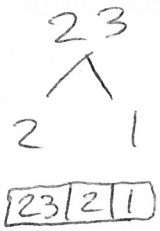
CS61B Week 9: Backtracking Search, Minimax, Heaps

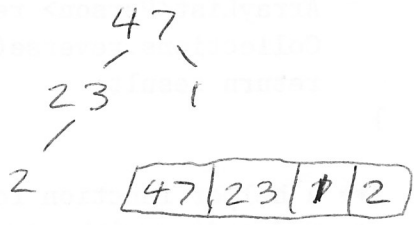
1. Complete the following table of running times. Assume each structure has n elemenents, and each range query looks for k elements.

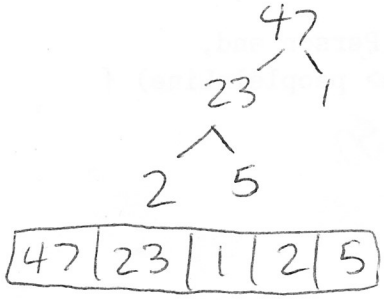
Function	Unsorted Array	Sorted Array	Binary Search Tree	Hash Table	Heap
find	$\Theta(n)$	$\Theta(log(n))$	$\Theta(log(n))$	$\Theta(1)$	$\Theta(n)$
add	$\Theta(1)$	$\Theta(n)$	$\Theta(log(n))$	$\Theta(1)$	$\Theta(log(n))$
build structure	$\Theta(n)$	$\Theta(n*log(n))$	$\Theta(n*log(n))$	$\Theta(n)$	$\Theta(n)$
range query	$\Theta(n)$	$\Theta(k+log(n))$	$\Theta(k+log(n))$	$\Theta(n)$	$\Theta(n)$
find largest	$\Theta(n)$	$\Theta(1)$	$\Theta(log(n))$	$\Theta(n)$	$\Theta(1)$
remove largest	$\Theta(n)$	$\Theta(1)$	$\Theta(log(n))$	$\Theta(n)$	$\Theta(log(n))$

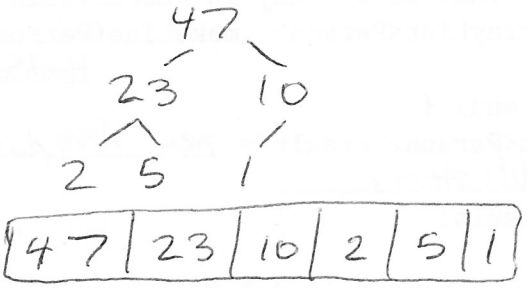
2. Assume we have a binary max-heap (largest value on top) data structue called Heap that stores integers and has properly implemented insert and removeLargest methods. Draw the heap and its corresponding array representation after each of the statements below.

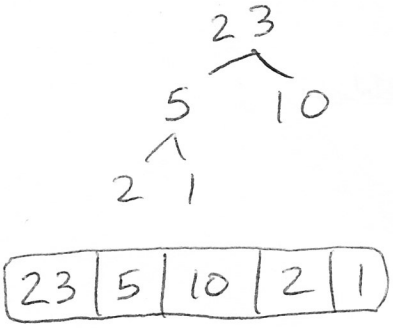
```
Heap h = new Heap(2, 23, 1); \\ Creates a heap by inserting 2, then 23, then 1
h.insert(47);
h.insert(5);
h.insert(10);
h.removeLargest();
h.removeLargest();
```

