HW 4

Intro Algorithms EN.601.433
Fall 2023
Due: Friday 10/27/2023 at 11:59 PM

Instructions

- This homework is worth 10% of your final grade.
- This homework is due Friday 10/27/2023 at 11:59 PM on Gradescope. Solutions will be posted on Canvas at midnight. No late homework will be accepted. To access Gradescope, click the Gradescope link on Canvas.
- You are **required** to type your homework. We will provide the LaTeX source code to the homework assignments, which you may optionally use as a template. If you install LaTeX on your computer, you can generate a PDF of your homework by running the command:

  ```
  latexmk -pdf my-homework.tex
  ```

  Alternately, you can use `https://www.overleaf.com/edu/jhu` as a web based LaTeX editor.
- You can collaborate with one other person. Collaboration should happen through verbal communication, scratch paper, whiteboards, etc. You are **not** allowed to copy homework solutions from another student. All students are required to submit their own write-up.
- You are **not** allowed to use any website to find solutions to these problems.
- Do not write your name on your homework. Instead, write your Johns Hopkins Student ID. This will allow us to grade your homework anonymously.
- For questions that require you to write an algorithm out, you can either use pseudo-code or English description.
- Make sure that you correctly assign which page a problem is located on when uploading to Gradescope.
- For problems that require graphs or diagrams, you are allowed to include an image of a drawing instead of typesetting the graph.[1]

[1] If you are using LaTeX, you can follow the instructions here `https://www.overleaf.com/learn/latex/Inserting_Images` to embed an image in LaTeX.

1 Better Quick Sort Pivots? (5 points)

Recall that when using the QuickSort algorithm to sort an array A of length n, we picked an element $x \in A$ which we called the *pivot* and split the array A into two arrays A_S, A_L such that $\forall y \in A_S, y \leq x$ and $\forall y \in A_L, y > x$.

We will say that a pivot from an array A provides $t|n-t$ separation if t elements in A are smaller than or equal to the pivot, and $n-t$ elements are strictly larger than the pivot.

Suppose Bob knows a secret way to find a good pivot with $\frac{n}{3}|\frac{2n}{3}$ separation in constant time. But at the same time Alice knows her own secret technique, which provides separation $\frac{n}{4}|\frac{3n}{4}$, her technique also works in constant time.

Alice and Bob applied their secret techniques as subroutines in the QuickSort algorithm to pick pivots. Whose algorithm works **asymptotically** faster? Or are the runtimes **asymptotically** the same? Prove your statement.

1.1 Solution

Let's consider Alice's pivot point first, with an array of size n. Alice's technique provides separation $\frac{n}{4}|\frac{3n}{4}$, meaning we have 2 arrays of size $\frac{n}{4}$ and $\frac{3n}{4}$ respectively, that we need to sort individually. If we continue this sorting algorithm recursively, and the level of the deepest final branch is level k, occuring at $n(\frac{3}{4})^k = 1$, so the level of the deepest final branch will occur at $k = \log_{\frac{4}{3}}(n)$.

Similarly, looking at the shallowest final branch of the recursion tree, the branch m will have depth $n(\frac{1}{4})^m = 1$, or $m = \log_4(n)$.

If we sum up the amount of work done on every level, we obtain $O(n)$. For example, in the 3rd level, we have $\frac{n}{16} + \frac{3n}{16} + \frac{3n}{16} + \frac{9n}{16} = n$.

Therefore, the minimum time complexity for Alice's algorithm is $O(n \log_4(n))$, and the maximum time complexity for her algorithm is $O(n \log_{\frac{4}{3}}(n))$. Our overall asymptotic time complexity will be $\theta(n \log(n))$.

Now, let's consider Bob's pivot point, with an equally size n array. Bob's technique uses separation $\frac{n}{3}|\frac{2n}{3}$, meaning we have 2 arrays of size $\frac{n}{3}$ and $\frac{2n}{3}$ respectively. Using the same method that we did for Alice's algorithm, the level k of the deepest final branch is $k = \log_{\frac{3}{2}}(n)$, and the level m of the shallowest final branch is $m = \log_3(n)$.

Similar to Alice's algorithm, Bob's algorithm also takes n work at every level, resulting in $O(n)$. This means that Bob's algorithm's minimum time complexity is $O(n \log_3(n))$, and the maximum time complexity is $O(n \log_{\frac{3}{2}}(n))$. It's overall asymptotic time complexity is $\theta(n \log(n))$.

The two algorithms have the same overall asymptotic time complexity of $\theta(n \log(n))$, which means that they have the exact same runtime, which means no algorithm is faster.

2 Sorted Matrix Search (5 points)

Alice and Bob are solving a problem. They are given a matrix A of size $n \times n$, where elements are sorted along every column and every row.

They need to provide an algorithm which takes integer x as an input, and checks if x is an element of matrix A or not. Both Alice and Bob decided to apply divide and conquer method.

Alice's algorithm is quite simple, she uses binary search routine to check each row.

