CS3000: Algorithms & Data — Summer I '21 — Drew van der Poel

Homework 2
Due Friday, May 28 at 11:59pm via Gradescope

Name: Gabriel Peter
Collaborators:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- This assignment is due Friday, May 28 at 11:59pm via Gradescope. No late assignments will be accepted. Make sure to submit something before the deadline.
- Solutions must be typeset. If you need to draw any diagrams, you may draw them by hand as long as they are embedded in the PDF. I recommend using the source file for this assignment to get started.
- I encourage you to work with your classmates on the homework problems. *If you do collaborate, you must write all solutions by yourself, in your own words.* Do not submit anything you cannot explain. Please list all your collaborators in your solution for each problem by filling in the `yourcollaborators` command.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class is strictly forbidden.

Problem 1. *Improve the MBTA (18 points)*

You have been commissioned to design a new bus system that will run along Huntington Avenue. The bus system must provide service to n stops on the eastbound route. Commuters may begin their trip at any stop i and end at any other step $j > i$. Here are some naïve ways to design the system:

1. You can have a bus run from the western-most point to the eastern-most point making all n stops. The system would be cheap because it only requires $n-1$ route segments for the entire system. However, a person traveling from stop $i = 1$ to stop $j = n$ has to wait while the bus makes $n-1$ stops.
2. You can have a special express bus from i to j for every stop i to every other stop $j > i$. No person will ever have to make more than one stop. However, this system requires $\Theta(n^2)$ route segments and will be expensive.

Using divide-and-conquer, we will find a compromise solution that uses only $\Theta(n \log n)$ route segments, but with the property that a user can get from any stop i to any stop $j > i$ making at most two stops in total. That is, it should be possible to get from any i to any $j > i$ either by taking a direct route $i \to j$ or by taking two routes $i \to m$ and $m \to j$.

(a) **[2 points]** For the base cases $n = 1, 2$, design a system using at most 1 route segment.

Solution:

n = 1: o
n = 2: $o \longleftrightarrow o$

(b) **[8 points]** For $n > 2$ we will use divide-and-conquer. Assume that we already put in place routes connecting the first $n/2$ stops and routes connecting the last $n/2$ stops so that if i and j both belong to the same half, we can get from i to j in at most 2 segments. Show how to add $O(n)$ additional route segments so that if i is in the first half and j is in the second half we can get from i to j making only two stops.

Solution:
DRAWING ATTACHED

Given that both halfs are now full connected, we find the midpoint as the pivot, where all other stops have a non-stop segment that connects them to this midpoint. This since each route will have a single segment, the total added segments will be $O(1n) = O(n)$

(c) **[4 points]** Using part (b), write (in pseudocode) a divide-and-conquer algorithm that takes as input the number of stops n and outputs the list of all the route segments used by your bus system.

Solution:

```
# Takes number of stops: n and returns the
# list of all the route segments needed.
def get_stops(n):
```

```
        stops = range(n)
        midpoint = int(len(stops)/2)
        return get_stops_helper(stops[:midpoint])
                + get_stops_helper(stops[midpoint:])


def get_stops_helper(stops):
        if len(stops) == 2: # base case from 1a.
                return (stops[0], stops[1])
        else:
                new_segments = []
                for i in range(len(stops):
                        # from 1b (O(n) stops) are added
                        new_segments.append(stops[0], stops[i])
                midpoint = int(len(stops) / 2)
                # only the lower half of the stops needs additional connectors
                return new_segments + get_stops_helper(stops[:midpoint)
                # recursive add to list
```

(d) **[4 points]** Write the recurrence for the number of route segments your solution uses and solve it. You may use any method for solving the recurrence that we have discussed in class.

Solution:

Each call results in a successive loop, which adds n segments to the line, then proceeds to recursively call itselfs in both halves (n/2):
Thus, $T(n) = 2T(n/2) + n$ AND $T(1,2) = 1$ and can now be analyzed with the master theorem:
$a = 2, b = 2, d = 1, \frac{2}{2^1} = 1 \rightarrow \Theta(n \log n)$

Problem 2. *Stop the Arsonist! (26 points)*

Boston FD receives a letter from an arsonist. In the letter they find a list of n words and a note that the building the arsonist plans to destroy is encoded within the list. The arsonist, who thinks he is clever, also provides a hint:

"The name of the building that I plan to destroy is the longest common prefix of all words in the attached list!"

Clearly, the arsonist is not very clever because (a) it is a terrible encoding and (b) he didn't think Boston FD had you to help them!

We will let w_i be the i-th word in the list ($1 \leq i \leq n$) and $l(w_i)$ be the length of w_i ($1 <= l(w_i) <= 25000\ \forall i$). We let L be the $\max(l(w_i))$ over all i.

Your task is to determine the longest prefix of all words in the arsonist's list, before he destroys his target.

Here is an example:

The arsonist's list reads: "northeasternkhouryccis", "northeasternkhouryccisbuilding", "northeasternkhouryyyyyyyy", "northeasternkhourybuilding", "northeasternkhourynortheastern"

The longest common prefix is "northeasternkhoury".

(a) **[10 points]** You want to show off your algorithmic prowess, and you realize that you can use divide-and-conquer to solve this problem. Design a divide-and-conquer algorithm that takes as input a list of n words of maximum length L and outputs the longest common prefix of the n words in time $O(Ln)$. You should pseudocode for your algorithm, accompany this pseudocode with a written description, and provide justification for why it runs in $O(Ln)$ time. Note that you should treat L as a variable, not as a constant.

(Hint: First develop an $O(L)$ algorithm for finding the longest common prefix of two words a and b, then use this algorithm as part of your divide-and-conquer approach.)

Solution:

```
# gets longest prefix of two words
def longest(a, b):
    i = 0
    while i < min(len(a), len(b)):
        if a[i] == b[i]:
            i +=1
        else:
            break
    return a[:i]

# gets longest prefix of list of words
def prefix_finder(words):
    if len(words) % 2 == 1:
        words.append(words[-1])
    if len(words) == 2:
```

```
        return longest(words[0], words[1])
    else:
        n = len(words)/2
        return max(prefix_finder(words[:n]), prefix_finder(words[n:]))
```

(b) **[10 points]** In order to better your familiarity with divide-and-conquer algorithms, we have created a hackerrank challenge (www.hackerrank.com/cs3000-summer1-2021-programming-assignment-2). Please implement your divide-and-conquer strategy and submit it to the challenge. Your grade for this part will depend on (a) how many test cases your implementation passes and (b) actually implementing a divide-and-conquer strategy (we will check!).

In order to allow for efficient grading, please write your hackerrank account below.

Solution:
peter_g

(c) **[3 points]** Your friend, who has no training in algorithms, proposes the following method for solving the problem when there are at least two words in the list ($n \geq 2$): Pick an arbitrary word and find the longest common prefix between it and each of the other $n-1$ words. Then look at this new list of $n-1$ common prefixes, and whichever is the shortest one, is the solution.

What is the Big Oh runtime of your friend's approach (state in terms of n and L)? Provide a short justification of your claim. To receive full marks, your runtime should be reasonably tight (i.e. relevant to the algorithm).

Solution:

Runtime is $O(L(n-1)) + O(L(n-1)) = O(Ln) + O(Ln) = O(Ln)$
The first $O(L(n-1))$ is from the initial comparison loop amongst all the words which initializes the list of prefixes.
The second is from finding the shortest prefix in the list of prefixes with would be size n -1 and up to L characters in length each at the worst case. :)

(d) **[3 points]** Is your friend's algorithm correct? If yes, provide a proof of its correctness (**Hint:** I would use a combination of direction observations and contradiction, definitely avoiding induction!). If not, provide an instance of the problem where their approach does not work (**Hint:** I would think about relatively short lists of relatively short words).

Solution:

This solution is correct:

Say we receive this list of words:
northeasternkhouryccis
northeasternkhouryccisbuilding
northeasternkhouryyyyyyyy
northeasternkhourybuilding

northeasternkhourynortheastern
northeastern

It is a guarantee that by checking an arbitrary word with the rest of the word bank that longest prefix will be found and added to the list. Then we can be certain that this would be the shortest word in the list is the true ”longest common prefix” because the word that produced the shortest prefix would not have any of the longer prefixes by function design (which asserts that it will find the longest prefix common between the two words) otherwise it would have found that besides the shorter one. Therefore the shortest is the common prefix shared across all words as the longer words are only common amongst a subset of the words in the provided bank.

Problem 3. *A fault-tolerant OR-gate (14 points)*

Assume you are given an infinite supply of two-input, one-output gates, most of which are OR gates and some of which are AND gates. You may assume that all inputs are from the set $\{0, 1\}$. Recall that if at least one of the inputs to an OR gate is a 1, then the output is a 1, otherwise the output is a 0. If both inputs to an AND gate are 1s, then the output is a 1, otherwise the output is a 0.

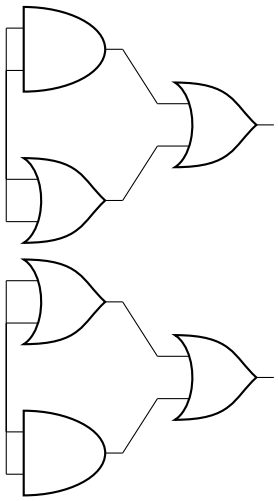
The OR and AND gates have been mixed together and you can't tell them apart. For a given integer $k \geq 0$, you would like to construct a two-input, one-output combinational "k-OR" circuit from your supply of two-input, one output gates such that the following property holds: If at most k of the gates are AND gates then the circuit correctly implements OR.

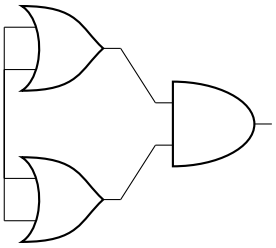
Note that the inputs to a gate can either be the two "circuit inputs" (which may be the inputs for multiple gates), or can be directed from the outputs of two other unique gates. There should be exactly one gate whose output is not directed to another gate, the output of which is returned by the circuit (is the circuit's output). A gate's output may be directed to more than one gate.

For a given integer $k \geq 0$, you would like to design a k-OR circuit that uses as few gates as possible. Assume for simplicity that k is a power of two. Note that AND and OR gates only differ in their outputs when one input is a 1 and the other is a 0. It suffices to only consider the case where the "circuit inputs" are a 1 and a 0.

(a) **[4 points]** Design a 1-OR circuit with the smallest number of gates. That is, your circuit should take two inputs (the "circuit inputs"), and always return the OR of the inputs, as long as at most one of the gates constituting the circuit is an AND gate, while the remainder are OR gates. Argue the correctness of your circuit.

Solution:: In all alterations of a 1-OR regardless of the placement of the single AND gate, the circuit should operate as an OR – Assume inputs are similar to attached drawing from 2a, gate rendering wasn't working too well.





(b) **[4 points]** Using a 1-OR circuit as a black box, design a 2-OR circuit. How many gates does your circuit have? Argue the correctness of your circuit.

Solution:
DRAWING ATTACHED

(c) **[6 points]** Generalizing the black box approach from part (b), design the best possible k-OR circuit you can and derive a Θ-bound (in terms of the parameter k) for the number of gates in your k-OR circuit. Show your work for deriving this bound. For full credit, the number of gates in your circuit must be a polynomial in k.

Solution:

The resulting circuit for k-OR will results in 3^k logic gates as shown in the illustration from 2b.

Problem 4. *Recurrences (10 points)*

Suppose we have algorithms with running times $T(n)$ given by the recurrences:

1. $T(n) = 4T(n/2) + n^2$
2. $T(n) = T(n/2) + n$
3. $T(n) = 6T(n/25) + n^{1/2}$
4. $T(n) = 8T(n/2) + n^{3.5}$
5. $T(n) = 6T(n/10) + 4$

Determine and state the asymptotic running time of each of these five algorithms, and then rank them in ascending order of their asymptotic running time. You do not need to justify your ranking.

Solution:

a	b	d	$\frac{a}{b^d}$	$>/</=$	$T(n)$
4	2	2	4/4	$=1$	$\Theta n^2 \log n$
1	2	1	1/2	<1	$\Theta n^{1/2}$
6	25	1/2	6/5	>1	$\Theta n^{\log_{25} 6}$
8	2	3.5	8/ 11	<1	$\Theta n^{3.5}$
6	10	0	6/1	>1	$\Theta n^{\log_{10} 6}$

Ranking Order: $2, 3, 5, 4, 1$