# Department of Computing

# School of Electrical Engineering and Computer Science

**CS-250: Data Structure and Algorithms**

**Class: BSCS 10C**

**Lab 08:  Merge Sort & Quick Sort**

**Date: 11th November, 2021**

**Time: 9:00 am – 11:50 am**

# Instructor: Prof. Dr. Faisal Shafait

# Lab Engineer: Mr. Aftab Farooq

# Lab 08: Sorting Algorithms

**Lab Tasks**

**Task 1:**

A). Implement Merge Sort algorithms in C++.

|  |
| --- |
| Code |
| #include <iostream>  using **namespace** std;  **void** merge(**int** arr[], **int** first, **int** mid, **int** last)  {  **int** sub1 = mid - first + 1;  **int** sub2 = last - mid;  **int** \*lefthalf = new **int**[sub1];  **int** \*righthalf = new **int**[sub2];  for (**int** i = 0; i < sub1; i++)  {  lefthalf[i] = arr[first + i];  }  for (**int** i = 0; i < sub2; i++)  {  righthalf[i] = arr[mid + 1 + i];  }  **int** index1 = 0;  **int** index2 = 0;  **int** mergedindex = first;  while (index1 < sub1 && index2 < sub2)  {  if (lefthalf[index1] <= righthalf[index2])  {  arr[mergedindex] = lefthalf[index1];  index1++;  }  else  {  arr[mergedindex] = righthalf[index2];  index2++;  }  mergedindex++;  }  while (index1 < sub1)  {  arr[mergedindex] = lefthalf[index1];  index1++;  mergedindex++;  }  while (index2 < sub2)  {  arr[mergedindex] = righthalf[index2];  index2++;  mergedindex++;  }  }  **void** mergeSort(**int** arr[], **int** first, **int** last)  {  if (first >= last)  {  return;  }  **int** mid = (first + last) / 2;  mergeSort(arr, first, mid);  mergeSort(arr, mid + 1, last);  merge(arr, first, mid, last);  }  **void** createArray(**int** arr[], **int** len)  {  srand(time(NULL));  for (**int** i = 0; i < len; i++)  {  arr[i] = (rand() % 100) + 1;  }  }  **int** main()  {  **int** arr[10];  createArray(arr, 10);  for (**int** i = 0; i < 10; i++)  {  cout << arr[i] << " ";  }  cout << endl;  cout << endl;  mergeSort(arr, 0, 9);  for (**int** i = 0; i < 10; i++)  {  cout << arr[i] << " ";  }  cout << endl;  cout << endl;  } |
| Output |
| Graphical user interface, text, application, chat or text message  Description automatically generated |

B). Implement the quick sort algorithm as per pseudo code give above using C++.

**int** partition(**int** arr[], **int** lower\_index, **int** higher\_index)

{

**int** pivot = arr[higher\_index];

**int** i = (lower\_index - 1);

for (**int** j = lower\_index; j <= higher\_index - 1; j++)

{

if (arr[j] < pivot)

{

i++;

swap(arr[i], arr[j]);

}

}

swap(arr[i + 1], arr[higher\_index]);

return (i + 1);

}

**void** quickSort(**int** arr[], **int** lower\_index, **int** higher\_index)

{

if (lower\_index < higher\_index)

{

**int** part\_int = partition(arr, lower\_index, higher\_index);

quickSort(arr, lower\_index, part\_int - 1);

quickSort(arr, part\_int + 1, higher\_index);

}

}

**Task 2: (average case complexity):**

A). The next step is to compute the time complexity of the Merge sort algorithm. Generate arrays of random numbers in the range 1 to 100 with sizes 100, 1000, 10000, 100000, and 1000000. Compare the running times of the algorithm on each array. Are the

100:

Text

Description automatically generated

1000:

Text

Description automatically generated

10000:

Text

Description automatically generated

100000:

Text

Description automatically generated

1000000:

Text

Description automatically generated

results what you expected, and why? Answer the questions in the solution section.   
 B). Implement same task by using Quick Sort.

100:

A screenshot of a computer

Description automatically generated with medium confidence

1000:

Text

Description automatically generated

10000:

Text

Description automatically generated

100000:

Text

Description automatically generated

1000000:

Text

Description automatically generated

**Task 3: (best and worst case complexity):**

A). Now sort the arrays using stl::sort, once in ascending order and then in descending order. Given both sorted arrays as inputs to the algorithm and compute their running time. Does the running time of algorithm shows variations based on the structure of the input and why? Plot the running time of the best and worst case complexities for different input sizes in a excel sheet and add it in the solution section.  
 B). Implement same task by using Quick Sort.

Graphical user interface, line chart

Description automatically generated with medium confidence  
**Important Note:** Practice your knowledge of OOP with C++ when creating a solution.

**Lab Grading:**

|  |  |
| --- | --- |
| **Task** | **Marks** |
| Lab Viva/Quiz | 5 |
| Comments/ Indentation | 2 |
| Solution Document | 2 |
| Output Screen Shots | 1 |
| -- | -- |
| Total | 10 |

**Deliverables**

This lab grading policy is as follows: The lab is graded between 0 to 10 marks. The submitted solution can get a maximum of 5 marks. At the end of each lab or in the next lab, there will be a viva related to the tasks. The viva has a weightage of 5 marks. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS. In case of any problems discuss it by emailing it to [aftab.farooq@seecs.edu.pk](mailto:aftab.farooq@seecs.edu.pk).

**Note:** Students are required to upload the lab on LMS before deadline.

Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks.