3013 - Alorithms
Spring 2020
Exam 1
March 9, 2020

Name: _____

READ THESE INSTRUCTIONS	
Use pencil only	Write name at the top of all pages turned in.
Staple pages together at the top left corner.	Illegible handwriting will lose points.
Make sure your pages are in order when stapled.	All questions written in order on paper provided.
Write answers on the paper provided, not directly on the test.	Indentation matters. Keep code aligned correctly.
Lists of items should be written vertically not horizontally.	
GOOD A: Answer a B: Answer b ..... N Answer n	BAD A) Answer a B) Answer b ..... N) Answer n
Failure to comply will result in loss of letter grade.	

This exam is 10 pages (without cover page) and 14 questions. Total of points is 270.

Grade Table (don't write on it)

Question	Points	Score
1	5	
2	20	
3	20	
4	20	
5	10	
6	15	
7	10	
8	30	
9	30	
10	5	
11	40	
12	40	
13	20	
14	5	
Total:	270	

1. (5 points) Flip back to the cover page. Write your name there.

---

2. (20 points) List the complexities from fastest to slowest.

Complexity Choices

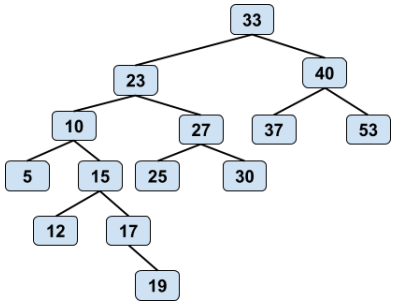
$O(n!)$ $O(2^n)$ $O(1)$ $O(n\ lg\ n)$ $O(n^2)$ $O(n^n)$ $O(n)$ $O(log\ n)$ None of These

Answer:

1. $O(1)$
2. $O(log\ n)$
3. $O(n)$
4. $O(n\ lg\ n)$
5. $O(n^2)$
6. $O(2^n)$
7. $O(n!)$
8. $O(n^n)$

---

3. (20 points) Given the following binary tree:



Answer:

1	What is the height of this tree?	5 or 6
2	What is the root of this tree?	33
3	How many leaves does this tree have? List them.	5,12,19,25,30,37,53
4	How many descendants does 23 have?	9
5	How many siblings does 23 have?	1
6	Who is the predecessor of 23?	19
7	Who is the successor of 23?	25

---

4. (20 points) Assign the correct complexity to each item below.

Complexity Choices

$O(n!)$ $O(2^n)$ $O(1)$ $O(n\ lg\ n)$ $O(n^2)$ $O(n^n)$ $O(n)$ $O(log\ n)$ None of These

Answer:

A	$O(log\ n)$	Inserting an element into a balanced binary search tree.
B	$O(n)$	Finding an element in a list.
C	$O(n)$	Finding an element in an ordered list.
D	$O(log\ n)$	Removing an element from a binary heap.
E	$O(log\ n)$	Finding an element in a binary search tree.
F	$O(log\ n)$	Adding an element to a binary heap.
G	$O(n)$	Building a binary heap given an array of values.
H	$O(n\ log\ n)$	Building a binary heap given a linked list of values.
I	$O(1)$	Remove an item from a linked list of values.
J	$O(n)$	Insert an item into an ordered linked list of values.

---

5. (10 points) **Heapify**: Describe what it does, and why it is significant. Be thorough.

Answer:

- **Heapify** takes an array of values, and turns it into a heap (min or max depending on need).
- It is significant because it can accomplish creating a heap faster than inserting items into an empty heap 1 item at a time.
- If we insert n items into a heap at a cost of $O(lg\ n)$ per insert, we can build a heap with a cost of $O(n\ lg\ n)$. Not bad, but using **Heapify** only costs us $O(n)$!!
- How does it do this in linear time? Well we could prove it by solving a summation and doing some substitutions and removals of constants etc. (Heap Proof), OR we can just understand that we don't need to process the entire array of values because the entire bottom row of our tree (more than half the nodes) are leaf's. And by processing only the top half of the array (swapping as necessary with children) we end up with an array in heap order. Yes, some values may need to be swapped more than once, but this also is shown to be a "constant" time cost, which can be ignored :/

---

6. (15 points) Write a pseudo code implementation for deleting a node from a binary search tree.

