**DIVIDED AND CONQUIRE METHOD**

In this problem in hand are divided into small sub problem utill if as no division, After this each sub problem Are to be solved and finally all of this sub solution are merged to obtain the actual solution of the problem

**EX: |**a b c d e f g h| in single box

**|a| |b| |c| |d| |e|………**

**|A| |B| |C|………………..** solutions for sub problem

**| A B C …**………| actual solution of given problem

**DIVIDE :** Problems are to be divided

**CONQUIRE:** SubProblems are to be solved

**COMBINE:** Merge sub problem

**EXAMPLE DIVIDE AND CONQUIRE METHOD**

1. Merge Sort
2. Quick sort
3. Binary Search
4. Stassen’s Matrix multiplicatin

**General method**

Algorith D&C(p){

if small(p) then return small(p)

else {

divided p into smaller instance(p1,p2...pn)

apply D&C method for all instance in p

return the combined smaller instance(D&C(p1),D&C(p2)..D&C(pn),)

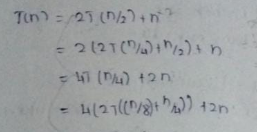
}

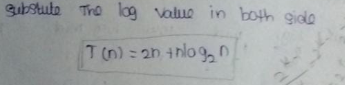
}

**Time complexity of D&C are describe from**

**T(n) = { g(n)**

**{ p(n1)+p(n2)....p(nk)**

****

****

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**BINARY SEARCH**

\* It is only used in sorted array

\* It follow divide and conqure method

\* If the element is found it returns the index

otherwise it jumps either right or left portion of

the array.

\* Bect case is O(1) worst and average case is O(log 2n)

**Algorithm BinarySearch(a,x,n){**

**low := 0, hign:=n;**

**while(low<=high) do{**

**mid=(low+high)/2;**

**if(x<a[mid]) then high=mid-1;**

**else if(x>a[mid]) then low=mid+1;**

**else return mid;**

**}**

**return 0;**

**}**

**RECURSIVE BINARY SEARCH**

**binarySearch (arr ,x, low, high)**

**{**

**If low >high then return False;**

**Else{**

**Mid=low+high/2;**

**If (x == arr[mid]) then return mid;**

**Else If (x>arr[mid]) then return binarySearch(arr,x,mid+1,high);**

**Else return binarySearch(arr,x,low,mid-1);**

**}**

**EXAMPLE PROBLEM:**

-15,-6,0,7,9,23,54,82,101,112,125,131,142,151

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

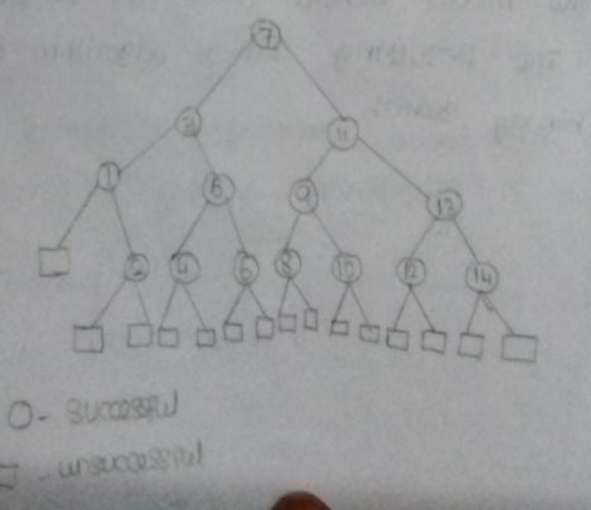
x=151

low high mid

0 14 7

8 14 11

12 14 13

14 14 14

**FORMULA FOR DIVIDING THE ARRAY**

E=I+2n E=2+4+6+8+10+12+14,I=31

I-Sum of Internal nodes, E-Sum of External node

n-Size of array

As(n) – Suceessful search As(n) = 1+I/n = 1+31/14

Au(n) – Unsuseccful search Au(n) = E/(n+1)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**MAX-MIN ALGORITHM**

Here we have to find min max element in array, for this we use two methods one is straightline, another one is recursion, In straightline method max and min element are to be found seperatly. Straightline maxmin requires 2(n-1) comparison as best,wort and average case.

Best case = n-1

Average case = 3n/2 -2

Worst case = 2(n-1)

**Algorith Straightline MAX-MIN(a){**

**min:=max:=a[1]**

**for i:=2 to n do{**

**if a[i] > max then max=a[i]**

**else if a[i] < min then min=a[i]**

**}**

**return(max,min)**

**}**

The number of comparision in this algorithm is 2n-2 this can be reduced by recursive approach.

**PROBLEM:**

a=[10,30,28,15,110,135,70,7]

10,30,28,15 110,135,70,7 – 4 parts

10,30 28,15 110,135 70,7- 2parts

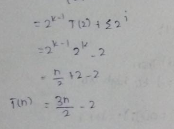
\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Min=10,max=30 min=7,max=135

Min=7,max135







**MERGE SORT (cn)**

Here the array is divided until it has one element,after that sorting operations are applied in each and every portion of the array, after that sorting is done all the sub arrays are merged with master array.

