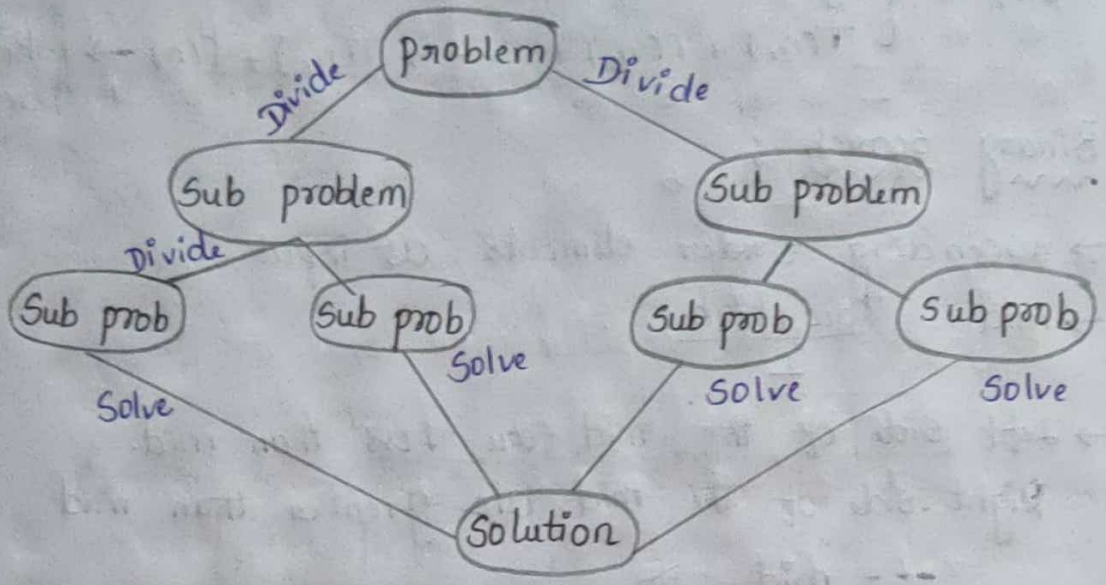
# Divide and Conquer




**Applications :**

→ Binary search
→ Quick Sort
→ Merge sort
→ Strassen's matrix multiplication.

**3 Steps :**

1. Divide
2. Conquer
3. Combine.

**Algorithm :**

```
Algorithm D and C (p)
{
    if small (p) then
        return S(p)
    else
    {
        divide p into smaller instances P1, P2 ---Pk, k≥1;
```

Apply D and C to each of these subproblems:
return Combine [D and C ($P_1$), D & C ($P_2$) --- D & C ($P_k$)]
}
}

Time Complexity:

$$T(n) = \begin{cases} g(n) & \text{if } n \text{ is small} \\ T(n_1) + T(n_2) + \cdots + T(n_k) + f(n) & \rightarrow \text{Where } n \text{ is large.} \end{cases}$$

Binary search:

→ Ascending Order elements as input

→ $mid = \frac{low + high}{2}$

→ Left side of the mid are Less than mid.
Right side of the mid are greater than mid

--- mid ---

→ $x == mid$ value.

→ If $x < mid$ value – Consider left sublist.

→ $x > mid$ value – Consider right Sublist

Ex:

11 15 20 35 42 50 75 100

Searching Element: $x_1 = 35$, $x_2 = 75$, $x_3 = 9$.

Given List

| 11 | 15 | 20 | 35 | 42 | 50 | 75 | 100 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

$mid = \frac{0+7}{2} = 3$

i) $x_1 = a[mid] = a[3] = 35$

35 is found at 3

ii) $x_2 = 75$

$mid = 3$

$x > mid$ – so we Consider right sublist.

| 42 | 50 | 75 | 100 |
|---|---|---|---|
| 4 | 5 | 6 | 7 |

$mid = \frac{4+7}{2} = \frac{11}{2} = 5$

Again $x > mid$

| 75 | 100 |
|---|---|
| 6 | 7 |

$mid = \frac{6+7}{2} = \frac{13}{2} = 6$

$x_2 = a[mid]$

75 found at 6.

iii) $x_3 = 9$

$mid = 3$

$x < mid$ ; Consider left sublist

| 11 | 15 | 20 |
|---|---|---|
| 0 | 1 | 2 |

$mid = \frac{0+2}{2} = 1$

$x > mid$, Consider left sublist.

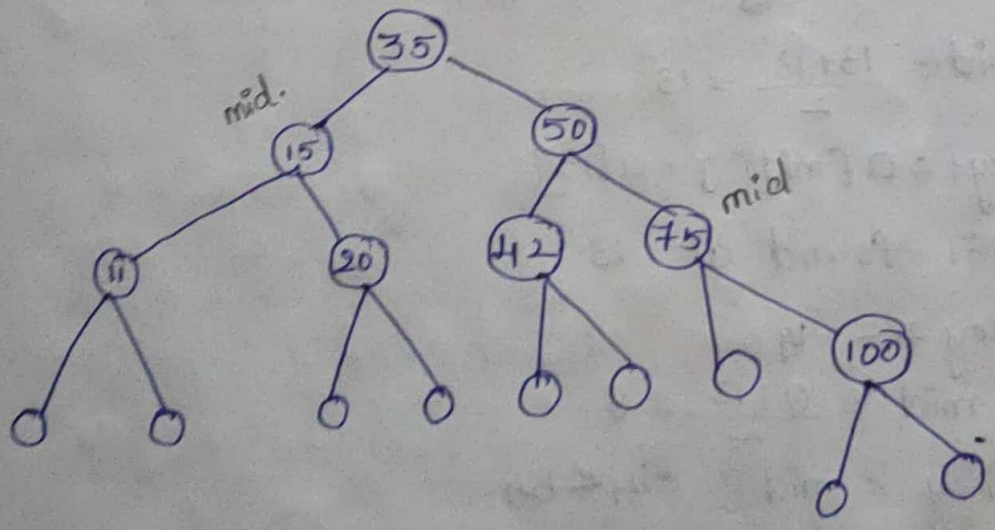
| 11 |
|---|
| 0 |

$mid = \frac{0}{2} = 0$

$x > mid$

∴ There is no element less than 11

Hence, 9 is not in the list.



Ex: -15 -6 0 7 9 28 54 82 101 112 125 131 142 151

Find the keys:

i) key 1 = 151

ii) key 2 = -14

iii) key 3 = 9

Given List.

| -15 | -6 | 0 | 7 | 9 | 28 | 54 | 82 | 101 | 112 | 125 | 131 | 142 | 151 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |

$mid = \frac{0+13}{2} = 6$

i) $key_1 = 151$

key 1 > mid ; so Consider right Sublist

| 82 | 101 | 112 | 125 | 131 | 142 | 151 |
|---|---|---|---|---|---|---|
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |

$mid = \frac{7+13}{2} = 10$

key 1 > mid $\Rightarrow$ 151 > 125

Again Considering right sublist.

| 131 | 142 | 151 |
|---|---|---|
| 11 | 12 | 13 |

$mid = \frac{11+13}{2} = 12$

key 1 > mid ; 151 > 142

| 151 |
|---|
| 13 |

$mid = \frac{13+13}{2} = 13$

key 1 = a[mid[] = a[3]

∴ 151 found at 13.

ii) key 2 = -14

$mid = \frac{0+13}{2} = 6.$

key < mid ; -14 < 54.

