Activity No. 8.1	
Sorting Algorithms 2	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 9/23/2025**
Section: CPE21S4	**Date Submitted: 9/23/2025**
Name(s): Anastacio, Lester Arvid P.	**Instructor: Engr. Jimlord M. Quejado**
6. Output	

7. Supplementary Activity

A. Quick Sort

- This type of sorting is something like a divide and conquer sorting algorithm in which works by choosing a pivot element and then dividing the list into two sublists, in which one of the element is smaller than the pivot in which will be listed into the left side and another element is greater than the pivot in which will be put into the right side and afterwards it will recursively sort out the sublists until the entire list of elements is ordered, it is a very efficient when it comes to time but may cause some problems if the pivots are chosen poorly.

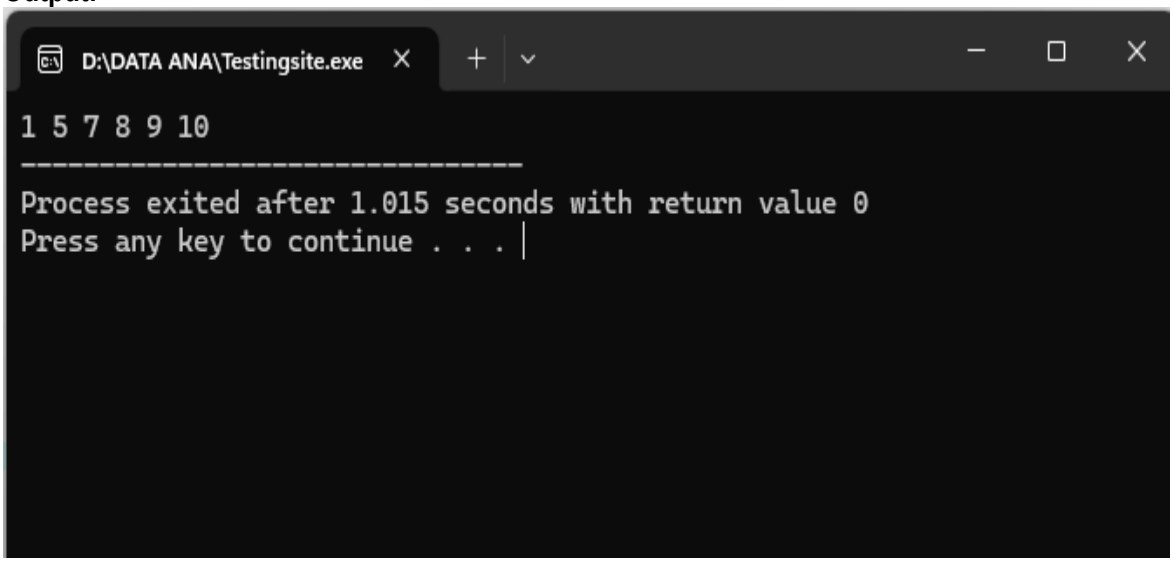
Sample Code:

```cpp
//testing site
#include <iostream>
#include <vector>
using namespace std;

int partition(vector<int>& arr, int low, int high) {

    // choose the pivot
    int pivot = arr[high];

    // index of smaller element and indicates
    // the right position of pivot found so far
    int i = low - 1;

    // Traverse arr[low..high] and move all smaller
    // elements on left side. Elements from low to
    // i are smaller after every iteration
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }

    // move pivot after smaller elements and
    // return its position
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

