**Experiment No. : 2**

**Title: Divide and Conquer Strategy**

**Batch: B2 Roll No.: 16010421119 Experiment No.: 2**

**Aim:** To implement and analyse time complexity of Quick-sort and Merge sort and compare both.

1. **Quick Sort----**

**Explaination:- Quick sort is a sorting algorithm that uses the divide and conquer approach to sort an array of elements. It chooses a pivot element from the array and partitions the array into two sub-arrays, one containing elements less than the pivot, and the other containing elements greater than the pivot. It then recursively applies the partitioning step to each sub-array until the entire array is sorted.**

1. **Merge Sort----**

**Explaination: - Merge sort is a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.**

**In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.**

**One thing that you might wonder is what is the specialty of this algorithm. We already have a number of sorting algorithms then why do we need this algorithm? One of the main advantages of merge sort is that it has a time complexity of O(n log n), which means it can sort large arrays relatively quickly. It is also a stable sort, which means that the order of elements with equal values is preserved during the sort.**

**Merge sort is a popular choice for sorting large datasets because it is relatively efficient and easy to implement. It is often used in conjunction with other algorithms, such as quicksort, to improve the overall performance of a sorting routine..**

**Algorithm of Quick sort & Merge sort:**

1. **Quick Sort**

**Algorithm:-**

* **Choose a pivot element from the array. This element will be used to divide the array into two parts.**
* **Partition the array by rearranging the elements such that all elements smaller than the pivot are on the left of the pivot, and all elements greater than the pivot are on the right of the pivot.**
* **Recursively apply the above steps to the sub-arrays on the left and right of the pivot until the entire array is sorted.**
* **When all elements have been processed, the list is sorted.**

1. **Merge Sort**

**Algorithm:-**

**step 1: start**

**step 2: declare array and left, right, mid variable**

**step 3: perform merge function.  
    if left > right  
        return  
    mid= (left+right)/2  
    mergesort(array, left, mid)  
    mergesort(array, mid+1, right)  
    merge(array, left, mid, right)**

**step 4: Stop**

**Derivation of Analysis** **Quick sort & Merge sort:**

Time complexity of Merge Sort is O(n\*Log n) in all the 3 cases (worst, average and best) as merge sort always divides the array in two halves and takes linear time to merge two halves.

It requires equal amount of additional space as the unsorted array. Hence its not at all recommended for searching large unsorted arrays.

**Program(s) of Quick sort & Merge sort:**

Code: - **Quick Sort**

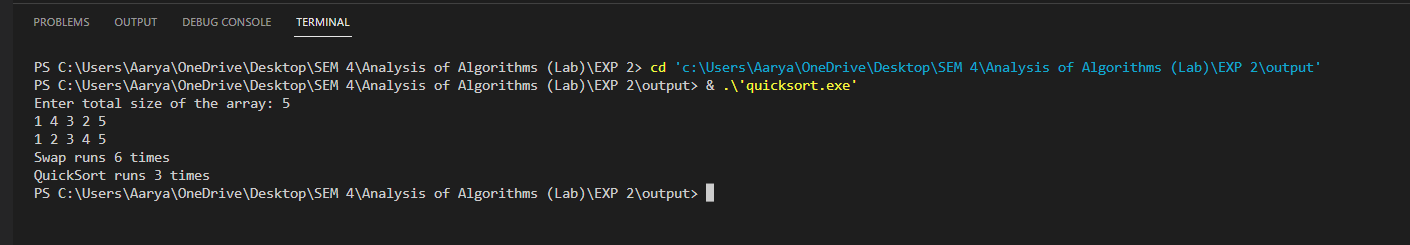
// Online C++ compiler to run C++ program online  
#include <iostream>  
using namespace std;  
int count1 = 0;  
int count2 = 0;  
int count3 = 0;  
void swapnumber(int \*a,int \*b)  
{  
    int x = \*a;  
    \*a = \*b;  
    \*b = x;  
}  
void print(int arr[],int size)  
{  
    for(int i=0;i<size;i++)  
    {  
        cout<<arr[i]<<" ";  
    }  
     
}  
int partition(int arr[] , int left , int right)  
{  
    int pivot = arr[right];  
    int i = left-1;  
     
    for(int j = left ; j<right ;j++)  
    {  
        if(arr[j] <= pivot)  
        {  
            i++;  
            swapnumber(&arr[i],&arr[j]);  
            count1++;  
        }  
    }  
    swapnumber(&arr[i+1],&arr[right]);  
    return i+1;  
     
}  
void quickSort(int arr[] ,  int left , int right)  
{  
    if(left < right)  
    {  
        int x = partition(arr,left,right);  
        quickSort(arr, left, x - 1);    
        quickSort(arr, x + 1, right);    
        count2++;  
    }  
}  
  