→ So Considering left sublist.

| -15 | -6 | 0 | 7 | 9 | 28 |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

$mid = \frac{0+5}{2} = 2$

Key2 < mid ⇒ -14 < 0

→ Again Considering left sublist.

| -15 | -6 |
|---|---|
| 0 | 1 |

$mid = \frac{0+1}{2} = 0$

Key2 > mid ; -14 > -15

→ Considering right sublist.

| -6 |
|---|
| 1 |

$mid = \frac{1+1}{2} = 1$

Key2 ≠ a[mid] ⇒ -14 ≠ a[1]

∴ -14 is not in the list.

iii) Key 3 = 9

$mid = \frac{0+13}{2} = 6$

Key3 < mid ⇒ 9 < 54

So, Considering left sublist.

| -15 | -6 | 0 | 7 | 9 | 28 |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

$mid = \frac{0+5}{2} = 2$

Key 3 > mid ; 9 > 0

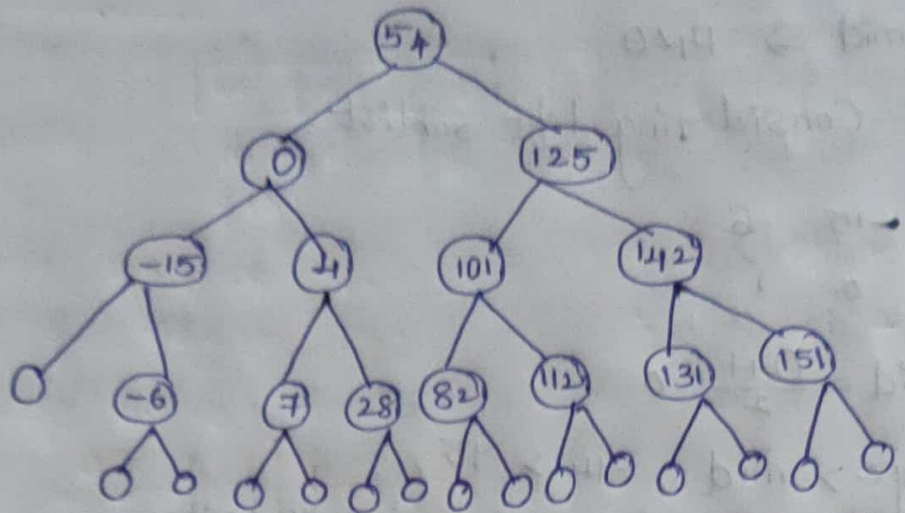
→ Considering Right sublist

| 7 | 9 | 24 |
|---|---|---|
| 3 | 4 | 5 |

mid = $\frac{3+5}{2}$ = 4

key 3 = a [mid] ⇒ key3 = a[4]

∴ key 3 = 9 found at 4.




Ex : 65 70 75 80 85 60 55 50 45

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 65 | 70 | 75 | 80 | 85 | 60 | 55 | 50 | 45 |
| ↓ pivot i | i | | | | | | | j |

pivot = 65

70 ≤ 65 → false ∴

j = [8] = 45 > 70 false

swap (a[i] & a[j])

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 65 | 45 | 75 | 80 | 85 | 60 | 55 | 50 | 70 |
| ↓ pivot | i | i | | | | | j | j |

45 ≤ 65 = i++

75 ≤ 65 = false

j - -

70 > 65 = True

50 > 65 = false

swap [a[i] & a[j]]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 65 | 45 | 50 | 80 | 85 | 60 | 55 | 75 | 70 |
| ↓ pivot | | i | i=3 | | | j=6 | j | |

pivot = 65 i++

j--

$50 \leq 65$ = True

$80 \leq 65$ = false

$75 > 65$ = true

$55 > 65$ = false.

Swap (a[i] & a[j])

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 65 | 45 | 50 | 55 | 85 | 60 | 80 | 75 | 70 |

i  i=4  J  ~~J=6~~

↓ pivot = 65

i++

$55 \leq 65$ = true

$85 \leq 65$ = False

j--

$85 > 65$ = true

$60 > 65$ = false.

Swap (a[i] & a[j])

| 65 | 45 | 50 | 55 | 60 | 85 | 80 | 75 | 70 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

j  i

i++

$60 \leq 65$ = true

$85 \leq 65$ = false

j--

$85 > 65$ = true

$60 > 65$ = false

$i \geq j$

Swap [a(j) & pivot)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 60 | 45 | 50 | 55 | 65 | 85 | 80 | 75 | 70 |

↓ pivot

→ whenever the pivot is swapped in divided the list into two sublist.

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 60 | 45 | 50 | 55 |
| i | ~~i~~ | i | j |

pivot.

$45 \leq 60$ = true  i++ i=1

$50 \leq 60$ = true  i++ i=2

$55 \leq 60$ = true  i++ i=3

i=4 Out of index.

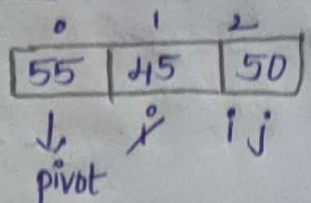
∴ i=3, j=3.

$i \geq j$ (i+ & time Swap swap (60, 55)

55 45 50 60 piv

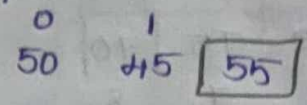
divide into two parts One is left sublist of pivot is right sublist of pivot. Here there is no right sublist Therefore.

(i) Left Sublist



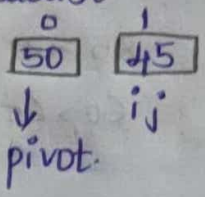

i=1 45 ≤ 55 i++
i=2 50 < 55 i++
i=3 out of index
∴ i=2, j=2

Swap (55, 50)



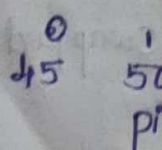

It is further divided but only ~~right~~ Left sublist is possible

Left sublist

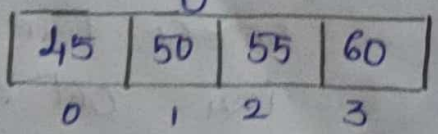



i=0 50 ≤ 50 - i++
i=1 45 ≤ 50 - true i++
i=2 out of index.

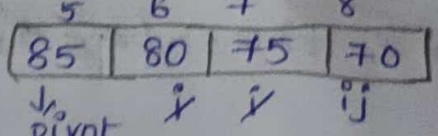
pivot
i=1, j=1 Swap (50, 45)




After sorting the Left elements are




2. Right Sublist :



i=5 85≤85 = True i++

i=6 80≤85 = true i++

i=7 75≤85 = i++

i=8 70≤85 i++

i=9 Out of index.

∴ (i=3 j=3)

swap (85, 70)

| 5 | 6 | 7 | 8 |
|---|---|---|---|
| 70 | 80 | 75 | 85 |

By dividing we get left sublist as :

| 5 | 6 | 7 |
|---|---|---|
| 70 | 80 | 75 |

pivot i j

i=6 80≤70 x false.

j=7 75>70 ; j--

j=6 80>70 ; j--

7=5 70=70 ; x

6>5 i≥j

swap (70, 70)

| 5 | 6 | 7 |
|---|---|---|
| 70 | 80 | 75 |

After dividing we get right sublist :-

| 6 | 7 |
|---|---|
| 80 | 75 |

pivot j

i=6 80≤80 i++

i=7 70≤80 i++

i=8 Out of index.

swap (80, 75)

75 80

pivot

We can able to divide further After sorting get right sublist as.

