Prerequisite Concepts for Analysis of Algorithms

Basic Data Structures (Version 7)

Name: ________________________________

Email: ________________________________

Code: 21481

Concept: *mathematics notation*

1. $\log_2 n$ is:

   (A) $o(\log_{10} n)$
   (B) $\omega(\log_{10} n)$
   (C) $\Theta(\log_{10} n)$

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

   (A) theta
   (B) little omega
   (C) big omicron
   (D) big omega
   (E) little omicron

3. $\log 2^n$ is $\Theta(?)$.

   (A) n
   (B) $n \log n$
   (C) $\log n$
   (D) 2^n

Concept: relative growth rates

4. 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) $h > g > f$
   (B) $g > f > h$
   (C) $f > g > h$
   (D) $f > h > g$
   (E) $h > f > g$
   (F) $g > h > f$

Concept: *order notation*

5. **T** or **F**: There exist algorithms that are $\omega(1)$.

6. **T** or **F**: There exist algorithms that are $O(1)$.

Concept: *comparing algorithms using order notation*

The phrase *by a stopwatch* means the actual amount of time needed for the algorithm to run to completion, as measured by a stopwatch.

7. **T** or **F**: If $f = \omega(g)$, then algorithm f always runs faster than g (by a stopwatch), in all cases.

8. **T** or **F**: If $f = \omega(g)$ and the input causes worst-case behaviors, then algorithm f always runs faster than g (by a stopwatch), regardless of input size.

9. **T** or **F**: If $f = \omega(g)$ and the input causes worst-case behaviors, then algorithm f always runs faster than g (by a stopwatch), above a certain input size.

10. **T** or **F**: If $f = \omega(g)$, then algorithm f always runs faster than g (by a stopwatch), above a certain input size.

11. **T** or **F**: If $f = \theta(g)$, then algorithm f always takes the same time as g (by a stopwatch), in all cases.

12. **T** or **F**: If $f = \theta(g)$ and the input causes worst-case behaviors, then algorithm f always takes the same time as g (by a stopwatch), regardless of input size.

13. **T** or **F**: If $f = \theta(g)$ and the input causes worst-case behaviors, then algorithm f always takes the same time as g (by a stopwatch), above a certain input size.

14. **T** or **F**: If $f = \theta(g)$, then algorithm f always takes the same time as g (by a stopwatch), above a certain input size.

15. **T** or **F**: If $f = \theta(g)$, then algorithm f always takes the same time as g (within a constant factor), in all cases.

16. **T** or **F**: If $f = \theta(g)$ and the input causes worst-case behaviors, then algorithm f always takes the same time as g (within a constant factor), regardless of input size.

17. **T** or **F**: If $f = \theta(g)$ and the input causes worst-case behaviors, then algorithm f always takes the same time as g (within a constant factor), above a certain input size.

18. **T** or **F**: If $f = \theta(g)$, then algorithm f always takes the same time as g (within a constant factor), above a certain input size.

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

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

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

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

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

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

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

Concept: *analyzing code*

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

Tracing recursive functions

When asked about the number of recursive calls, do not include the original call.

26. How many recursive calls are made if $n = 5$? Assume the initial value of i is zero.

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

(A) 5
(B) the function recurs infinitely
(C) 4
(D) 3
(E) 6
(F) none of the other answers are correct

27. How many recursive calls are made if $n = 81$. Assume the initial value of i is one.

```
function f(i,n)
    {
    if (i < n)
        {
        println(i);
        f(i*3,n);
        }
    return 0;
    }
```

(A) 6
(B) none of the other answers are correct
(C) 4
(D) 3
(E) 5
(F) the function recurs infinitely

Time complexity, recursive functions, single recursion

28. What is the time complexity of this function? Assume the initial value of i is one.

```
function f(i,n)
    {
    if (i < n)
        {
        println(i);
        f(i*3,n);
        }
    return 0;
    }
```

