Directions: Save the Word document to your computer. Sign or digitally sign the document and insert an image of the student ID. Complete all of the questions. Show all of your work if applicable. Save the Word document as a PDF. In your PDF clearly identify each problem, please keep the questions in order and clearly identify the answers to the question and numbering it.

1) (2 pts ea.) Use Master Theorem to solve the following recurrence relations or indicate “not applicable” if Master Theorem does not apply. Explain why.

1. T(n) = T(n/4) + 6 (n2) + n

1. T(n) = 1/6 T(n/4) + sqrt(n) + O(1)

1. T(n) = 5T(n/2) + 4n + 1

2) (4 pts) A function Max-Heapify(A, i) ensures that element *i* is in its correct location in a max heap. Show the array after a call to Max-Heapify(Array, 3).

Input Array = [ 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 ]



Output Array = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3) (4 pts) In the following array representation of a heap, which of the following values are contained in leaf nodes?

A = [ 5, 3, 17, 10, 84, 19, 6, 22, 9 ]

4) (2 pts) In an undirected graph, the sum of the degrees of all vertices is equal to?

5) (4 pts) The post-order traversal of a binary tree is D,E,B,F,C,A. What is the pre-order traversal?

6) (4 pts) True or false – In a max-heap, the smallest element must be in a leaf node. Justify your answer.

7) (2 pt ea.) Indicate the space complexity (i.e. the extra space required, in big-O terms) for each of the following algorithms:

A) Insertion sort

B) Merge sort

C) Quicksort

8) (4 pts) The height of a binary search tree with *n* nodes is on the order of?

9) (4 pts) Given below are functions f(n) and g(n). Assume, . Show work.

(a) f(n) = 30n2 and g(n) = 6.

10) (2 pt ea.) Consider the following nearly complete **balanced** binary tree in array form:

[18, 12, 10, 7, 4, 2, 21, 5]

1. True or false - the tree is a Heap. Explain your answer.
2. True or false – the tree is a Binary Search Tree. Explain your answer.

11) (4 pts) If an algorithm is O(n), that means: (choose one)

 A. The algorithm always performs in O(n) for all inputs

 B. The algorithm always performs in O(n) for all values of n

 C. The algorithm performs at worst in O(n) for all values of n above some n0

 D. The algorithm performs on average in O(n)

12) (4 pts) What is the runtime complexity of the following algorithm?

int i = N;
while (i > 0) {
 int Sum = 0;
 int j;
 for (j = 0; j < i; j++)
 Sum++;

 print(Sum);
 i--;
}

13) (4 pts) True or false – in Quicksort, selecting the smallest element of the list as the pivot is likely to provide even splits.

14) (4 pts) A recursive algorithm divides and conquers the problem space into two subproblems, each half the size of the original. The combine step takes on the order of the length of the input. In “Big O” terms, what is the complexity of the algorithm?

15) (4 pts) If an algorithm is O(n), it might also be: (choose one)

 A. o(n2)

 B. o(1)

 C. O(n2)

 D. Ω(n2)

16) (10 pts) Draw the binary tree after each element of the following array is inserted into the tree: Array = [ 12, 2, 54, 62, 34, 12, 17, 54, 29, 5, 82, 9]

17) (2 pts each) What is the output of the traversal of the Binary tree in question 4 for the following algorithms: (List the values of the nodes)

1. In order traversal
2. Pre order traversal
3. Post order traversal

18) (4 pts each) The following array is a BST

* + Values: 35, 78, 89, 13, 2, 14, 45
* Identify if possible:
	1. A node with a missing successor
	2. A node whose successor is in the right subtree of a node

19) (5 pts each) Given the following problem:



For the following functions f(n) and constants c, give as tight a bound as possible for f\*c(n).

1. f(n) = n-1 c = 0
2. f(n) = n(lg n) c = 1

20) (2 pt) True or False. An algorithm that runs in O(n log2 n) is faster than an algorithm that runs in O (n).

21) (2 pts) Given the following set of equations identify if the statements are incorrect or incorrect:

*f* ↔ *g* ≈ *a* ↔ *b*

1. *f* (*n*) *= O*(*g*(*n*))≈ *a* ≥ *b*
2. *f* (*n*) *=* Ω(*g*(*n*))≈ *a* ≤ *b*
3. *f* (*n*) *=* Θ(*g*(*n*))≈ *a = b*
4. *f* (*n*) *= o*(*g*(*n*))≈ *a* < *b*
5. *f* (*n*) *= w* (*g*(*n*))≈ *a* > *b*