CS2040S Data Structures and Algorithms **AY2022/2023 Semester 2**

CS2040S: Midterms Extra Practice

By: Audrey Felicio Anwar

Problem 1

Solve the following recurrence relations, give the tightest bound possible.

1. $T(n) = 2T(n-1) + \Theta(1)$
2. $T(n) = \sum_{i=1}^{n-1} T(i) + \Theta(1)$
3. $T(n) = 3T(\frac{n}{2}) + \Theta(n^2)$
4. $T(n) = T(\frac{n}{2}) + T(\sqrt{n}) + \Theta(n)$

Solution:

1. $T(n) = \Theta(2^n)$
2. Note that $T(n-1) = \sum_{i=1}^{n-2} T(i) + \Theta(1)$, hence $T(n) = T(n-1) + \sum_{i=1}^{n-2} T(i) + \Theta(1) = 2T(n-1) + \Theta(1) + \Theta(1)$. Hence $T(n) = \Theta(2^n)$
3. Draw out the recursion tree, at the first level cn^2, second level $\frac{3}{4}cn^2$, third level $(\frac{3}{4})^2cn^2$, and so on. In total, $T(n) = \Theta(n^2)$
4. Notice that $T(n) \geq cn$. Now, to upper bound $T(n)$, notice that $T(\sqrt{n}) \leq T(\frac{n}{4})$, hence $T(n) \leq T(\frac{n}{2}) + T(\frac{n}{4}) + \Theta(n)$. Draw out the recursion tree, in total $T(n) = \Theta(n)$. Hence $\Theta(n)$ is the tightest possible bound.

Problem 2

Audrey is not impressed with the sorting algorithms taught in lectures and decided to design his own sorting algorithm. He would like to sort an array A of N integers in the following way.

1. Run `QuickSort` algorithm on A, but Audrey modified the `QuickSort` algorithm to stop when the current size of the array during recursion is $\leq K$.
2. After the modified `QuickSort` has finished, Audrey runs `InsertionSort` on the modified array A.

Does this algorithm always sorts A correctly? What is the running time of this algorithm?

Solution:

Yes it always sorts A correctly, because in the end `InsertionSort` is run, regardless of the modified array A after `QuickSort`. In step 1, `QuickSort` runs $\frac{N}{K}$ recursion levels deep, hence $O(N \log \frac{N}{K})$. In step 2, notice that due to `QuickSort`, each element in the array is now at most K distances away from it's correct location. Hence, `InsertionSort` runs in $O(NK)$. In total, the running time is $O(NK + N \log \frac{N}{K})$.

Problem 3

Audrey is busy with his midterms, however he still has K homeworks that needs to be finished this week. Due to the lack of time, Audrey decided to outsource his homeworks to his friends. Audrey has N friends. Each friend f_i has an initial annoyance level of a_i and every time Audrey ask to do an homework, the friend's annoyance level increases by d_i. Each friend is represented by the tuple (a_i, d_i). Audrey does not want to annoy his friends too much. Design an efficient algorithm that will assign all of Audrey's homeworks to his friends, such that the maximum level of annoyance among his friends is as small as possible.

As an example, when $K = 4$ and `friends` $= [(1, 2), (2, 3), (3, 4), (4, 5)]$, a possible desired assignment would be give 2 homeworks to f_1, 1 homework to f_2, and 1 homework to f_3, resulting in a maximum annoyance level of 7. In this test case, we cannot have maximum annoyance less than 7, no matter what the assignments are.

Solution:

There are at least two ways to solve this efficiently. The first one is to store `friends` in an `AVL` tree and rank them by their $a_i + d_i$ value. For each homework, we just assign it to the `MinElement` of the current `AVL` tree, and then update it's $a_i + d_i$ value to be $a_i + d_i + d_i$, since we annoy him/her once. Finally, return the assignments of homework. The time complexity for this is $O(K \log N)$.

Another way to solve this is to Binary Search The Answer (BSTA). We can do this because if you can assign K homeworks with maximum annoyance level of X, then definitely you can assign with maximum annoyance level of anything $> X$. Hence, we can just binary search for this minimum value of X. To quickly check whether a given X works, we just assign as much homework as possible to each friend, without making his/her annoyance level $> X$ and output `true` if we can assign all the homeworks. Finally, return the assignments of homework, once we found the minimum value of X. The time complexity for this is $O\left(N \log \left(\max_{1 \le i \le N} a_i + K d_i\right)\right)$.

Problem 4

You are given an array A of N integers. You want to construct an array B of N integers, such that

$$B[i] = \begin{cases} x & \text{where } x \text{ is the smallest index} > i \text{ such that } A[x] \ge A[i] \\ -1 & \text{if all the elements after } A[i] \text{ are smaller than } A[i] \end{cases}$$

Design an efficient algorithm to construct B. As an example, if `A` $= [1, 3, 2, 2, 1]$, then `B` $= [1, -1, 3, -1, -1]$.

