CSC4424: Analysis of Algorithm

Practical I [Sorting Algorithms]

OBJECTIVES:

1. Design and implement C++ programs in an object-oriented language demonstrating the use of the Sorting.
2. Analyze the complexity and performance of sorting algorithms.

INSTRUCTION:

1. Design and construct a complete C++ program to solve the following problem.
2. Solve the problem independently.
3. You must use abstract data type (ADT) in your solution, and you must also implement any two of the following sorting algorithms
    a. Shell sort
    b. Quick sort
    c. Radix Sort
    d. Bucket Sort
4. Each student need to prepare the following submission:
    a. Design and develop a C++ program.
    b. Analysis Report: Analyze the heuristics you generated and write a report concerning what you discovered about the two chosen sorting algorithms.

5. Use the following table, one for random data and one for the nearly ordered data. Calculate the ratio to one decimal place.

Number	Sort Algorithm 1 (SA1)	Sort Algorithm 2 (SA2)	Ratio (SA1/SA2)
Compare			
100			
500			
1000			
Moves			
100			
500			
1000			

6. Submit the softcopy solution online through **https://forms.gle/aV7mvm4HjKiWLPwC9** before/on April 10th, 2021.

PROBLEM

Write a program that sorts an array of random numbers using the two chosen algorithms. Both sorts should use same data. Each sort should be executed twice. For the first sort, fill the array with random numbers between 1 and 999, For second sort, fill the array with a nearly ordered list. Construct your nearly ordered list by reversing elements 19 and 20 in the sorted random list. For each sort, count the number of comparisons and moves necessary to order this list.

Run the program three times, once with an array of 100 items, once with an array of 500 items, and once with an array of 1000 items. For the first execution only (100 elements), print the unsorted data followed by the sort data 10x10 matrixes (10 rows of 10 numbers each). For all runs, print the number of comparison and the number of moves required to order the data.

To make sure your statistics are as accurate as possible, you must analyze each loop limit condition test and each

selection statement in your sort algorithms. The following note should help in the analysis.

a. All loops require a count increment in their body.
b. Pretest loop (while and for) also require a count increment either before (recommended) or after the loop to count the last test.
c. Remember C++ uses the shortcut rule for evaluating Boolean or expression.

The best way to count them is with comma expression, as follow. Use similar code for the selection statements.
while ((count++,a) && (count++, b))