G. H. RAISONI COLLEGE OF ENGG., NAGPUR

(An Autonomous Institute under UGC Act 1956)

Department of Artificial Intelligence

Date: 27/07/2020

Practical Subject: Data Structures and Algorithms
Session: 2020-21

Student Details:

Roll Number	58
Name	Shivam Tawari
Semester	3
Section	A
Branch	Artificial Intelligence

Practical Details: Practical Number- 3

Practical Aim	Design, develop and implement a menu driven program in C++ for implementing the following sorting methods to arrange a list of integers in ascending order: a) Insertion sort b) Merge sort c) Quick sort d) Heap sort
Theory	**Insertion Sort:** Insertion sort is the sorting mechanism where the sorted array is built having one item at a time. The array elements are compared with each other sequentially and then arranged simultaneously in some particular order. The analogy can be understood from the style we arrange a deck of cards. This sort works on the principle of inserting an element at a particular position, hence the name Insertion Sort. **Merge Sort:** In Merge sort the idea is to split the unsorted list into smaller groups until there is only one element in a group. Then, group two elements in the sorted order and gradually build the size of the group. Every time the

	merging happens, the elements in the groups must be compared one by one and combined into a single list in the sorted order. This process continues till all the elements are merged and sorted. **Quick Sort:** Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value. **Heap Sort:** Heap sort is performed on the heap data structure. We know that heap is a complete binary tree. Heap tree can be of two types. Min-heap or max heap. For min heap the root element is minimum and for max heap the root is maximum. After forming a heap, we can delete an element from the root and send the last element to the root. After these swapping procedures, we need to re-heap the whole array. By deleting elements from root, we can sort the whole array.
Procedure	1. User enters the size of array. 2. Then it enters the integers in the array. 3. We have 4 sorting algorithms; Insertion sort, merge sort, quick sort, heap sort. 4. After selecting the sorting algorithm, it sorts and displays the sorted array. 5. Then we ask the user if it again wants to run the program.
Algorithm	**Insertion Sort:** Step 1: START Step 2: Consider the first element to be sorted and the rest to be unsorted Step 3: Compare with the second element: 1. If the second element < the first element, insert the element in the correct position of the sorted portion 2. Else, leave it as it is Step 4: Repeat 2 and 3 until all elements are sorted. Step 5: STOP **Merge Sort:**

	Step 1: START Step 2: Split the unsorted list into groups recursively until there is one element per group Step 3: Compare each of the elements and then group them Step 4: Repeat step 3 until the whole list is merged and sorted in the process Step 5: STOP **Quick Sort:** Step 1: START Step 2: Choose the highest index value has pivot Step 3: Take two variables to point left and right of the list excluding pivot Step 4: left points to the low index Step 5: right points to the high Step 6: while value at left is less than pivot move right Step 7: while value at right is greater than pivot move left Step 8: if both step 5 and step 6 does not match swap left and right Step 9: if left ≥ right, the point where they met is new pivot Step 10: STOP **Heap Sort:** Step 1: START Step 2: Construct a Binary Tree with given list of Elements. Step 3: Transform the Binary Tree into Min Heap. Step 4: Delete the root element from Min Heap using Heapify method. Step 5: Put the deleted element into the Sorted list. Step 6: Repeat the same until Min Heap becomes empty. Step 7: Display the sorted list. Step 8: STOP
Program	

main.cpp

Run

```cpp
#include <iostream>

using namespace std;

void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

void insertion(int arr[], int siz)
{
    int temp, i, j;
    for(i=1; i<siz; i++)    {
        temp = arr[i];
        j = i;
        while(j>0 && temp<arr[j-1]) {
            arr[j] = arr[j-1];
            j -= 1;
        }
        arr[j] = temp;
    }
}

void merge(int A[] , int start, int mid, int end)
```

```
{
    int p=start, q=mid+1;
    int Arr[end-start+1], k=0;

    for(int i=start; i<=end; i++)   {
        if(p > mid)
            Arr[k++] = A[q++];
        else if(q > end)
            Arr[k++] = A[p++];
        else if(A[p] < A[q])
            Arr[k++] = A[p++];
        else
            Arr[k++] = A[q++];
    }
    for(int p=0; p<k; p++) {
        A[start++] = Arr[p];
   }
}
void mergeSort(int A[], int start, int end)
{
    if(start < end) {
    int mid = (start+end)/2 ;
    mergeSort(A, start, mid);
    mergeSort(A, mid+1, end);
    merge(A, start, mid, end);
    }
}

