ACTIVITY NO. 8

SORTING ALGORITHMS: SHELL, MERGE, AND QUICK SORT	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:** Engr. Roman M. Richard

1. Objective(s)

Create C++ code to sort through a dataset using additional sorting algorithms

2. Intended Learning Outcomes (ILOs)

After this activity, the student should be able to:

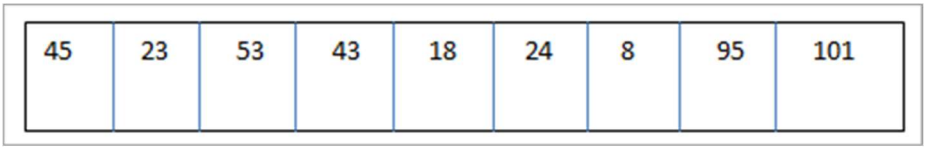
- Create C++ code for implementation of shell, merge and quick sort.
- Solve given data sorting problems using appropriate sorting algorithms

3. Discussion

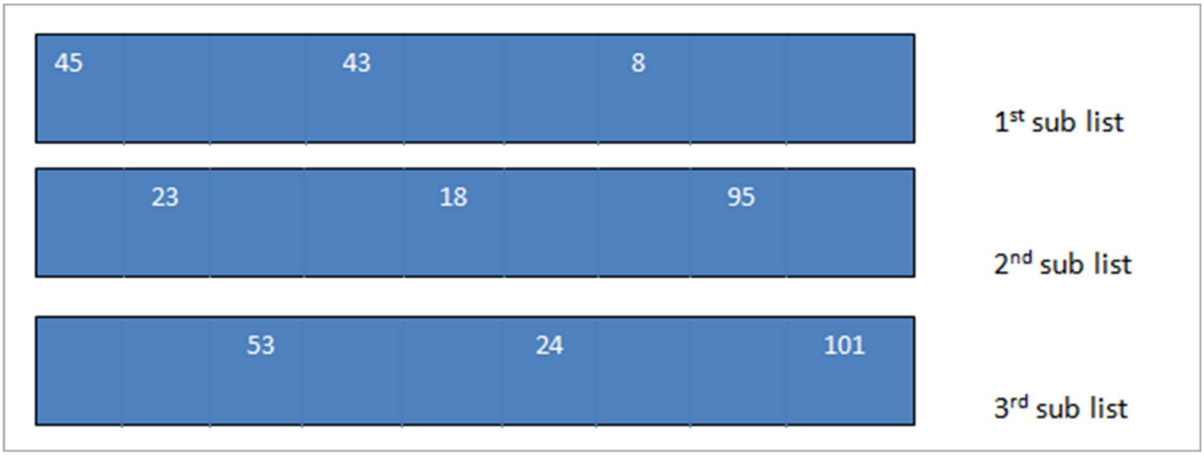
PART A: Shell Sort

Shell sort is often termed as an improvement over insertion sort. In insertion sort, we take increments by 1 to compare elements and put them in their proper position. In shell sort, the list is sorted by breaking it down into several smaller sub lists. It's not necessary that the lists need to be with contiguous elements. Instead, shell sort technique uses increment "i", which is also called "gap" and uses it to create a list of elements that are "i" elements apart.