// the QuickSort function implementation
void quickSort(vector<int>& arr, int low, int high) {

    if (low < high) {

        // pi is the partition return index of pivot
        int pi = partition(arr, low, high);
```

```
        // recursion calls for smaller elements
        // and greater or equals elements
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    vector<int> arr = {10, 7, 8, 9, 1, 5};
    int n = arr.size();
    quickSort(arr, 0, n - 1);

    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Source: https://www.geeksforgeeks.org/dsa/quick-sort-algorithm/

Output:



Explanation

- I think this program works by selecting a pivot and then dividing the array into smaller and larger elements around the pivot in which it recursively sorting the subarray in which can be found in the quicksort function if the current segment has more than one element then the partition function is called to find the correct position of the pivot and then in the main function a vector array is created with its elements and the size of the vector is stored in n and quicksort is called to sort the array from the first to the last index printing the fully sorted array

B. Shell Sort

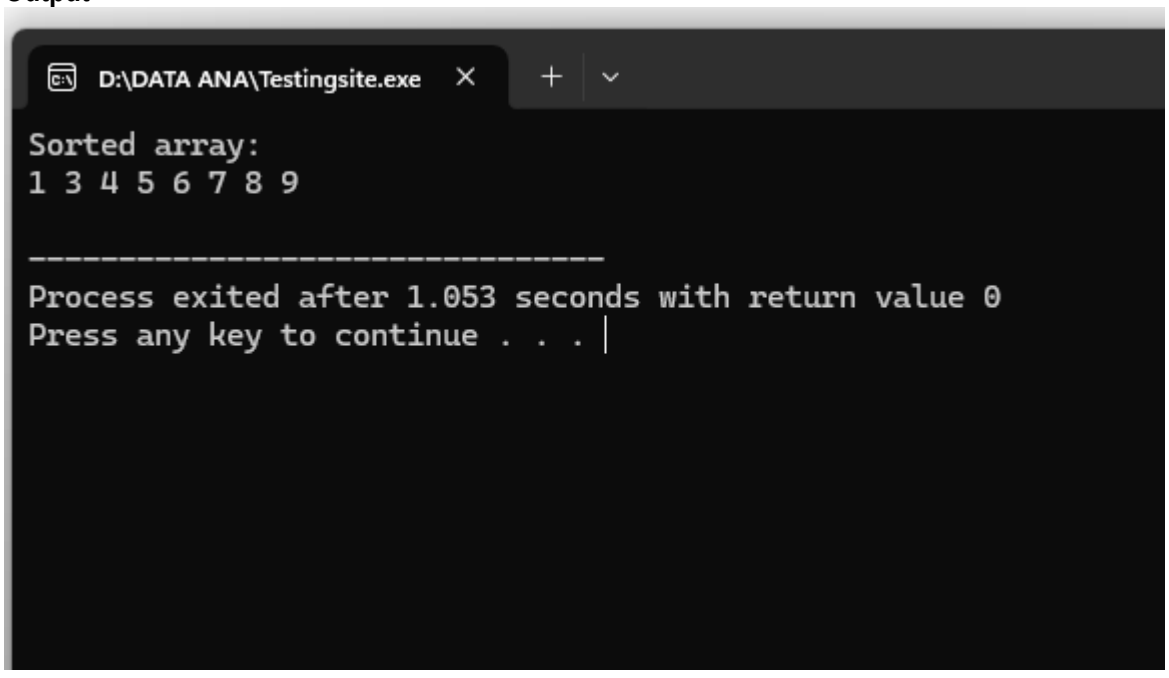
- This type of sorting is quite similar to insertion sorting but more improved in which works by comparing and sorting the elements that are far apart and then using a sequence of gaps in which gradually reduces until it becomes one similar to what insertion sort finishes.

Sample Code

