

## CptS 223 - Fall 2020

Due Date: Friday, December 18, by midnight

### *Take-home Final Exam*

Your Name: \_\_\_\_\_ **Koji Natsuhara** \_\_\_\_\_

ID Number: \_\_\_\_\_ **11666900** \_\_\_\_\_

READ THE FOLLOWING INSTRUCTIONS:

1. When your finished, add, commit, and push a file called FinalExam.pdf to remote origin on GitLab.
2. You may use your book and online resources. However, you may not work with others.

	Points Possible	Points Earned
Total	120	

## ~~ General Stuff ~~

1. [2] Assume that we have two functions:  $f(n)$  and  $g(n)$ . On a benchmark where  $n = 10,000$ , the execution time of  $f(n)$  was 15 seconds and the execution time of  $g(n)$  was 2.3 seconds. What can we conclude about these two functions?

$F(n)$  has a higher time complexity than  $g(n)$  when  $n = 10,000$ .

2. [2] If  $f(n)$  took 30 seconds for an  $n = 100$  then how long would it take to process  $n = 400$  if  $f(n)$  was of order:

1.  $O(N)$ : 120 seconds

2.  $O(N^2)$ : 480 seconds

3. [4] Big-O Dating Game. Match each data structure with the correct desired behavior so that each algorithm achieves its best Big-O (runtime efficiency). Feel free to justify each match.

**Structure A (2):** I'm the kind of algorithm who wants a data structure that isn't slow minded. When I ask it a question, I expect my data structure to be able to find the answer quickly and efficiently, but I don't need to keep my data in any particular order. (What data structure would be appropriate in a find-heavy algorithm?)

Hash Tables have extremely fast look up times  $O(1)$  using a vector and a search algorithm. The order of elements is irrelevant for this data structure.

**Structure B (2):** I like to keep things tidy. I like it best when everything is in its rightful place and would like to meet a data structure that isn't going to make that difficult. (What data structure would be appropriate for an algorithm that needs to maintain the ordering of my data?)

Self-Balancing binary search trees have to maintain their balance condition to keep the insert, remove, and find operations at  $O(\log n)$ . These data structures rotate or recolor to sufficiently keep the height of the tree at the lowest possible height.

4. [5] Supply both the names of these interfaces and code to complete the Big Five in this partial class implementation. Note helper functions at the end:

```
class AvlTree {
private:
    AvlNode *root;

public:
    ~AvlTree( ) {                // ← Interface name: __Destructor_____

makeEmpty(this->root);
delete root;

    }
    AvlTree( const AvlTree &other ) : root( NULL ) // ← Interface name: __Copy Constructor_____
    {
this->root = clone(other.root);
    }

    AvlTree( AvlTree &&other ) : root( NULL )      // ← Interface name: __Move Constructor_____
    {
        this->root = other.root;
        other.root = nullptr;
    }

    const AvlTree & operator= ( const AvlTree & other ) // ← Interface name: __Copy Assignment Operator_____
    {
        this->root = clone(other.root);

        return *this;
    }

    const AvlTree & operator= ( AvlTree && other )      // ← Interface name: __Move Assignment Operator_____
    {
        if(this != &other)
        {
            If(this->root != nullptr) {
                makeEmpty(this->root);
            }
            this->root = other.root;
            other.root = nullptr;
        }

        return *this;          // Return ourselves to caller
    }

    AvlNode * clone( AvlNode *t ) {                // Clone the whole tree
        if( t == NULL ) { return NULL; }
        else { return new AvlNode( t->element, clone( t->left ), clone( t->right ), t->height ); }
    }
    void makeEmpty( AvlNode * &t ) {                // Delete the whole tree
        if( t != NULL ) { makeEmpty( t->left ); makeEmpty( t->right ); delete t; }
        t = NULL; }
};
```

~~ Hashing ~~

5. [10] Suppose we want to insert the keys (k) 7, 10, 17, 20, 27 (in this order) into an initially empty hash table of size  $TS = 7$  and using the primary hash function  $h(k) = k \% TS$ . Show the final hash table for each of the following scenarios.

a. (4) Collision resolution by separate chaining.

0	1	2	3	4	5	6
7			10			20
			17			27

b. (4) Collision resolution with a linear probing hash function for  $i$  in  $\text{range}(0..TS-1)$ :

$$\text{probe}(i) = ( \text{hash}(k) + (i + 1) ) \% TS$$

0	1	2	3	4	5	6
7	27		10	17		20

c. (2) What is lazy deletion and why is it important for linear and quadratic probing?

