Name: *(as it would appear on official course roster)*	
UCSB email address:	**@ucsb.edu**

Homework 03: Heaps
Points: *100*

- MAY ONLY BE TURNED IN ON **GRADESCOPE as a PDF file**.
- There is NO MAKEUP for missed assignments.
- We are strict about enforcing the LATE POLICY for all assignments (see syllabus).
  Only use the space provided for answers. Use clear and clean handwriting (or typing).
  NOT USING THIS TEMPLATE WILL LOSE YOU 20 POINTS!

Course Textbook – I'll use the following 'shorthand':

• **DS**: *Data Structures and Algorithm Analysis in C++, 4th Edition* by Mark Allen Weiss

This book is required for this course and homework assignments will often ask you to read and comment on sections from this book.

---

1. **(20pts) Finding Nodes Less Than a Given Value in a Binary Heap**. Design an algorithm that, given a binary *min-heap* and a value `X`, outputs all nodes in the heap whose values are less than `X`. Assume that the heap is implemented with **an array** and you have access to the **size** of the heap. Your algorithm should run in **O(K)** time, where **K** is the number of nodes returned.
   a. **(10pts)** Provide an algorithm.

b. **(5pts)** Provide a justification for the correctness of your algorithm.

c. **(5pts)** Explain why your algorithm meets the time bound**.**

2. (20pts) Locating a Node by Implicit Position in a Pointer-Based Heap. Binary heaps are often represented as arrays, where the node at position `i` has children at positions `2i` and `2i + 1`. Suppose instead that a binary heap is implemented as a binary tree with explicit left and right child pointers. Design an algorithm that, given an integer `i`, finds the node that would be at position `i` in the array representation. Assume the root is at position 1.

a. **(10pts)** Provide an algorithm that runs in $O(log\ i)$ time in the worst-case.

<u>Hint</u>: Look at the **binary representation** of the index i. After the first bit, how could the remaining bits serve as a "map" to tell you whether to turn left or right at each level?

b. **(5pts)** Provide a justification for the correctness of your algorithm.

c. **(5pts)** Explain why your algorithm meets the time bound.

3. **(10pts) Insertion into a Binary Min-Heap**. Starting from an empty binary min-heap, insert the following elements one at a time in the given order:

   `10, 12, 1, 14, 6, 5, 8, 15.`

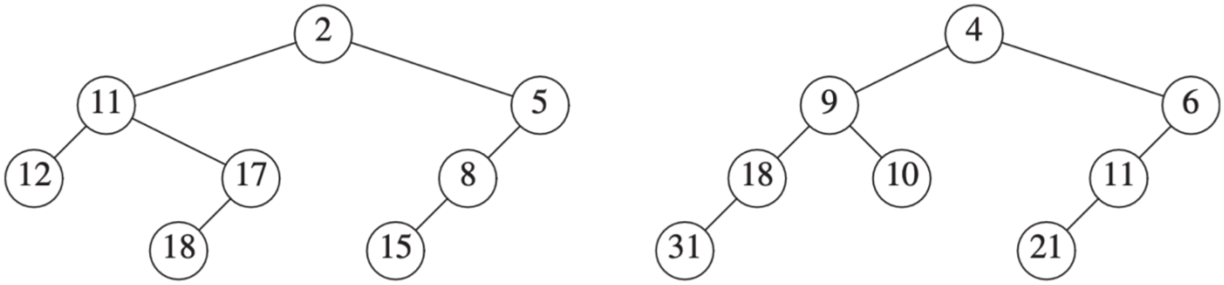
   a. **(5pts)** Use the heap's insert operation for each element. Show the state of the heap (as a tree) after each insertion.

b. **(5pts)** Use the O(n) `buildHeap` function to create the heap. Show the state of the heap (as a tree) after each insertion.

4. **(10pts) Performing deleteMin on a Binary Heap**. Starting from the heap you constructed in part b of the previous question, perform three `deleteMin` operations in sequence. Show the state of the heap after each deletion.

5. **(10pts) Exploring Leftist Heaps and Null Path Lengths**. A leftist heap maintains the **null path length (npl)** property at each node. For each of n = 1, 2, 3, 4, draw all structurally distinct valid leftist heaps. Annotate each node in your drawings with its null path length.

6. **(10pts) Merging Two Leftist Heaps.** Given the two leftist heaps shown below, perform a merge operation on them. Show the intermediate steps of the merge process, and annotate the final merged heap with the null path lengths of each node.



7. **(20pts)** This is a general algorithm design question that covers topics from any lecture.

   Design a matchmaking system for a competitive game that manages a pool of players based on their **skill ratings**. The system must support the following operations:

   1. **`join(rating)`**: Add a new player's rating to the pool.
   2. **`leave(rating)`**: Remove a specific rating from the pool when a player goes offline.
   3. **`findClosest(rating)`**: Return the rating currently in the pool that is numerically closest to the target rating (to ensure a fair match).

   To handle high-traffic servers, all three operations (`join`, `leave`, and `findClosest`) must run in $O(logn)$ **worst-case time**, where n is the number of players in the pool.

   a. **(8pts)** Which data structure would you use to meet this specific worst-case time complexity? Explain your rationale.

   b. **(12pts)** Explain how the `findClosest` operation works within your chosen structure to ensure it never exceeds $O(logn)$.