```cpp
// Shell Sort in C++ programming

#include <iostream>
using namespace std;

// Shell sort
void shellSort(int array[], int n) {
  // Rearrange elements at each n/2, n/4, n/8, ... intervals
  for (int interval = n / 2; interval > 0; interval /= 2) {
    for (int i = interval; i < n; i += 1) {
      int temp = array[i];
      int j;
      for (j = i; j >= interval && array[j - interval] > temp; j -= interval) {
        array[j] = array[j - interval];
      }
      array[j] = temp;
    }
  }
}

// Print an array
void printArray(int array[], int size) {
  int i;
  for (i = 0; i < size; i++)
    cout << array[i] << " ";
  cout << endl;
}

// Driver code
int main() {
  int data[] = {9, 8, 3, 7, 5, 6, 4, 1};
  int size = sizeof(data) / sizeof(data[0]);
  shellSort(data, size);
  cout << "Sorted array: \n";
  printArray(data, size);
}
```

Source: https://www.programiz.com/dsa/shell-sort

Output



Explanation

- The program that I found here works by sorting an array using Shell sort, in which it starts by picking a large interval in which is the half of the array size and then compares and shifts the elements that are that far apart and after each pass the interval will then be reduced until it finally became 1 whereas the array becomes gradually more ordered with each pass and finally the last part is where the sorted array will be printed out.

C. Merge Sort

- This type of sorting is also a divide and conquer sorting algorithm but what makes it different to quick sort is that this works by repeatedly splitting the list into halves until only single elements remains and then it will merge these sublists back together in sorted order it will continue to do this until the whole list is arranged.

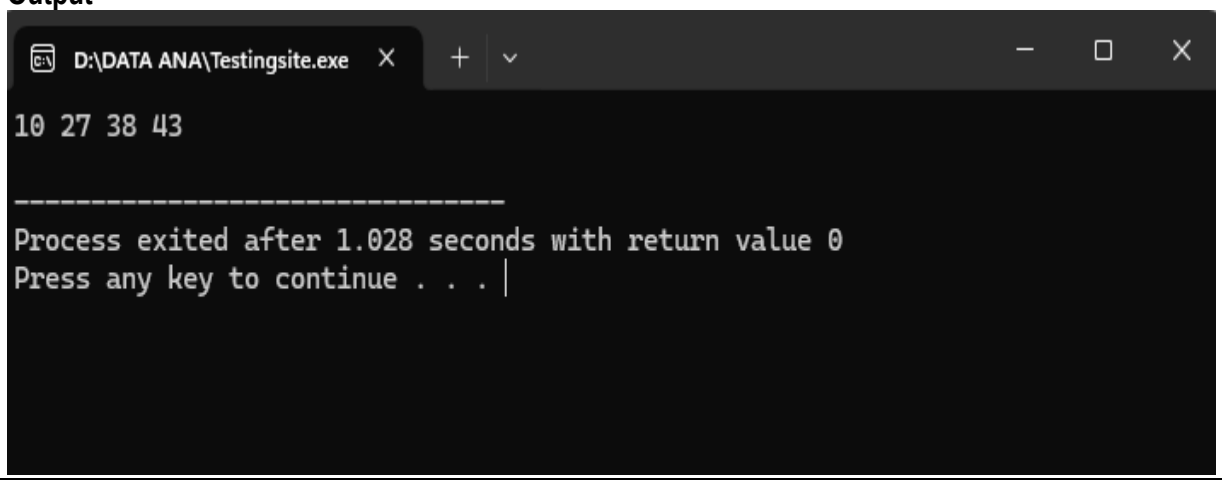
Sample Code

```
#include <iostream>
#include <vector>
using namespace std;

// Merges two subarrays of arr[].
// First subarray is arr[left..mid]
// Second subarray is arr[mid+1..right]
void merge(vector<int>& arr, int left,
           int mid, int right){

    int n1 = mid - left + 1;
    int n2 = right - mid;

    // Create temp vectors
    vector<int> L(n1), R(n2);

    // Copy data to temp vectors L[] and R[]
    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0;
    int k = left;

    // Merge the temp vectors back
    // into arr[left..right]
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
```

```cpp
    }

    // Copy the remaining elements of L[],
    // if there are any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy the remaining elements of R[],
    // if there are any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// begin is for left index and end is right index
// of the sub-array of arr to be sorted
void mergeSort(vector<int>& arr, int left, int right){

    if (left >= right)
        return;

    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);
    merge(arr, left, mid, right);
}

// Driver code
int main(){

    vector<int> arr = {38, 27, 43, 10};
    int n = arr.size();

    mergeSort(arr, 0, n - 1);
    for (int i = 0; i < arr.size(); i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

Source: https://www.geeksforgeeks.org/dsa/merge-sort/

Output



Explanation

- The program is sorting an array by the merge sort algorithm which, logically speaking, is a situation where the solution to the problem is found by breaking it down into smaller problems (divide and conquer paradigm). The mergeSort function keeps dividing the array into halves until both have only one element. Thus the merge function combines two sorted subarrays into one sorted by comparing the elements from both sides (L[] and R[]) and putting the smaller one in the original array. If any elements are left in L[] or R[], they are copied. This action is repeated recursively until the entire array is sorted. Lastly, the main function creates the array and then calls mergeSort and displays the sorted array.

8. Conclusion

- This activity has exposed us to various sorting methods besides those more commonly known. We have been able to look into and understand C++ programs that implement quick sorting, shell sorting, and merge sorting algorithms. By these programs, we learned the operations of each algorithm and recognized their different ways of sorting data. We also segregated the sorting algorithms types by visualizing their methods—divide and conquer he used for Quick Sort and Merge Sort, and gap-based insertion for Shell Sort. Last but not least, through code writing and code execution, we fulfilled the goal of producing the working implementations of sorting algorithms.

9. Assessment Rubric