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

29. What is the time complexity of this function? Assume the initial value of i is one.

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

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

Time complexity, recursive functions, double recursion

Space complexity, recursive functions, single recursion

30. What is the space complexity of this function? Assume the initial value of i is one.

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

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

31. What is the space complexity of this function? Assume the initial value of i is one.

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

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

Space complexity, recursive functions, double recursion

Time complexity, iterative loops, single loops

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

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

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

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

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

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

Time complexity, iterative loops, double loops

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

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

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

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

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

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

Space complexity, iterative loops, single loops

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

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

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

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

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

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

Space complexity, iterative loops, double loops

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

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

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

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

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

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

Concept: *analysis of classic, simple algorithms*

40. Which of the following describes the classic recursive fibonacci's time complexity?

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

41. Which of the following describes iterative fibonacci's space complexity?

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

Concept: searching

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

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

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

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

Concept: sorting

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

Concept: *space and time complexity*

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

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

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

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

Concept: *simple arrays*

Assume zero-based indexing for all arrays.

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

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

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

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

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

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

49. What is a *not* a major characteristic of a simple array?

    (A) swapping two elements can be done in constant time
    (B) finding an element 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

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

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

(A) yes, if all the elements are negative
(B) yes, if all the elements are positive
(C) never
(D) only if the true minimum value is zero
(E) only if all elements have the same value
(F) always

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

```
v = array[0];
for (i from 0 until array.length)
    if (array[i] > v)
        v = array[i];
```

(A) always
(B) never
(C) yes, if all the elements are positive
(D) only if all elements have the same value
(E) yes, if all the elements are negative
(F) only if the true minimum value is at index 0

52. Does this *find* function return the expected result? Assume the array has at least two elements.

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

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

53. Does this *find* function return the expected result? Assume the array has at least two elements.

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

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

Concept: *simple fillable arrays*

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

54. What is *not* a property of a simple fillable array?

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

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

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

56. 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.

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

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

Concept: *circular arrays*

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

57. 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

Concept: *dynamic arrays*

58. Suppose array capacity grows by 10 every time a dynamic array fills, If the only events are insertions, the growing events:

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

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

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

Concept: *singly-linked lists (insertions)*

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

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

61. Suppose you have a pointer to a node near the end of a long singly-linked list. You can then insert a new node just prior in:

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

62. 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) 5
(B) 4
(C) 2
(D) 3
(E) 1

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

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

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

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

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

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

68. Suppose you have a pointer to a node in the middle of a singly-linked list. You can then delete that node in:
    - (A) $n \log n$ time
    - (B) log time
    - (C) constant time
    - (D) linear time

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

69. Appending to a non-circular, doubly-linked list without a tail pointer takes:
    - (A) log time
    - (B) $n \log n$ time
    - (C) linear time
    - (D) constant time

70. Appending to a non-circular, doubly-linked list with a tail pointer takes:
    - (A) constant time
    - (B) log time
    - (C) $n \log n$ time
    - (D) linear time

71. Removing the first item from a non-circular, doubly-linked list without a tail pointer takes:
    - (A) $n \log n$ time
    - (B) log time
    - (C) linear time
    - (D) constant time

72. Suppose you have a pointer to a node in the middle of a doubly-linked list. You can then insert a new node just after in:
    - (A) $n \log n$ time
    - (B) constant time
    - (C) linear time
    - (D) log time

73. Suppose you have a pointer to a node in the middle of a doubly-linked list. You can then insert a new node just prior with as few pointer assignments as:
    - (A) 1
    - (B) 4
    - (C) 3
    - (D) 5
    - (E) 2

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

Concept: *doubly-linked lists (deletions)*

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

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

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

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

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

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

Concept: *input-output order*

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

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

79. LIFO ordering is the same as:

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

Concept: *time and space complexity*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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.

```
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);
        }
    }
```

89. 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 2
(B) 5 6
(C) 1 6
(D) 3 3

Concept: *input-output order*

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

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

91. FIFO ordering is the same as:

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

Concept: *complexity*

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

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

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

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

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

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

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

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

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

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

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

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

Concept: *complexity*

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

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

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

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

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

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

Concept: *balance*

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

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

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

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

Concept: *tree shapes*

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

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

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

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

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

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

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

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

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

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

108. **T** or **F**: All *perfect* trees are *full* trees.

109. **T** or **F**: All *complete* trees are *perfect* trees.

