COS 226 **Algorithms and Data Structures** **Fall 2009**

Midterm Solutions

1. **8 sorting algorithms.** 0 3 5 2 6 4 8 7 9 1

2. **Analysis of algorithms.**

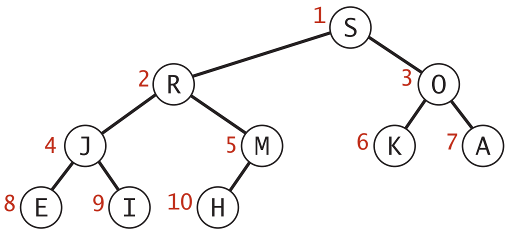
   (a) I and II only. III is a true statement, but tilde notation also suppresses lower order terms so it is not a reason why tilde notation is more precise.

   (b) $\sim 10^{-9}N^3$ seconds

3. **Binary heaps.**

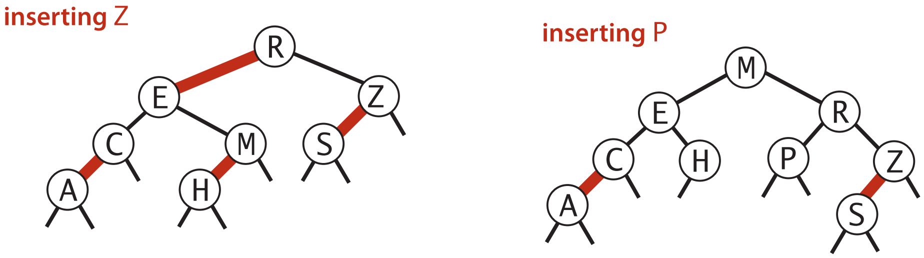
   (a) K M R

   (b)



4. **Red-black trees.**

   (a)



   (b) 2 left rotation (one while inserting Z, one while inserting P), 1 right rotations (while inserting P), 2 color flips (while inserting P).

5. **Hashing.**

I only.

- I results from inserting the keys in the order B D F A C E G.
- II cannot result. The first key inserted will end up in the table entry corresponding to its hash value. But no key has this property.
- III cannot result. Both A and F end up in the table entry corresponding to their hash values, so we can assume they were inserted first and second. So, the third key inserted will also end up in the table entry corresponding to its hash value. But no keys (beside A and F) have this property.

6. **Data structures.**

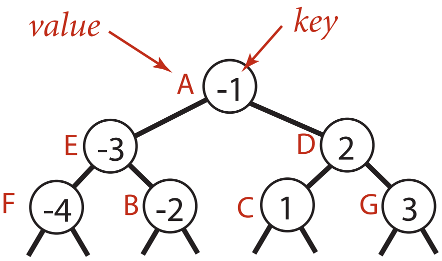
(a) N^4. Deleting the ith element takes linear time in the worst case (and average case). In this example, the size of the array decreases from N^2 to 0.

(b) A randomized queue supports deleting a random element in constant (amortized) time.

(c) N^2. Starting from an empty randomized queue (as implemented in Assignment 2), any sequence of $2N^2$ operations takes time proportional to $2N^2$ in the worst case.

7. **Generalized queue.**

(a) Create a BST where the keys are integers and the values are the generic items such that the ith value in the queue is associated with the ith largest key in the BST. *That is, an inorder traversal of the BST yields the items in the queue in order.*



(b) To implement `get(i)`: return the value corresponding to the `ith` largest key. The `ith` largest key is `select(i)`.

(c) To implement `addFirst()` and `addLast()`, we will maintain two instance variables `lo` and `hi`, which we initialize to `0`.

- To implement `addFirst(item)`, associate the new item with the key `lo - 1` and decrement `lo`. Thus, the new item has the smallest key.
- To implement `addLast(item)`, associate the new item with the key `hi + 1` and increment `hi`. Thus, the new item has the largest key.

(d) To implement `remove(i)`, delete the `ith` largest key and its associated value from the symbol table. The `ith` largest key is `select(i)`.

If we use a red-black tree for the BST, all operations take logarithmic time in the worst case. Here's a complete Java implementation.

```
public class GeneralizedQueue<Item> {
    private RedBlackBST<Long, Item> st = new RedBlackBST<Long, Item>();
    private long lo = 0, hi = 0;
    public Item get(int i)          { return st.get(st.select(i)); }
    public void delete(int i)       { st.delete(st.select(i));     }
    public void addFront(Item item) { st.put(--lo, item);          }
    public void addLast(Item item)  { st.put(++hi, item);          }
}
```