3013 - Algorithms
Spring 2023
Exam 1
February 17 2023

Name: _____

READ THESE INSTRUCTIONS

- Use pencil only
- Initial top right corner of all pages (except the first one).
- Do not remove the staple from your exam.
- Do not crumple or fold your exam.
- Handwriting that is illegible (messy, small, not straight) will lose points.
- Indentation matters. Keep code aligned correctly.
- Answer all questions on the answer sheet(s) provided.
- It may be the test itself, or it may be sheets given to you separately. If you are not sure ask. But I can assure you it is either one or the other and not a mixture of both.
- Help me ... help you!

Failure to comply will result in loss of letter grade

Grade Table (don't write on it)

Question	Points	Score
1	16	
2	20	
3	15	
4	20	
5	28	
6	35	
7	10	
8	10	
Total:	154	

Use the following complexity choices for the following two questions:

$O(n!)$	$O(2^n)$	$O(1)$	$O(n\ lg\ n)$	$O(n^2)$	$O(n^n)$	$O(n)$	$O(log\ n)$
A	B	C	D	E	F	G	H

---

1. (16 points) List the complexities from fastest to slowest. Look at the complexities from above and place the corresponding letters from fastest to slowest in the boxes below:

Solution: The spreadsheet below shows the growth of each choice based on the N value in column 1. The columns go from least cost on the left to greatest cost on the right

Fastest Slowest

	C	H	G	D	E	B	A	F
N	O(1)	O(lg N)	O(N)	O(N lg N)	O(N^2)	O(2^N)	O(N!)	O(N^N)
2	1	1	2	2	4	4	2	4
3	1	2	3	5	9	8	6	27
4.5	1	2	4.5	10	20.25	22.627417	24	869.8739234
6.75	1	3	6.75	19	45.5625	107.6347412	720	396096.0091
10.125	1	3	10.125	34	102.515625	1116.679918	3628800	15122538466
15.1875	1	4	15.1875	60	230.6601563	37315.82598	130767436800	8.78664E+17
22.78125	1	5	22.78125	103	518.9853516	7208411.797	1.124E+21	8.45889E+30
34.171875	1	5	34.171875	174	1167.717041	19353494041	2.95233E+38	2.56058E+52
51.2578125	1	6	51.2578125	291	2627.363342	2.6924E+15	1.55112E+66	4.35072E+87
76.88671875	1	6	76.88671875	482	5911.56752	1.39704E+23	1.88549E+111	9.92934E+144
115.3300781	1	7	115.3300781	790	13301.02692	5.22171E+34	2.92509E+188	6.36795E+237

---

2. Use the corresponding letters from the complexities above to answer the following questions (or none if an appropriate value doesn't exist).

   (a) (2 points) Inserting an element into a balanced binary search tree.

   (a) **H**

   (b) (2 points) Finding an element in a list.

   (b) **G**

   (c) (2 points) Finding an element in an ordered list.

   (c) **G**

   (d) (2 points) Removing an element from a binary heap.

   (d) **C**

   (e) (2 points) Finding an element in a binary search tree.

   (e) **H**

(f) (2 points) Adding an element to a binary heap.

(f) **H**

(g) (2 points) Building a binary heap given an array of values.

(g) **G**

(h) (2 points) Building a binary heap given a linked list of values.

(h) **D**

(i) (2 points) Remove an item from a linked list of values.

(i) **G**

(j) (2 points) Insert an item into an ordered linked list of values.

(j) **G**

3. (15 points) **Heapify**: Describe what it does, and why it is significant. Be thorough.

Solution: Heapify is the generic heap function that takes an array of unordered values, and puts them into "heap order". The significance of this method is that it can complete its task in ***O(N)*** time, and is what allows heap sort to work in ***O(n lg n)***.

Why is this possible? It's because of the relationship between inner nodes and leaves. Remember that a full complete tree has more leaves than inner nodes. So if we start process at the first inner node, and work our way up the tree, we only have to process half the array. If were only processing half of an array that represents a tree that's already bounding by a height of ***lg n***, (along with some hand waving) we end up with a cost of O(N).

4. Given a sequence of numbers: **19, 6, 8, 11, 4, 5**

   (a) (10 points) Draw a binary min-heap (**in array form**) by inserting the above numbers reading them from left to right.

- Insert 19 at index 1, and its a heap! Nothing needs done.

Insert 19:						
X	19					
0	1	2	3	4	5	6

- Insert 6 at index 2, but its smaller than parent so swap to index 1.

Insert 6:						
X	6	19				
0	1	2	3	4	5	6

- Insert 8 at index 3, it is NOT smaller than parent so nothing happens.

Insert 8:						
X	6	19	8			
0	1	2	3	4	5	6

- Insert 11 as next available (index 4). It is smaller than parent (19) so we swap.
- It is not smaller than 6 so it stops bubbling up.

Insert 11:						
X	6	11	8	19		
0	1	2	3	4	5	6

- Insert 4 at next available (index 5) then start bubbling up.
- 4 is smaller than its parent 8 so swap.
- 4 is smaller than its new parent 6, so swap again.

Insert 4:						
X	4	11	6	19	8	
0	1	2	3	4	5	6

- Insert 5 at next available spot (index 6).
- 5 is smaller than parent (value 6 at index 3) so we swap.
- 5 is not smaller than the next parent (index 1) so we stop.

