## **LEC-8 Recursion**

**Recursion:**

A recursive call of a function to itself

**When to stop recursive:**

There should be condition, called base conditions, on which termination of recursive call depends.

* **Merge Sort:**

It’s a divide and conquer

(We divide big problem into sub-problem and solve them separately, then merge them to get the output of given problem).

(0) (1) (2) (3) (4) (5)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 | 5 | 12 | 10 | 9 | 1 |

## **Merge Sort**

//Merge Sort

//Implementation in C++

#include <iostream>

using namespace std;

void merge(int\* a,int l1,int l2,int r1,int r2);

void mergesort(int\* arr,int start, int end){

    if(start<end){

        int mid=(start+end)/2;

        mergesort(arr,start,mid);

        mergesort(arr,mid+1,end);

        merge(arr,start,mid,mid+1,end);

    }

}

//Definition of merge

void merge(int\* arr,int l1,int l2,int r1,int r2){

    int\* temp=new int[r2-l1+1];

    int i=l1,j=r1,k=0;

    while(i<=l2 && j<=r2){

        if(arr[i]<arr[j]){

            temp[k]=arr[i];

            i++;

        }//end of if

        else{

            temp[k]=arr[j];

            j++;

        }

        k++;

    }//end of while

    //copying left side remaining elements

    while(i<=l2){

        temp[k]=arr[i];

        i++;

        k++;

    }

    //copying right side remaining elements

    while(j<=r2){

        temp[k]=arr[j];

        j++;

        k++;

    }

    //copying elements back to orignal array

    int start;

    for(start=l1,k=0;start<=r2;start++,k++){

        arr[start]=temp[k];

}

delete [] temp;

    temp=0;

    // end of merge

}

int main(){

    int n;

    int arr[]={1,2,3,4,5};

    //take size from user

    //for array, create and

    //initialize array display array

    mergesort(arr,0,n-1);

    //display array again

    return 0;

}

## **Analysis of Merge Sort**

T(n) =2T(n/2) +O(n)------k

(complexity of = (complexity of

Merge sort) merge)

T(1)= O(1) // base’s case equation

**>>>To T(n/2) put n=n/2 in eq k, we get**

**T(n/2) = 2T ((n/2)/2) + O(n/2)**

T(n/2) = 2T (n/4) + O(n/2)

**Put T(n/2) in eq k, we get**

T(n) = 2[2T (n/4) + O(n/2)] + O(n)

= 4T(n/4) + 2O(n/2) + O(n)

=4T(n/4)+O(n)+O(n)

=2^2 T (n/4) + 2O(n)

T(n) = 2^2 T(n/4) + 2O(n)-------k1

**>>>Lets find T(n/4), put n=n/4 in eq k1 , we get**

T(n/4) = 2T ((n/4)/2) + O(n/4)

T(n/4) = 2T (n/8) + O(n/4)

Put T(n/4) in eq k1, we get

T(n) = 2^2 [2T (n/8)+O(n/4)]+2O(n)

= 2^2 . 2T (n/8) + 2^2 . O(n/4)+ 2O(n)

= 2^3 (n/8) + O(n) + 2O(n)

= 2^3 (n/2^3) + 3O (n)

= 2^k (n/ 2^k) + kO (n)----------k2

Eventually n/n^k (array size) reaches to 1;

So,

n/n^k = 1

**n = 2^k**

logn = log2^k ⬄ logn = klog2

**logn = k 🡪**log2=1

lets rewrite eq k2,

=n T(1) + logn O(n) 🡪T(1) =O(1)

=n O(1) + logn O(n)

=O (n) + 0(logn . n)

**OR**

=O(n) + 0(nlogn)

**T(n) = O(nlogn) //both best and average case scnerio**