Sherwyn Fernandes
SE COMP-B
191105063

**PRACTICAL EXAM**

**Date:- 25-08-2021**

**Programs to be implemented:**

**A] Write a C/C++ program to implement merge sort algorithm using divide & conquer and estimate its time and space complexity**

**B] Write a C/C++ program to implement master's theorem to solve recurrence relations(code all 3 conditions) and estimate its time and space complexity**

## PROGRAM [A] IMPLEMENTATION:

```
#include<iostream>
using namespace std;

void mergesort(int,int);
void merge(int,int,int);
int count=0;
int *a,*b;
int main()
{
	int n;
	cout<<"Enter number of elements\n";
	count++;
	cin>>n;
	count++;
	a = new int[n];
	b = new int[n];
```

Sherwyn Fernandes
SE COMP-B
191105063

```
        cout<<"Enter "<<n<<" elements to be sorted\n";
        count++;
        for(int i=0;i<n;i++)
            {
                count++;//for
                cin>>a[i];
                count++;
            }

        mergesort(0,n-1);
        for(int i=0;i<n;i++)
            cout<<a[i]<<" ";

        cout<<endl;
        cout<<"Count="<<count<<endl;
        return 0;
    }

    void mergesort(int low, int high)
    {
        count++;
        if(low<high)
        {

        int mid = (low+high)/2;
        count++;
        mergesort(low,mid);
        mergesort(mid+1,high);
        merge(low,mid,high);
        }

    }

    void merge(int low, int mid, int high)
```

Sherwyn Fernandes
SE COMP-B
191105063

```
	{
		int i=low,j=mid+1,k=low;

		while(i<=mid && j<=high)
		{
			count++;  //while
			count++; //if
			if(a[i]<a[j])
				{
					count+=3;
					b[k++] = a[i++];
				}

			else
				{
					b[k++] = a[j++];
					count+=3;
				}
		}
		count++; //while


		while(i<=mid)
			{
				count++;      //while statement
				count++;
				b[k++] = a[i++];
			}
		count++;      //Last while


		while(j<=high)
		{
			count++;      //while statement
```

Sherwyn Fernandes
SE COMP-B
191105063

```
                count++; //for
                b[k++] = a[j++];
        }
        count++;      //Last while

        for(k=low;k<=high;k++)
        {
                count++; //for
                a[k]=b[k];
                count++;
        }
}
```

OUTPUT:

1. When elements are already sorted

   **Count=238**

```
C:\WINDOWS\SYSTEM32\cmd.exe
Enter number of elements
10
Enter 10 elements to be sorted
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
Count=238
Press any key to continue . . .
```

2. When elements are in random order

   **Count=240**

```
C:\WINDOWS\SYSTEM32\cmd.exe
Enter number of elements
10
Enter 10 elements to be sorted
33 69 44 12 10 2 29 5 3 1
1 2 3 5 10 12 29 33 44 69
Count=240
Press any key to continue . . .
```

Sherwyn Fernandes
SE COMP-B
191105063

3. When n=20 and elements are in sorted order

**Count=582**

```
C:\WINDOWS\SYSTEM32\cmd.exe
Enter number of elements
20
Enter 20 elements to be sorted
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Count=582
Press any key to continue . . .
```

## Time Complexity:

Merge sort uses the technique of Divide and Conquer. Hence, each time the problem is divided into equal halves and once it is broken to atomic level, we start arranging them either in Ascending or Descending order based on our requirement.

As the array is divided until there's only one element, the n value will decrease by a factor of 2. Hence, the recurrence relation for this can be written as:

$$T(n) = 2T(n/2) + n$$

Hence, by using master's theorem, the time complexity of the Recurrence Relation is computed to be O(n*logn).

**Therefore, Time complexity of MergeSort Algorithm is O(n*logn)**

Sherwyn Fernandes
SE COMP-B
191105063

## Space complexity:

MergeSort requires space for the auxiliary array as well as the main array. Hence total space required is 2*n.

### Therefore, Space Complexity of MergeSort is of the order of O(n)

Sherwyn Fernandes
SE COMP-B
191105063

## PROGRAM [B] IMPLEMENTATION:

```
#include<iostream>

#include<math.h>


int count=0;

using namespace std;


float compute_UN( float lg, float n)

{

        count++;    //for if else

        if(n>lg)

                {

                        n-=lg;         //for case O(n^r) r>0

                        count++;

                }


        else if((n==0 && n!=lg)  || (n<lg && n!=0))

                {

                        n=-1; //for case O(1)

                        //here h(n) = 1/n^r, hence r<0 or     (n)=n^x/n^y where y>x

                        count+=2;

                }
```

Sherwyn Fernandes
SE COMP-B
191105063

```
        else if(n==lg)
                {
                        n=0;            //case when h(n)=1 hence r=0;
                                        //ie i>=0 so logn^(i+1)/i+1
                        count+=2;
                }
        count++;
        return n;

}



float log_a_base_b(float a, float b)
{
        count++;
        return log(a)/log(b);
}
```

