

CS5234 : Combinatorial and Graph Algorithms
 Homework Set #0 (For Information Only)
 [Problems on Pre-requisite Materials]

All undergraduates from SoC must have done CS3230 before taking this course. For graduate students who have not taken CS3230 Analysis of Algorithms, or similar algorithms course, I have added in some problems on these pre-requisite materials. I will be assuming that students in CS5234 will be able to solve these problems easily. If these problems pose some difficulties or if you have never solved these types of problems before, it is strongly advisable to either drop the course (and take CS3230 instead) or be prepared to work doubly hard to make up the background materials.

[HW0 is for your information only - I will not grade HW0. If you turn them in, I'll look at them.]

Pre-Requisite Problems -- For HW0 & HW1 only, I have added in some problems on assumed pre-requisite materials (these are marked with [PRn Pre-Req: ...]) that you should be able to solve easily if you the right pre-requisite.

PR1 [Pre-Req: Mathematical Analysis] Consider recurrence relations of the following form, where $f(k)$ is a non-decreasing function of integer k :

$$T(1) = 1;$$

$$T(n) = 2T(n/2) + f(n)$$

Such recurrences occur when analyzing the running time of a divide-and-conquer algorithm in which the original problem is split into two disjoint equal parts, and where the cost to combine subproblem solutions is $f(n)$. For each of the recurrences below of this form, give a tight upper bound on $T(n)$, in O notation. Prove that your answer is correct.

(a) $f(n) = 1$, (b) $f(n) = 2n$, (c) $f(n) = n^2$.

[Note: n^2 means n -squared, or n "to the power of" 2.]

PR2 [PreReq: Basic Algorithm and its Analysis] The AAS autoclub gives its members free maps and trip planners that include the distances between each major city along the journey. Suppose you are given a journey consisting of a list of n cities, and the distances between consecutive pairs. Then you could easily make a grid chart that gives you the distance between any two cities in $O(1)$ time, but that would take $O(n^2)$ time to preprocess the data. Devise an $O(1)$ algorithm that gives you the distance between any two cities on the route, but uses only $O(n)$ time to preprocess the data.

PR3 [PreReq: Search Tree] Let T be an AVL tree (or any other balanced binary search tree), and let x be a key. Give an $O(\log n)$ algorithm for finding the smallest key y in T such that $y > x$. Note that x may or may not be in T . Explain why your algorithm has $O(\log n)$ running time.

PR4 [PreReq: Basic Data Structure and Algorithms] We define *parenthetically correct* strings as strings in which the parenthetic symbols $()$, $[]$, and $\{\}$ are balanced and properly nested. Assuming a string is represented as an array of characters, the following quadratic-time algorithm determines if a string is parenthetically correct:

Scan through the array A , removing all non-parenthetic symbols. Repeatedly scan through the result, removing all adjacent pairs of symbols $()$, $[]$, and $\{\}$, until there are no more symbols left or no more pairs can be removed. A is parenthetically correct if and only if all of the symbols have been removed.

Consider the following examples:

```
input:  ab(de{fg}h[(i)jk]l)m
pass 1: ({}[()])
pass 2: ([])
pass 3: ()
pass 4: nothing left → string is parenthetically correct
```

```
input:  ab(de{fg}h[(i)jk]l)m
pass 1: ({}[{}])
pass 2: ([{}])
pass 3: ([{}]) → nothing more can be removed,
                so string is not parenthetically correct
```

The algorithm is quadratic-time because there are at most $n/2+1$ scans of the array -- one to remove the non-parenthetic symbols and at most $n/2$ to remove pairs of symbols, since if any symbols are removed in a pass, then at least two are. Each scan can be done in linear time by copying all of the symbols that aren't removed to a new array, since deleting them "in place" requires a lot of extra time to shift elements of the array to the left to fill the gap.

This is not the most efficient solution for the problem, however. Show how to solve the problem in linear time, and explain why your algorithm has the running time it does.

Hint: Use a stack as an auxiliary data structure.

PR5 [PreReq: Graph and Representation] Would you use the adjacency list or the adjacency matrix representation in each of the following cases? Justify your choice.

- The graph has 10,000 vertices and 20,000 edges, and it is important to use as little space as possible.
- The graph has 10,000 vertices and 20,000,000 edges, and it is important to use as little space as possible.
- You need to answer as fast as possible the query `areAdjacent`, no matter how much space you use.

PR6 [Pre-Req: Graph Algorithms] Would you prefer DFS or BFS (or both equally) for the following tasks? (Assume the graph is undirected and connected.) Justify your answer.

- Determining if the graph is acyclic
- Finding the connected components of the graph (if it is not connected)
- Finding a path to a vertex known to be near the starting vertex
- Finding the farthest vertex from a given vertex, where the farthest vertex is one with the longest minimum-length path (in other words, to find the farthest vertex from u , take the shortest paths from u to every other vertex v in the graph and report the v for which the shortest path is the longest). Here length of a path is defined as the number of edges in the path.