  
  
  
int main() {  
    int n;  
    cout<<"Enter total size of the array: ";  
    cin>>n;  
    int arr[n];  
    for(int i = 0;i<n;i++)  
    {  
        cin>>arr[i];  
    }  
    quickSort(arr,0,n-1);  
    print(arr,n);  
    cout<<endl;  
    cout<<"Swap runs "<<count1<<" times"<<endl;  
    cout<<"QuickSort runs "<<count2<<" times"<<endl;  
    return 0;  
}

**Merge Sort: -**

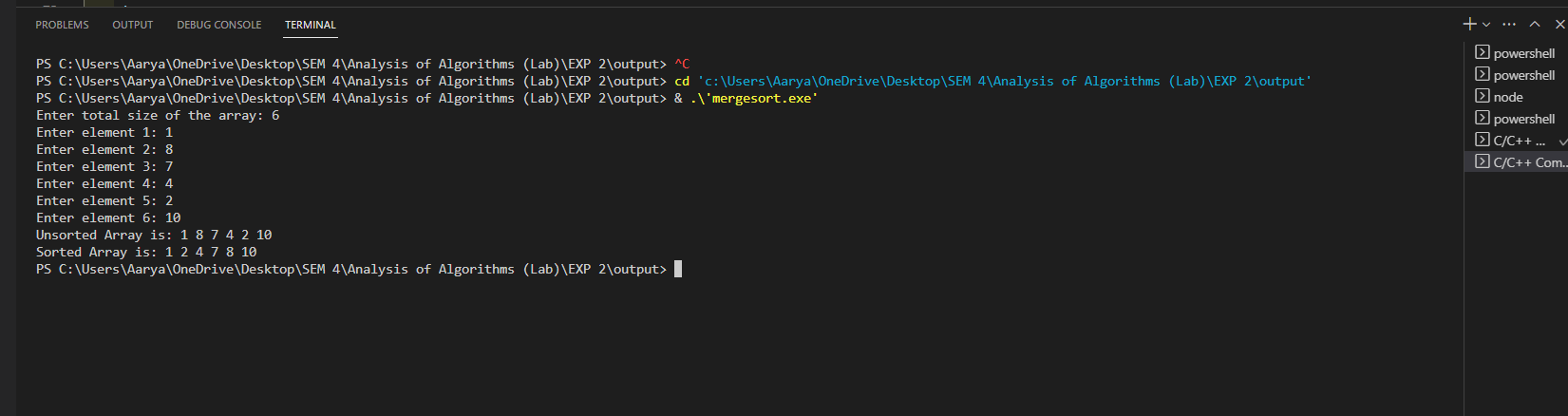
#include <iostream>  
using namespace std;  
void printarr(int arr[],int size)  
{  
    for(int i=0;i<size;i++)  
    {  
        cout<<arr[i]<<" ";  
    }  
}  
void merge(int arr[],int left, int mid,int right)  
{  
    int size1 = mid-left+1;  
    int size2 = right - mid;  
    int array1[size1];  
    int array2[size2];  
    for(int i=0;i<size1;i++)  
    {  
        array1[i] = arr[left+i];  
    }  
    for(int j = 0;j<size2;j++)  
    {  
        array2[j] = arr[mid+1+j];  
    }  
    int a,b,c;  
    a = 0;  
    b = 0;  
    c = left;  
    while(a<size1 && b<size2)  
    {  
        if(array1[a]<=array2[b])  
        {  
            arr[c] = array1[a];  
            a++;  
        }  
        else  
        {  
            arr[c] = array2[b];  
            b++;  
        }  
        c++;  
    }  
    while(a<size1)  
    {  
        arr[c] = array1[a];  
        a++;  
        c++;  
    }  
    while(b<size2)  
    {  
        arr[c] = array2[b];  
        b++;  
        c++;  
    }  
}  
void mergeSort(int arr[] , int left , int right)  
{  
    if(left < right)  
    {  
        // int mid = left + (right-left)/2;  
        int mid = (left + right)/2;  
         
        mergeSort(arr,left,mid);  
        mergeSort(arr,mid+1,right);  
         
        merge(arr,left,mid,right);  
    }  
}  
int main() {  
    int n;  
    cout<<"Enter total size of the array: ";  
    cin>>n;  
    int arr[n];  
    for(int i = 0;i<n;i++)  
    {  
        cout<<"Enter element "<<i+1<<": ";  
        cin>>arr[i];  
    }  
    cout<<"Unsorted Array is: ";  
    printarr(arr,n);  
    cout<<endl;  
    cout<<"Sorted Array is: ";  
    mergeSort(arr,0,n-1);  
    printarr(arr,n);  
    return 0;  
}