```

```
55  int partition (int arr[], int low, int high)
56  {
57      int pivot = arr[high];
58      int i = (low - 1);
59
60      for (int j = low; j <= high - 1; j++)   {
61          if (arr[j] < pivot)
62          {
63              i++;
64              swap(&arr[i], &arr[j]);
65          }
66      }
67      swap(&arr[i + 1], &arr[high]);
68      return (i + 1);
69  }
70  void quickSort(int arr[], int low, int high)
71  {
72      if (low < high)  {
73          int pi = partition(arr, low, high);
74          quickSort(arr, low, pi - 1);
75          quickSort(arr, pi + 1, high);
76      }
77  }
78
79  void heapify(int arr[], int n, int i)
80  {
81      int largest = i;
82      int l = 2*i + 1;
```

```
    int r = 2*i + 2;

    if (l < n && arr[l] > arr[largest])
        largest = l;

    if (r < n && arr[r] > arr[largest])
        largest = r;

    if (largest != i)   {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n)
{
    for (int i = n/2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for (int i=n-1; i>0; i--)   {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

void display(int arr[], int n)
{
    for (int i=0; i<n; ++i)
        cout << arr[i] << " ";
```

```
111     cout << "\n";
112 }
113
114 int main()
115 {
116     int arr[100], n, ch;
117     char ext;
118     cout << "\n\tName: Shivam Tawari (A - 58)";
119     do {
120         cout << "\n Enter the size of Array: ";
121         cin >> n;
122         cout << "\n Enter the Array: ";
123         for(int i=0; i<n; i++)  {
124             cin >> arr[i];
125         }
126         cout << "\n\n -----MAIN_MENU-----";
127         cout << "\n 1. Insertion Sort";
128         cout << "\n 2. Merge Sort";
129         cout << "\n 3. Quick Sort";
130         cout << "\n 4. Heap Sort";
131         cout << "\n Enter your choice: ";
132         cin >> ch;
133         switch(ch)  {
134             case 1: insertion(arr, n);
135                     break;
136             case 2: mergeSort(arr, 0, n-1);
137                     break;
138             case 3: quickSort(arr, 0, n-1);
139                     break;
140             case 4: heapSort(arr, n);
141                     break;
142             case 5:
143             default: cout << "\n Err!!! Wrong Choice";
144                     break;
145         }
146         cout << "\n Sorted array is: ";
147         display(arr, n);
148         cout << "\n Do you want to enter again?(Y/N): ";
149         cin >> ext;
150     } while(ext != 'N');
151 }
```

Output

Insertion Sort:

Output Clear

```
g++ -o /tmp/xiAMJDejTy.o /tmp/xiAMJDejTy.cpp
/tmp/xiAMJDejTy.o
Name: Shivam Tawari (A - 58)
 Enter the size of Array: 3
 Enter the Array: 6
 9
 5
 -----MAIN_MENU-----
 1. Insertion Sort
 2. Merge Sort
 3. Quick Sort
 4. Heap Sort
 Enter your choice: 1
 Sorted array is: 5 6 9

 Do you want to enter again?(Y/N):
```

Merge Sort:

```
 Enter the size of Array: 4
 Enter the Array: 2
 90
 22
 4
 -----MAIN_MENU-----
 1. Insertion Sort
 2. Merge Sort
 3. Quick Sort
 4. Heap Sort
 Enter your choice: 2
 Sorted array is: 2 4 22 90
```

Quick Sort:

	`` Enter the size of Array: 5 Enter the Array: 3 5 0 44 58 -----MAIN_MENU----- 1. Insertion Sort 2. Merge Sort 3. Quick Sort 4. Heap Sort Enter your choice: 3 Sorted array is: 0 3 5 44 58 ``` **Heap Sort:** ``` Enter the size of Array: 3 Enter the Array: 58 11 49 -----MAIN_MENU----- 1. Insertion Sort 2. Merge Sort 3. Quick Sort 4. Heap Sort Enter your choice: 4 Sorted array is: 11 49 58 ```
Conclusion	Hence, successfully implemented a menu driven program for Insertion sort, Merge sort, Quick sort and Heap sort in a C++.