CS 577: Introduction to Algorithms | Homework 1

Out: 02/02/21 | Due: 02/09/21

Name: Aiden Tepper | Wisc ID: 908242969

Ground Rules

- Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.
- The homework is to be done and submitted individually. You may discuss the homework with others in either section but you must write up the solution *on your own*.
- You are not allowed to consult any material outside of assigned textbooks and material the instructors post on the course websites. In particular, consulting the internet will be considered plagiarism and penalized appropriately.
- The homework is due at 11:59 PM CST on the due date. No extensions to the due date will be given under any circumstances.
- Homework must be submitted electronically on Gradescope.

Problem 1:

1. You are given 2^s arrays of integers that are already sorted, and each one has n elements. You want to combine these into a single sorted array of $2^s n$ elements.

   (a) One way of doing this is to repeatedly apply the MERGE subroutine from MERGESORT. That is, first merge the 1st and 2nd array, then merge the resulting array with the 3rd array, and so on. Asymptotically, how many times will this algorithm perform a comparison of two integers? Briefly justify your answer. (*Hint*: Merging two sorted arrays of size m and n requires $O(m + n)$ pairwise comparisons of integers.)

By line 3 and 4, the 2^k arrays are paired up and merged to 2^k arrays of size $2n$
These 2^k arrays then are used as an input to combine
By I.H., Combine for 2^{k+1} arrays returns a correctly merged/sorted array

$$T(n) = 2n + 3n + 4n + \ldots + 2^s n = (2+3+4+\ldots+2^s)\cdot n$$

$$n(2^s - 1) + n$$

$$= \frac{(2^s+2)(2^s-1)}{2} \cdot n = O(n \cdot 2^s \cdot 2^s)$$

$$= O(n \cdot 4^s)$$

(b) Devise a divide and conquer algorithm that uses $O(2^s sn)$ comparisons of integer pairs to combine the arrays into one sorted array.

Procedure Combine($A_1, A_2, \ldots, A_{2^s}$)

1. If $2^s = 1$: return A_1
2. Else:
3. For $i = 1, 2, \ldots 2^{s-1}$:
4. $B_i := \text{merge}(A_{2i}, A_{2i-1})$
5. return Combine($B_1, B_2, \ldots, B_{2^{s-1}}$)

Use induction to prove that your algorithm correctly combines the 2^s arrays into one sorted array.

(c)

Induct on s

Base case: s=0

There is only $2^s=1$ array which is already sorted

So Combine returns a sorted array

IH: s=k

Combine returns a sorted array for 2^k arrays

Inductive step: s=k+1

By line 3 and 4, the 2^{k+1} arrays are paired up and merged to 2^k arrays of size 2n

These 2^k arrays then are used as an input to combine

By I.H., Combine for 2^{k+1} arrays returns a correctly merged/sorted array

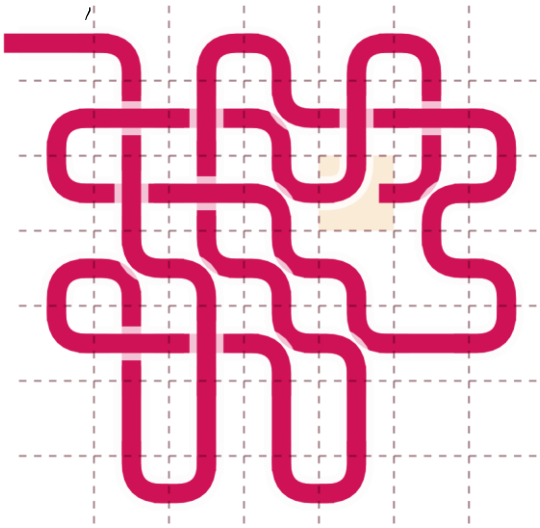
(d) Write a recurrence as a function of *n* and *s* for the number of times your algorithm performs a comparison of two integers. State the solution of your recurrence in asymptotic notation. No explanation is required.