**Output(o) of Quick sort & Merge sort:**

**Quick Sort :-**



**Merge Sort: -**



**Results:**

**Time Complexity of Quick sort:**

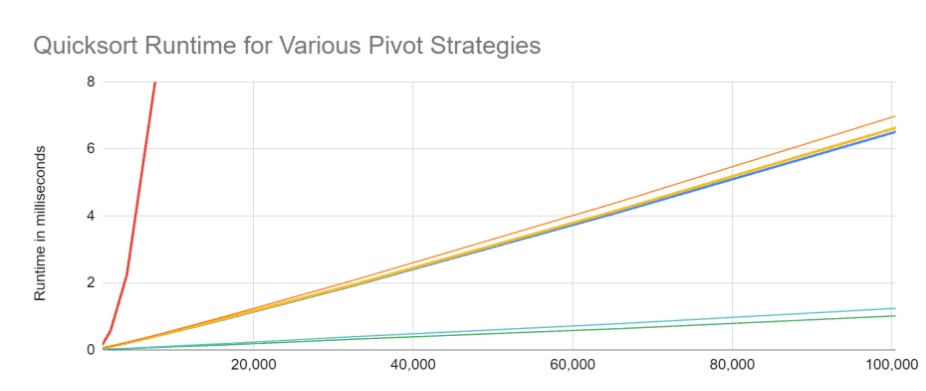
**Worst Case Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Input size** | **No: of steps from Algorithm analysis** | **No: of steps from Theoretical analysis** |
| 1. | 5 | 13 | 13 |
| 2. | 10 | 28 | 28 |
| 3. | 20 | 59 | 59 |
| 4. | 30 | 105 | 105 |

**Best Case Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Input size** | **No: of steps from Algorithm analysis** | **No: of steps from Theoretical analysis** |
| 1. | 5 | 4 | 4 |
| 2. | 10 | 17 | 17 |
| 3. | 20 | 28 | 28 |
| 4. | 30 | 58 | 58 |

**GRAPH**



**Time Complexity of Merge sort:**

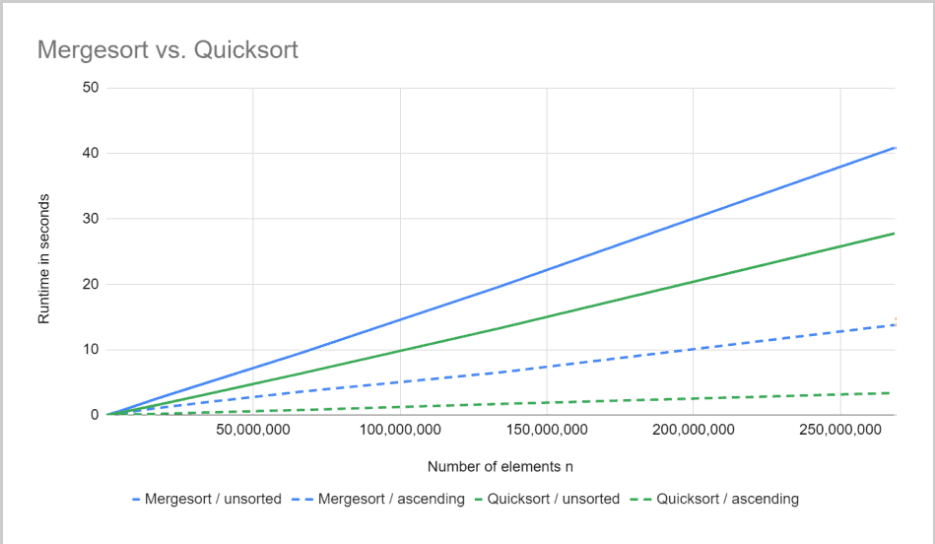
**Worst Case Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Input size** | **No: of steps from Algorithm analysis** | **No: of steps from Theoretical analysis** |
| 1. | 5 | 13 | 13 |
| 2. | 10 | 28 | 28 |
| 3. | 20 | 59 | 59 |
| 4. | 30 | 105 | 105 |

**Best Case Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Input size** | **No: of steps from Algorithm analysis** | **No: of steps from Theoretical analysis** |
| 1. | 5 | 4 | 4 |
| 2. | 10 | 17 | 17 |
| 3. | 20 | 28 | 28 |
| 4. | 30 | 58 | 58 |

**GRAPH**



**Conclusion: (Based on the observations):**

**We can conclude that we have learnt about Quick Sort and Merge Sort.**

**Outcome:**

**CO 1. Analyze time and space complexity of basic algorithms**

**References:**

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.