

81. 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) selection sort
   (b) heapsort
   (c) mergesort
   (d) quicksort
   (e) insertion sort

82. 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) selection sort
   (b) mergesort
   (c) quicksort
   (d) insertion sort
   (e) heapsort

83. 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) selection sort
   (c) mergesort
   (d) heapsort
   (e) insertion sort

84. 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) mergesort
   (c) heapsort

(d) insertion sort
(e) selection sort

85. What is the best case complexity for merge sort?
    (a) cubic
    (b) $n \log n$
    (c) quadratic
    (d) linear
    (e) $\log n$

86. What is the worst case complexity for merge sort?
    (a) linear
    (b) $n \log n$
    (c) cubic
    (d) $\log n$
    (e) quadratic

87. What is the best case complexity for quicksort?
    (a) cubic
    (b) $n \log n$
    (c) $\log n$
    (d) quadratic
    (e) linear

88. What is the worst case complexity for naive quicksort?
    (a) quadratic
    (b) linear
    (c) cubic
    (d) $n \log n$
    (e) $\log n$

89. What is the best case complexity for selection sort?
    (a) quadratic
    (b) linear
    (c) $\log n$
    (d) cubic
    (e) $n \log n$

90. What is the worst case complexity for selection sort?
    (a) $n \log n$
    (b) quadratic
    (c) $\log n$
    (d) cubic
    (e) linear

91. What is the best case complexity for insertion sort?
    (a) $\log n$

(b) linear
(c) quadratic
(d) cubic
(e) $n \log n$

92. What is the worst case complexity for insertion sort?
    (a) linear
    (b) $n \log n$
    (c) quadratic
    (d) $\log n$
    (e) cubic

Concept: stacks, queues, and binary search trees

93. These values are pushed onto a stack in the order given: 1, 5, 9. A pop operation would return which value?
    (a) 9
    (b) 5
    (c) 1
    (d) 1 and 5
    (e) 1 and 9

94. Consider a stack based upon a regular array with pushes onto the front of the array. What is the best one can do in terms of worst case behavior for *push* and *pop*, respectively?
    (a) constant and constant
    (b) constant and linear
    (c) linear and linear
    (d) linear and constant

95. Consider a stack based upon a circular array with pushes onto the front of the array. What is the best one can do in terms of worst case behavior for *push* and *pop*, respectively? You may assume there is sufficient space for each operation.
    (a) linear and linear
    (b) constant and constant
    (c) linear and constant
    (d) constant and linear

96. Consider a stack based upon a circular array with pushes onto the front of the array. What is the best one can do in terms of worst case behavior for *push* and *pop*, respectively? You may assume the array grows when it becomes full.
    (a) linear and constant
    (b) constant and linear
    (c) constant and constant
    (d) linear and linear

97. Consider a stack based upon a singly-linked list without a tail pointer with pushes onto the front of the list. What is the best one can do in terms of worst case behavior for *push* and *pop*, respectively?
    (a) constant and constant
    (b) linear and constant

(c) constant and linear
(d) linear and linear

98. Consider a stack based upon a singly-linked list with a tail pointer with pushes onto the front of the list. What is the best one can do in terms of worst case behavior for *push* and *pop*, respectively?

(a) constant and linear
(b) constant and constant
(c) linear and linear
(d) linear and constant

99. Consider a stack based upon a doubly-linked list with a tail pointer with pushes onto the front of the list. What is the best one can do in terms of worst case behavior for *push* and *pop*, respectively?

(a) linear and constant
(b) constant and linear
(c) linear and linear
(d) constant and constant

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

(a) 1 and 5
(b) 9
(c) 1 and 9
(d) 1
(e) 5

101. Consider a queue based upon a regular array with enqueues onto the front of the array. What is the best one can do in terms of worst case behavior for *enqueue* and *dequeue*, respectively?

(a) constant and linear
(b) linear and linear
(c) linear and constant
(d) constant and constant

102. Consider a queue based upon a circular array with enqueues onto the front of the array. What is the best one can do in terms of worst case behavior for *enqueue* and *dequeue*, respectively?

(a) linear and linear
(b) constant and constant
(c) linear and constant
(d) constant and linear

103. Consider a queue based upon a singly-linked list without a tail pointer with enqueues onto the front of the list. What is the best one can do in terms of worst case behavior for *enqueue* and *dequeue*, respectively?

(a) constant and linear
(b) constant and constant
(c) linear and linear
(d) linear and constant

104. Consider a queue based upon a singly-linked list with a tail pointer with enqueues onto the front of the list. What is the best one can do in terms 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

105. Consider a queue based upon a doubly-linked list with a tail pointer with enqueues onto the front of the list. What is the best one can do in terms of worst case behavior for *enqueue* and *dequeue*, respectively?

(a) linear and linear
(b) constant and linear
(c) linear and constant
(d) constant and constant

106. Consider a binary search tree with n nodes. What is the best case time complexity for finding a value at a leaf?

(a) constant
(b) $\sqrt{n}$
(c) linear
(d) quadratic
(e) log n
(f) n log n

107. 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

108. 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) 4 2 1 6 3 8 7
(c) 1 2 3 4 5 7 6

109. 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

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

(a) less than or equal to, if the child is a left child
(b) greater than or equal to, if the child is a left child
(c) greater than or equal to
(d) less than or equal to
(e) there is no relationship

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

(a) there is no relationship
(b) greater than or equal to
(c) equal to
(d) less than or equal to

112. Do all these deletion strategies for non-leaf nodes reliably preserve BST ordering?

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

Concept: hash tables:

113. Consider chaining as a collision strategy, with the most efficient implemention possible. What are the worst case behaviors for insertion and finding, respectively?

(a) constant, linear
(b) linear, quadratic
(c) constant, constant
(d) quadratic, linear
(e) linear, linear

114. Consider open addressing as a collision strategy, with the most efficient implemention possible. What are the worst case behaviors for insertion and finding, respectively?

(a) linear, linear
(b) linear, constant
(c) constant, linear
(d) constant, constant
(e) linear, quadratic
(f) quadratic, linear

115. Consider rehasing with a perfect hash as a collision strategy, with the most efficient implemention possible. What are the worst case behaviors for insertion and finding, respectively?

(a) constant, constant
(b) linear, linear
(c) linear, constant
(d) constant, linear
(e) linear, quadratic
(f) quadratic, linear