Mihir Jham
CS251
Written HW #2
03/24/2013

CS25100 Homework 2: Spring 2014

Due Monday, March 24th, 2014, by 11:59 PM. Only PDF files will be accepted. No extensions or late submissions since solutions will be released immediately after the deadline. Submit from lore or data using the command: turnin -c cs251 -p hw2 yourFirstName_yourLastName where yourFirstName_yourLastName is a folder (i.e., subdirectory) that contains a single file named hw.pdf; the command should be typed while in the parent directory of that subdirectory.

1 Algorithm Properties (6 pts)

Match up each worst-case quantity on the left with the best matching order-of-growth term on the right. You may use a letter more than once.

__B__	Height of a binary heap with N keys.	A. 1.
__B__	Height of a BST with N keys.	B. log N
__E__	Number of comparisons to sort N equal keys using our standard version of quicksort (with 2-way partitioning).	C. N .
__C__	Number of comparisons to sort N equal keys using 3-way quicksort.	D. N log N .
__C__	Time to iterate over the keys in a BST using inorder traversal.	E. N^2.
__B__	Number of array access to insert a key into a BST of size N .	F. 2^N .

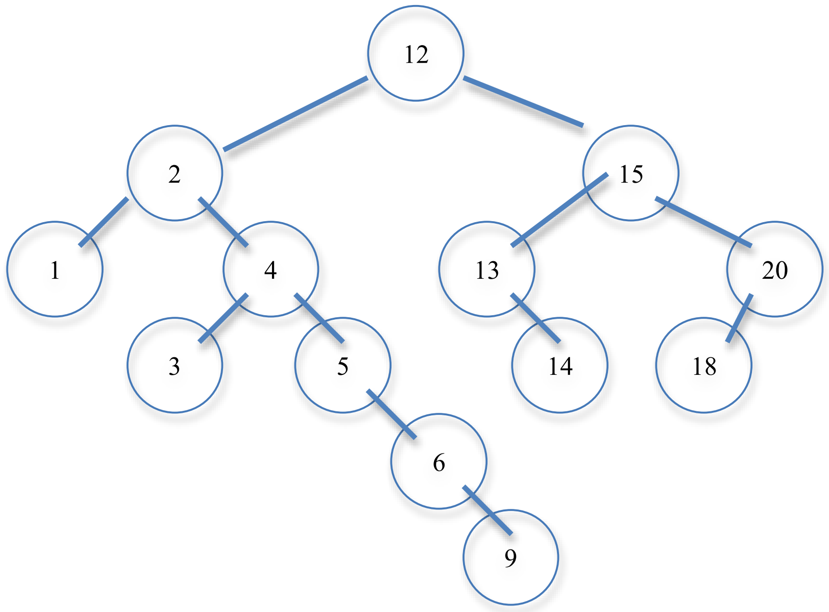
2 Tracing Algorithm Operation (4 x 5 = 20 pts)

- Trace the operation of selection sort on the following input, using the notation from the slides (e.g., elementary sorts slide 15):
  22, 15, 36, 44, 10, 3, 9, 13, 29, 25, 11
- Give an example of a worst case input with n elements for insertion sort, and describe how insertion sort runs in n^2 time on such input.
- Trace the keys 'H' 'O' 'M' 'E' 'W' 'O' 'R' 'K' 'N' in a bottom-up merge sort, using the notation from the slides (e.g., merge sort slide 9).
- Draw the binary search tree that results from the insertion of:
  12 2 4 5 15 1 20 3 6 18 13 9 14

					a[]							
		0	1	2	3	4	5	6	7	8	9	10
i	min	22	15	36	44	10	3	9	13	29	25	11
0	5	22	15	36	44	10	3	9	13	29	25	11
1	6	3	15	36	44	10	22	9	13	29	25	11
2	4	3	9	36	44	10	22	15	13	29	25	11
3	10	3	9	10	44	36	22	15	13	29	25	11
4	7	3	9	10	11	36	22	15	13	29	25	44
5	6	3	9	10	11	13	22	15	36	29	25	44
6	6	3	9	10	11	13	15	22	36	29	25	44
7	9	3	9	10	11	13	15	22	36	29	25	44
8	8	3	9	10	11	13	15	22	25	29	36	44
9	9	3	9	10	11	13	15	22	25	29	36	44
10	10	3	9	10	11	13	15	22	25	29	36	44
		3	9	10	11	13	15	22	25	29	36	44