Lazy deletion is important for linear and quadratic probing so that the search and insert functions can differentiate between an empty and deleted slot. If the search function were to encounter a slot that is “empty” the search can end. If the search function were to encounter a slot that is “deleted” the search can continue. Lazy deletion is needed or else the searching algorithm within the hash table would break.

6. [4] Which of the following hashing algorithms is likely to produce a faster HashTable implementation? Why?

Hashing Algorithm A	Hashing Algorithm B
<pre>int hash(string text):   int result = 0;   for(char ch : text)     result += ch;   return result;</pre>	<pre>int hash(string text):   int result = 1;   for(char ch : text)     result *= ch * ch + (11 * ch) + 7;   return result;</pre>

Hashing algorithm B is more likely to produce a faster HashTable implementation. The more variance that the algorithm produces when inserting each element into an empty index is important because it will generate fewer collisions. The less collisions that the hashing algorithm produces, the faster it is to rehash and insert elements into the table.

## ~~ Heaps ~~

**7. [6] Binary Heaps (Priority Queues!).** Enqueue (insert) the value 6 to the following binary heap, then do a Dequeue (deleteMin):

Starting heap:

3	7	5	9	7	12	8	14		
---	---	---	---	---	----	---	----	--	--

After enqueue (insert):

3	6	5	7	7	12	8	14	9	
---	---	---	---	---	----	---	----	---	--

After dequeue (deleteMin):

5	6	8	7	7	12	9	14		
---	---	---	---	---	----	---	----	--	--

I suggest you draw it out:

**8. [4]** List the fundamental structural rules of a binary heap (priority queue) and also let me know what the average height of a binary heap is:

The fundamental structural rules of a binary heap (priority queue) is that the dequeue function returns the minimum element in the heap. Each level is (except the most bottom level) is completely filled and is filled left to right, and each node gets a maximum of 2 children.

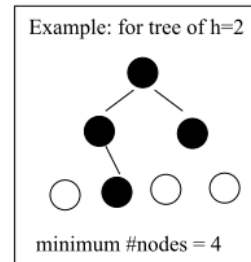
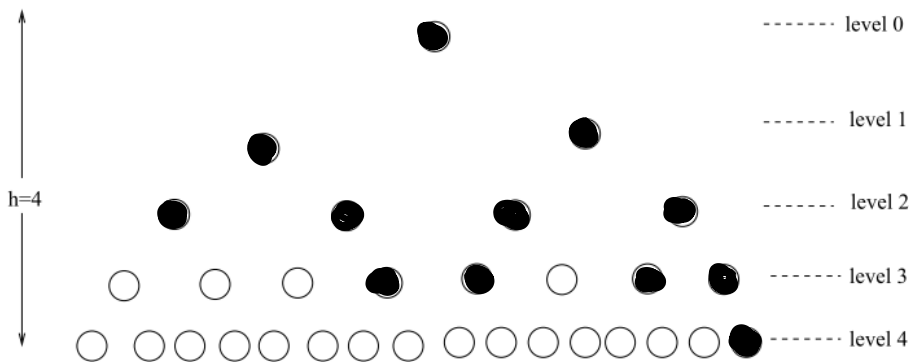
The average height of a binary heap is  $\log_2(N)$ .

## ~~ Trees ~~

9. [2] For Depth-first (pre, post, & in-order) tree traversals, it is often convenient to use a stack data structure to help process the tree nodes.

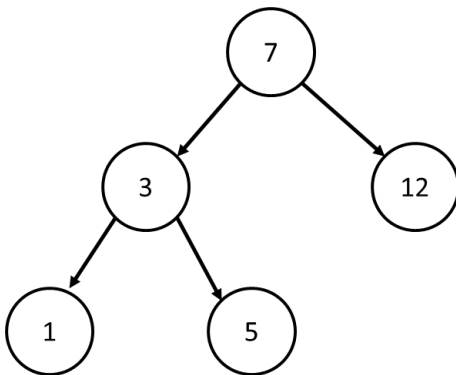
10. [2] For Breadth-first (level order) tree traversals, it is often convenient to use a queue data structure to help process the tree nodes.

11. [2] What are the minimum number of nodes allowed in an AVL tree of height 4? Give your answer both as a number and by shading in the necessary nodes below. Provided is a possible answer to the same question with an AVL tree of height 2.



Height of 4: 15

12. [7] Assuming we will be inserting 2 into the following AVL tree:

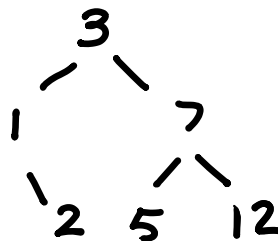


a.