Answer:

- Assuming we find the node with the "key" in it that needs deleting.
- **1) No children:** Simply delete the node with the key. Set parents pointer to $NULL$.
- **2) One child:** Delete the node with the key, and have the parent take over its single child.
- **3) Two children:** this is indeed the "complicated" case.
  1. Find the rightmost node of the left child (or the leftmost node of the right child) of the node with key in it.
  2. Swap the found value with node that needs deleting (its either its predecessor or successor so it doesn't "break" the order in the tree.
  3. Delete the node it was swapped into since it should be an instance of case 1 or case 2

,

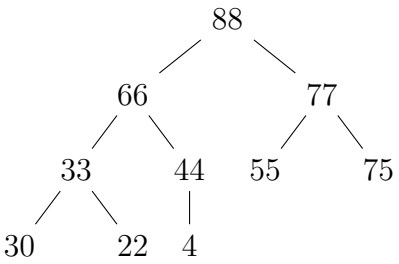
Answer (w/ code):

```
1   if ( key not in BST )
2   {
3           return;         // Nothing to delete
4   }
5
6   if ( key has no children )              //     x           x
7   {                                       //    / \    ==>   /
8           delete key from keys parent;        //   y   k        y
9           return;                             //
10  }
11
12  if ( k has 1 child )                            //     x           x
13  {                                               //    / \    ==>   / \
14          make keys parent point to keys child;       //   y   k        y   z
15          return;                                     //        \
16  }                                                           z
17
18  /* k has 2 subtrees - TOUGH */
19
20  //(1)  find the successor of key:
21  //         go right once
22  //       go left all the way down
23  //
24  // (2) Replace key with keys successor;
25  //
26  // (3) Make the successors parent point to the successors right subtree;
```

7. (10 points) Take the values from the max heap below and draw an array that would represent this tree.

   Put the answer on your answer sheet using something similar to:

0	1	2	3	4	5



Answer:

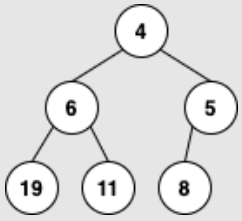
x	88	66	77	33	44	55	75	30	22	4
0	1	2	3	4	5	6	7	8	9	10

---

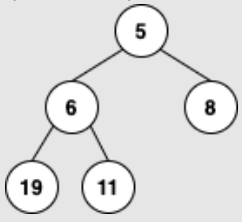
8. (15 points) Given a sequence of numbers: 19, 6, 8, 11, 4, 5
   (a) Draw a binary min-heap (in a tree form) by inserting the above numbers reading them from left to right
   (b) Show a tree that can be the result after the call to deleteMin() on the above heap
   (c) Show a tree after another call to deleteMin()

Answer:

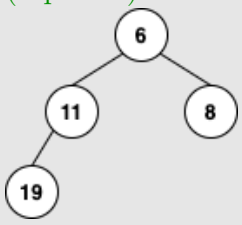
(a) (5 points) Draw a binary min-heap (in a tree form) by inserting the above numbers reading them from left to right



(b) (5 points) Show a tree that can be the result after the call to deleteMin() on the above heap



(c) (5 points) Show a tree after another call to deleteMin()



9. (30 points) List vs Array based data structures. Given a statement below, choose:
List, **Array**, **Both**, **None**
to indicate what the statement is implying.

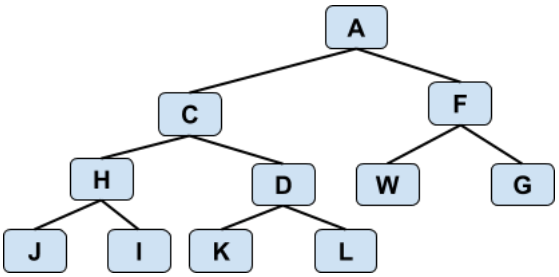
Choices: List Array Both None

Answer:

A.	Array	Directly access element in this structure.
B.	Array	Bounded by size.
C.	List	Easy to insert and delete from.
D.	Array	Easy to implement.
E.	List	More overhead.
F.	Both	Can be sorted.
G.	List	Must be allocated in the heap.
H.	Array	Expensive to resize.
I.	List	Grows and shrinks easily.
J.	Array	Binary search can be performed on this.
K.	Both	Easily access each element in this structure.
L.	None	Cannot be allocated in the heap.
M.	None	Must be statically declared.
N.	List	Items added to front or rear.
O.	List	Easier to delete from middle of structure.
P.	Both	Can be used to represent a binary tree.

Reference: Array vs Linked List

10. (5 points) Give a pre-order / post-order /in-order traversal of the following tree:



Answer:

Pre	A	C	H	J	I	D	K	L	F	W	G
In	J	H	I	C	K	D	L	A	W	F	G
Post	J	I	H	K	L	D	C	W	G	F	A

11. (20 points) Stack based memory VS Heap based memory.
    (a) (10 points) Pros and cons of a Stack
    (b) (10 points) Pros and cons of the Heap

Answer:

- **Stack Pro's**
  1. There is no need to manage the memory yourself.
  2. The stack grows and shrinks as it needs to based on need.
  3. Reading from and writing to stack variables is very fast!
  4. No fragmented memory (cpu managed).
- **Stack Con's**
  1. Local variables only.
  2. Limit on stack size (OS-dependent).
  3. Variables cannot be resized.
- **Heap Pro's**
  1. Variables can be accessed globally.
  2. No limit on memory size.
- **Heap Con's**
  1. Relatively slower access.
  2. No guaranteed efficient use of space.