An input that is reverse-sorted is a worst-case input for insertion sort. Every iteration of the inner loop will scan and shift the entire sorted subsection of the array before inserting the next element. It makes $\sim\frac{1}{2}n^2$ comparisons and $\sim\frac{1}{2}n^2$ exchanges, thus giving a quadratic running time of $O(n^2)$.

	a[]								
	0	1	2	3	4	5	6	7	8
sz=1	H	O	M	E	W	O	R	K	N
merge(a,0,0,1)	H	O	M	E	W	O	R	K	N
merge(a,2,2,3)	H	O	E	M	W	O	R	K	N
merge(a,4,4,5)	H	O	E	M	O	W	R	K	N
merge(a,6,6,7)	H	O	E	M	O	W	K	R	N
sz=2									
merge(a,0,1,3)	E	H	M	O	O	W	K	R	N
merge(a,4,5,7)	E	H	M	O	K	O	R	W	N
sz=4									
merge(a,0,3,7)	E	H	K	M	O	O	R	W	N
sz=8									
merge(a,0,7,8)	E	H	K	M	N	O	O	R	W



3 Binary Search Trees (4 pts)

```
public Key mystery(Key key) {
   Node best = mystery(root, key, null);
   if (best == null) return null;
   return best.key;
}

private Node mystery(Node x, Key key, Node best) {
   if (x == null) return best;
   int cmp = key.compareTo(x.key);
   if      (cmp < 0) return mystery(x.left, key, x);
   else if (cmp > 0) return mystery(x.right, key, best);
   else              return x;
}
```

Consider the binary search tree method above. What does mystery(key) return?

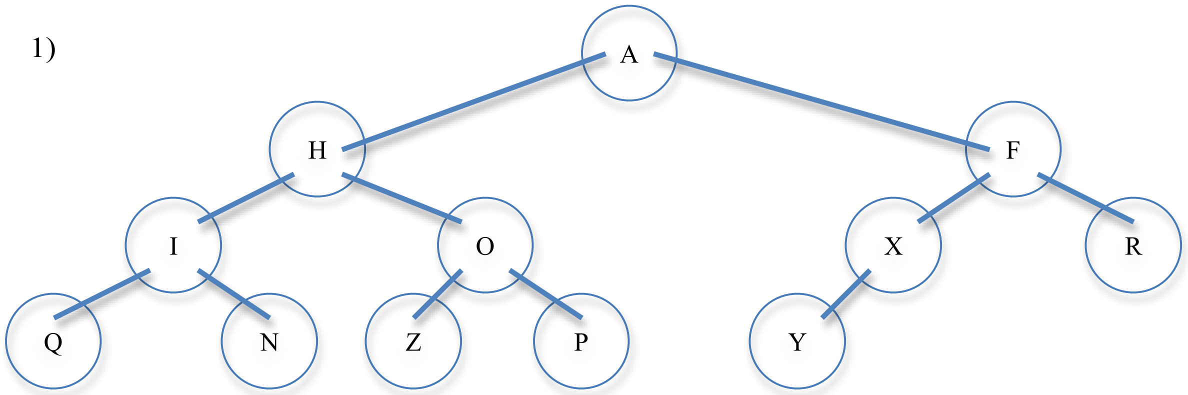
This function returns the ceiling of the key input, which is the smallest key in the tree that is greater than or equal to the key input.

4 Heap Operations (8 pts)

Consider a MinPQ implemented as a heap in which following keys have been inserted in this order: Q, F, Y, I, O, R, A, N, H, Z, P, X.