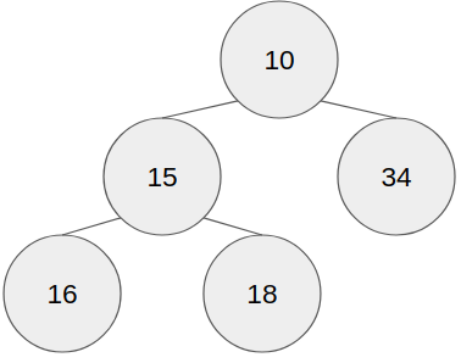
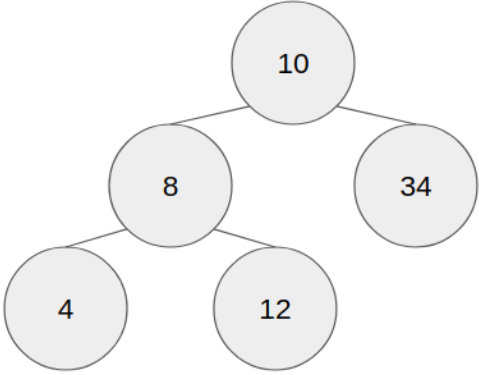
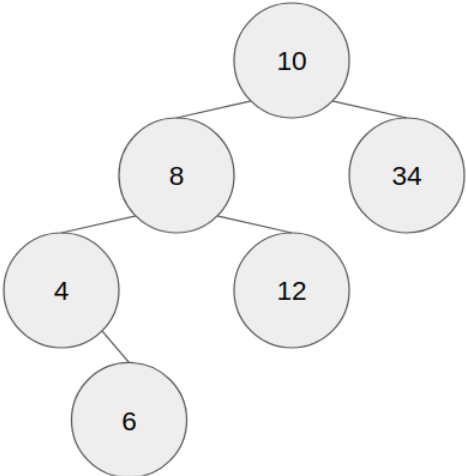
(2) What kind of rotation will be performed?

Right rotation

b. (5) Draw the resulting tree:

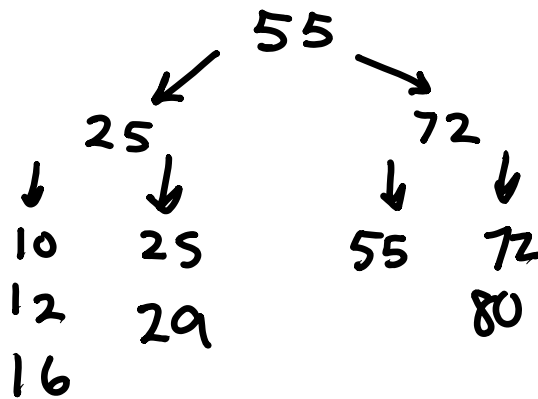
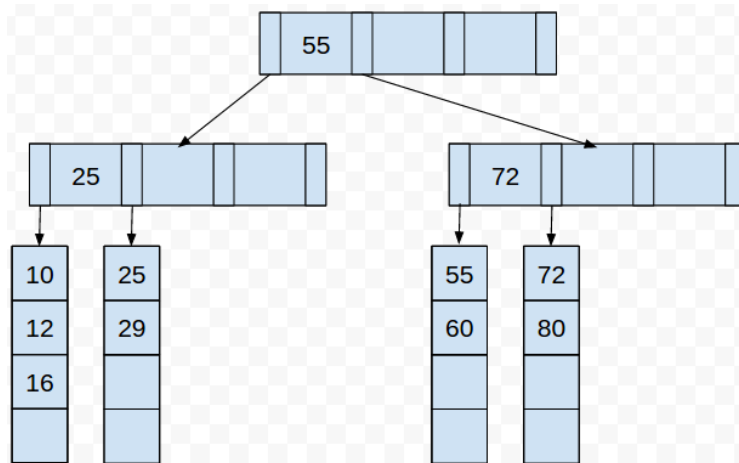


13. [9] Circle whether each tree is a valid Binary Search Tree (BST), AVL tree and/or Binary Heap

Tree	Valid BST?	Valid AVL?	Valid Heap?
 <pre> graph TD     10((10)) --&gt; 15((15))     10 --&gt; 34((34))     15 --&gt; 16((16))     15 --&gt; 18((18)) </pre>	<p>Yes</p> <p>No</p>	<p>Yes</p> <p>No</p>	<p>Yes</p> <p>No</p>
 <pre> graph TD     10((10)) --&gt; 8((8))     10 --&gt; 34((34))     8 --&gt; 4((4))     8 --&gt; 12((12)) </pre>	<p>Yes</p> <p>No</p>	<p>Yes</p> <p>No</p>	<p>Yes</p> <p>No</p>
 <pre> graph TD     10((10)) --&gt; 8((8))     10 --&gt; 34((34))     8 --&gt; 4((4))     8 --&gt; 12((12))     4 --&gt; 6((6)) </pre>	<p>Yes</p> <p>No</p>	<p>Yes</p> <p>No</p>	<p>Yes</p> <p>No</p>

14. [4] The following B+ tree has  $M = 3$ ,  $L = 4$ .