70 75 80 85

finally, the sorted elements are

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 45 | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 |

Algorithm for Binary Search :

```
Algorithm Binary Search (a, n, x)
{
   low = 1, high = 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 if (x == a[mid]) then
              return mid
        else
              printf (" elements not found ");
   }
}
```

Time Complexity :

Best Case :

If the search element is at the middle of all element. In thate Case we need only one Comparision.

→ O(1)

Worst Case :

$O(\log_2 n)$

Quick Sort :

30 20 10 50 60 40

i
pivot

1. While a[i] ≤ pivot do i++
2. while a[j] > pivot do j--
3. if i<j Swap a[i] and a[j]
4. If i≥j swap a[j] and pivot

pivot = 30

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 30 | 20 | 10 | 50 | 60 | 40 |
| pivot | i | i | i | | j |

20 ≤ 30 ; i++

10 ≤ 30 ; i++

50 ≤ 30 ✗

| 0 | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| 10 | 20 | 30 | ~~50~~ | 60 | ~~40~~ | ~~4~~ |

→ whenever the pivot is swapped then divide the list into two sublists.

| 0 | 1 |
|---|---|
| 10 | 20 |
| i | j |

pivot

10 = 10 i++

| 0 | 1 |
|---|---|
| 10 | 20 |

pivot j, i

20 > 10 j-1

| 0 | 1 |
|---|---|
| 10 | 20 |

20

consider the right sublist.

| 3 | 4 | 5 |
|---|---|---|
| 50 | 60 | 40 |

pivot
i

50 < 50 i++

| 3 | 4 | 5 |
|---|---|---|
| 50 | 60 | 40 |

pivot i

60 < 50 ×

| 50 | 60 | 40 |
|---|---|---|

pivot i

40 > 50 ×

4 < 5

i < j swap a[i] and a[j]

| 3 | 4 | 5 |
|---|---|---|
| 50 | 40 | 60 |

pivot i j

40 ≤ 50 True i++

| 50 | 40 | 60 |
|---|---|---|

pivot i,j

60 > 50 j--

| 40 | 50 | 60 |
|---|---|---|

∴ After sorting

10 20 30 40 50 60.

Quick Sort : Quick Sort is the widely used sorting Algorithm that makes n log n comparisions. divide first pick a pivot element After that partition.

Algorithm for Quick sort :

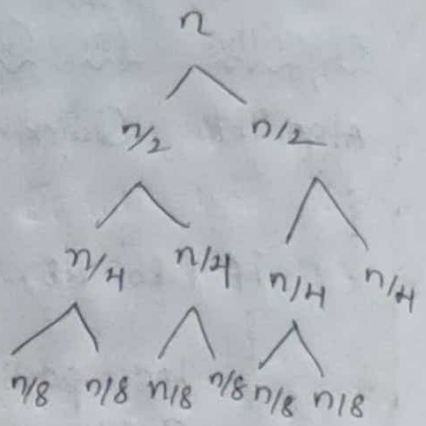
```
Algorithm Quick sort (a, lb, ub)
{
    if (lb < ub)
    {
        loc = partition (a, lb, ub); ⟶ c.n
        Quicksort (a, lb . loc-1); ⟶ T(n/2)
        quick sort (a, loc+1, ub); ⟶ T(n/2)
    }
}
Algorithm partition (a[], lb, ub)
{
   p = a[lb];
   Start = lb , end = ub;
   While (Start < end)
   {
      While (a[Start] ≤ p && Start < end)
      {
         Start = Start + 1 ;
      }
      While (a[end] > p)
      {
         end = end - 1 ;
      }
      if (Start < end)
      {
         Swap (a[Start], a[end]) ;
      }
   }
   a[lb] = a[end] ;
   return end;
}
```

Time Complexity :

Best Case :

The pivot is placed at mid position.

*Average Case :




$O(n \log n)$

This is for partition

For n elements

total time Complexity

$n \log_2 n$

$O(n \log_2 n)$

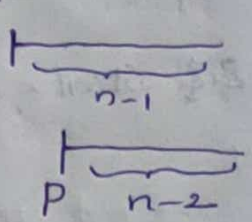
$\frac{n}{2^k} = 1$

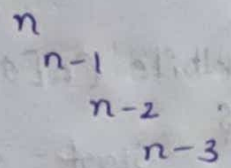
$n = 2^k$

$k = \log_2 n$

Worst Case :

→ The pivot is placed at either beginning or ending position




$n + n-1 + n-2 + n-3 + \cdots + 1$

$1 + 2 + 3 + \cdots + n-2 + n-1 + n$

$= \frac{n(n+1)}{2} = \frac{n^2+n}{2}$

$= O(n^2)$

* If we give sorted elements as i/p, then we will have worst Case.
* If elements are in ascending order, then pivot will be placed at beginning.
* If elements are in descending order, then pivot will be placed at ending.

General Method :

In divide & Conquer method, a given problem is divided into smaller sub-problems. These sub-problems are solved independently. Combining the solutions of all the sub problems into a solution of whole.

→ If the sub problem are large enough then divide & Conquer is reapply.

→ The generated sub problems are usually of same type has the original problem.

→ A control Abstraction were divide & Conquer is given below :

Using Control abstraction a flow of control of a procedure is given ;

⇒ Binary Search Steps :

1) In the Binary search method the list of elements in the ascending order are taken at the input Here a key element is given to search in the list.

2) The list is divided into two parts initially by finding the mid position and the mid = $\frac{\text{start}+\text{end}}{2}$

3) All the elements in the left sub lise careless than the mid value. All the elements in the right sublist greater than the mid value.

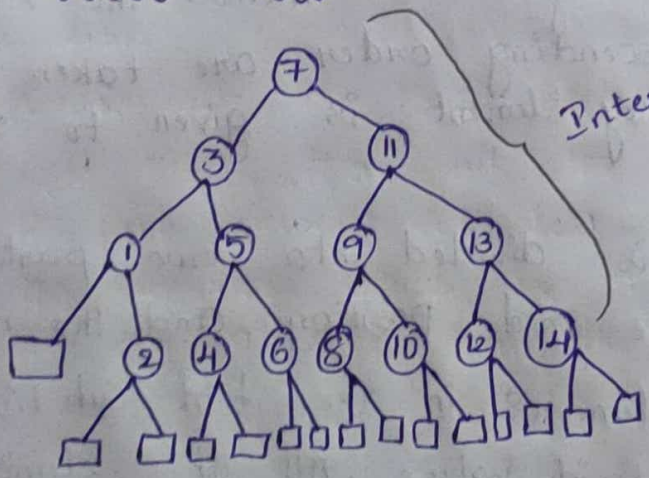
4) The key element is now compared with the mid value then one of the three conditions may be true.

* If key is less than a[mid] then left sublist is taken and again the same process is repeated for searching the element in the.

Left sublist the mid is that again in the left sublist the mid position will be found and the key element is compared with the mid value and again three cases will occur the same process is repeated in any case till element is found.

* If key is greater than a[mid] then right sublist is taken and again the same process is repeated for searching the element in the right sublist. It means that again right sublist mid position will be found and the key element is compared with the mid element and again three cases will occur the same process is repeated in any case till the element is found.
* If key is equal to a[mid] the return the mid position.

