Self-Assessment Review

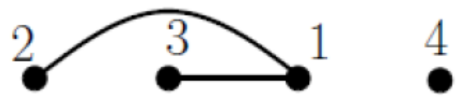
Hiba Mallick - 24015

CSE 317 Design and Analysis of Algorithms
Spring 2024

Question 2 - Midterm

Question:

You are given as input a permutation = (1, 2, . . . , n) of 1, 2,...,n. Let G be an undirected graph constructed from as follows: It has n vertices, and there's an undirected edge between i and j where i < j and $\pi(i) > \pi(j)$. For example, if the permutation was = (2, 3, 1, 4), then Gπ is the following: Develop an algorithm for computing the connected components of G , i.e.,



the output of the algorithm on the above input could be 1, 2, 3, 4. For full credit your algorithm should run in O(n) time.

Score:

My answer received 4/10 points

LLM:

ChatGPT 4o

Assessment:

For this Self-Assessment, I first gave the LLM instructions to act as a very smart algorithmic analysis expert, and gave it the entire question and my solution to the question and asked it about its correctness or lack thereof and to correct what is incorrect.

GPT identified issues with my solution's efficiency when adding elements to an array and checking for their existence in each iteration, which can become inefficient. Checking if an element exists in an array is typically $O(n)$, making the overall complexity higher than $O(n)$. It also pointed out my incorrect handling of connected components, using a sequential method instead of something more efficient to merge the connected components.

GPT gave me its solution using a Union Find data structure with n elements. It built the graph by iterating through the array to determine the edges based on the condition $\pi(i) > \pi(j)$ for $i < j$ and performing a union operation at each valid edge. After this, it extracted all the connected components and returned them. However, Union Find's final time complexity is amortized $O(\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function.

So, I prompted GPT to simply improve upon my solution's weak points without using UF. To this, it gave an algorithm that only compared each i with i+1 and iterated through each index i from 0 to n-2. It checks if there is an inversion $\pi(i) > \pi(j)$; it adds them to the current component or starts a new component otherwise. However, this is incorrect since this only checks for each i and i+1, i.e., its immediate next number when it should be checking for each i and j where j>i. This solution should avoid nested loops and work in O(n) complexity.

Upon pointing this out, GPT returned an improved approach using a stack to maintain the current sequence of elements and indices for efficiency and to identify the connected components as we process the given array.

For each element in the array, we check the stack to detect any inversions $\pi(i) > \pi(j)$ where j>i, and if found, we pop from the stack and add I and j to a list of elements in the current component. Then, we push the current index on the stack. If the current component is not empty, and the last element is not the current index, we finalize the current component as a connected component. After the loop, we handle any remaining indices in the stack, which form the last connected component.

This method correctly handles all pairs (i,j) with i<j and $\pi(i) > \pi(j)$, ensuring an O(n) solution for finding the connected components.

Chat link:

Due to a bug, it wasn't letting me generate a shareable link; however, I uploaded screenshots of my conversation at the link below.
`https://drive.google.com/drive/folders/1qyZw23Og2AWmSDlHQYrOdWkEP1BaIEP0?usp=drive_link`

Question 1, part (b) - Problem Set 2

Question:

Let S be a set of n arbitrary but distinct numbers.

1. Give a deterministic algorithm to output the largest $n^{3/4}$ of the numbers in S in sorted order. Your algorithm must use $O(n)$ comparisons.
2. Show that any deterministic comparison-based algorithm that correctly outputs the median on inputs of n distinct elements must use at least $n-1$ comparisons. To be completely precise, the median is defined to be the $\lceil n/2 \rceil^{th}$ smallest element.

Score:

My answer received full credit for (a) but 0 credit for (b) (3/6 points)

LLM:

ChatGPT 4o

Assessment:

For this Self Assessment, I used ChatGPT 4o to obtain the answer for not just part (b), but the entire question. Since I answered part (a) correctly, I will attempt to see if the LLM can answer it correctly. Once this is done, I will try to obtain the correct answer for part (b) as well.

My first attempt was to prompt it to answer the entire question to see what approach it would pick.

For part (a), it correctly fulfilled the first grading requirement, which was to recognize that the question required finding the $n - n^{\frac{3}{4}}$ th largest element to serve as the threshold. Elements greater than this element are among the largest $n^{\frac{3}{4}}$th numbers.

Just to confirm, I asked GPT if it would be correct if I used a list containing the last $n^{\frac{3}{4}}$ elements, to which it agreed and gave me a far more incorrect solution than the initial one.

Then, in the initial solution, it partitions the array into GREATER and LESSER sub-arrays and then sorts the GREATER list using Merge Sort, which takes $nlogn$, or O($n^{\frac{3}{4}} log(n^{\frac{3}{4}})$) as correctly identified in the solution. It calculates the total complexity of this solution and correctly shows that the dominant term is $O(n)$. Thus, GPT's initial solution to part (a) correctly solves the question even when subjected to further questioning and clearly convinces me of the right answer.

For part (b), GPT initially solves it using a binary comparison tree model as a representation of a deterministic comparison-based algorithm. Each node in the tree is a comparison between two elements. This seems like a solid approach; however, it then leads to using the concept of n! possible permutations of n distinct elements, leading to a different outcome for the median, which is incorrect since it implies there are n! permutations of the median in a sorted list.

To guide GPT to the correct answer, I explained that its n! permutations approach is incorrect, to which it corrected itself and refined its previous answer to prove that n-1 comparisons are needed. It again uses the binary comparison tree model as a representation of a deterministic comparison-based algorithm but still descends into an inelegant argument using n! possible permutations of the n elements without proper proof.

Finally, I asked it to use a representation of a graph to solve this. Then, GPT used a graph where each node is an element, and each edge represents a comparison between two elements. It uses a fully connected graph. Otherwise, there would be elements we don't know about. It uses the concept of a minimum spanning tree (MST), a connected graph with n vertices that has n-1 edges, which is the minimum number of edges required to ensure all vertices are connected. Similarly, in the comparison graph, n−1 comparisons (edges) are the minimum required to ensure that the graph is connected and thus every element is reachable from every other element through comparisons.

To determine the median, it realizes that a connected comparison graph with n vertices and n−1 edges ensures that there are no isolated vertices. With n−1 comparisons, we can build a spanning tree that includes all n elements.

Thus, applying this to our comparison graph shows that it takes at minimum n-1 comparisons to find the median using a connected graph and the concept of an MST.

Chat link:

`https://chatgpt.com/share/67904faf-e9d5-4105-bfee-51e1efbe9299`