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

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

111. 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) $2^{k-1} - 1$
(B) 2^{k+1}
(C) $2^{k-1} + 1$
(D) $2^{k+1} - 1$
(E) $2^{k} - 1$

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

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

113. 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
(B) $k + 1$
(C) $\log k$
(D) $2^{k+1} - 1$
(E) 2^{k+1}
(F) $(\log k) + 1$

Concept: *ordering in a BST*

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

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

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

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

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

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

117. 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
(B) it is always an interior node
(C) it is always a leaf node
(D) has no right child
(E) it may be an ancestor

Concept: *traversals*

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

(A) 25
(B) none of the other answers are correct
(C) 63
(D) 38
(E) 54
(F) 0

119. 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 4 2 7 8 10 12 6
(B) 7 0 2 4 6 8 10 12
(C) 0 2 4 6 7 8 10 12
(D) 6 2 10 0 4 8 12 7
(E) 12 10 8 7 6 4 2 0
(F) 0 2 4 6 8 10 12 7

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

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

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

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

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

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

Concept: *insertion and deletion*

123. **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?

124. **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?

125. Suppose 10 values are inserted 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 10
(B) 5 and 9
(C) 5 and 10
(D) 3 and 10
(E) 3 and 9
(F) 4 and 9

126. 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* and *iii*
   (B) *i* and *ii*
   (C) *ii* and *iii*
   (D) none
   (E) *ii*
   (F) *iii*
   (G) *i*
   (H) all

Concept: heap shapes

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

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

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

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

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

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

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

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

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

132. **T** or **F**: All heaps are perfect trees.

133. **T** or **F**: No heaps are perfect trees.

134. **T** or **F**: All heaps are complete trees.

135. **T** or **F**: No heaps are complete trees.

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

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

138. **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.

139. **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

140. In a min-heap, what is the relationship between a parent and its left child?

(A) there is no relationship between their values
(B) the parent has a smaller value
(C) the parent has the same value
(D) the parent has a larger value

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

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

142. **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.

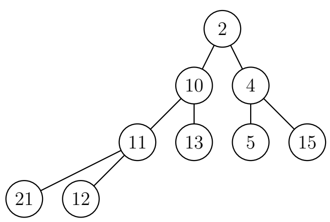
143. **T** or **F**: The largest value in a max-heap can be found at the root.

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

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

Concept: heaps stored in arrays

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



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

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

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

148. Suppose the heap has n values. The root of the heap can be found at which index?

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

149. Suppose the heap has n values. The left child of the root can be found at which index?

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

150. Left children in a heap are stored at what kind of indices?

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

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

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

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

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

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

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

154. 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) $(i + 1)/2$
(C) $i/2$
(D) $(i - 1)/2$

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

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

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

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

Concept: *heap operations*

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

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

158. Suppose a min-heap with n values is stored in an array a. In the *extractMin* operation, which element immediatelyreplaces the root element (prior to this new root being sifted down).

(A) `a[2]`
(B) the minimum of `a[1]` and `a[2]`
(C) `a[n-1]`
(D) `a[1]`

159. The *findMin* operation in a min-heap takes how much time?

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

160. The *extractMin* operation in a min-heap takes how much time?

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

161. Merging two heaps of size n and m, $m < n$ takes how much time?

(A) $\Theta(n \log m)$
(B) $\Theta(n + m)$
(C) $\Theta(\log n * \log m)$
(D) $\Theta(n * m)$
(E) $\Theta(\log n + \log m)$
(F) $\Theta(m \log n)$

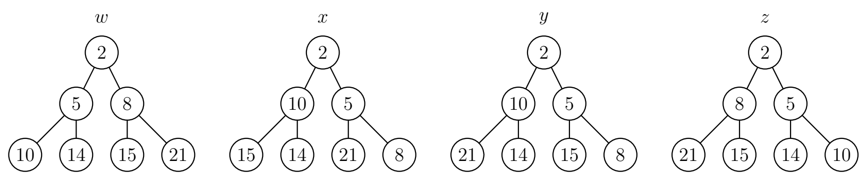
162. The *insert* operation takes how much time?

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

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

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

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



(A) z
(B) w
(C) y
(D) x

165. Using the standard *buildHeap* operation to turn an unordered array into a *max*-heap, how many parent-child swaps are made if the initial unordered array is `[5,21,8,15,25,3,9]`?

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