Binary Decision tree :




* A Binary decision tree traces the way in the which these values are produced in Binary search.
* The first Comparision is x with a[7]
If x is less than a[7] the next Comparison is a[3] Similarly If x is greater than a[7] then the next Comparison is with a[11]

Each path through the tree represents a sequence of Comparision in the binary search method.

* If 'x' is present then algorithm will end at one of the circular (internal) nodes. If 'x' isn't present then the algorithm will terminate at one of the square nodes.
* Circular nodes are called internal nodes and square nodes are reffered to as leaf nodes

Analysis of Binary search :

The Analysis of Binary search depends on the position of the element.

Best Case : The Best Case occur when the position of the search element is present in the middle of all the elements. It takes only one unit of time so the time taken to search for an element in the best case is "O(1)"

Worst Case :

It occurs when the position of the search element is starting (or) ending of all the positions of elements. In order to find out the search element we have to partition all the elements into two parts the size of the each part in $n/2$. we only choose one part based on the search element. Again the select size $n/4$, here again we choose only one part like this the process is continued.

$n, n/2, n/4, n/8, n/16, n/32 \cdots n/2^k$

Let us Consider the last partition size is 1

i.e $\frac{n}{2^k} = 1$ Applying log on both sides.

$k = \log_2 n$

So the time taken to search the element in the worst case $O(\log n)$

Average Case :

The position of search element may be at the middle (or) at the starting (or) at the ending the Average time taken to search a key element is Overally O(log n)

→ The time taken for unsuccessful search is O(log n)

⇒ Quick sort :

In this method the array of elements are taken as the Input. Here one element is choosen as the pivot, the pivot can be the first element (or) the last element (or) the middle element (or) any random element in the list

* Usually the first element in the list taken as the pivot. The correct position of the pivot in the list is determined by comparing the pivot element with remaining elements in the list.
* The position of the pivot element is find in such a way that all the elements left of the pivot element are less than the pivot and all the elements which are right of the pivot element are greater than the pivot Element
* Once the pivot is placed in its correct position then the list is divided into two parts (or) Sublists Again in each sublist the above procedure is repeated.

Procedure to find the pivot in the list :

1) Lets take an array of n elements a[1] to a[n]. The first element in the array a[1] is taken as pivot, let us say i, j indicates starting and ending positions of the array.

i.e., i=1 and j=n

2) Increment i everytime if a[i] ≤ pivot
Once a[i] > pivot stop incrementing i

3) Decrement j everytime if a[j] > pivot.
Once a[j] ≤ pivot stop Decrementing j

4) check the positions of i and j if (i<j) then swap a[i] and a[j] and then repeat the above steps from (2)

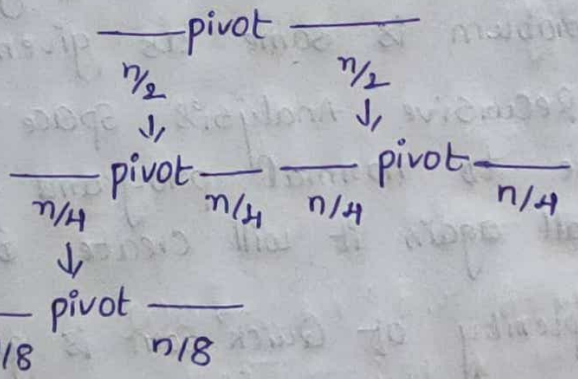
5) If i and j are crosed i.e., if (i≥j) then swap pivot and a[j], Once the pivot is swapped then the List is divided into two parts.

6) Repeat the same process from step 1 and step 5 for each and every sublist.

Analysis of Quick Sort :

The running time of the Quick sort depends on whether the partition is balanced (or) unbalanced

* A very good partition splits an array into 2 parts.
i.e. If there are 'n' elements in the array.




Like wise in each step the whole number of elements to be sorted are 'n' which will take the time $\log_2 n$

The best case time Complexity is $O(a\log_2 n)$

* A bad partition splits an array into very different sized arrays
i.e., If there are 'n' elements in an array.

pivot ——— n

pivot ——— n

——— pivot
n

It means in each step there are almost 'n' elements which takes time of n elements Sorting

The worst case time Complexity $O(n^2)$

* An Average partition splits in both ways in different levels.

∴ The Average time Complexity is $O(n \log_2 n)$

## Recurrence Relations / Recurrence Equations :

If we represent the time Complexity of a recursive algorithm in the form of relation / equation, then it said to be "Recurrence relation / equation"

**Recursion** : A function calling itself is called "Recursion"

Recursion is preferable when, In a given problem every subproblem is same as given problem.

For Recursive Analysis space Complexity is more Compare to normal programs. for every recursive call again it will create stack.

Time Complexity of Quick sort is given by

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

$$T(n) = 2T(n/2) + C \cdot n$$

This equation is called Recurrence Equation.

(Since time Complexity of Quick sort (recursive Algorithm is represented in the form of equation).

# Dividing the time Complexity form recurrence equation

(Solving recurrence relation)

Methods :

1. Back substitution method
2. Masters method
3. Tree method

## 1. Back substitution method :

* Simply substitute the terms.

Ex :- $T(n) = \begin{cases} T(n-1)+1 & n>0 \\ 1 & n=0 \end{cases}$

$T(n) = T(n-1)+1 \longrightarrow ①$

$T(n-1) = T(n-2)+1$

$T(n-2) = T(n-3)+1$

$① \Rightarrow T(n) = T(n-2)+1+1$

$= T(n-3)+1+1+1$

$= T(n-3)+n.$

$\vdots$

$= T(n-k)+k$

$n-k=0$

$n=k$

$= T(0)+k$

$= 1+k$

$= 1+n$

$O(n)$

Time Complexity is $O(n)$

Ex : Quick Sort

$T(n) = \begin{cases} 2T(n/2)+C.n & n>1 \\ C_1 & n=1 \end{cases}$

Recursive relation.

Sol: Finding time Complexity through back Substitution Method.

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

$= 2[2T(n/4)+\frac{Cn}{2}]+Cn$

$T(n/2) = 2T(n/4)+\frac{Cn}{2}$

$T(n/4) = 2T(n/8)+C.\frac{n}{4}$

$= 2^2 T(n/4) + cn + cn$

$= 2^2 [2 + (n/8) + cn] + cn + cn$

$= 2^3 + n/8 + 3cn$

$= 2^k T(n/2^k) + kcn$

$= n \cdot c_1 + \log_2 n \cdot c \cdot n$ $\quad \frac{n}{2^k} = 1$

$= nc_1 + n \log_2 n\, c$ $\quad n = 2^k$

$\boxed{k = \log_2 n}$

Consider higher order $(n \log_2 n)$ (neglect Constant terms)

∴ time Complexity of Quick sort is $O(n \log_2 n)$

Steps :

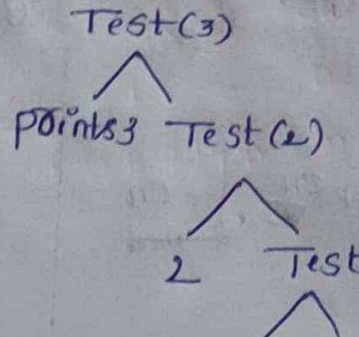
1. In order to Analyse any Recursive program we need to write Recurrence relation.
2. By solving that recurrence relation only will get Time Complexity.

Eg:

```
void Test(int n)
{
    if(n>0) → O(1)
    {
        printf("%d", n); → O(1)
        Test(n-1); → T(n-1)
    }
}
```

Tree method :

Let $n=3$



Test (0)
x (stop)

The amount of work done here → printing value + calling function.

As I passed '3' → 3 times printing If I passed '5' → 5 times prints

For each call it is printing 1 unit of time so

no. of calls 3+1 = 4

no. of prints - 3

In general no. of calls 'n+1'

for any 'n' no. of prints 'n'

Time function $f(n) = O(n)$

∴ Time Complexity = $O(n)$

eg 1 :

```
void Test (n) {  ⟶ T(n)
If (n>0) → O(1)
{
   printf ("%d", n); ⟶ 1
   Test (n-1);  ⟶ T(n-1)
}
}
```

Step 1 : write down recurrence relation, for if and printf statement it will take constant amount of time and again it will call Test (n-1) so the relation is.

$T(n) = T(n-1) + 1$ [∵ as if and printf takes O(1)]

$$T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+1 & n>0 \end{cases}$$

Step 2 : Solve recurrence relation by Back substitution method.

$T(n) = 1 + T(n-1) \rightarrow (1)$

$T(n-1) = 1 + T(n-2) \rightarrow (2)$

$T(n-2) = 1 + T(n-3)$

⋮

$T(n-k) = 1+T(n-k)$

Consider ① $T(n) = 1+T(n-1)$

Substitute $T(n-1)$ value from ②

$\Rightarrow T(n) = 1+T(n-1)$

$T(n) = 1+1+T(n-2) = 2+T(n-2)$

$T(n) = 2+1+T(n-3) = 3+T(n-3)$

⋮

$T(n) = K+T(n-K)$

When it will stop whenever $n-k=0 \Rightarrow n=k$.

$T(n) = T(n-K)+K$

Substitute $n=K$

$T(n) = T(n-n)+n$

$T(n) = T(0)+n \Rightarrow 1+n$

$\boxed{T(n) = O(n)}$

Eg 2:

```
void Test (n)
{
  if (n>0) {                    → 1
    for (i=0; i<n; i++)         →
    {
      printf ("%d", n); n       recursive call
    }
    Test (n-1);
  }                               ↳ T(n-1)
```

Step 1:

$T(n) = T(n-1)+1+n+1+n$

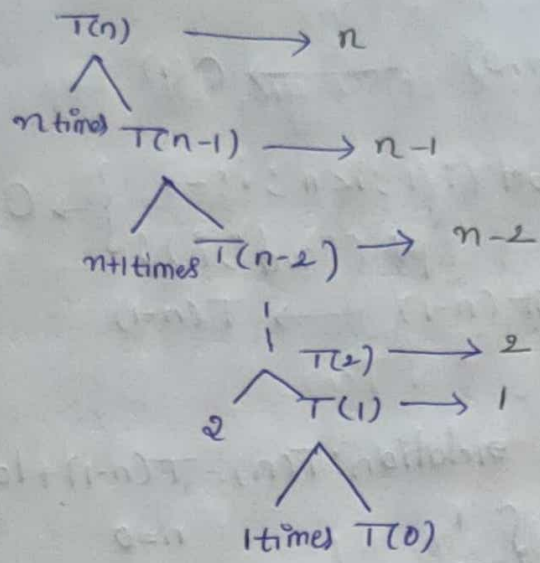
$= T(n-1)+2n+2$

Writing it as Asymptotically $O(n)$

So recurrence relation now

$$T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+n & n>0 \end{cases}$$

Tree Method :




total time is :

$$T(n) = n + \underbrace{(n-1)+(n-2)+ \cdots 2+1}_{\text{Sum of 1st natural numbers.}}$$

$$T(n) = \frac{n(n+1)}{2} \simeq O(n^2)$$

$$\therefore T(n) = O(n^2)$$

Backsubstitution Method :

$$T(n) = T(n-1) + n$$
$$T(n-1) = T(n-2) + n-1$$
$$T(n-2) = T(n-3) + n-2$$
$$\vdots$$
$$T(n-K) = T(n-(K+1)) + n-K$$

from 1

$$T(n) = T(n-1) + n$$
$$= [T(n-2) + n-1] + n$$
$$= T(n-3) + (n-2) + (n-1) + n$$

K times ↓

$$= T(n-(K+1)) + n-K + \cdots T(n-3) + (n-2) + (n-1) + n$$
$$= T(n-K) + (n-(K-1)) + n-(K-2) + \cdots + (n-1) + n$$

$n-K=0 \Rightarrow n=K$ Substitute.

$$T(n) = T(0) + (n-n+1) + (n-n+2) + \cdots (n-1) + n$$
$$= \underbrace{1+1+2+3+\cdots (n-1)}_{} + n$$
$$= n(n+1)/2 \simeq O(n^2)$$

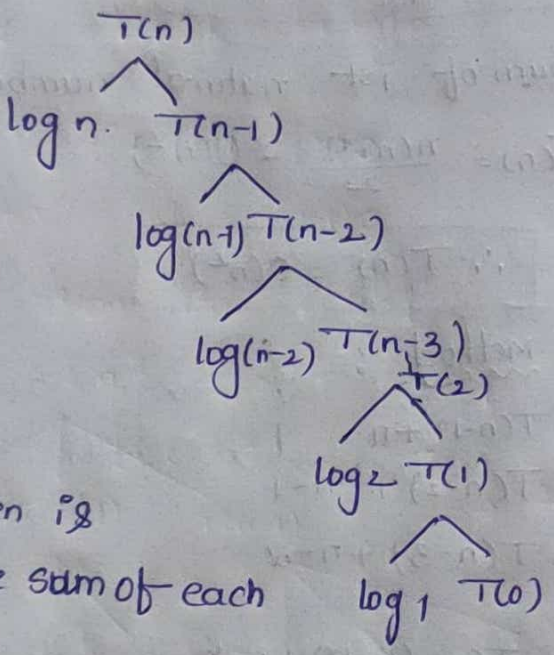
Eg 3:

```
void Test (n)
{
    if (n>0)  ⟶ O(1)
    {
        for (i=1; i<n; i*2) } → O(logn)
            pf ("%d", i);
        Test (n-1) ⟶ T(n-1)
    }
```

write Recurrence relation: $T(n) = T(n-1) + \log n + O(1)$

$$T(n) = \begin{cases} 1 & n=0 \\ T(n-1) + \log n & n>0 \end{cases}$$

Tree method:




∴ Time taken is workdone sum of each level.

$T(n) = \log n + \log(n-1) + \log(n-2) + \cdots + \log 1$

$\log [n*(n-1)*(n-2)*(n-3)* \cdots 1)$

$= \log [n!]$ $[\because n! \simeq n$

$= \log n^n$ There is no bound so upperbound $n! \simeq n^n$

$\boxed{T(n) \Rightarrow O(n \log n)}$

Back Substitution Method :

$T(n) = T(n-1) + \log n$

$T(n) = [T(n-2) + \log(n-1)] + \log n$

$= T(n-2) + \log(n-1) + \log n$

$= [T(n-3) + \log(n-2)] + \log(n-1) + \log n$

$= T(n-3) + \log(n-2) + \log(n-1) + \log n$

After k times :

$T(n) = T(n-k) + \log 1 + \log 2 + \log 3 + \cdots \log(n-1) + \log n \rightarrow (1)$

We know

$n - k = 0$

$n = k \rightarrow$ substitute in (1)

$T(n) = T(0) + \log 1 + \log 2 + \log 3 + \cdots + \log n$

$= 1 + \log n!$

$= 1 + \log n^n$

$= 1 + n \log n$

$\boxed{T(n) = O(n \log n)}$

Conclusions :

$T(n) = T(n-1) + 1 \rightarrow O(n)$

$T(n) = T(n-1) + n \rightarrow O(n^2)$

$T(n) = T(n-1) + \log n \rightarrow O(n \log n)$

↓

fun is multiplied by 'n' times.

Instead by $T(n-1)$ we can take $T(n-2)$

Then also same thing

$T(n) = T(n-2) + 1 \rightarrow n/2 \simeq O(n)$

$= T(n-100) + 1 \rightarrow O(n)$

↓

constant.

∴ In general we can write from above observations If any recurrence relation in the form of

$T(n) = T(n-K) + f(n)$

↓ Constant ↓ Asymptotic fun.

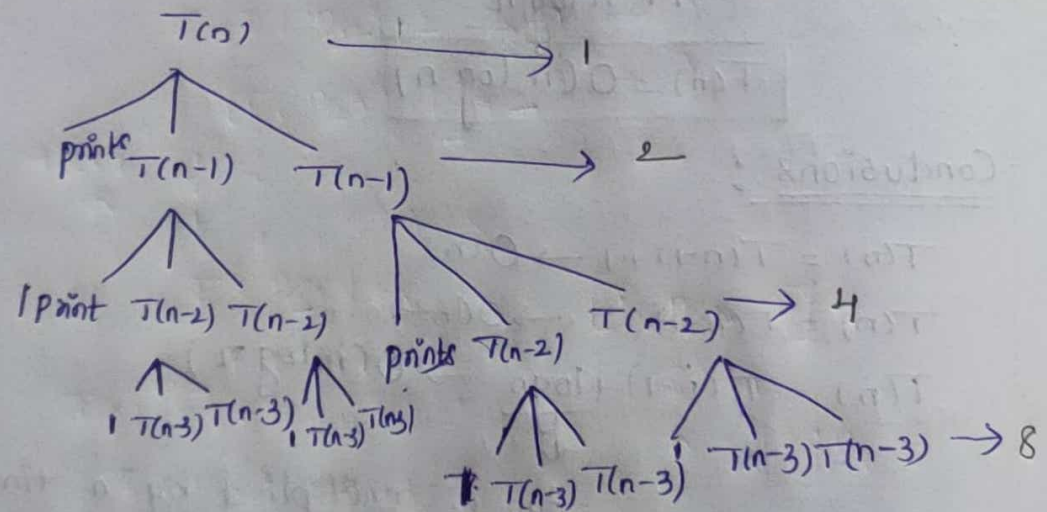
then time Complexity is n times of $f(n)$

Ex 4:

```
void Test(n) {          ⟶ T(n)
if(n>0)                 ⟶ 1
  {
    pf("%d", n);        ⟶ 1
    Test(n-1);          ⟶ T(n-1)
    Test(n-1);          ⟶ T(n-1)
  }
}
```

$$T(n) = \begin{cases} 2T(n-1)+1 & n>0 \\ 1 & n=0 \end{cases}$$

using Recursion Tree method:




After K;

$T(0)\ T(0)\ \cdots\ T(0)\ T(0) - 2^K$.

total work done Sum of work in each level.

$1 + 2 + 2^2 + 2^3 + \cdots 2^K$ [∵ Sum of G.P series]

$a = 1$

$r = 2$ $\quad a + ar + ar^2 + \cdots ar^K$

$$\text{Sum} = \frac{a(r^{K-1})}{r-1} = 2^{K+1} - 1 \simeq O(2^n)$$

Assume $n-K = 0 \Rightarrow n = K$

Back Substitution method :

$T(n) = 2T(n-1)+1 \rightarrow ①$

$= 2[2T(n-2)+1]+1$

$= 2^2 T(n-2)+2+1 \rightarrow ②$

$= 2^2[2T(n-3)+1]+2+1$

$= 2^3 T(n-3)+2^2+2+1 \rightarrow ③$

After K times!

$T(n) = 2^K T(n-K)+2^{K-1}+2^{K-2}+\cdots 2^3+2^2+2+1$

Assume $n-K=0$

$n=K$

Substitute K values as n'

$T(n) = 2^K T(0)+2^{K-1}+2^{K-2}+\cdots 2^3+2^2+2+1$

$= 2^n * 1 + 2^n - 1$

$= 2^n + 2^n - 1 \Rightarrow 2^{n+1}-1$

$T(n) = 2^{n+1}-1$

$\boxed{T(n) \simeq O(2^n)}$

Ex 5 : Test (n) { $\longrightarrow T(n)$

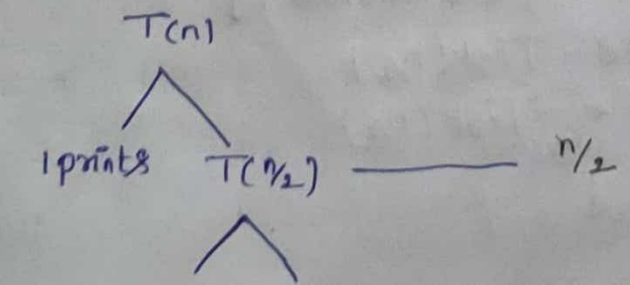
```
if (n>1)
{ pf("%d", n); ⟶ 1
  Test (n/2); ⟶ T(n/2)
}
}
```

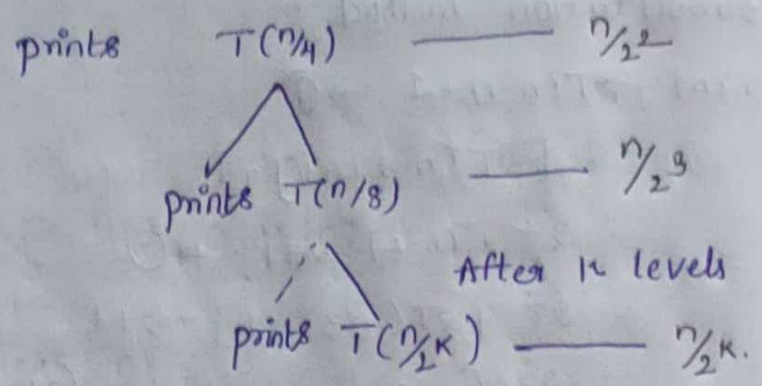
$T(n) = T(n/2)+1$

Recurrence relation :

$$T(n) = \begin{cases} T(n/2)+1 & n>1 \\ 1 & n=1 \end{cases}$$

Recursion Tree method :






We know it will stop at

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

$$k = \log n$$

It will stop after k level so total time is:

$$1 * k = 1 * \log n = O(\log n)$$

Back substitution method :

$T(n) = T(n/2) + 1 \rightarrow ①$

$T(n/2) = [T(n/2^2) + 1] + 1$

$= T(n/2^2) + 2 \rightarrow ②$

$= [T(n/2^3) + 1] + 2$

After k :

$T(n/2^k) + k$ $\quad [\because n/2^k = 1]$

$= T(0) + k$ $\quad 2^k = n$

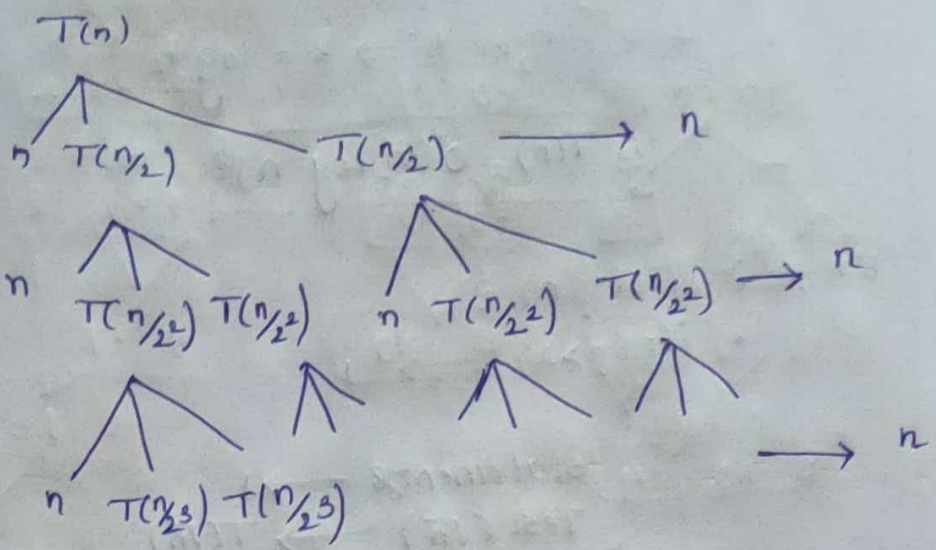
$T(n) = 1 + \log n$ $\quad k = \log n$

$\boxed{T(n) = O(\log n)}$

Ex 6 :

```
Void Test (n) {
if (n>1) {
    for (i=0; i<n; i++)
n ← { statements ; }
T(n/2) ← Test(n/2)
T(n/2) ← Test(n/2)
}
```

Recursive Tree method :




After k level :

$T(n/2^k)$

total work done after k levels is $\Rightarrow k*n$

We know it will stop when $n/2^k = 1$

$n = 2^k$

$k = \log n$

Substitute

$T(n) = k*n$

$= \log n * n$

$\therefore T(n) = O(n \log n)$

Recurrence relation :

$$T(n) = \begin{cases} 2T(n/2)+n & n>1 \\ 1 & n\leq 2 \end{cases}$$

Back Substitution Method :

$T(n) = 2T(n/2) + n \rightarrow (1)$

$= 2[2T(n/2^2) + n/2] + n$

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

$= 2^2 T(n/2^2) + 2n \rightarrow (2)$

$= 2^2 [2T(n/2^3) + n/2^2] + 2n$

$= 2^3 T(n/2^3) + 3n \rightarrow (3)$

After $k^{th}$ recursive calls. :

$= 2^k T(n/2^k) + kn$

$n/2^k = 1 \Rightarrow k = \log n$

$\therefore T(n) = O(n \log n)$

Ex 7 :

```
Test (n) {
if (n>2) ——> 1
  {
    statements ——> 1
    Test (√n) ——> T(√n)
  }
}
```

$$T(n) = \begin{cases} T(\sqrt{n}) + 1 & n > 2 \\ 1 & n = 2 \end{cases}$$

Using Back Substitution Method :

$T(n) = T(n^{1/2}) + 1 \rightarrow (1)$

$= [T(n^{1/2^2}) + 1] + 1$

$= T(n^{1/2^2}) + 2 \rightarrow (2)$

$= T(n^{1/2^3}) + 3 \rightarrow (3)$

After k :

$T(n^{1/2^k}) + k$

$n^{1/2^k}$ Assume $n = 2^m$

$T(2^m) = T(2^{m/2^k}) + k$

$= \log_2 m$

$\Rightarrow \log_2 \log n$

$n = 2^m$

$m = \log n$

$1 = \frac{m}{2^k}$

$2^k = m$

$k = \log_2 m$

$T(n) = O(\log \log n)$

## Merge Sort :

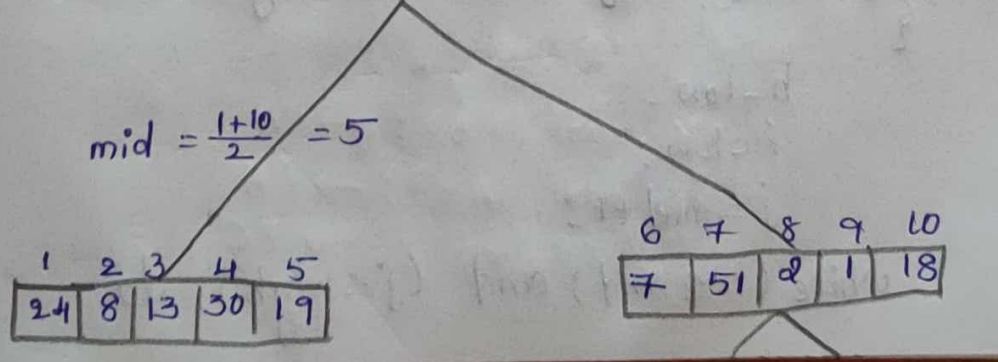
In this method list of elements are given, We have to arrange the elements in the order it follows the divide & Conquer technique.

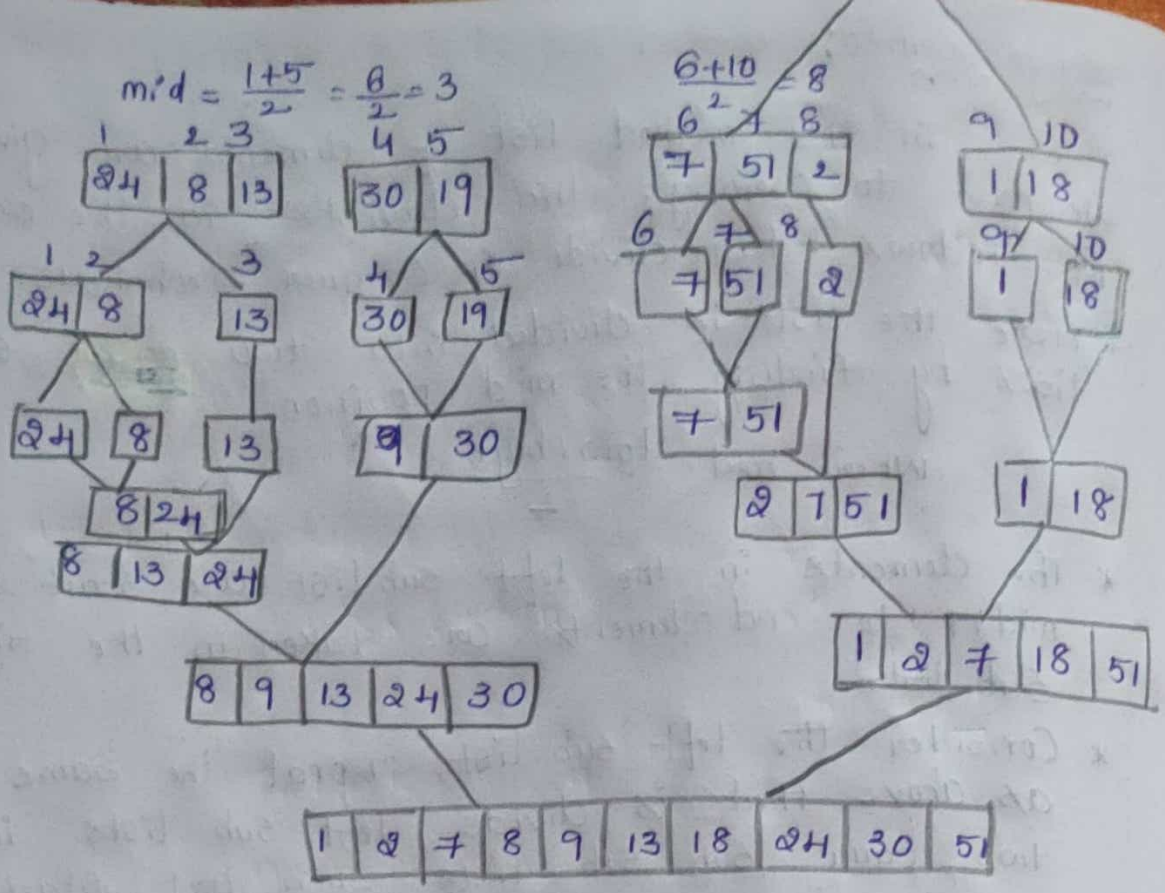
* Here the list is divided into two equal sub lists by finding the mid position.