Sherwyn Fernandes
SE COMP-B
191105063

```
void compute_RR( float n, float a, float b)

{

    float lg;     //this will be log value

    lg = log_a_base_b(a,b);      //this will be power of n while calculating T(n)

    n= compute_UN(lg,n);         //computes U(n) based on h(n)

    count+=2;


    count++;    //if else

    if(lg > n && n<0)

        cout<<"Time Complexity of the recurrence relation is:
        O(n^"<<lg<<")"<<endl;


    else if ((lg > n && n>0) || n>lg )

        {

            cout<<"Time Complexity of the recurrence relation is:
            O(n^"<<lg+n<<")"<<endl;

            count++;

        }


    else if(lg==0 && n==0)

        {

            cout<<"Time Complexity of the recurrence relation is:
            O(logn)"<<endl;
```

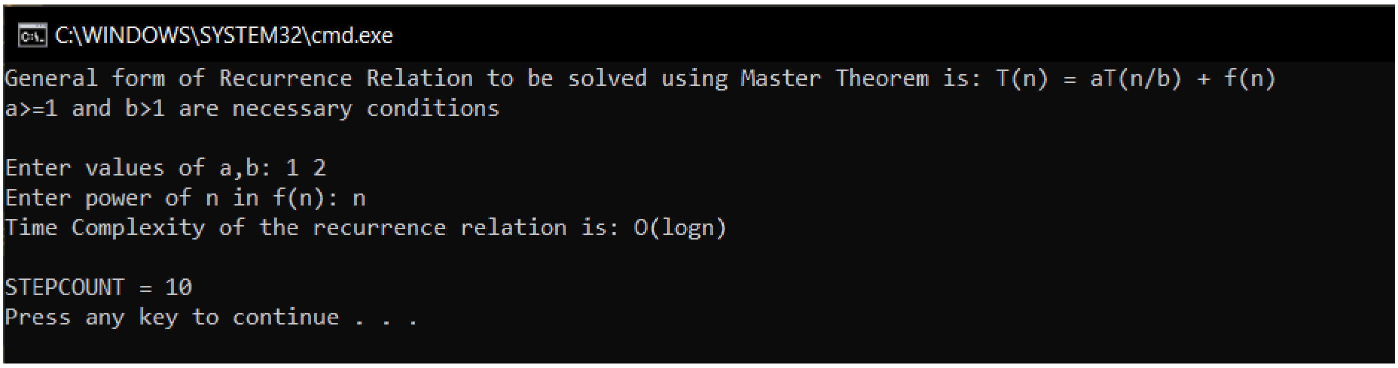
Sherwyn Fernandes
SE COMP-B
191105063

```
                count++;
            }

        else if (lg>0 && n==0)
            {
                cout<<"Time Complexity of the recurrence relation is:
                O((n^"<<lg<<")*logn)"<<endl;
                count++;
            }

}

int main()
{
        cout<<"General form of Recurrence Relation to be solved using Master
        Theorem is: T(n) = aT(n/b) + f(n)"<<endl;
        cout<<"a>=1 and b>1 are necessary conditions"<<endl;

        float n;        //variable n is power of n
        float a,b;
        cout<<"\nEnter values of a,b: ";
        cin>>a>>b;
```

Sherwyn Fernandes
SE COMP-B
191105063

```
    count++;

    if(a<1 || b<=1 )

        cout<<"This Recurrence Relation cant be solved using Master
        Theorem"<<endl;


    else

        {

            cout<<"Enter power of n in f(n): ";

            cin>>n;

            compute_RR(n,a,b);

        }


    cout<<endl<<"STEPCOUNT = "<<count<<endl;

    return 0;

}
```

## OUTPUT:



```
C:\WINDOWS\SYSTEM32\cmd.exe
General form of Recurrence Relation to be solved using Master Theorem is: T(n) = aT(n/b) + f(n)
a>=1 and b>1 are necessary conditions

Enter values of a,b: 1 2
Enter power of n in f(n): n
Time Complexity of the recurrence relation is: O(logn)

STEPCOUNT = 10
Press any key to continue . . .
```

Sherwyn Fernandes
SE COMP-B
191105063

C:\WINDOWS\SYSTEM32\cmd.exe

```
General form of Recurrence Relation to be solved using Master Theorem is: T(n) = aT(n/b) + f(n)
a>=1 and b>1 are necessary conditions

Enter values of a,b: 28 3
Enter power of n in f(n): 3
Time Complexity of the recurrence relation is: O(n^3.0331)

STEPCOUNT = 9
Press any key to continue . . .
```

C:\WINDOWS\SYSTEM32\cmd.exe

```
General form of Recurrence Relation to be solved using Master Theorem is: T(n) = aT(n/b) + f(n)
a>=1 and b>1 are necessary conditions

Enter values of a,b: 2 2
Enter power of n in f(n): 1
Time Complexity of the recurrence relation is: O((n^1)*logn)

STEPCOUNT = 10
Press any key to continue . . .
```

## Time Complexity:

This algorithm runs in constant time since it does not contain loops, recursion and call to any other non-constant time function.

**Hence, the time complexity is of the order of O(1)**

## Space Complexity:

Space needs to be allocated for a fixed number of variables in this algorithm., and no auxiliary space is needed.

**Hence, the space complexity is of the order of O(1)**