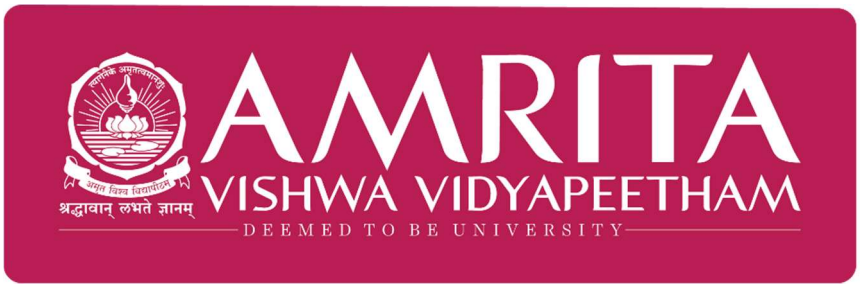

DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK

WEEK – 4

NAME : **ROHITH S**

ROLL NUMBER : **CH.SC.U4CSE24140**

CLASS : **CSE-B**

Question 1: Write a C program that reads N integers from the user and sorts it using Merge sort

CODE:

mergesort.c > main(void)

```c
#include <stdio.h>
static void ms_merge(int*a,int*tmp,int l,int m,int r)
{
    int i=l;
    int j=m;
    int k=l;
    while(i<m&&j<r)
    {
        if(a[i]<=a[j])
            tmp[k++]=a[i++];
        else
            tmp[k++]=a[j++];
    }
    while(i<m)
        tmp[k++]=a[i++];
    while(j<r)
        tmp[k++]=a[j++];
    for(i=l;i<r;i++)
        a[i]=tmp[i];
}
static void ms_run(int*a,int*tmp,int n,int sz)
{
    if(sz>=n)
        return;
    for(int s=0;s<n;s+=2*sz)
    {
        int m=s+sz;
        int e=s+2*sz;
        if(m>n)
            m=n;
        if(e>n)
            e=n;
        if(m<e)
            ms_merge(a,tmp,s,m,e);
    }
    ms_run(a,tmp,n,sz*2);
}
```

```c
void ms(int*a,int n,int buf[])
{
    ms_run(a,buf,n,1);
}
int main(void)
{
    int a[]={157,110,147,122,111,149,151,141,123,112,117,133};
    int n=sizeof(a)/sizeof(a[0]);
    int buf[n];
    ms(a,n,buf);
    for(int i=0;i<n;i++)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```

OUTPUT:

```
110 111 112 117 122 123 133 141 147 149 151 157
PS C:\Users\123sr\OneDrive\Desktop\COLLEGE STUFF\SEM IV\DAA LAB\Week 4>
```

Space Complexity: O(N)

Time Complexity: O(N log N)

Justification:

- The array is divided into two equal halves at each step, and the merge operation compares all n elements. Since this division continues for log n levels, the overall Time Complexity becomes O(N log N).
- An extra array is used during the merging process to temporarily store elements. This additional storage requires space proportional to the number of elements, resulting in O(n) Space Complexity.

Question 2:

Create a C program that implements Quick Sort to sort an array of integers.

CODE:

```c
#include <stdio.h>
static int part(int*a,int l,int r)
{
    int p=a[r];
    int i=l-1;
    for(int j=l;j<r;j++)
    {
        if(a[j]<=p)
        {
            i++;
            int t=a[i];
            a[i]=a[j];
            a[j]=t;
        }
    }
    int t=a[i+1];
    a[i+1]=a[r];
    a[r]=t;
    return i+1;
}
static void qs_run(int*a,int l,int r)
{
    if(l<r)
    {
        int p=part(a,l,r);
        qs_run(a,l,p-1);
        qs_run(a,p+1,r);
    }
}
void qs(int*a,int n)
{
    qs_run(a,0,n-1);
}
int main(void)
{
    int a[]={157,110,147,122,111,149,151,141,123,112,117,133};
    int n=sizeof(a)/sizeof(a[0]);
```

```
38      qs(a,n);
39      for(int i=0;i<n;i++)
40          printf("%d ",a[i]);
41      printf("\n");
42      return 0;
43  }
```

OUTPUT:

```
110 111 112 117 122 123 133 141 147 149 151 157
PS C:\Users\123sr\OneDrive\Desktop\COLLEGE STUFF\SEM IV\DAA LAB\Week 4>
```

Space Complexity: O(N)

Time Complexity: O(N^2)

Justification:

- In the worst case, Quick Sort makes highly unbalanced recursive calls, reducing the problem size by only one element at each step. As a result, the recursion depth becomes n, and the recursion stack stores up to n function calls, leading to O(N) Space Complexity.

- This occurs when the pivot divides the array into highly unbalanced parts, such as in already sorted or reverse sorted array. Hence the Time Complexity is O(N^2).