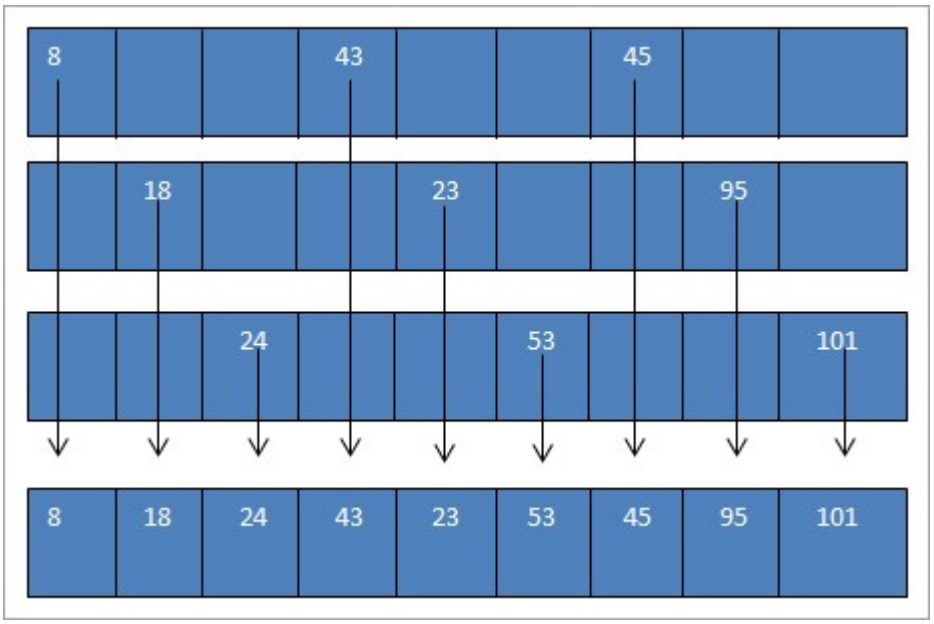
Consider the following array of 10 elements.



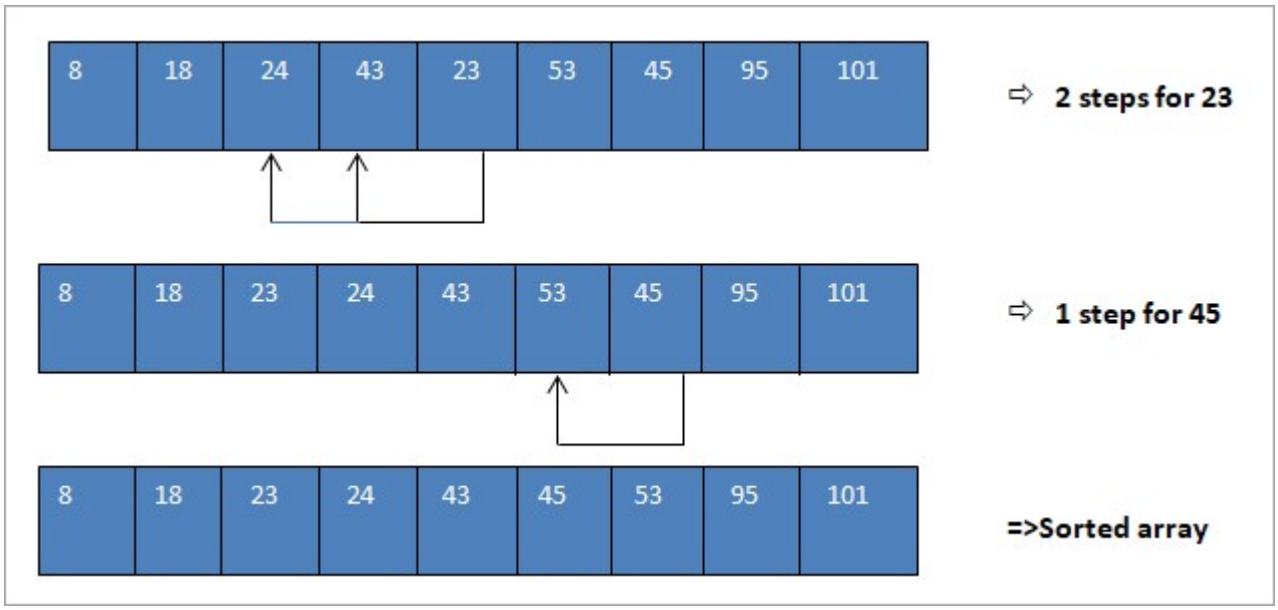
If we provide a gap of 3, then we will have the following sub-lists with each element that is 3 elements apart. We then sort these three sublists.



The sorted sub-lists and the resultant list that we obtain after combining the three sorted sublists are shown below.