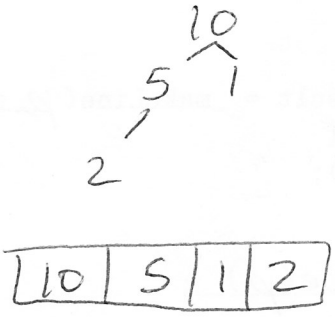












3. Complete the following program to implement a backtracking search algorithm.

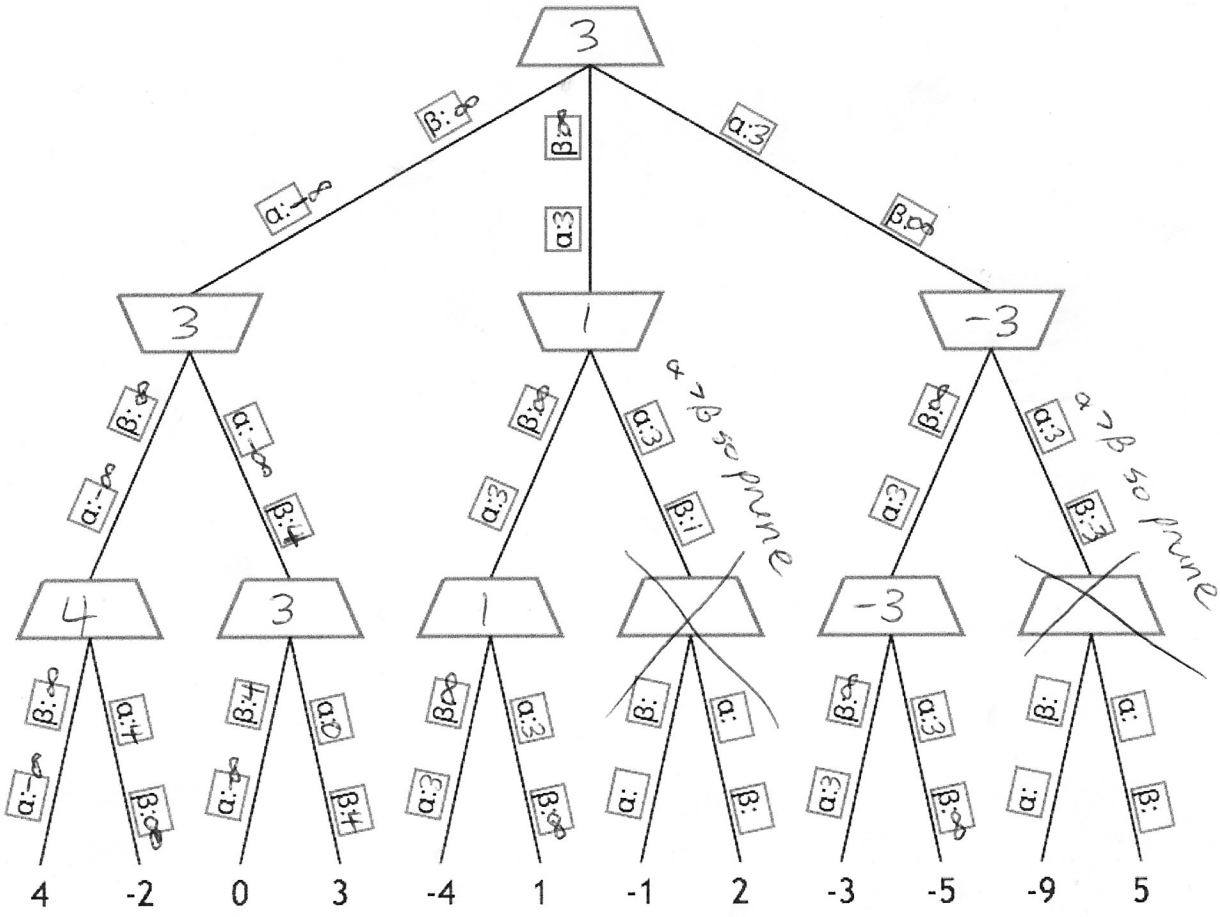
```
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

/** A person. All friends are mutual - if b is in a's friend array,
 *  then a is in b's friend array. */
class Person {
    Person[] friends;

    /** Returns an list of people such that for every position k, the person in the
     *  kth position is friends with the people in the (k-1)th and (k+1)th positions.
     *  Assumes there are a finite number of people in the world. */
    public static ArrayList<Person> makeLine(Person start, Person end) {
        HashSet<Person> peopleInLine = new HashSet<Person>();
        peopleInLine.add(start);
        ArrayList<Person> result = _makeLine(start, end, peopleInLine);
        Collections.reverse(result);
        return result;
    }

    /** A helper function for makeLine. Makes a line starting with person START
     *  and ending with person END. Does not use any people that are in PEOPLEINLINE.
     *  Assumes that START is already in PEOPLEINLINE. */
    private static ArrayList<Person> _makeLine(Person start, Person end,
                                               HashSet<Person> peopleInLine) {
        if (start == end) {
            ArrayList<Person> result = new ArrayList<Person>();
            result.add(start);
            return result;
        }
        for (Person p : start.friends) {
            if (peopleInLine.contains(p)) {
                continue;
            }
            peopleInLine.add(p);
            ArrayList<Person> result = _makeLine(p, end, peopleInLine);
            if (result != null) {
                result.add(start);
                return result;
            }
            peopleInLine.remove(p);
        }
        return null;
    }
}
```

4. Fill out the following minimax tree using alpha-beta pruning. Draw an X through branches that are pruned.



5. Fill in the following functions according to their comments. Both use the provided Node class.

```
public class Node {
    int value;
    Node left;
    Node right;
}
```

(a)
```
/** Removes the node with value VAL from the binary search tree T. */
public static void removeBinarySearchTreeNode(Node t, int val) {
```

See RemoveBSTNode.java