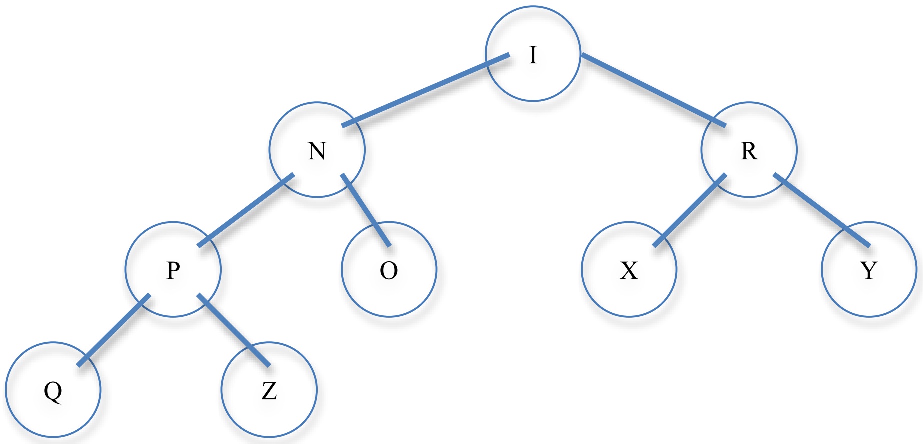
1. Draw the heap and show its contents in the corresponding array representation.
2. Now assume that the RemoveMin operation has been invoked 3 times. Draw the resulting heap and its array representation.

1)



0	1	2	3	4	5	6	7	8	9	10	11	12
-	A	H	F	I	O	X	R	Q	N	Z	P	Y

2)



0	1	2	3	4	5	6	7	8	9
-	I	N	R	P	O	X	Y	Q	Z

5 Median (12 pts)

Design a data type that supports (a) insert in logarithmic time, (b) find the median in constant time, and (c) remove the median in logarithmic time.
Give pseudocode or Java code for the three operations (a), (b), and (c) above. Discuss the complexities of the three methods. Your answer will be graded on correctness, efficiency, and clarity.

The data structure that I will use consists of 2 heaps, a MaxHeap and a MinHeap. The MaxHeap has the maximum value of that heap at its root and the MinHeap has the minimum value of that heap at its root. Also, each heap will have a size variable that consists of the number of elements in the heap. The will both be array-based heap implementations.

Both the heaps have the functions that are defined in the book we are using, which I will write code for below after discussing how the median can be found in constant time.

For the first 2 elements that are entered, add the smaller element to the MaxHeap and the larger element to the MinHeap.

After the first 2 elements have been entered, check if the next element entered is smaller than the the element at MaxHeap's root. If it is, add the element to the MaxHeap. Otherwise, add it to the MinHeap.

Check if the number of elements in either one of the heaps is greater than the other by more than 1. If it is, remove the root element from the heap containing more elements and add it to the other one. This step balances the heaps and after this step, the heaps will either be balanced or one of them will contain 1 more item.

At any time, to calculate the median, check if both the heaps are of equal size. If they are, then the median = (root of MaxHeap + root of MinHeap)/2
Otherwise,
median = root of the heap with more elements

To delete the median, you delete the root of the heap with more elements.

The following functions have been taken from the Algorithms, 4th edition book.
The insert and delete root functions are the same for both MaxHeap and MinHeap.

```
public void insert(Key x)
{
        pq[++N] = x;
        swim(N);
}

public Key delRoot()
{
        Key root = pq[1];
        exch(1, N--);
        sink(1);
        pq[N+1] = null;
}
```

The swim and sink functions differ very slightly. Here are the functions for MaxHeap.

```
private void swim(int k)
{
        while (k > 1 && less(k/2, k))
        {
                exch(k, k/2);
                k = k/2;
        }
}

private void sink(int k)
{
        while (2*k <= N)
        {
                int j = 2*k;
                if (j < N && less(j, j+1)) j++;
                if (!less(k, j)) break;
                exch(k, j);
                k = j;
        }
}
```

Here are there functions for the MinHeap.

```
private void swim(int k)
{
        while (k > 1 && greater(k/2, k))
        {
                exch(k, k/2);
                k = k/2;
        }
}

private void sink(int k)
{
        while (2*k <= N)
        {
                int j = 2*k;
                if (j < N && greater(j, j+1)) j++;
                if (!greater(k, j)) break;
                exch(k, j);
                k = j;
        }
}
```

The insert and delete functions are both of logarithmic time as the sink and swim functions step up and down the levels respectively. The finding of the median is done in constant time as it just a matter of looking at the respective roots of the MaxHeap and MinHeap.