Lab # 3 – **Data Structures and Algorithms – taught by Sahar Waqar**
Pre-Lecture Exercise for Lecture 3 Spring 2021
TAs: Fatima Muzaffar, Maham Fatima, Usama Ashfaq

Lab goals:

- Announcement
- Reading and understanding Pre-lecture exercises
- Solving recurrence questions
- Project Announcement and Idea Selection

ANNOUNCEMENT:

- **Lab Evaluation and Running Scheme:** It will be announcement during lab.

SECTION I:

Pre-lecture exercises will not be collected for credit. However, you will get more out of each lecture if you do them, and they will be referenced during lecture. We recommend **writing out** your answers to pre-lecture exercises before class. Pre-lecture exercises usually should not take you more than 30 minutes.

In this pre-lecture exercise, you'll explore *recurrence relations.* A recurrence relation defines a function $T(n)$ recursively. For example, for $n = 2^i$ which is a power of 2, we might define:

$$T(n) = \begin{cases} 2 \cdot T(n/2) + n & n > 1 \\ T(n) = 1 & n = 1 \end{cases}.$$

Why is a function like this relevant to us? It turns out that it is a good way to write down the running time of divide-and-conquer algorithms. For example, we saw with MergeSort that we broke up one problem of size n into two problems of size $n/2$; and then it took us an extra $O(n)$ operations to merge the solutions together. Let's say for concreteness that it takes $11n$ operations, where the 11 is arbitrary. So if $T(n)$ is the number of operations it takes to run MergeSort on a list of size n, we could write something like

$$T(n) = 2T(n/2) + 11 \cdot n,$$

which is similar to the function $T(n)$ above. The way to interpret the base case $T(1) = 1$ is that it takes one operation to sort a list of length 1, since we just return it; it's already sorted. We saw in class that $T(n) = O(n \log(n))$. That is, the running time of MergeSort is $O(n \log(n))$.

Question 1. The first problem in the pre-lecture exercise is to understand the above text, and make sure you understand the connection between the function $T^{\sim}(n)$ defined above to the running time of MergeSort.

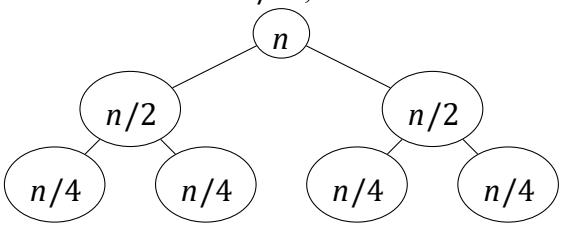
For the rest of the pre-lecture exercise, you'll see if you can generalize the argument that we saw in class to different recurrence relations. For reference, **on the next page we've shown two different ways of showing that** $T(n) = O(n \log(n))$**.** (We went with $T(n)$ instead of $T^{\sim}(n)$ because it's a little bit cleaner to write down without carrying that factor of "11" around everywhere, and the point still gets across.) If you found the tree method confusing, you might like the second method which just uses algebra.

Question 2. Now try to generalize the argument for $T(n)$ to the following two recurrence relations. That is, for each of $T_1(n)$ and $T_2(n)$ below, figure out an expression of the form $T_j(n) = O(\_\_\_)$. **Assume that** n **is a power of** 2 **if it helps.**

- $T_1(n) = \begin{cases} T_1(n/2) + n & n > 1 \\ T_1(n) = 1 & n = 1 \end{cases}$.
- $T_2(n) = \begin{cases} 4 \cdot T_2(n/2) + n & n > 1 \\ T_2(n) = 1 & n = 1 \end{cases}$.

Here are two different ways to understand the running time of $T(n)$**, when** n **is a power of** 2:

SOLUTION 1. Just like we did in class, imagine a tree with $\log(n)+1$ levels. The top node is labeled "n", its two children are labeled "$n/2$", and so on.



… etc.

Consider $T(n) = T(n/2) + T(n/2) + n$. In the context of the tree above, that means that $T(n) = n +$ (stuff contributed by things in the tree lower than the root). That is,

$T(n)$ = (label on the root) + (stuff contributed by things lower than the root).

We can repeat this logic recursively to figure out what that second term is, all the way down to the bottom of the tree, where we have $T(1) = 1$. We conclude that each node in the tree that's labeled k contributes k to the sum.[1] Now we add everything up:

[1] Notice that this is a special consequence of the fact that the term we are adding in the definition of $T(n)$ is exactly n; if it were, say $11 \cdot n$, the contribution of a node labeled k would be $11k$.

- The zeroth layer contributes n, since there is one problem of size n, which contributes n.
- The first layer also contributes n, since there are two problems of size $n/2$, each of which contributes $n/2$.
- ...
- The t'th layer also contributes n, since there are 2^t problems of size $n/2^t$, each of which contributes $n/2^t$.
- ...
- The $\log(n)$'th layer (which is the bottom one) also contributes n, since there are n problems of size 1, each of which (by the base case $T(1) = 1$) contributes 1.

Altogether there are $\log(n)+1$ layers, each contributing n, so we conclude that, when n is a power of 2,

$$T(n) = n(\log(n)+1).$$

Notice that this is an exact answer when n is a power of 2, we don't even need a $O(\cdot)$.

SOLUTION 2. We can do the exact same calculation without the tree, by repeatedly applying our formula.

$$\begin{aligned} T(n) &= 2T(n/2)+ n \\ &= 2(2T(n/4)+ n/2)+ n \\ &= 4T(n/4)+2n \\ &= 4(2T(n/8)+ n/4)+2n = \\ &8T(n/8)+3n \end{aligned}$$

and at this point we can spot the pattern: for all $j \le \log(n)$,

$$T(n) = 2^jT(n/2^j)+ jn.$$

In order to formally prove that this is true, we should use a proof by induction; that's called the *substitution method* and we'll talk about it soon. But for now you can convince yourself that this is true.

Once we have this, we can just plug in $j = \log(n)$, and get

$$T(n) = 2^{\log(n)}T(n/2^{\log(n)})+ n\log(n) = n \cdot T(1)+ n\log(n) = n(\log(n)+1),$$ just as

before.

SECTION II:

Select a project title for your data structure project. Project can cover variety of domains. It can be web-based or desktop application. There is no restriction on programming language. Your main job will be to use algorithms for solving particular problems. We will encourage students to look around their surroundings and see what are the issues culminating in our environment? Find those problems, make them your project idea and develop your screens. For understanding and ideas, you can discuss with your TAs.

SECTION III:

Please solve following questions from your book "Introduction to Algorithms by Cormen".

- ➔ **Substitution Method:** 4.3.1, 4.3.2, 4.3.3, 4.3.7, 4.3.8, 4.3.9
- ➔ **Recurrence Tree Method:** 4.4.2, 4.4.3, 4.4.4, 4.4.5, 4.4.6, 4.4.7
- ➔ **Master Theorem:** 4.5.1, 4.5.2, 4.5.3, 4.5.4

Note:

Submit these questions as pdf document. Work on it using Word first and LaTeX later as second submission.

For LaTeX, we will schedule a tutorial session on Monday next week. If you are unable to work on it, you can submit the Word document temporarily but will have to convert it to LaTeX later and re-submit or re-evaluate your lab3.