Final Examination Sample Solution (Partial)

1. Consider the following methods implementing a queue with two stacks:

```java
stack S1, S2;

boolean isEmpty()
    return S1.isEmpty() && S2.isEmpty(); //&& is logical-and

tvoid enQ(int x)
    S1.push(x);

tint deQ()
    if (isEmpty()) return EmptyQueueError;
    if S2.isEmpty()
        while (not S1.isEmpty())
            S2.push(S1.pop());
    return S2.pop();
```

(a) What is the total time used by `enQ(1)`, `enQ(2)`, ..., `enQ(n)`, `deQ()`, `deQ()`, ..., `deQ()` (n enqueues followed by n dequeues)?
(b) What is the total time of n enqueues, followed by n pairs of alternating dequeues and enqueues? As in `enQ(1)`, `enQ(2)`, ..., `enQ(n)`, `deQ()`, `enQ(n+1)`, `deQ()` `enQ(n+2)`, ..., `deQ()`, `enQ(2n)`.
(c) Pick a time bound t(n) and argue that “Any sequence of O(n) enqueues and dequeues, in any order, takes total time t(n)”.

sol’n: All the bounds are O(n). The idea is to imagine the route of any value: it gets pushed onto S1, sometime later it migrates to S2, to get popped after that. Thus n values cause 3n operations overall.

2. Professor Pequalsnp claims to have an algorithm that will take a series of n integers and construct a BST containing them. It does not perform one-by-one insertion, but it is comparison based. The amazing thing is that this algorithm is claimed to operate in time $O(n\sqrt{\log n})$.

Do you believe Prof. Pequalsnp? Explain your reason.

sol’n: If one could build a BST in time $O(n\sqrt{\log n})$, then by performing an inorder traversal of the tree (which is $O(n)$), one would have an $O(n\sqrt{\log n})$ comparison based sorting routine. Due to the $\Omega(n \log n)$ lower bound, this is impossible. Hence, the claim of Prof. Pequalsnp cannot be true.

3. Into an initially empty red-black tree
(a) insert the values 22, 30, 25, 10, 8, 6, 12, 15, 24, 20, 18.
(b) then delete 30

4. Describe how to find the $\sqrt{n}$ smallest items, listed in increasing order, in $O(n)$ time. The input list is unsorted (but the output should be sorted.) (Hint: $\sqrt{n}\log\sqrt{n}$ and $\sqrt{n}\log n$ are both $O(n)$.)

sol’n: Most everyone got this one. There are two obvious ways to do this.
Method 1: Find the item $x$ of rank $\sqrt{n}$ (time: $O(n)$ expected). Scan the list, finding everything $\leq x$ (time: $O(n)$). Sort those items (time: $O(\sqrt{n}\log\sqrt{n}) = O(n)$).
Method 2: Build a min-heap containing all the items (time: $O(n)$). Perform an extractMin operation on the heap $\sqrt{n}$ times (time: $O(\sqrt{n}\log n) = O(n)$).

5. Consider an array containing the values 10, 14, 6, 7, 4, 11, 16, 5, 8, 12, 2, 1, 13, 3, 15, 9 in locations 1-16. Illustrate the build-heap method converting this array into a max-heap.

6. Into an initially empty binomial heap (which, as in the text, is a MIN-heap):
   (a) insert the values 3, 15, 1, 9, 7, 14, 12, 2, 6, 5, 4, 11, 10
   (b) then remove the min.

7. Write a recursive routine which, given a BST T and integer k, will traverse T and print the contents of all nodes at height k, from left to right.

sol’n: Most people interpreted this as asking for nodes at depth k. That was on a number of previous exams, but we do tend to change the questions somewhat. What we need to do is to incorporate a routine calculating the height into one printing out the keys of nodes at the given height:

```c
int nodesAtHeight(node p, int k)
    if (p==null)
        return -1
    int curHeight = 1 + max(nodesAtHeight(p.left(),k), nodesAtHeight(p.right(),k))
    if (curHeight==k)
        print p.key()
    return curHeight
```

Note that although this is a postorder traversal, all the nodes at some fixed height will still be visited from left to right.

8. Suppose you have an array $S$ of size $n$, where each element of the array represents a vote for class president. A vote is represented by the student ID of the candidate. We do not know the number of candidates. A person wins if they receive a majority of the votes. Give an efficient procedure to determine if there is a winner. How fast is your method?

sol’n: Any vote which is an absolute majority will have to appear in the median position - that is, it will have to have rank $\left\lceil \frac{n}{2} \right\rceil$, based on positions 1 through $n$. So, the algorithm is simple

```c
int medPos = n/2
int median = S.randomizedSelect(medPos)
```
scan through S, determine number of votes for median
if number of votes is at least n/2
    then "median has majority"
else "no winner"

The time for this is $O(n)$ expected time. Most people used a sort (or built a BST), which would be $O(n \log n)$ time.