The above array that we have obtained after merging the sorted subarrays is nearly sorted. Now we can perform insertion sort on this list and sort the entire array. This final step is shown below for your reference.



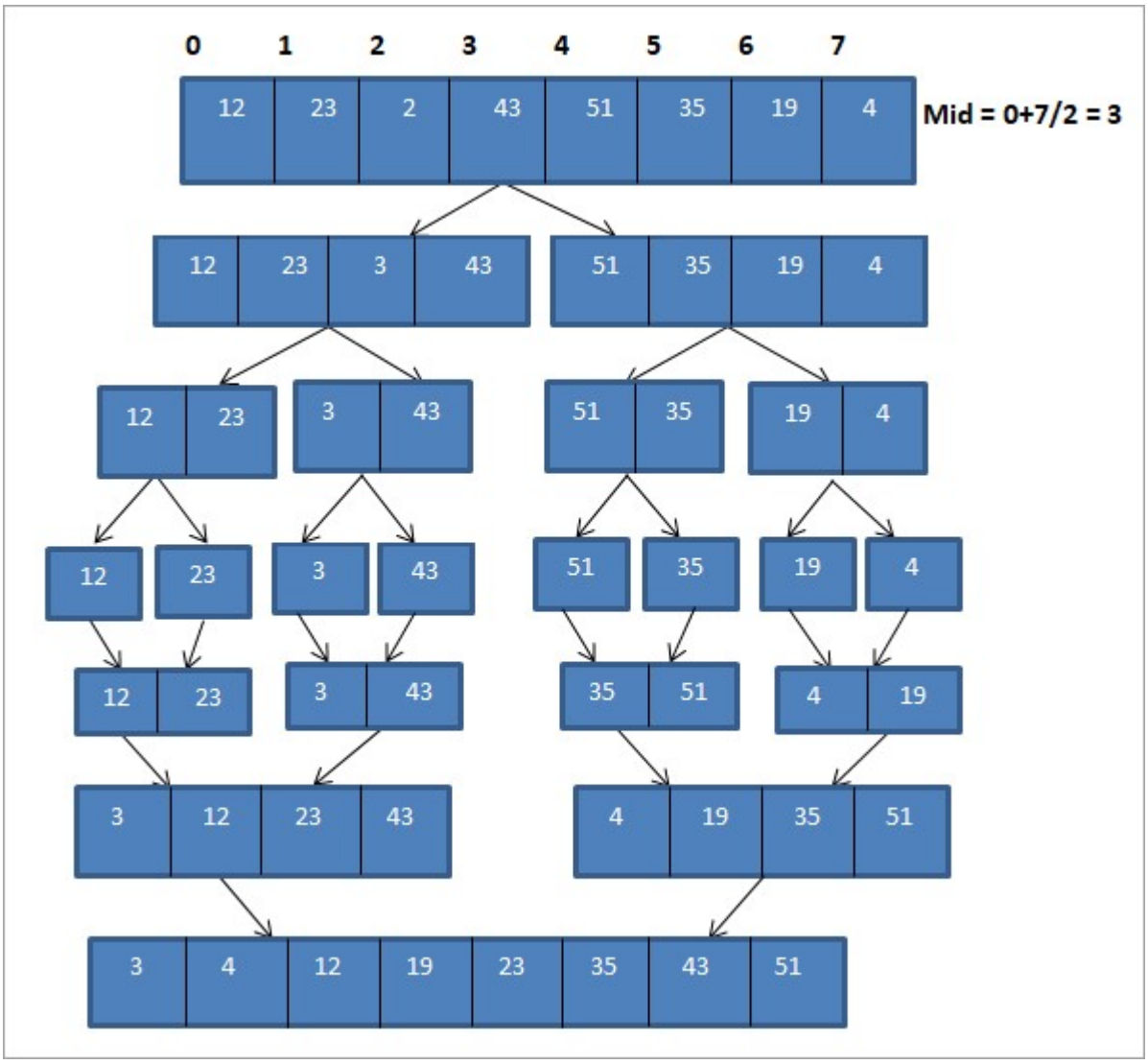
As seen above, after performing shell sort and merging the sorted sub lists, we only required three moves to completely sort the list. Thus, we can see that we can significantly reduce the number of steps required to sort the array. The choice of increment to create sub-lists is a unique feature of shell sort.