Bob's algorithm is more advanced. Bob found out that if matrix A is represented in block-view:

$$A = \left[\begin{array}{c|c} A_1 & A_2 \\ \hline A_3 & A_4 \end{array}\right],$$

where all matrices A_1, A_2, A_3, A_4 are of the size $\frac{n}{2} \times \frac{n}{2}$, then he can compare element x with $A_4[1,1]$ and consider only three options:

1. $x = A_4[1,1]$ then his algorithm outputs "YES".
2. $x < A_4[1,1]$ then he knows that $\forall i,j < n/2 : \quad x < A_4[i,j]$, thus he only needs to recursively check if $x \in A_1$, if $x \in A_2$ and if $x \in A_3$.
3. $x > A_4[1,1]$ then he knows that $\forall i,j < n/2 : \quad x > A_1[i,j]$, thus he only needs to recursively check if $x \in A_2$, if $x \in A_3$ and if $x \in A_4$.

Whose algorithm is **asymptotically** faster on the worst case input? Prove your statement.

2.1 Solution:

Let's look at Alice's algorithm first. Because her algorithm is a simple binary search on each row, the complexity of every row is $\mathcal{O}(\log n)$, and since there are n rows, the overall complexity is $\mathcal{O}(n \log n)$.

We can calculate the runtime of Bob's algorithm using the Master Theorem. Since there are 3 subproblems, $a = 3$. $b = 2$ because the problem decreases by a factor of 2, and $c = 0$ because the work to combine or divide the subproblems is constant. Substituting these values in, we obtain $T(n) = \Theta(n^{log_2 3})$.

Comparing these two runtimes, Alice's algorithm is faster on the worst case.

3 Local Search on a Tree (5 points)

Consider an n-node complete binary tree T, where $n = 2^d - 1$ for some d. Each node v of T is labeled with a real number x_v. You may assume that the real numbers labeling the nodes are all distinct. A node v of T is a *local minimum* if the label x_v is less than the label x_w for all nodes w that are joined to v by an edge.

You are given such a complete binary tree T, but the labeling is only specified in the following *implicit* way: for each node v, you can determine the value x_v by *probing* the node v. Show how to find a local minimum of T using only $O(\log n)$ probes to the nodes of T.

3.1 Solution

Since all the labels of the tree are distinct real numbers and the tree is finite, there exists a smallest element in the tree. In a complete binary tree, every node has a maximum of three neighbours, one parent and two children. In case of a root node, there is no parent, and in case of leaf nodes, there are no children. In order to know if a given node is a minimum, we have to compare it to its parent and child nodes. In case of root node, we have to just compare it with it's children.

1. Set the root node as the current node
2. Compare the label of current node to the label of its children
3. If the current node is smaller, then return it as local minimum

4. Otherwise, pick the smallest of the children nodes and repeat from step 2.

This algorithm is $O(\log(n))$ because we are traversing the levels of the tree only once in its maximum time complexity, which is the depth of the complete tree, $\log(n)$.

4 Sorting Algorithms (5 points)

1. Show the steps that the merge sort takes when sorting the following array

$$L = [5, 0, 2, 1, 6, 4, 3]$$

2. Show the steps that non-randomized quicksort will take on the following array. Choose the pivot element as the first element in the sub-sequence (element at the lowest index of the array).

$$L = [0, 1, 2, 3, 4]$$

What is the time complexity of non-randomized quicksort in this case?

4.1 Solution

4.1.1 Merge Sort Steps

$$L = [5, 0, 2, 1, 6, 4, 3]$$

$$[5, 0, 2][1, 6, 4, 3]$$

$$[5][0, 2][1, 6][4, 3]$$

$$[5][0][2][1][6][4][3]$$

$$[5][0, 2][1, 6][3, 4]$$

$$[0, 2, 5][1, 3, 4, 6]$$

$$L = [0, 1, 2, 3, 4, 5, 6]$$

4.1.2 Non-Randomized Quicksort

$$L = [0, 1, 2, 3, 4]$$

$$[0][1][2, 3, 4], n \text{ comparisons}$$

$$[0][1][2][3, 4], n \text{ comparisons}$$

$$[0][1][2][3][4], n \text{ comparisons}$$

$$L = [0, 1, 2, 3, 4], n \text{ comparisons}$$

In this case, the time complexity of the non-randomized quick sort is $O(n^2)$, which is the worst-case scenario.

5 Recurrences Bounds (3 points)

Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible and justify your answer.

1. $T(n) = T(3n/11) + n$
2. $T(n) = 6T(n/3) + n^2$
3. $T(n) = 2T(n/4) + \sqrt{n}$

5.1 Solution

1. $1 > \log_{\frac{11}{3}}(1)$, therefore $T(n) = \Theta(n)$
2. $2 > \log_3(6)$, therefore $T(n) = \Theta(n^2)$
3. $\frac{1}{2} = \log_4(2)$, therefore $T(n) = \Theta(n^{\frac{1}{2}} \log n)$