Data Structure	Lab 4
Spring 2021	Wed, April 21

Lab Goals:

- Project Idea Approval (This lab)
- Project Screens Finalization (Next week)
- Pre-lecture Exercises Complete Understanding (Next Week)
- Python notebooks Complete and Thorough Understanding (Next Week)
- Homework Questions
- Recurrence Questions
- LaTeX Submission for Lab3

Project Idea Approval:

Get your project ideas approved in this lab.

Project Screens Finalization:

In this week, you will work with your group partner and decide screens for your project. You can use any prototyping tool like JustInMind or Pencil or any other. Prototype screens must include all buttons, headings, text areas, images and alignment or positioning of components etc. Next week you will discuss your screens and project flow with your TAs.

Pre-lecture Exercises

In this pre-lecture exercise, you'll get acquainted with the *k*-select problem which we'll see in lecture 4.

Consider the problem *k*-select: When given an array *A* of *n* distinct numbers, find the *k*-th smallest one. For example, if the input was

$$A = [6, 4, 8, 9, 5, 2, 1],$$

then the output to 3-select should be "$A[1] = 4$," since 4 is the 3rd smallest item in this array.

1. Give an $O(n \log(n))$-time algorithm for *k*-select (for any fixed *k*).

2. Give an $O(n)$-time algorithm for 1-select. (That is, k-select for $k = 1$).

3. Give an $O(n)$-time algorithm for 2-select.

4. Question to ponder before Lecture 4: can you come up with an $O(n)$-time algorithm for k-select for general k? (It's okay if not – as we'll see in lecture 4, this is tricky!)

Python Notebooks:
Solve and thoroughly understand Python notebook provided in this lab.

Homework Question:
Matrix multiplication. Suppose that we have $n \times n$ matrices X and Y and we'd like to multiply them. (For details see **Strassen** method in CLRS)

(a) (2 pt.) What is the running time of the standard algorithm (that computes the inner product of rows of X and columns of Y)? You can assume that simple arithmetic operations, like multiplication, take constant time ($O(1)$).

[We are expecting: A detailed analysis of the runtime of your algorithm including the Big-O time in terms of n.]

(b) (3 pt.) Now let's divide up the problem into smaller chunks like this, where the eight n/2 x n/2 sub-matrices (A, B, C, D, E, F, G, H) are each quarters of the original matrices, X and Y :

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

We now have a divide and conquer strategy! Find the recurrence relation of this strategy and the runtime of this algorithm.

[**We are expecting:** A detailed analysis of the runtime of your algorithm including the Big-O time in terms of n.]

(c) (3 pt.) Can we do better? It turns out we can by calculating only 7 of the subproblems:

$$P_1 = A(F - H) \qquad P_5 = (A + D)(E + H)$$

$$P_2 = (A + B)H \qquad P_6 = (B - D)(G + H)$$

$$P_3 = (C + D)E \qquad P_7 = (A - C)(E + F)$$

$$P_4 = D(G - E)$$

And we can solve XY by

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

We now have a more efficient divide and conquer strategy! What is the recurrence relation of this strategy and what is the runtime of this algorithm?

[We are expecting: A detailed analysis of the runtime of your algorithm including the Big-O time in terms of n.]

Historical note: Sophisticated variants of this strategy have resulted in (asymptotically) better-and-better algorithms for matrix multiplication over the years. For a humorous take on the most recent improvement, see here (just for fun): https://www.smbc-comics.com/comic/mathematicians.

(d) (2 pt.) Your friend tried to solve part (c) of the problem, and came to the following conclusion:

Claim: The algorithm runs in time $T(n) = O(n^2 \log(n))$.

Proof: At the top level, we have 7 operations adding/subtracting $n/2 \times n/2$ matrices, which takes $O(n^2)$ time. At each subsequent level of the recursion, we increase the number of subproblems by a constant factor (7), and the subproblems only become smaller. Therefore, the running time per level can only increase by a constant factor. By induction, since for the top level it is $O(n^2)$, it will be $O(n^2)$ for all levels. There are $O(\log(n))$ levels, so in total the running time is $T(n) = O(n^2 \log(n))$.

What is the error in this reasoning?

[We are expecting: A clear and concise description, in plain English, of the error with this proof.]

Recurrence Questions:

1. (4 pt.) There exist different ways to solve the recurrence relation $T(n) = 2 \cdot T(n/2) + n$ with $T(1) = 1$. From lectures, we have seen that $T(n)$ is exactly $n(1 + \log(n))$ when n is a power of 2. In this exercise, you'll analyze a few more recurrences for a few variants.

   (a) What is the exact solution to $T(n) = 3 \cdot T\left(\frac{n}{3}\right) + n$ with $T(1) = 3$, when n is a power of 3?

   (b) What is the exact solution to $T(n) = 3 \cdot T\left(\frac{n}{3}\right) + 3n$ with $T(1) = 1$, when n is a power of 3?

[We are expecting: Your answer — no justification required. Notice that we want the exact answer, so don't give an *O*() statement.]

2. (4 pt.) Use any of the methods we've seen in class so far to give big-Oh solutions to the following recurrence relations. You may treat fractions like *n*/2 as either *floor(n/*2) or *ceiling(n/*2*)*, whichever you prefer.

   (a) $T(n) = 3T\left(\frac{n}{9}\right) + \sqrt{n}$ for $n \geq 9$, and $T(n) = 1$ for $n < 9$.

   (b) $T(n) = T(n-4) + n$ for $n \geq 4$, and $T(n) = 1$ for $n < 4$. (You may assume $n \bmod 4 = 0$.)

   (c) $T(n) = 6T\left(\frac{n}{4}\right) + n^2$ for $n \geq 4$, and $T(n) = 1$ for $n < 4$.

   (d) $T(n) = 5T\left(\frac{n}{2}\right) + n^2$ for $n \geq 2$, and $T(n) = 1$ for $n < 2$.

   [**We are expecting:** For each item, the best answer you can give of the form $T(n) = O(\ )$ and a justification. (That is, all of these satisfy $T(n) = O(2^n)$, but you can do better). You do not need to give a formal proof, but your justification should be convincing to the grader. You may use the Master Theorem if it applies.]

3. (2 pt.) Consider the following algorithm, which takes as input an array *A*:

```
def printStuff(A):
    n = len(A)
    if n <= 4: return
    for i in range(n):
        print(A[i])
    printStuff(A[:n/3]) # recurse on first n/3 elements of A

    printStuff(A[2*n/3:]) # recurse on last n/3 elements of A

    return
```

   What is the asymptotic running time of printStuff?

   [**We are expecting:** The best answer you can give of the form "The running time of printStuff is *O*( )" and a short explanation.]

LaTeX Submission:
Please submit lab#3 in LaTeX, no later than 30th April, 2021.