CST 370 – Homework 1

Due: **2:00pm on February 12, 2019**

Final Submission Deadline: **2:00pm on February 19, 2019**

Changelog:

- 2/1/19: Updated Sample Input 2 in Problem 2
- 1/30/19: Added note in submission section about HackerRank contests.

About

For this homework assignment you have multiple problems focused on Trees, Heaps, Searching, and Sorting. For each problem, you may reuse code or data structures you wrote on previous problems.

Submissions

Each problem on this assignment will need to be submitted on either HackerRank or Kattis (the platform will be specified on the problem).

In order to submit work on HackerRank, you must join the homework 1 contest at https://www.hackerrank.com/cst370-s19-hw1.

You may submit as many times as you would like, I will only look at the final submission before the due date. You must submit something that shows you've put in the effort before the due date in order to resubmit before the final submission deadline (an empty function does not show effort).

After the due date, you may resubmit as many times as you'd like before the final submission deadline. I will only look at the final resubmission before the deadline. Resubmissions will be assessed a 10% deduction.

Scoring

Your score on HackerRank or Kattis will give you *an approximation* of what your grade will be; however, I will read your code and determine your final grade myself, as detailed on the syllabus.

Problems

1	Max Binary Heaps	10 points
2	Complete Trees	12 points
3	Beating Bogosort	12 points
4	Finding Students	8 points
5	Akcija	8 points

1. Max Binary Heaps (10 points)

https://www.hackerrank.com/contests/cst370-s19-hw1/challenges/problem-1-max-binary-heap

For this problem, you will implement a max heap using an array for underlying storage. For example, in C++, the class definition would look like:

```
class MaxHeap {
    vector<int> data;
};
```

You will need to implement the following operations on heaps:

`size` which returns the number of items in the heap.

`max lookup` which returns the max element in the heap.

`extract max` which removes the max element from the heap.

`insert` which takes an integer and adds it to the max heap.

`delete` which takes an index and removes the item at that index from the heap.

For example, in C++, the function headers would be the following:

```
class MaxHeap {
    vector<int> data;
  public:
    MaxHeap() {
        // ...
    }
    int size() {
        // ...
    }
    int maxLookup() {
        // ...
    }
    void extractMax() {
        // ...
    }
    void insert(int data) {
        // ...
    }
    void remove(int index) {
        // ...
    }
};
```

Input

- The first line contains an integer $n, 0 \leq n \leq 100$, denoting the number of commands following it.
- The next n lines will consist of one of 5 commands ("size", "maxLookup", "extractMax", "insert", and "delete").
- The commands "insert" and "delete" will be followed by a space and an integer to perform the command on.
- At the end of the input there will be a blank line.

Your program should initialize an empty max heap and perform the commands given by the input, in sequence, on it. See sample input and output below for a concrete example.

(Note that the argument to "delete" represents the index of the element in the underlying array, not the element itself.)

Constraints

You can assume there will be no invalid input including never calling maxLookup on an empty heap or calling delete on out-of-bounds index. You can assume the max heap will consist only of positive integers with no duplicates.

Output

- For the "maxLookup" command, print the max.
- For the "size" command, print the number of elements in the heap.
- For the "insert", "delete", and "extractMax" commands, print nothing.

See the sample output section for a concrete example.

Sample Input 1	Sample Output 1
25 size insert 5 maxLookup insert 18 maxLookup insert 3 maxLookup insert 12 insert 19 maxLookup extractMax maxLookup insert 25 insert 6 insert 1 size extractMax extractMax extractMax size delete 1 delete 1 delete 0 size maxLookup	0 5 18 18 19 18 7 4 1 1

This page is intentionally left blank.

2. Complete Trees (12 points)

https://www.hackerrank.com/contests/cst370-s19-hw1/challenges/problem-2-complete-trees

For this problem, we will implement a Complete tree using nodes. For example, in C++, the node and class definitions would look like this:

```
struct Node {
    int data;
    Node* left;
    Node* right;
    Node* parent;
};
class CompleteTree {
    Node* root;
    ...
};
```

You will need to implement the following operations on complete trees:

`fromArray` which creates nodes for a complete tree represented by the given array.

`toArray` which returns the array representation of the complete tree.

`isBST` which returns true if the complete tree is a Binary Search Tree.

`preOrder` which prints a pre-order traversal of the tree.

`postOrder` which prints a post-order traversal of the tree.

`numNodesInLookup` which prints the number of nodes visited when looking up the given value.

For example, in C++, the function headers would be the following:

```
class CompleteTree {
    Node* root;
  public:
    CompleteTree() {
        // ...
    }
    void fromArray(vector<int> list) {
        // ...
    }
    vector<int> toArray() {
        // ...
    }
    bool isBST() {
```

```
        // ...
    }
    void preOrder() {
        // ...
    }
    void postOrder() {
        // ...
    }
    int numNodesInLookup(int value) {
        // ...
    }
};
```

Input

- The first line contains an integer $n, 0 \leq n \leq 100$, denoting the number of nodes in the tree.
- The next line will consist of n integers that make up the values to be stored in the tree.
- The third line contains an integer $m, 0 \leq m \leq 10$, denoting the number of commands in the following lines.
- The next m lines have one of the following commands ("toArray", "isBST", ""preOrder", "postOrder", "numNodesInLookup") per line.

Constraints

In this problem, a binary search tree must have unique values. You can assume the value given to "numNodesInLookup" is in the tree.

`lookupOperations` should try to look at as few nodes as possible. This means that if the tree is a Binary Search Tree, it should use binary search. If the tree is not a binary tree, use level-order traversal.

Output

- The "fromArray" command will print nothing.
- "toArray" will print out the array representation of the complete tree.
- "isBST" will print out either "true" or "false".
- "preOrder" and "postOrder" will print out the order they visit each node respectively.
- "numNodesInLookup" will print out an integer.

Sample Input 1

```
10
1 2 3 4 5 6 7 8 9 10
4
isBST
preOrder
toArray
numNodesInLookup 4
```

Sample Output 1

```
false
1 2 4 8 9 5 10 3 6 7
1 2 3 4 5 6 7 8 9 10
4
```

Sample Input 2

```
12
100 53 172 24 64 150 200 12 33 60 98 130
3
isBST
postOrder
numNodesInLookup 98
```

Sample Output 2

```
true
12 33 24 60 98 64 53 130 150 200 172 100
4
```

This page is intentionally left blank.

3. Beating Bogosort (12 points)

https://www.hackerrank.com/cst370-s19-hw1

James has a student roster list where each row looks like "Student Name, Graduation Year". He believes the only way to sort this list is to randomize the rows and manually check if it's sorted. Prove him wrong by implementing 3 faster ways to sort James's student roster.

The student roster must be sorted according to graduation year (soonest first), and for students with the same graduation year, alphabetically by first name (and if there are any students with the same first name and graduation year, ties should be broken by last name). **Rather than just printing the final sorted list, you'll print the partially sorted list after each pass.**

Input

- The first line contains an integer $n, 0 \leq n \leq 10$, denoting the number of students you will need to sort.
- The next line n lines will each consist of student data wtih space-separated first name, last name, and graduation year, in that order.
- The input will terminate with a blank line.

Constraints

You can assume n is a non-negative integer. You can further assume the list contains no students with the same first name, last name, and graduation year.

Output

You will be printing the partially sorted list as many times as necessary based on the sorting algorithm you are implementing. Look at the hacker rank challenge for sample input and output for the different algorithms.

Algorithms

- Insertion Sort (4 points)
  https://www.hackerrank.com/contests/cst370-s19-hw1/challenges/problem-3-a-beating-bogosort-insertion-sort
- Merge Sort (4 points)
  https://www.hackerrank.com/contests/cst370-s19-hw1/challenges/problem-3-b-beating-bogosort-merge-sort
- Quick Sort (4 points)
  https://www.hackerrank.com/contests/cst370-s19-hw1/challenges/problem-3-c-beating-bogosort-quick-sort

This page is intentionally left blank.

4. Finding Students (8 points)

https://www.hackerrank.com/contests/cst370-s19-hw1/challenges/problem-4-finding-students

Now that James has a sorted student roster, he wants your help to write a program to quickly find students.

Input

- The first line contains an integer $n, 0 \leq n \leq 100$, denoting the number of students in his list.
- The next line n lines will each consist of student data wtih space-separated first name, last name, and graduation year, in that order. These lines will be sorted alphabetically by first name.
- The next line will consist of an integer, m, the number of students you have to look for.
- The next m lines will each consist of student's first name to look up.
- The input will terminate with a blank line.

Constraints

You can assume n is a non-negative integer. You can further assume the list contains no students with the same first name. When finding the middle element, if the array has even length, choose the left of the two middle elements.

Output

- For each query, output a line in the format "i: x" where i is the index of the students and x is the number of students you looked at to find it.
- If the student is not in the roster, give -1 as i to indicate it was not found.

Sample Input 1	**Sample Output 1**
7 Emma Watson 2016 Laura Laham 2018 Mason Park 2017 Nathan Spaun 2019 Percy Wesley 2016 Princess Peach 2017 Simon Marlow 2018 3 Emma James Nathan	0: 3 -1: 3 3: 1

5. Akcija (8 points)

https://csumb.kattis.com/problems/akcija

The problem statement for problems on the kattis platform are found at the link above.

Hints

Think about the data structure and algorithms we've covered so far. Are there any that can help you solve this problem?

This page is intentionally left blank.