Remove the record with the key 60 from the tree and draw the results.



15. [2] What is the primary reason to use a B+ Tree?

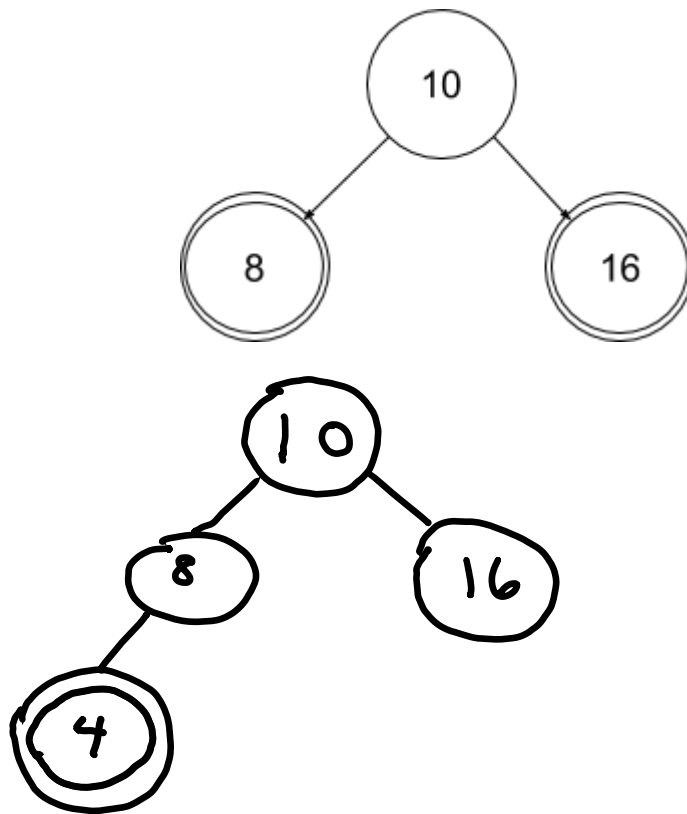
The main reason we use B+ trees is to reduce the number of disk accesses and height of the tree. Since the height of the B-tree is low so total disk accesses for most of the operations are reduced significantly compared to binary search trees.



16. [2] List the four fundamental structural rules of a red-black tree:

- 1: Every node is colored either red or black
- 2: The root is always black
- 3: If a node is red, the children must be black
- 4: Every path from a node to a null pointer must contain the same number of black nodes.

17. [4] Please draw the tree as it should be when we're done with an insert(4) operation:



## ~~ Parallel Programming ~~

18. [4] What is parallel programming? Explain.

Parallel programming is the usage of multiple cores/processors to perform a set of partitioned various tasks in a program so that they can run in coordination to speed up the runtime of the program/operations. Parallel programming is mainly used to decrease the runtime of a program.

19. [4] When should we use critical regions in OpenMP? Explain.

We use critical regions in OpenMP when you want threads to perform an operation one at a time. We do this if we need to perform tasks in sequential order.

20. [4] What is meant by synchronizing tasks? Explain.

Synchronizing tasks makes sure that each core works at the same speed so that one core does not get too far ahead of the rest. This imposes constraints and is used to protect access to shared data.

## ~~ Sorting ~~

21. [6] For each of the following sorting algorithms, give the best-case and worst-case running times for sorting an array of size  $N$  and note the stability of the sort.

<u>Algorithm</u>	<u>Best Case</u>	<u>Worst Case</u>	<u>Is Sort Stable?</u>
InsertionSort	$O(n)$	$O(n^2)$	Yes
HeapSort	$O(n \log(n))$	$O(n \log(n))$	No
QuickSort	$O(n \log(n))$	$O(n^2)$	No
Radix Sort	$O(nk)$	$O(nk)$ (k is the # of digits in the largest number)	Yes

22. [8] Perform quick sort on the following numbers: [8, 3, 9, 6, 2, 5, 4, 1].

8 3 9 6 2 5 4 1

8 3 9 1 2 5 4 6

4 3 9 1 2 5 8 6

4 3 5 1 2 9 8 6

4 3 5 1 2 6 8 9

4 3 2 1 5 6 8 9

4 1 2 3 5 6 8 9

2 1 4 3 5 6 8 9

2 1 3 4 5 6 8 9

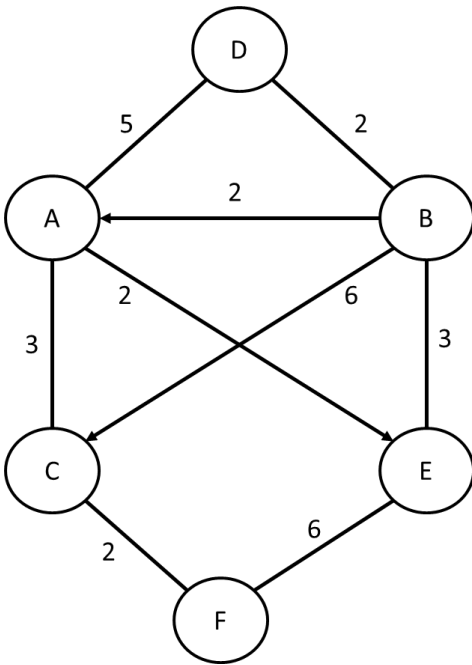
1 2 3 4 5 6 8 9

## ~~ Graphs ~~

23. [1] What are the requirements for a graph to be considered a DAG?

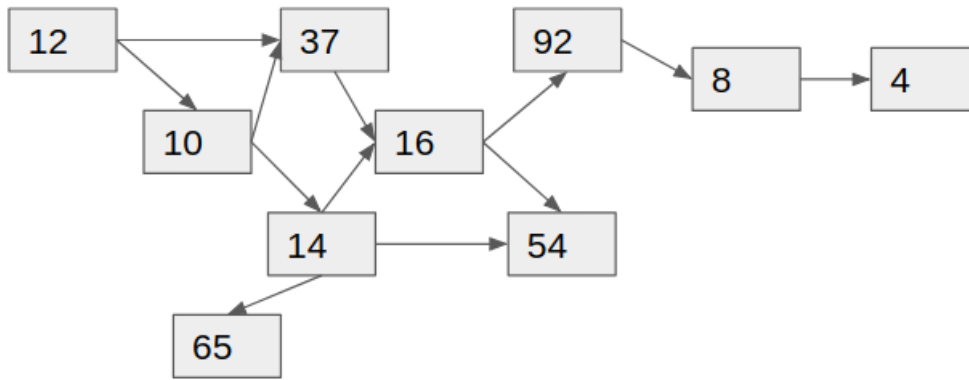
A graph is considered a DAG if it has no cycles.

24. [8] **Dijkstra's Algorithm.** Use Dijkstra's Algorithm to determine the shortest path starting at E. Note that edges without heads are bi-directional. To save time, you do not have to add items to the "priority queue" column after it has been discovered (listed in the "distance" column). Use the table below to show your work.



Node: Distance	Priority Queue
E: 0	(E, E) 0
B: 3	(E, B) 3
F: 6	(E, F) 6
D: 5	(B, D) 5
A: 5	(B, A) 5, (D, A) 10,
C: 8	(B, C) 9, (F, C) 8, (A, C) 8

25. [2] Emit a valid vector of results from topological sort when run on this graph:



{12, 10, 14, 65, 37, 16, 54, 92, 8, 4}

26. [6] Time for some graph definitions in short answer form. What are:

Directed graph: Are when the edges are directed (have a front and back) and are shown in arrow form

Undirected graph: Are when the edges do not have a front and back and are shown in line form.

Adjacency: Vertex  $w$  is adjacent to  $v$  if and only if  $(v, w) \in E$ . When you can traverse the edge from  $v$  to  $w$ .

Path: A sequence of vertices

Cycle: Is a non-empty trail in which the only repeated vertices are the first and last vertices.

Connected Graph: A graph in which there's a path from every vertex to every other vertex.

**27. [2]** In your own words, describe the purpose behind doing and using Big-O analysis:

The purpose of using Big-O analysis is to identify the efficiency of the programs/algorithms you make. By learning the time complexity of your algorithms, you can learn how efficient your algorithms are and how much time they will take to execute.

**28. [2]** What is Git and name 3 key things (not commands, but kinds of activities) it enables developers to do:

Git is a tool that programmers use to upload their code to servers online so they can write a program from many different computers, create different versions of their programs, revert their changes if they upload broken code, as well as collaborating with other team members. We are going to use Git to upload our programming assignments to our repositories.

**29. [2]** What is your favorite data structure or algorithm? Why?

My favorite data structure are hash tables because they have  $O(n)$  time complexity when inserting, deleting and searching for elements within the list.