CISC2006: Algorithm Design and Analysis
Homework 1 **Due Date:** 23:59, Feb 02, 2024

Rules: Discussion of the problems is permitted, but writing the assignment together is not (i.e. you are not allowed to see the actual pages of another student). You can get at most 100 points if attempting all problems. Please make your answers precise and concise.

1. (15 pts) Give "True" or "False" for each of the following statements.

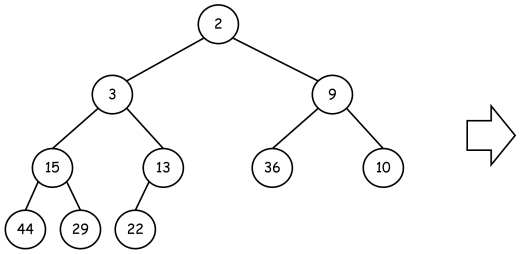
   No proof is needed.

   - $T(n) = 4^{\log_2 n} + 9\sqrt{n} = O(n^2)$.
   - $T(n) = n^2 + n = O(n^3)$.
   - $T(n) = 5n^2 + n \cdot \log_2 n = \Omega(n)$.
   - $T(n) = 20n \cdot \log_2 n + 5n = \Omega(n^2)$.
   - $T(n) = 4 \cdot (\log_2 n)^5 + 5\sqrt{n} + 10 = \Theta(\sqrt{n})$.

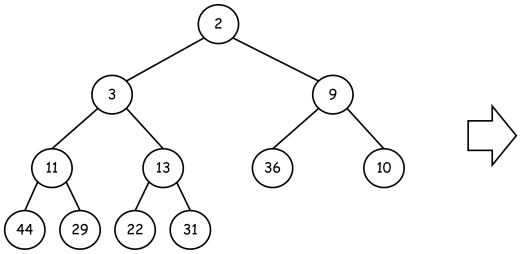
2. (10 pts) Prove that function $T(n) = \sum_{i=1}^{n} \log i = \Theta(n \log n)$.

3. (15 pts) Draw the heap (as a binary tree) after executing the following operations

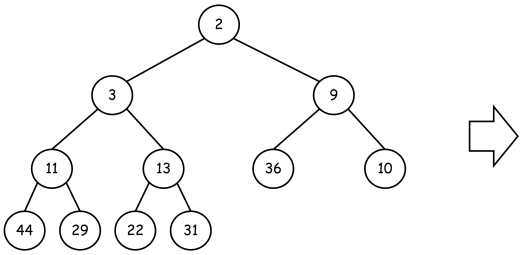
- removeMin from the following heap:



- removeMin from the following heap:

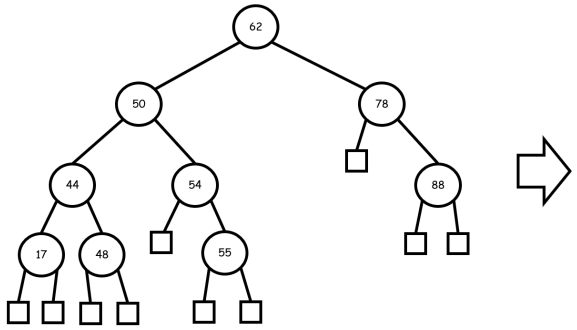


- Insert element 5 into the following heap:

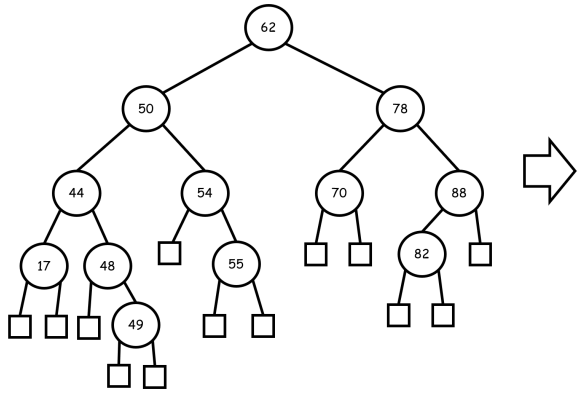


4. (20 pts) Draw the AVL tree (as a binary tree) after executing the following operations

   - Insert element 49 into the following AVL tree:



   - Delete element 62 from the following AVL tree:



5. (10 pts) You are given an implementation of binary search tree (e.g., AVL tree) that supports find(k) (finding the element with key $= k$) in $O(\log n)$ time, where n is the total number of elements. Design a function findAll(k_1, k_2) to find all elements with keys in $[k_1, k_2]$ in $O(\log n + s)$ time, where s is the output size.

   Present your algorithm in pseudocode, prove its correctness and analyze its complexity. You can assume that all elements have different integer key values.

6. (20 pts) **Different implementations of Priority Queue**.

Implement a priority queue data structure class to store a collection of different numbers that supports the following operations:

- insert(e) : insert an element e into the priority queue;
- min() : return the minimum element in the priority queue;
- removeMin() : remove the minimum element in the priority queue;
- size() : return the total number of elements in the priority queue;
- isEmpty() : return True if the priority queue is empty; False otherwise.
- printPQ() : list all elements in the priority queue. For a heap, list the elements from top-level to bottom-level, and for each level from left to right.

Use the following data structures for three different implementations:

- Unsorted Doubly Linked List;
- Sorted Doubly Linked List;
- Heap (implemented using an array).

In the main function, we read an array $A = \{a_1, a_2, \ldots, a_n\}$ of n different integers, and use the three different implementations of priority queue to do sorting.

In particular, we do the following for each version of priority queues.

We first initialize a priority queue, which is empty. Then we insert the numbers in A one-by-one, and sort the numbers into another array $B = \{b_1, b_2, \ldots, b_n\}$ for output by repeatedly calling min() and removeMin().

7. (30 pts) **Different implementations of Binary Search Tree**.

Implement a binary search tree (BST) data structure class to store a collection of different numbers that supports that following operations

- insert(e) : inset element e into the BST;
- find(k) : return the pointer that points to an element with key = k; if no such element exists, return Null;
- remove(k) : remove the element with key = k if such element exists;
- remove(p) : remove the element pointed by pointer p;
- size() : return the total number of elements in the BST;
- isEmpty() : return True if the BST is empty; False otherwise.
- printTree() : print the whole BST (use indentation to show the structure).

Use the following data structures for the two different implementations:

- Binary tree without height balance property;
- AVL tree.

In the main function, we will read an array $A = \{a_1, a_2, \ldots, a_n\}$ of n different integers, and insert the numbers into the two different implementations of BST one-by-one. Then we read another array $B \subseteq A$ of integers, each of which appeared in A, and remove the integers in B from the BST.

Finally, we output the resulting BST using printTree().