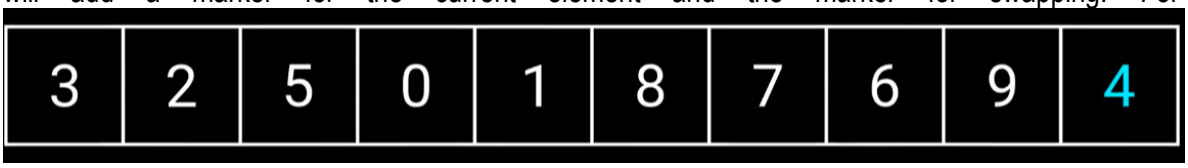
Activity No. 8.1	
Using Sorting Algorithms 2	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: Sept. 23, 2024**
Section: CPE21S4	**Date Submitted: Sept. 23, 2024**
Name(s): Crishen Luper S. Pulgado	**Instructor: Sir Jimlord Quejado**

6. Output

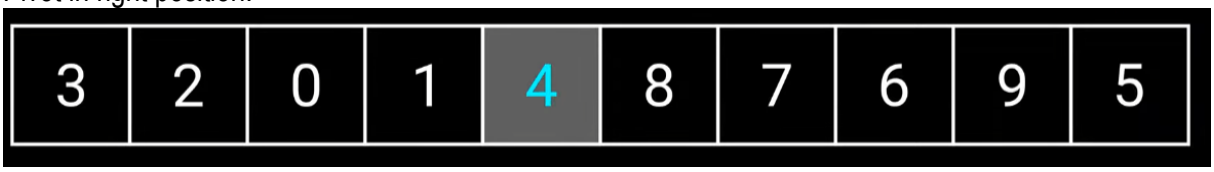
1.) Explain the quick sort, shell sort and merge sort types of sorting algorithms.

In quick sort, it divides the elements in the array and choose an element as pivot and partitions the array around the picked pivot by placing the pivot in its correct position in the sorted array (GeeksforGeeks, 2025). The pivot tells whether it will pass the elemen. Choosing a pivot can vary like last element, first element, or median. Now, we will add a marker for the current element and the marker for swapping. For example:



Here, we pick the last element 4 as our pivot. The current index (i) is zero and the swap marker is at index (j) is -1. Now, the condition is i>pivot we will pass and i<=pivot we will move the swap so j+1. There will be two conditions if we do not pass, if i>j then it will swap and if i==j then pass the elements. This will continue to repeat until the pivot is in the correct position. After it is in the correct position, it will divide the array and the elements smaller than 4 will be on the right and the elements higher than 4 is on the left. These elements around 4 is still unsorted so it will repeat the same process as choosing the pivot and there will be a condition whether it will pass or swap the I and J and dividing again until there is nothing to divide.

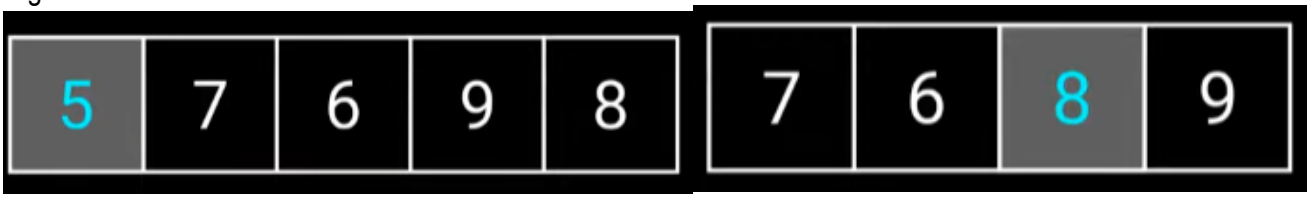
Pivot in right position:



3,2,0,1 will be sorted separately and the 8,7,6,9,5 will also do the same as the previous sorting.

Left:

Right:

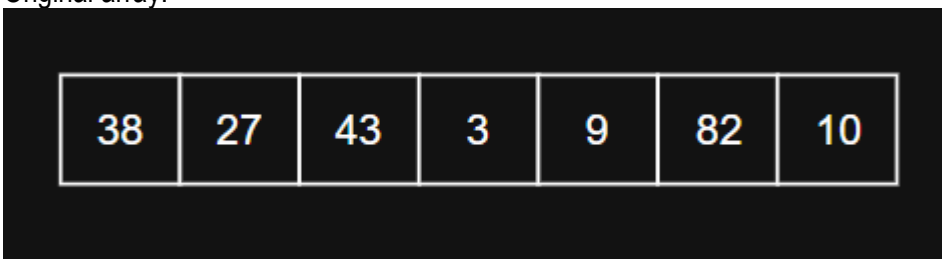


For the shell sort, it is a improve version of insertion sort. So, it is faster than insertion sort. What happens in the shell sort is it has gaps where in it breaks the elements into sub-list. Let's say I have 12,34,54,2,3. The number of the elements is 5 so we divided into 2 in order to make the gaps. 5 divided by 2 is 2.5 and we will round down this which makes it 2. The gaps tell us the positions of the elements which makes the list turn into sub-list. So, in this case, 12, 54, 3 are in a group. We operate the insertion sort so 54 and 3 are swap. So, this subgroup is now 12, 3, and 54. Next, is we apply the insertion sort for the next index which is 34 and their gap is 2 which makes the number 2 are grouped together (34 and 2). We apply insertion sort, which makes the 34 and 2 swap positions. We

put back together the list of elements which is now (12, 2, 3, 34, 54). We repeat the previous method which gap equals to 1. So, we can apply the normal insertion sort into the whole elements.

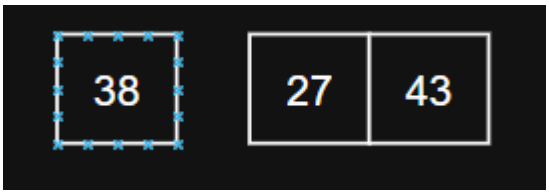
For merge sort, it follows the divide and conquer approach since it divides the array until it is small size of array and conquers and merge it until it is sorted (GeeksforGeeks, 2025b). For example:

Original array:

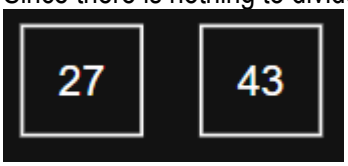


Next is we divided into half:

We divide the left hand first:

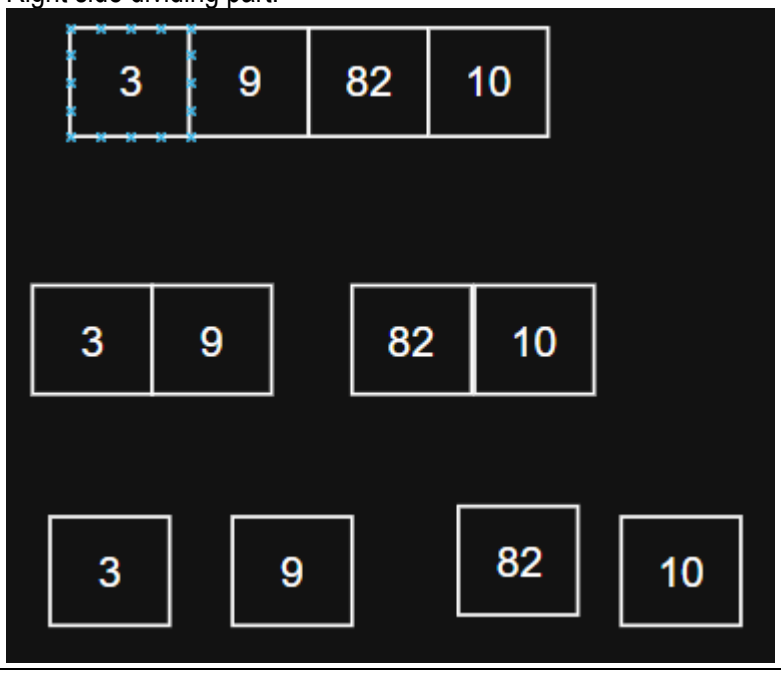


Since there is nothing to divide in the left, we divide the right side:

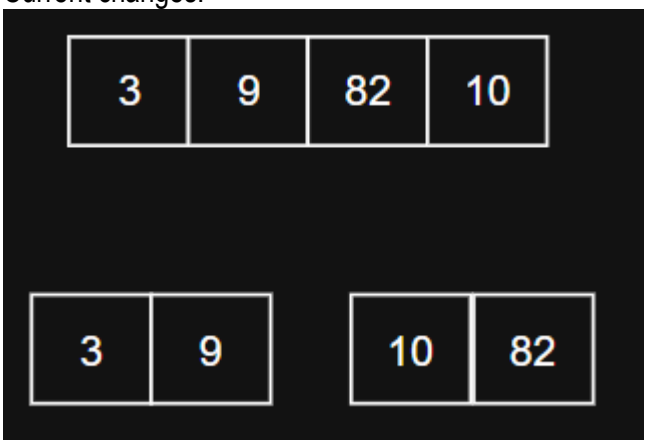


Next, is we compare the 27 and 43. Since there is nothing to change, we can already put it back and merge it. After merging it, we can compare and merge again the (38) and the (27, 43). Therefore, it is (27, 38, 43). We will apply the same method to the right side until they are merge into 4 elements again.