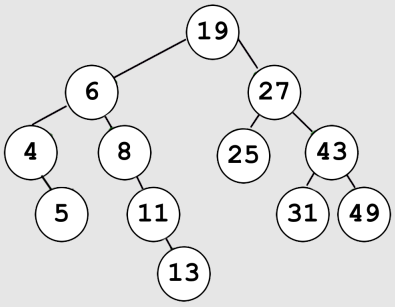
Reference: Stack Vs Heap

12. (20 points) (a) (10 points) Draw the resulting binary search tree by inserting the following values from left to right: **19, 6, 8, 11, 4, 13, 5, 27, 43, 49, 31, 25**

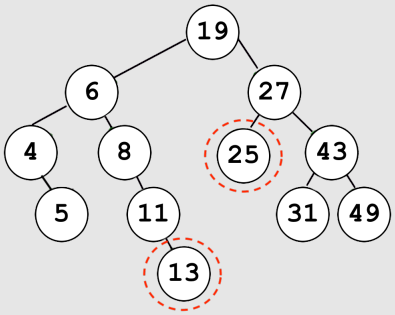
(b) (10 points) Delete **19** from the binary search tree in part (a) using a standard removal algorithm for binary search trees. Draw the TWO potential binary search trees that you can end up with.

Answer:

12a The resulting binary search tree:



12a What are my 2 choices?



12b The TWO potential binary search trees that you can end up with

SEE ABOVE CHOICES

13. (20 points) You are attending a party with n other people. Each other person i arrives at the party at some time s_i and leaves the party at some time t_i (where $s < t_i$). Once a person leaves the party, they do not return. Additionally, each person i has some coolness c_i. At all times during the party, you choose to talk to the coolest person currently at the party. (All coolness values are distinct.) If you are talking to someone, and someone else cooler arrives at the party, you leave your current conversation partner and talk to the new person. If the person you are talking to leaves the party, you go talk to the coolest person remaining at the party. (This might or might not be a person with whom you have already talked.) You are the first to arrive at the party and the last to leave. Additionally, you are the most popular person at the party, so everyone wants to talk with you.

    Describe a data structure which allows you to decide in $O(1)$ time to whom you should talk at any moment. You should be able to update this data structure in $O(log\ n)$ time when someone arrives or leaves.

Answer:

Note:
The problem states that we are the most **popular** person at the party. Not necessarily the **coolest**.

Correct Answer - Heap:
A heap will always have the value with the highest (or lowest) priority at the top of the heap. If we set the priority to be **coolness**, then the coolest person will always be at the top of the heap. This also lets us access the coolest person in $O(1)$ (constant) time. Whenever someone arrives or leaves, a heap can be updated in $O(log\ n)$ time. Why? Because a heap is always a **complete tree** which inherently means it is a balanced tree with a height of no more that $lg\ n$.

Binary Search Tree:
A binary search tree has similar properties to a heap, but is not perfect for the job as stated in the problem. We can always find the coolest person if we build the tree using "coolness" as the key. BUT! It would take $O(log\ n)$ time to find the coolest person not $O(1)$ or constant time. This also assumes that we maintain a balanced tree somehow. When individuals arrive or leave, it takes $O(log\ n)$ time to insert or remove them (same as a heap) as long as we can maintain a balanced tree. This is all good, but NOT as good as a heap.

Array:
What if we maintained an ordered array based on coolness? We could then access the "coolest" person in constant time $O(1)$, right? Sure. But whats the cost to build and maintain this structure? The problem states that we are the first one at the party and the last one to leave. So, we end up updating the array for everyone that shows up (all n people). The cost to put them into the ordered array is $O(n)$. Remember, worst case: they might be so un-cool they have to go to the very end of the array. You also have to do this n times. This means that maintaining an ordered array would cost $O(n^2)$! Not even close to a heap.

Stack:
Does a stack fit this problem? Hopefully you can see why it does not. A stack is totally based on an order of occurrence. We cannot "sort" or alter a stack based on anything other than "when" something happened. So, this is really not a good fit.

Queue:
If we use a very specific type of queue (aka a Priority Queue) then yes! OH! The ordered array is a priority queue and its performance was not as good as a heap .... darn it! Wait! A heap IS a priority queue! So yes, if you used a queue data structure, implemented using an array based binary tree, then you get full credit! (aka look at first answer ;))

14. (5 points) Write your name on your answer sheets.