Insert 5:						
X	4	11	5	19	8	6
0	1	2	3	4	5	6

(b) (5 points) Show the min-heap after a after you deleteMin()

- Swap the minimum item with the last item (4 with 6).
- Now bubbleDOWN if necessary by comparing the new root to its children and choosing the smallest to swap with.
- 6 is bigger than its child 5, so we swap.
- Nothing else is needed.

Delete Min						
X	5	11	6	19	8	
0	1	2	3	4	5	6

(c) (5 points) Show the min-heap after another call to deleteMin()

- Swap the minimum item with the last item (8 with 5).
- Now bubbleDOWN if necessary by comparing the new root to its children and choosing the smallest to swap with.
- 8 is bigger than its child 6, so we swap.
- Nothing else is needed.

Delete Min						
X	6	11	8	19		
0	1	2	3	4	5	6

5. List vs Array based data structures. Given a statement below, choose:

List, **Array**, **Both**, **None**

to indicate what the statement is implying.

(a) (2 points) Possible to directly access an element in this structure.

(a) **Array**

(b) (2 points) Is bounded by size.

(b) **Array**

(c) (2 points) Easy to insert and delete from while not having empty nodes / slots.

(c) **List**

(d) (2 points) Easiest to implement.

(d) **Array**

(e) (2 points) Has more overhead.

(e) **List**

(f) (2 points) This structure can easily be sorted.

(f) **Array**

(g) (2 points) Must be allocated in the heap.

(g) **List**

(h) (2 points) This structure is expensive to resize.

(h) **Array**

(i) (2 points) Which structure grows and shrinks easily.

(i) **List**

(j) (2 points) Binary search can easily be performed on this sorted structure.

(j) **Array**

(k) (2 points) Searching this structure can be done in $O(N)$.

(k) **Both**

(l) (2 points) Can add items added to either end of this structure easily.

(l) **List**

(m) (2 points) It is easier to manipulate the middle of this structure without big costs.

(m) **List**

(n) (2 points) Can be used to represent a binary tree.

(n) **Both**

6. Given the following list of numbers: **27, 11, 6, 8, 19, 4, 43, 49, 31** process them from left to right and draw your resulting structure as indicated below.

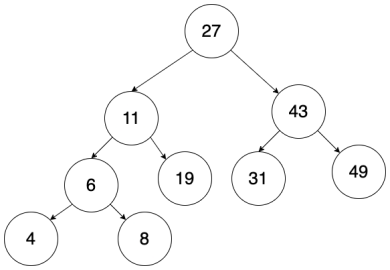
   (a) (10 points) Array based binary search tree

Solution:

	27	11	43	6	19	31	49	4	8
0	1	2	3	4	5	6	7	8	9

   (b) (10 points) Traditional binary search tree

Solution:



For the next three questions, if you fear that the tree you created in the previous question is wrong, then explain what each of the following terms actually mean for partial credit.

   (c) (5 points) Is this tree **complete**?

Solution: Yes *(Filled in row wise from left to right with no gaps in the tree.)*

   (d) (5 points) Is this tree **full**?

Solution: No *(Every available spot is filled OR there are N+1 leaf nodes vs inner nodes.)*

   (e) (5 points) Is this tree **balanced**?

Solution: Yes *(The left and right subtrees differ in height by no more than 1.)*

7. (10 points) Describe one or more algorithms that you could use to implement a function that takes in a list of integers and ***returns the two numbers in the list that sum up to a given target value***. The list your function receives is not sorted. What data structure and what alogorithm would you use to return the two numbers in sorted order? What is the cost of your algorithm?

Solution: A heap data structure implemented as a "priority queue" could work. This allows for efficient insertion and removal, but also will "sort" our items as needed.

Here's how the algorithm works:

Initialize a priority queue, which is a min-heap in this case. Iterate through the input list and insert each element into the priority queue that is smaller than the requested sum (since anything larger cannot be added to another value and obtain the sum). For each element in the priority queue:

- Calculate the difference between the target value and the current element.
- Check if the difference exists in the priority queue (linear search).
- If the difference exists, return the current element and the difference.

If no two values sum up to the target value, return an empty list or a message indicating that no such pair exists.

I will take into consideration any of your "algorithms" that include sorting and a viable list of steps. However, we have only discussed a handful of algorithms in class which should have helped you decide on your solution.

8. (10 points) You have a large dataset of employee records, each containing employee information that includes an employee's ID. Describe which data structure and which algorithm you would use to ***efficiently look up an employee's name given their ID***. Your cost should be no more than O(log n) or better.

Solution: One data structure that can be used to efficiently look up an employee's name given their unique ID is a binary search tree (BST) with the employee ID as the key. We **WANT** the tree to be balanced so we get guaranteed performance! Otherwise it is not ***O(lg N)*** search ... it degrades to ***O(h)***!

A BST is a binary tree in which the heights of the left and right subtrees of any node differ by at most one. This ensures that the worst-case time complexity of lookup, insertion, and deletion operations is ***O(log n)***, where ***n*** is the number of nodes in the tree.

I understand that "self balancing" trees were not on this exam, so, simply addressing the fact that BST's may degrade to be unbalanced would suffice in justifying your answer. But if you do mention AVL or Red BLack trees I will accept it.