PART B: Merge Sort

Merge sort algorithm uses the "divide and conquer" strategy wherein we divide the problem into subproblems and solve those subproblems individually. These subproblems are then combined or merged to form a unified solution.

Merge sort is performed using the following steps:

1. The list to be sorted is divided into two arrays of equal length by dividing the list on the middle element. If the number of elements in the list is either 0 or 1, then the list is considered sorted.
2. Each sub list is sorted individually by using merge sort recursively.
3. The sorted sub lists are then combined or merged to form a complete sorted list.



As shown in the above representation, first the array is divided into two sub-arrays of length 4. Each sub-array is further divided into two more sub arrays of length 2. Each sub-array is then further divided into a sub-array of one element each. This entire process is the "Divide" process.

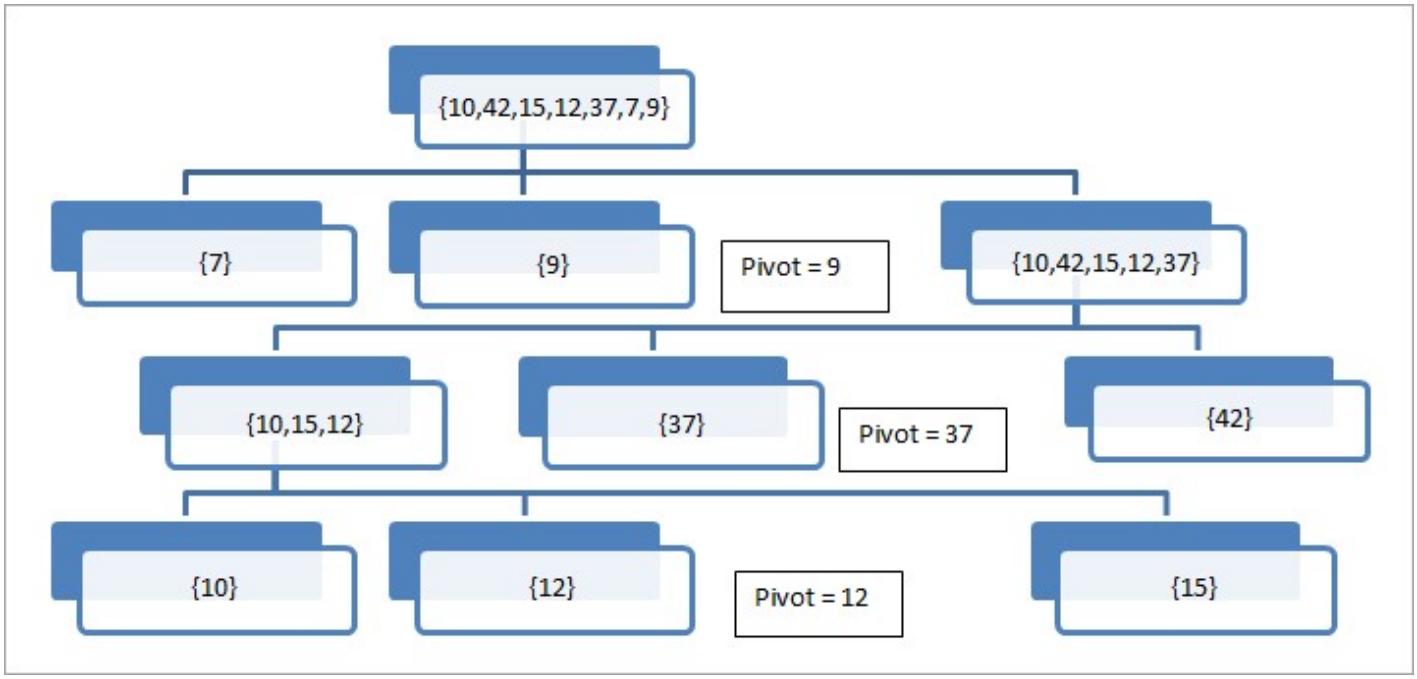
Once we have divided the array into sub-arrays of single element each, we now have to merge these arrays in sorted order.