Right side dividing part.



Here, 3 and 9 has no changes since they are in the right order. However, the 82 and 10 should be compare which it will be arranged and merge into (10, 82).
Current changes:



Now, we can merge these 2 sub-groups by comparing per element. So, 3 and 10 are compared, which makes 3 is putted first. Next is 9 is compared to 10 which makes no changes so we can put the 9 already. Next is 10 and 82 which the 2 elements are already arranged so we can merge it. We repeat again the same steps for comparing and merging the (27, 38, 43) and the (3, 9, 10, 82).

2.) **Give simple sample programs in C++ that uses the above sorting algorithms. Use a user input of 10 integer values in your example elements in an array to be sorted. Explain how the programs work.**
Shell sort:

```cpp
#include <iostream>

void shellSort(int arr[], int n) {
    for (int gap = n/2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j-gap] > temp; j -= gap) {
                arr[j] = arr[j-gap];
            }
            arr[j] = temp;
        }
    }
}

int main() {
    int arr[10];
    std::cout << "Enter 10 integers: " << std::endl;
    for (int i = 0; i < 10; i++) std::cin >> arr[i];

    shellSort(arr, 10);

    std::cout << "Sorted array: ";
    for (int i = 0; i < 10; i++) std::cout << arr[i] << " ";
    return 0;
}
```

```
Z:\Crishen\cpp\practice.exe
Enter 10 integers:
6
5
2
9
8
1
3
4
7
10
Sorted array: 1 2 3 4 5 6 7 8 9 10
--------------------------------
Process exited after 27.44 seconds with return value 0
Press any key to continue . . .
```

Here, the outer loop indicates the gap which divides it into 5, then 2, then 1. The next loop goes through elements starting from index gap. The inner loop operates the insertion sort but with elements that is spaced apart or gap. If the previous element is bigger, shifts it forward (arr[j-gap > temp]). Then, insert the temp in its correct position.

Merge sort:

```cpp
#include <iostream>

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    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, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    int arr[10];
    std::cout << "Enter 10 integers: " << std::endl;
    for (int i = 0; i < 10; i++) std::cin >> arr[i];
```

```
Z:\Crishen\cpp\practice.exe
Enter 10 integers:
6
5
2
9
8
1
3
4
7
10
Sorted array: 1 2 3 4 5 6 7 8 9 10
--------------------------------
Process exited after 35.4 seconds with return value 0
Press any key to continue . . .
```

In the merge sort, there are two function which is for comparing and merging. Merge function splits the array into two halves which is the left-side and right-side arrays. Then, it merges them back into the main array, the two current elements is always choosing the smaller of the two. Any leftover elements from either half are copied back. The mergSort function recursively divides the array into halves until only single elements remain. In order to combine them back in sorted order, we call the merge() function. So we merge the sorted right and sorted left then merges the sorted halves.

Quick sort:

```cpp
#include <iostream>

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            std::swap(arr[i], arr[j]);
        }
    }
    std::swap(arr[i+1], arr[high]);
    return (i+1);
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    int arr[10];
    std::cout << "Enter 10 integers: " << std::endl;
    for (int i = 0; i < 10; i++) std::cin >> arr[i];

    quickSort(arr, 0, 9);

    std::cout << "Sorted array: ";
    for (int i = 0; i < 10; i++) std::cout << arr[i] << " ";
    return 0;
}
```

```
Z:\Crishen\cpp\practice.exe
Enter 10 integers:
6
5
2
9
8
1
3
4
7
10
Sorted array: 1 2 3 4 5 6 7 8 9 10
--------------------------------
Process exited after 27.29 seconds wi
Press any key to continue . . .
```

In the quick sort, there are also 2 functions which are the partition function and quicksort function. In the partition(), the pivot chooses the last element in the array. Then, it rearranges the array, which all elements that are less than the pivot is on the left and elements greater than the pivot are on the right side of the pivot. Lastly, it returns the pivot's new position. For the quicksort(), it recursively applies partitioning to left and right halves. When low is greater and equal to high, the subarray has 1 or 0 elements.
9. Assessment Rubric