CS2110 Fall 2017

Recitation 9: Analysis of Algorithms

(1) Consider the following (inefficient, but correct) implementation of a function to find the minimum element of a LinkedList of integers:

```
public static int findMinimum(LinkedList<Integer> l) {
    int tempMin = l.get(0);
    for (int i = 1; i < l.size(); i++){
            int current = l.get(i);
            if (current < tempMin) tempMin = current;
    }
    return tempMin;
}
```

Note: The get method above traverses the nodes of the LinkedList, starting at index 0, until it finds the element at index i, then returns the element.

(a) Give the runtime complexity (asymptotic complexity) of running this function on a list with n elements.

Running time is $O(n^2)$. There are order n iterations through the loop and in each iteration, operation `l.get(i)` takes time $O(n)$.

(b) Rewrite the function to run in O(n) time.

```
Integer tempMin = Integer.MAX_VALUE
for(Integer i : l){
    if(i < tempMin) tempMin = i;
}
return tempMin.intValue();
```

(2) Give the runtime complexity of the following operations on data structures, and a brief explanation:

1. Inserting a new element into an ArrayList at an arbitrary index.

Takes time $O(n)$ since you need to copy the full array.

2. Inserting a new element as the first element of the LinkedList.

Takes time $O(1)$ since you just need to change the the pointers

3. Removal of an arbitrary element of an ArrayList.

Takes time $O(n)$ since you need to copy all the subsequent elements to their new location

4. Accessing an arbitrary index of an ArrayList.

Takes time $O(1)$ since arrays have constant time lookup

5. Counting the number of nodes in a Tree.

Takes time $O(n)$ to traverse the tree since you have to visit all the nodes

6. Computing the depth of a balanced tree.

Takes time $O(\log n)$, since a balanced tree with branching factor b has depth $\log_b n$)

7. Searching for an element in a tree.

Takes time $O(n)$ since you have to check all the elements of the tree.

8. Searching for an element in a binary search tree.

Takes time $O(d)$ in a tree of depth d since you only have to check one branch. ($O(\log n)$ in a balanced binary tree.)

9. Reversing the order of words in a string. (Java is fun becomes fun is Java)

Depends on the implementation. A naïve implementation akes time $O(n^2)$ since there can be $O(n)$ word and each time you rewrite the sting (move one word) takes time $O(n)$

10. [Extra credit] Calculating whether a number is prime or not.

Depends on the algorithm. The best known deterministic algorithm takes time $O(\log^{6+\varepsilon} p$.)

11. [Hard] Computing the median of numbers in a linked list.

The best worst-case running time is $O(n \log n)$, which can be achieved by sorting the list using merge sort and then walking down the list to the middle element. Average-case time $O(n)$ can be achieved with a version of QuickSort that only recurses on one side, but it has worst-case time $O(n^2)$.)

(3) Write down the asymptotic complexity of each of the following functions:

1. $f(n) = \sqrt{n^3 + n^2 + 10}$ $\quad O(n^{3/2})$
2. $f(n) = n!$ $\quad O(n^n)$
3. $f(n) = 2^n + n^2$ $\quad O(2^n)$
4. $f(n) = n \log n + (\log n)^2$ $\quad O(n \log n)$
5. $f(n) = 100000n + \log n$ $\quad O(n)$
6. $f(n) = n + 0.0000000012^n$ $\quad O(2^n)$
7. $f(n) = \sum_{k=1}^{K} \log n^k$ $\quad O(K^2 \log n)$

8. $f(n) = \log\log n$ $O(\log\log n)$

9. $f(n) = (n+10)^4$ $O(n^4)$

10. $f(n) = \sqrt{n} + 10\log n$ $O(\sqrt{n})$

(4) Give the asymptotic complexity of the following recursive function from lecture notes:

```
/** = a**n. Precondition: n >= 0 */
static int power(int a, int n) {
    if (n == 0) return 1;
    if (n%2 == 0) return power(a*a, n/2);
    return a * power(a, n-1);
}
```

Each iteration takes constant time, and there are at most a logarithmic number of iterations (since at least half of the operations divide n by 2) so this function takes time $O(\log n)$.