**Algorithm Merge Sort (low, high) {**

**If (low < high) {**

**mid := (low+high)/2; -> float**

**MergeSort(low,mid);**

**MergeSort(low,mid+1);**

**Merge(low,mid,high);**

**}**

**}**

**Algorithm Merger(low,high){**

**H,I := 0, j=mid+1;**

**While( H<=mid and j<=high) do {**

**If(a[j] <a[h]) then {**

**B[i]=a[h]; h=h+1;**

**}**

**Else{**

**B[i]=a[j]; j=j+1;**

**}**

**I=i+1;**

**}**

**If (h>mid) then {**

**For j:= k to high do {**

**B[i]=a[k]; i=i+1;**

**}**

**Else{**

**For j:= k to mid do{**

**B[j]=a[j]; j=j+1;**

**}**

**}**

**For k :=low to high do a[k]: =b[k];**

**}**

**EXAMPLE:**

To understand merge sort, we take an unsorted array as the following −

Unsorted Array

We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.

Merge Sort Division

This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

Merge Sort Division

We further divide these arrays and we achieve atomic value which can no more be divided.

Merge Sort Division

Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.

We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.

Merge Sort Combine

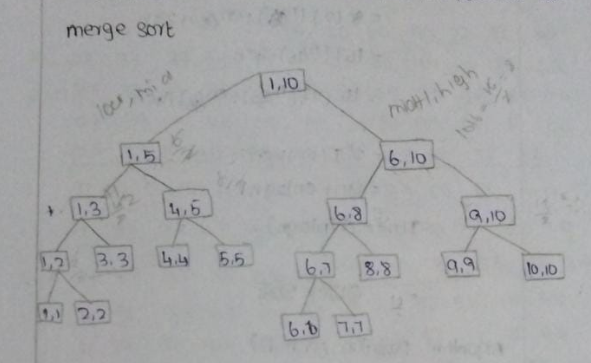
In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.

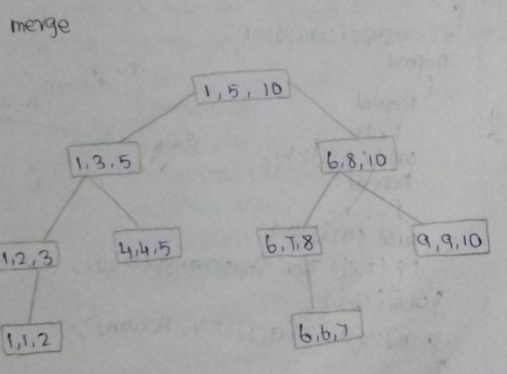
Merge Sort Combine

After the final merging, the list should look like this −

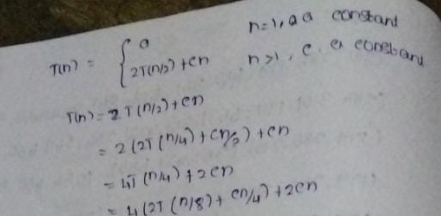
Merge Sort

Now we should learn some programming aspects of merge sorting.





**TIME COMPLXITY OF MERGE SORT IS**

****

**…………………………………..**

**………………………………….**

O(n logn)

**QUICK SORT**

**\***  It is also follows divide and conqure method

\* We can use recursion to build this sort

\* In each recursion we choose one pivot element and all the lesser elements compare to pivot

are going to left and all the greater elements compare to pivot are going to rigth is such a way it is sorted.

\* After every sort pivot occupies correct sorted position in array

\* So in each recusion our array is reduced by 2.

\* Pivot can be last,first or random element of the array.

**ALGORITHM:**

Algorithm partition (a, lower, higher)

{

Pivot := a[lower];

i := lower; and j := higher;

Repeat ( i<j ){

// finds larger element compare to pivot

Repeat {

i=i+1;

} until (a[i] <= Pivot)

// finds smaller element compare to pivot

Repeat {

j=j+1;

} until (a[j]>=Pivot)

If (i<j) then interchange (a,I,j);

}

a[lower]=a[j];

}

Algorithm Interchage(a, i ,j)

{

Temp := a[i];

a[j] := a[j];

a[j] := Temp;

}

Algorithm QuickSort(array, low, upper){

If (lower<right) {

Q=partition(arraym,right);

Quicksort(array,left,q-1);

Quicksort(array,q+1,right);

}

}

}

**EXAMPLE:**

44, 33, 11, 55, 77, 90, 40, 60

i j

**Step1:** Select pivot element first then find then larger element (from left to right) compare with pivot.

44-pivot

33>44 no, 11>44 no, 55>44 yes move start,now find a element that is lesser than the pivot from right to left 60<44 no 40<44 yes, so now start and are in this place

44, 33, 11, 55, 77, 90, 40, 60

i j

Now we have to interchange I and j

44, 33, 11, 40, 77, 90, 55, 60

I j

Again we have to find the greater element and lesser element util i>j

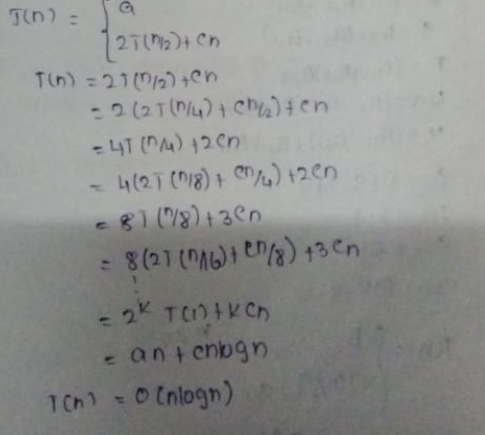
44, 33, 11, 40, 77, 90, 55, 60

j I

Now I is greater than j so we have to swap pivot with j

40, 33, 11, 44, 77, 90, 55, 60

**TIME COMPLEXITY**

****