CSM 61B Spring 2019

Final Review

Mentoring 13: April 29, 2019

1 Running Out of Time

1.1 Give a tight runtime bound for `mystery` as a function of N, the length of the input array.

```
public static int[] mystery(int[] array) {
    boolean done = false;
    while (!done) {
        done = true;
        for (int i = 1; i < array.length; i++) {
            if (array[i-1] > array[i]) {
                done = false;
                int tmp = array[i-1];
                array[i-1] = array[i]
                array[i] = tmp;
            }
        }
    }
    return array;
}
```

1.2 Give a tight asymptotic bound for `convoluted` as a function of N, the length of the input arrays `a` and `b`. If possible, give a $\Theta(\cdot)$ bound for the overall runtime. Otherwise, provide a $\Theta(\cdot)$ bound for both the best case and worst case runtime.

```
int[] convoluted(int[] a, int[] b) {
    assert a.length == b.length;
    int[] result = new int[a.length];
    for (int i = 0; i < result.length; i += 1) {
        for (int j = 0; j <= i; j += 1) {
            result[i] += a[j] * b[i];
        }
    }
    return result;
}
```

1.3 Give a tight asymptotic bound for `debugBinarySearch` as a function of N, the length of the input array, `a`. If possible, give a $\Theta(\cdot)$ bound for the overall runtime. Otherwise, provide a $\Theta(\cdot)$ bound for both the best case and worst case runtime.

```
boolean debugBinarySearch(int[] a, int target) {
    int start = 0;
    int end = a.length - 1;

    while (start <= end) {
        int mid = ((end - start) / 2) + start;
        if (a[mid] == target) {
            return true;
        } else if (a[mid] < target) {
            start = mid + 1;
        } else {
            end = mid - 1;
        }

        System.out.print("Searching: [ ");
        for (int i = start; i <= end; i++) {
            System.out.print(a[i] + " ");
        }
        System.out.println("]");
    }
    return false;
}
```

1.4 Give a tight asymptotic bound for `mystery`.

```
int mystery(int N) {
    if (N == 0) {
        return 0;
    }
    return mystery(N/3) + mystery(N/3) + mystery(N/3);
}
```

2 Out of Sorts

2.1 Each column below gives the contents of a list at some step during sorting. Match each column with its corresponding algorithm.

· Merge sort · Quicksort · Heap sort · LSD radix sort · MSD radix sort

For quicksort, choose the topmost element as the pivot. Use the recursive (top-down) implementation of merge sort.

	Start	A	B	C	D	E	Sorted
1	4873	1876	1874	1626	9573	2212	1626
2	1874	1874	1626	1874	7121	8917	1874
3	8917	2212	1876	1876	9132	7121	1876
4	1626	1626	1897	4873	6973	1626	1897
5	4982	3492	2212	4982	4982	9132	2212
6	9132	1897	3492	8917	8917	6152	3492
7	9573	4873	4873	9132	6152	4873	4873
8	1876	9573	4982	9573	1876	9573	4982
9	6973	6973	6973	1897	1626	6973	6152
10	1897	9132	6152	3492	1897	1874	6973
11	9587	9587	7121	6973	1874	1876	7121
12	3492	4982	8917	9587	3492	9877	8917
13	9877	9877	9132	2212	4873	4982	9132
14	2212	8917	9573	6152	2212	9587	9573
15	6152	6152	9587	7121	9587	3492	9587
16	7121	7121	9877	9877	9877	1897	9877

3 T,F,G,V,E

3.1 State if the following statements are True or False, and justify. For all graphs, assume that edge weights are positive and distinct, unless otherwise stated.

(a) Adding some positive constant k to every edge weight does not change the shortest path tree from vertex S.

(b) Doubling every edge weight does not change the shortest path tree.

(c) Adding some positive constant k to every edge weight does not change the minimum spanning tree.

(d) Doubling every edge weight does not change the minimum spanning tree.

(e) Let $(S, V - S)$ be a specific cut of the graph. If an edge e is not the lightest edge across this cut, it cannot be a part of any MST.

(f) If an edge e is the lightest edge connected to vertex S, it must be a part of the shortest path tree from vertex S.

4 Roleplaying Game

4.1 You are the king of a large kingdom! In order to manage your kingdom, you have appointed lords to rule towns within your kingdom. Every lord can govern over his town and any town that he is connected to by road. Your job as king is to figure out the optimal way to allocate lords and build roads.

Formally, consider a graph G with vertices V and edges E. Each vertex v represents a town. It has an associated cost c, the cost of installing a lord in the town. Each edge e represents a potential road. It has an edge weight w, the cost of building that road. Devise an algorithm that can efficiently compute which towns to install lords in and which roads to build, such that every town in the kingdom is governed (either has a lord in it or is connected by some number of roads to a town with a lord in it).

5 Largest Perimeter Triangle

5.1 Given an array A of positive lengths, return the largest perimeter of a triangle with non-zero area, formed from 3 of these lengths. Recall the Triangle Inequality, which states that for any triangle, the sum of the lengths of any two sides must be greater than or equal to the length of the remaining side ($a + b > c$). If it is impossible to form any triangle of non-zero area, return 0.

For example, A = [2, 1, 2] returns 5. A = [1, 2, 1] returns 0. A = [3, 2, 3, 4] returns 10.

What is the runtime of your solution?

```
public int largestPerimeter(int[] A) {




}
```

Note: this problem was adapted from LeetCode (https://leetcode.com/problems/largest-perimeter-triangle/).