$T(n,s) = 1 \cdot T(2n, s-1) + O(2^s \cdot n)$ where n = size of array, s = log(# of arrays)

$T(n,s) = O(2^s \cdot n \cdot s)$

Problem 2:

2. You are given an image depicting a rope that is wrapping around itself. The image is split into $n \times n$ cells. You are given that the top-left cell contains one end of the rope and you want to find the cell that contains the other end. When the rope comes in from one of the edges of a cell, it leaves out of another edge (unless it is the end of the rope). The rope never goes through the same edge of a cell twice. An example of such an image split in cells is given below for $n = 7$.



Your goal is to identify the cell that contains the other end. Every time you can ask to see how the image looks like at a cell of your choice. Find an algorithm that lets you identify the cell containing the other end of the rope while looking at as few of the cells of the image as possible.

(a) Give an example showing that following the rope around requires looking at $\Omega(n^2)$ cells.
(b) Suppose you iterate over the middle column of cells. How can you tell if the endpoint is to the left of themiddle column, to the right of this column, or in this column?

a) In the worst case, the rope passes through each cell at least once. This could require looking at every cell at least once, or at least $n \times n$ times. Thus, we would be required to look at $\Omega(n^2)$ cells.

b) Because the rope always starts on the left side, we can devise a pattern. Iterate down the column and count how many times the rope passes through the left vs. right

edges. If the left & right counts are both odd, we know the end is on the right side, because it's passed from left to right one more than right to left. If they're both even, then the end is on the left by similar logic. If they're not equal, then we know the rope exited the middle column without exiting; thus, the endpoint is in the column.

(c) Devise a divide and conquer algorithm that is more efficient than following the rope in terms of the numberof cells it queries. Describe your algorithm in pseudocode. Aim for a total of $O(n \log n)$ queries.

Algorithm: FindEndpoint | input: 2D array of rope cells (arr) | output: cell with endpoint

```
1.  leftCount := 0
2.  rightCount := 0
3.  length := len(arr[0])
4.  for i in arr[length/2]:
5.      if cell has endpoint:
6.          return i
7.      if rope enters left edge:
8.          leftCount++
9.      if rope enters right edge:
10.         rightCount++
11. elif leftCount is odd:
12.     return FindEndpoint(subarray of columns length/2 +1 to length)
13. else
14.     return FindEndpoint(subarray of columns 0 to length/2 -1)
```

(d) Prove the correctness of your algorithm.

given $n :=$ side length of grid, we will prove termination.

Base case: $n=1$

If $n=1$, FindEndpoint(image) will iterate through the 1 cell and terminate upon seeing the endpoint.

Inductive step: assume FindEndpoint terminates for all $n \in \mathbb{N}$ s.t. $1 \le n \le j$. We will prove that $j+1$ holds.

For $n=j+1$, the algorithm will always iterate first through the middle column, then determine the correct half to recurse on. The arms will narrow down onto the target column after each recursion, then, as shown above, successfully find the endpoint and terminate.

Assuming FindEndpoint terminates, we will prove program partial correctness.

Base case: $n=1$

If $n=1$, the algorithm returns on the one call, which is correct.

Inductive step: assume FindEndpoint holds for all $n \in \mathbb{N}$ s.t. $1 \le n \le k$. We will prove that FindEndpoint holds for $k+1$.

By logic found in b), the algorithm will correctly determine if the endpoint is within the current column, to the right, or to the left. Assuming the recursive lines (12 & 14) call FindEndpoint with the correct 2D-subarray, the algorithm will then repeat on the middle column of said subarray. At the worst case, the algorithm will recurse until it reaches the subarray of the 1 column containing the endpoint, then will correctly terminate after reading it.

Because the algorithm exhibits termination and partial correctness, it holds program correctness.

(e) How many cells does your algorithm need to query in the example image?
For general $n \times n$ images, write down a recurrence as a function of n for the number of queries that suffice for your algorithm to find the other end of the rope. State the solution of your recurrence in asymptotic notation. No explanation is required.

My algorithm needs to query 17 cells in the example image.

$T(n) = 3 + 6n \cdot T(n/2)$

$3 + 6(n/2) \cdot T(n/4)$

$3 + 6(n/4) \cdot T(n/8)$

$12n \cdot \frac{3 \log n}{\log 2} - 12$

$= O(n \log n)$