**Group Members :**

* Benedict Zevanno Christabel - 2602056851
* Evander Octavianus Layardi – 2602075983
* Florenze Albertine Tatuwey - 2602072022
* Hans William Christianto Wijaya – 2602079331
* Jeffline Kristian D. – 2602077156

For this project, our group decided to make a comparison between 6 different types of sorting algorithms which are Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort, and Heap Sort.

How our code works is, we made an array of integer which will be randomized by the program and to visualize each of the sorting algorithms we made a 2D array of characters which will show the data as a histogram.

To visualize the sorting algorithm, the first thing the program will do is randomized the array of integer and also setting up the histogram. After the user choose a sorting algorithm to be visualized the histogram will always be updated every single iteration. Most of the algorithms update the histogram every single time two elements is swapped, but for selection sort because it will swap the elements at the end of the inner for loop it will not represent the time complexity that it supposed to.

**Sorting algorithms comparison :**

1. Bubble Sort

This sorting works by comparing each element with the element next to it, if it’s higher or lower (depends if you want to sort it in ascending or descending order) it will swap those two elements until all of the elements are sorted.

Bubble sort is a sorting algorithm which uses brute force approach. This sorting is using a nested for loop which make this algorithm has the time complexity of O(n2) on all the cases either the best, average, or worse case.

1. Selection Sort

Selection sort works by finding the smallest or largest element(depends on how you want to sort it) of the unsorted part of the array and at the end swapping it with the first index of the array, then the second, then the third, and so on until all of the elements are sorted.

Selection sort uses a greedy approach. Despite using a greedy approach, the time complexity of this algorithm is the same as bubble sort on all the cases which is O(n2) because it also uses a nested for loop.

1. Insertion Sort

Insertion sort works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Insertion sort is a comparison-based, incremental sorting algorithm that is not classified as either brute force or greedy. This algorithm by far is the better one than both bubble and selection sort, because at the best case, this algorithm has the time complexity of O(n) but the average and worst is still O(n2).

Notes : Three of the algorithms above are a good algorithm to use on a small dataset, but for a larger dataset the algorithms below are much more efficient.

1. Quick Sort

Quick sort is a highly efficient sorting algorithm that divides a large data array into smaller ones. It works by selecting a 'pivot' element from the array and partitioning the other elements into two groups: one containing values smaller than the pivot and the other containing values greater than the pivot. This process is applied recursively to the two smaller sub-arrays.

This algorithm uses divide and conquer approach. It has an average time and best time complexity of O(n\*log n), but the downside of this algorithm is the time complexity can be O(n^2) if pivot selection leads to unbalanced partitions.

1. Merge Sort

Merge Sort, a divide and conquer sorting algorithm. It recursively breaks down a problem into sub-problems, solves them, and then combines the solutions. Merge Sort first divides an array, then merges the smaller lists in a sorted manner.

This algorithm has the time complexity of O(n \* log n) in all of the cases.

1. Heap Sort

Heap sort works by first making a min heap or a max heap from our array then swapping the root of the heap to the last element of the array then heapify the rest of the heap, it then repeats that process over and over until all of the elements are sorted either ascendingly or descendingly.

Heap sort has the time complexity of O(n \* log n) in all of the cases which makes it consistent like merge sort.