Solution:

We can use a stack to construct B, we store indexes in the stack. The idea is to use the stack to store all the possible greater than indexes for the current $A[i]$. Iterate through the array from right to left (highest index to lowest index). For each element $A[i]$, whenever the stack is empty, we just add -1 to B. Otherwise, we check the top of the stack, let the top of the stack be x. If $A[x] \ge A[i]$, then we have found the required index, add x to B, then if $A[x] = A[i]$, pop the top. Otherwise if $A[x] < A[i]$, we pop it from the stack, and

repeat checking the top until we get $A[x] \geq A[i]$ or until the stack is empty. Finally, we add i to the stack. Lastly, we reverse B and output it as our answer. Time complexity is $O(N)$.

Problem 5

You are given an array A of N distinct integers. You are also given an array B consisting $N-1$ characters `<` or `>`. You want to find a permutation of A such that it satisfies the constraints given in array B. For example, if $\texttt{A} = [100, 2, 9, 19]$ and $\texttt{B} = [>, <, >]$, then a desired result can be $\texttt{A} = [19, 9, 100, 2]$, since $19 > 9, 9 < 100$, and $100 > 2$. Design an efficient algorithm to solve this problem.

Solution:

Sort the array A in increasing order. Initiate a `low` pointer pointing to the first element of A and a `high` pointer pointing to the last element of A. Now iterate through B. If $B[i]$ is `>`, then push $A[\texttt{high}]$ to the output array and decrement `high` by one. Similarly, if $B[i]$ is `<`, then push $A[\texttt{low}]$ to the output array and increment `low` by one. This works because at each step, you are selecting the highest possible for each `>` and the lowest possible for each `<`. Time complexity is $O(N \log N)$ using `QuickSort`, can run in $O(N)$ if we use `RadixSort`.

Problem 6

You are given N distinct points on a plane. The manhattan distance between two points $(x_1, y_1), (x_2, y_2)$ is defined as $|x_1 - x_2| + |y_1 - y_2|$. Design an efficient algorithm to compute the sum of manhattan distances between all pairs of points. As an example, when $\texttt{points} = [(1, 2), (2, 3), (3, 5)]$, the answer is $2 + 3 + 5 = 10$.

Solution:

Notice that we can separately count the sum of x coordinates and y coordinates. Given the x coordinates $x_1 \leq x_2 \leq ... \leq x_N$ in an increasing sorted order, the total manhattan distance is

$$\sum_{i=1}^{N} \sum_{j=i+1}^{N} (x_j - x_i) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} x_j - \sum_{i=1}^{N} (N-i)x_i = \sum_{i=2}^{N} (i-1)x_i - \sum_{i=1}^{N} (N-i)x_i$$

Notice that we can compute both sums in $O(N)$ time. Hence, we just need to sort the points in ascending x coordinate and do the computation of those sums to get the total manhattan distance over the x coordinates. Similarly, we can calculate the sum of the y coordinates. Time complexity is $O(N \log N)$.

Problem 7

Audrey has been kidnapped and is forced to do some tasks in order to survive. For each day, a stream of tasks will be given to Audrey and has to be executed in the order of arrival. The tasks that arrive are of two types.

1. Audrey will be asked to write down a name X.
2. Audrey will be asked to convert all written names X to Y.

At the end of the day, after executing all the tasks, Audrey is asked to scream out loud the final names of each name that he has been asked to write, in order of their arrival.

As an example, if the tasks are `(1, "Alice"), (1, "Bob"), (2, "Alice", "Charlie"), (1, "Alice"), (2, "Bob", "Charlie"), (2, "Charlie", "Malory")`, then at the end of the day, Audrey will have to scream `["Malory", "Malory", "Alice"]`, since the arrival order is `"Alice", "Bob", "Alice"` and the first `"Alice"` and `"Bob"` has been changed to `"Malory"`.

Audrey will be given punishment if he cannot figure out what names to scream quickly. Design an efficient algorithm to help Audrey survive.

Solution:

Just tell Audrey to do nothing and wait until all the tasks has been received :D. Create a hash table `T` that maps a string X to it's final conversion `T[X]`. Initialize `T[X] = X` for all string X that are seen in all the tasks. The key idea is to do the tasks backwards, from the most recent to the oldest one because for each name we only care about the final conversion of it.

Iterate through the tasks from the most recent to the oldest. For each task we do one of the following two procedures, depending on the task type.

```
1: procedure TypeOne(X, Output)
2:     Output.add(T[X])
3: end procedure
4:
5: procedure TypeTwo(X, Y)
6:     T[X] = T[Y]
7: end procedure
```

Since `T` indicates the final conversion, we simply just add `T[X]` for each task of type 1. Now the magic happens in the task of type 2. Notice that `T[Y]` indicates what will be the final conversion of name `Y` and now we want to convert `X` into whatever the final conversion of `Y` is. Hence we just need to do `T[X] = T[Y]`. Finally after we have done all the tasks, we reverse the output array and give it to Audrey. Time complexity is $O(N)$, where N is the number of tasks given to Audrey.

All the best for midterms on Monday!