As shown in the illustration above, we consider each subarray of a single element and first combine the elements to form sub-arrays of two elements in sorted order. Next, the sorted subarrays of length two are sorted and combined to form two sub-arrays of length four each. Then we combine these two sub-arrays to form a complete sorted array.

PART C: Quick Sort

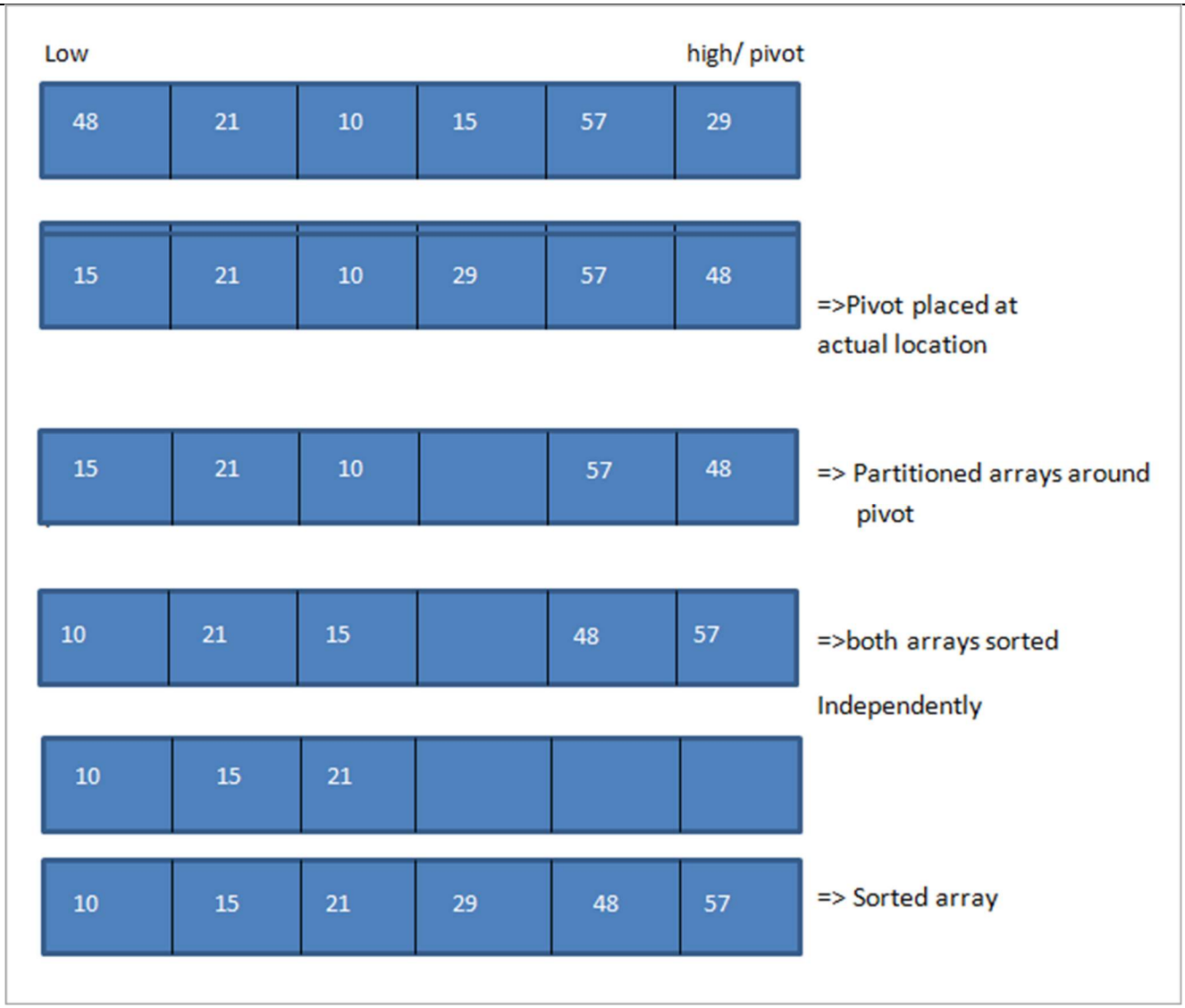
Quicksort is a widely used sorting algorithm which selects a specific element called “pivot” and partitions the array or list to be sorted into two parts based on this pivot s0 that the elements lesser than the pivot are to the left of the list and the elements greater than the pivot are to the right of the list.

