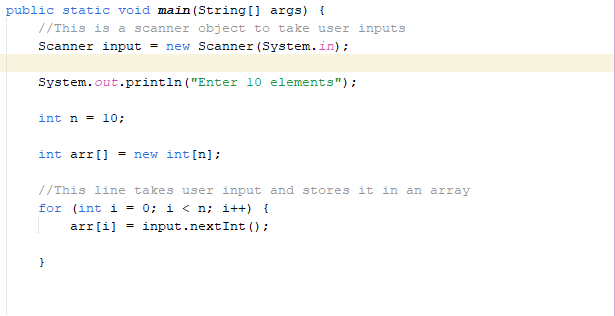
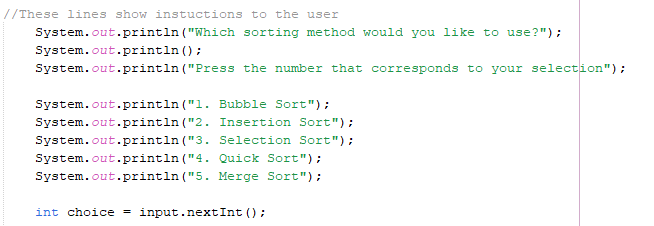
Game Implementation description

Inputting 10 elements





Declaration of a scanner called input, the user is prompted to enter 10 elements and a declared array named “arr[]” which has a size of n = 10.The for loop repeats from when i = 0 until i >= n while incrementing by one with each iteration, inside the loop as the loop iterates to a larger value of i representing the array index, stores the user’s 10 element input thus creating an unsorted array with 10 elements. User is prompted to pick a sorting algorithm.

From the main method a switch case is created to allow the user to pick a sorting algorithm of which each case calls the corresponding method with the required sorting algorithm implementation (Bubble, insertion, selection, merge or quick sort).

Bubble Sort

A method was created, which takes an array as input (parameter) and performs the Bubble Sort algorithm. A for loop was created to display the array elements to be sorted. The Outer loop was implemented and will run n-1 (i < arr.length) times, where n is the length of the array. This loop represents the number of passes needed to sort the array. The Inner loop was implemented, it will compare each pair of adjacent elements and swap them if they are in the wrong order. It runs from the start of the array up to n-i-1(i < arr.length - 1), because the last i elements are already sorted. Another inner was created within the first outer loop, which displays each pass and another which displays the fully sorted one.

Insertion Sort

A method was created, which takes an array as input (parameter) and performs the Insertion Sort algorithm. A for loop was created to display the array elements to be sorted. The Outer loop was created and runs from i = 0 to n-1, this loop represents the number of passes needed to sort the array and initializes j as i – 1 as to represent the first element. If the temporary element is smaller than the elements in the sorted portion, we shift those larger elements one position to the right to make space for it. This was done using another loop that continues until we find the correct position for the key or reach the beginning of the array (the first index). This process is continued until the array is sorted. Another inner was created within the first outer loop, which displays each pass and another which displays the fully sorted one.

Selection Sort

A method was created, which takes an array as input (parameter) and performs the selection sort algorithm. A for loop was created to display the array elements to be sorted. An outer loop was set up which will iterate through each element of the array, except for the last one because, after each pass, the smallest element is placed in its correct position, and we need to check only the unsorted part. After identifying the smallest element in the unsorted portion, we swap it with the first element of that unsorted portion. This moves the smallest element to the end of the sorted portion of the array. The outer loop then continues to the next index, and the process repeats until the whole array is sorted. Another inner was created within the first outer loop, which displays each pass and another which displays the fully sorted one.

Merge sort

Two methods where created was created, one which sorts the array and one which merges the two sorted sub arrays, the first one taking an array as a parameter and the second one taking the array as a parameter as the left and right sub arrays. A for loop was created to display the array elements to be sorted. Instead of using recursion, we use an iterative loop that starts with subarrays of size 1 and gradually increases the size of the subarrays until the entire array is sorted. For each iteration of the loop, we perform the merge operation on all pairs of subarrays of the current size. The merge function, it merges two sorted subarrays into a single sorted array using a temporary array. After merging the subarrays, we copy the sorted elements back to the original array. Also adding a for-loop to print after each pass.

Quick Sort

Two methods where created, one to perform the quick sort and one to partition the array around the pivot. The partition method rearranges the elements based on the pivot Whenever an element less than the pivot is found, swap it with the element at the position of the first pointer, and then move the first pointer to the right. After traversing the array, the pivot was swapped with the element at the position of the first pointer. This places the pivot in its correct position. Once the quicksort method was defined, it can be called from the main method.

Difficulties faced during, the game’s creation

Working as a group made added some complexity to the project, Effectively dividing tasks among team members based on their strengths and ensuring a fair workload distribution. Maintaining clear and consistent communication to prevent misunderstandings and ensure everyone is aligned with project's goals. Balancing these collaborative challenges along with the technical aspects of implementing and visualizing sorting algorithms was necessary for us to successfully complete the project. Through teamwork and collective problem solving, the group was able to overcome these challenges and create a functional sorting game.

Code References

The lecture slides provided where the ones used for bubble, selection and insertion sort algorithms, for merge and quick sort we managed with the help of websites such as geek for geeks and w3school.

Screenshot of the game in action

