Dissertation Progress Report

Title: Visualising Divide-and-Conquer Algorithms
Submitted by: Ashwin Harish (s2710401)

1 Goal of the project

- The goal of this project is to develop an interactive visualization tool that illustrates the functioning of divide-and-conquer algorithms across at least three distinct problem domains, showcasing the algorithm's versatility.
- It aims to visualise examples in a conceptual approach rather than following a code based approach while maintaining a user-driven interface that enables learners to control the algorithm's execution step-by-step and explore at their own pace.

2 Methodology

The project currently has three use cases: Quick Sort, Convex Hull, and Binary Tree construction.

In Quick Sort, each step involves selecting a pivot element and rearranging the array so all numbers smaller than the pivot are on the left side and all numbers larger than the pivot are on the right side of it. The algorithm uses the "median-of-three" strategy to select the pivot element. This is the median value among the first, middle, and last elements of the current partition (it can be the whole array at the very beginning). In the very first click of 'Divide' there can be just one pivot, in the second 'Divide', there could be a maximum of two pivots, in the third, there could be a maximum of four pivots and so on. The base case is when there is only one number in each partition. This is how the array is sorted in the background. Section 3 mentions how it is visualised and connected to buttons.

In Convex Hull, the set of points is divided into halves again and again. The first divide operation creates two partitions, the second divide operation creates a maximum of four partitions and so on. The base case is when all the partitions are having either one point or two points where the hull will be the single point or line joining the two points respectively. A case where there are more than two collinear points having the same x-coordinate is treated similarly to having two points in the partition. Their convex hull will be the line that joins the two end points. The conquer phase merges these partitions to create larger convex hulls. The convex hull is constructed using a two-tangent merge approach. When merging two hulls, the algorithm identifies the upper and lower tangents connecting the left and right hulls. The endpoints of these tangents become part of the merged convex hull. All points on the tangents but not at the endpoints are excluded, as they lie on the straight line connecting two convex boundaries without contributing to the final shape. Additionally, points inside the quadrilateral formed by the tangent endpoints are discarded, as they are enclosed within the convex shape. However, any point that lies between the tangents but outside this quadrilateral is retained, as it contributes to the outer boundary of the combined hull.

In Binary Tree use-case, the algorithm requires two inputs - in-order and pre-order traversals to compute the final tree. The algorithm identifies the root node as the first element in the pre-order traversal. This root is located within the in-order sequence, and the elements to the left and right of it represent the left and right subtrees respectively. This forms two partitions in the in-order sequence. These partitions are then located within the pre-order sequence to identify the root/parent of that partitions/subtree. The process continues until each partition contains only one element. While reconstructing the tree, the root/parent is attached to their left and right subtrees. Repeating this until the main root is attached gives a final connected binary tree.

3 Things accomplished

Currently, the tool has three different pages created for all three use-cases. The user can navigate to each of the pages using direct link present in each page.

In the Quick Sort use-case, users can input up to 12 numbers either one at a time or as a comma/space-separated string. The entered numbers are then represented as distinct bars with their height being same as the number. The recursive nature of Quick Sort is adapted into an iterative approach, with each iteration triggered by the user clicking the 'Divide' button. During the divide phase, each click of 'Divide' button processes all currently active partitions, creating sub-partitions using the median-of-three pivot. If no further division is possible, the 'Divide' button is disabled. Alternatively, the user can press 'Solve' at any time during the divide phase to immediately display the sorted solution array based on the current state, skipping any remaining divide and conquer steps. Clicking on 'Solve' even before clicking 'Divide' immediately displays the final solution. To highlight the pivot(s) created in divide phase, the bar representing the pivot is coloured differently and is placed with wider gap compared to non-pivot elements. The conquer phase de-highlights the pivots and brings it closer to other array elements.

In the Convex Hull use-case, users can input points by clicking on an 11x11 grid. They have the liberty to input any number of points - from a single point to all 121 points. On clicking 'Divide', the current set of points is split into two halves based on their x coordinates, and a vertical line is drawn to indicate the division. Repeated clicks continue dividing all eligible subsets, with the 'Divide' button being disabled when there are no more partitions to make. The vertical line showing the partitions are of varying thickness and colour to highlight the order in which those partitions were created. The conquer phase merges the convex hulls of partitions starting from the most recently created partition. The conquer phase is complete when the convex hull for the entire set of points is generated. Like the previous use-case, clicking 'Solve' at any point during the divide phase immediately constructs and displays the convex hull based on the current partition state, skipping the remaining divide and its associated conquer steps.

In the Binary Tree use-case, the users can enter comma/space-separated strings for both in-order and pre-order traversals. The divide phase displays both the traversals with the root highlighted. It also shows the left and right partitions in the in-order traversal. Subsequent divide operations take each partition and then pick a parent node along with left and right partitions until each partition contains only one element. During the conquer phase, nodes created at the most recent level are rendered on screen. Leaf nodes are displayed without attachment, while internal nodes are connected to their respective children from lower levels. This process continues level by level until the entire binary tree is assembled. Clicking 'Solve' during the divide phase constructs the partial tree (or the complete tree if clicked before 'Divide') based on existing partitions and skips the remaining divide and conquer operations.

4 Future work

The following tasks need to be done on this project in the future to enhance its usability and visual appearance:

- Make minor alignment adjustments to improve the overall visual consistency of the interface.
- Add contextual pop-ups to guide users through navigation and explain briefly the changes that took place after any button (Divide, Conquer or Solve) click.
- Implement undo and redo functionality to give users greater control, allowing them to revisit or repeat previous actions for understanding and/or experimentation.