Thus, the list is partitioned into two sub lists. The sub lists may not be necessary for the same size. Then Quicksort calls itself recursively to sort these two sub lists.



In this illustration, we take the last element as pivot. We can see that the array is successively divided around the pivot element until we have a single element in the array.

Now we present an illustration of the Quicksort below to better understand the concept.



From the illustration, we can see that, we move the pointers high and low at both the ends of the array. Whenever low points to the element greater than the pivot and high points to the element lesser than the pivot, then we exchange the positions of these elements and advance the low and high pointers in their respective directions.

This is done until the low and high pointers cross each other. Once they cross each other the pivot element is placed at its proper position and the array is partitioned into two. Then both these sub-arrays are sorted independently using quicksort recursively.

4. Materials and Equipment

Personal Computer with C++ IDE
Recommended IDE:

- CLion (must use TIP email to download)
- DevC++ (use the embarcadero fork or configure to C++17)

5. Procedure

ILO A: Create C++ code for implementation of shell, merge and quick sort.

Preparation Task:

- Create an array of elements with random values. The array must contain 100 elements that are not sorted. Use the created array for each sorting algorithm below. Show the output of the preparation task on table 8-1 in section 6.
- Create a header file for the implementation of the different sorting algorithms.
- Import this header file into your main.cpp.

A.1. Shell Sort

General Algorithm
The general algorithm for shell sort is given below.

```
shellSort(array, size)
  for interval i <- size/2n down to 1
    for each interval "i" in array
        sort all the elements at interval "i"
end shellSort
```

A.2. Merge Sort

General Algorithm
The general pseudo-code for the merge sort technique is given below.

```
Declare an array Arr of length N
If N=1, Arr is already sorted
If N>1,
Left = 0, right = N-1
Find middle = (left + right)/2
Call merge_sort(Arr,left,middle) =>sort first half recursively
Call merge_sort(Arr,middle+1,right) => sort second half recursively
Call merge(Arr, left, middle, right) to merge sorted arrays in above steps.
Exit
```

A.3. Quick Sort

```
quicksort(A, low, high)
begin
Declare array A[N] to be sorted
    low = 1st element; high = last element; pivot
if(low < high)
    begin
    pivot = partition (A,low,high);
    quicksort(A,low,pivot-1)
    quicksort(A,pivot+1,high)
    End
end
```

6. Output

Code + Console Screenshot	
Observations	

Table 8-1. Array of Values for Sort Algorithm Testing

Code + Console Screenshot	
Observations	

Table 8-2. Shell Sort Technique

Code + Console Screenshot	
Observations	

Table 8-3. Merge Sort Algorithm

Code + Console Screenshot	
Observations	

Table 8-4. Quick Sort Algorithm

7. Supplementary Activity

ILO B: Solve given data sorting problems using appropriate basic sorting algorithms

Problem 1: Can we sort the left sub list and right sub list from the partition method in quick sort using other sorting algorithms? Demonstrate an example.

Problem 2: Suppose we have an array which consists of {4, 34, 29, 48, 53, 87, 12, 30, 44, 25, 93, 67, 43, 19, 74}. What sorting algorithm will give you the fastest time performance? Why can merge sort and quick sort have O(N • log N) for their time complexity?

8. Conclusion

Provide the following:

- Summary of lessons learned
- Analysis of the procedure
- Analysis of the supplementary activity
- Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement?

9. Assessment Rubric