$$\text{where mid} = \frac{\text{low+high}}{2}$$

* The elements in the left sub list are start to mid, mid+1 to end elements are taken in the right sub list.
* Consider the left sub list repeat the same process as above that is divide left sub lists into two equal sub sub lists. Such that start mid are the elements of the list and mid+1, end are the elements in the left right sub list for each and every sublists the above process is repeated until the sublist contain element.
  (sub. → single.)
* The same procedure is repeated for right sub list also until it is divided into a sub sub list contains a single element.
* Consider the two Adjacent Singleton sub lists, Merge them by arranging the elements in order.
* Repeat the process until the entire list is merged into a single list (The sublists are merged in a manner in which they are divided).

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Ex : | 24 | 8 | 13 | 30 | 19 | 7 | 51 | 2 | 1 | 18 |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |






Algorithm for Merge Sort :

```
Algorithm merge sort (low, high)
                        start  end
{
    if (low < high) then
    {
        mid = (low+high)/2 ;
        Merge sort (low, high) ; ⟶
        Merge sort (mid+1, high) ; ⟶
        Merge (low, mid, high) ; ⟶
    }
}
Algorithm merge (low, mid, high)
{
    h = low ;
    i = low ;
    j = mid+1 ;
while (h ≤ mid) and (j ≤ high)) do
```

```
{
if (a[h] ≤ a[j]) then
 {
   b[i] = a[h]
   h = h+1
 }
 else
 {
    b[i] = a[j] ;
    j = j+1 ;
 }
 i = i+1;
}
if (n>mid) then                    (:: To copy the leftlover
for k=j to high do                    elements in right sublist
{                                     to b array.)
   b[i] = a[k];
   i = i+1;
}
else
 for (k=h to mid do                (:: To copy the left over
 {                                    elements in left sublist
     b[i] = a[k];                     to b array).
     i = i+1;
 }
 For k= low to high do a[k] = b[k];
}.
```

Time Complexity :

Recurssive relation

$$T(n) = \begin{cases} 2T(n/2) + Cn & n \geq 1 \\ 1 & n = 1 \end{cases}$$

By Substitution Method,

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

$= 2(2T(\frac{n}{4}) + (\frac{n}{2}) + cn$

$= 2^2 T(\frac{n}{2^2}) + 2cn$

$T(n/2) = 2T(n/4) + \frac{n}{2}$

$2^n T(n/2^n) + nc$

$\vdots$

$= 2^3 T(\frac{n}{2^3}) + 3cn$

After $k^{th}$ iteration

$= 2^k T(n/2^k) + kcn$

$= 2^k T(1) + kcn$

$= 2^k \cdot 1 + \log_2 n \cdot c \cdot n$

$= n + n \log_2 n \cdot c$

$= c \cdot n \log_2 n$

$= O(n \log_2 n)$

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log_2 n$

Time Complexity in all Cases.

$T(n) = O(n \log_2 n)$

Ex :

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 21 | 15 | 11 | 6 | 5 | 23 | 8 | 9 | 13 |

Merge sort (1, 10)

if (1<10) true

{

mid = $\frac{1+10}{2} = 5.5 = 5$

Merge sort (1, 5) ; → I

Merge sort (6, 10) ; → II

Merge Sort (1, 5, 10); → III

}

I ; if (1<5) true

mid = $\frac{1+5}{2} = \frac{6}{2} = 3$

merge Sort (1, 3) ; → IA

Merge sort (4, 5) ; → IB

Merge (1, 3, 5) ; → IC.

IA :

if (1<3) true

mid = $\frac{1+3}{2} = \frac{4}{2} = 2$

Merge Sort (1,2) ; → IAa

Merge Sort (3,3) ; → IAb

Merge ~~sort~~ (1,2,3) ; → IAC.

IAa : if (1<2) true

mid = $\frac{1+2}{2} = 1.5 = 1$

Mergesort (1,1); → $IAa_1$

Merge sort (2,2); → $IAa_2$

Merge (1,1,2) ; → $IAa_3$.

$IAa_1$ : If (1<1) → false

$IAa_2$ : If (2<2) → false.

$IAa_3$ :

Merge (1,1,2)

{

h=1; i=1, j=2;

while ((1≤1) and (2≤2)) true

{

if (10≤21) → true

{

b[1] = a[1]

b[1] =10

h = 1+1 = 2

}

i=2;

while((2<1) and (2 ≤ 2)) → false

}

if (2>1) → true then

for K=2 to 2

{

b [2] = a[2]

b [2] = 21

}

for K=1 to 2,

a[1] =10

a [2] = 21

IAb :

Merge sort (3,3)

if (3<3) → false

IAC :

Merge (1,2,3)

{

h=1, i=1, j=3; mid = 2

while((1≤2) and 3≤3)) → true

{ if (a[1]≤a[3]) ⇒ if (10≤5) → true.

b[1]=10

h=2, i=2

while((2≤2) and (3≤3) true.

if (a[2]≤ a[3]) ⇒ if (21 ≤ 15) → false

Else

b[i]=a[j] ⇒ b[2]=15

j=4,

i=3

while (2≤2 and 4≤3) → false

}

if (h>mid) → (2>2) false

Else

for k=h to mid

k=2 to 2

b[3] = a[2]

b[3] = 21

for k=low to high ⇒ for k=1 to 3

a[1] = b[1] ⇒ a[1] = 10

a[2] = b[2] ⇒ a[2] = 15

a[3] = b[3] ⇒ a[3] = 21

IB : Merge sort (4,5)

if (4<5) → true

mid = $\frac{4+5}{2}$ = 4.5 = 4

Merge sort (4,4) → IBa

Merge sort (5,5) → IBb

Merge (4,4,5) → IBC.

IBa :

Mergesort(4,4)

if (4<4) false.

IBb :

Mergesort(5,5)

if(5<5)false

IBC :

Merge(4,4,5)

{

h=4, i=4, mid=4, j=5

while (4≤4 and 5≤5) → true

{

if (a[4] ≤ a[5])

if (11≤6] → false

Else

b[4] = a[5]

b[4] = 6;

j = 6,

i=5;

while (4≤4 and 6≤5) false.

}

if (4>4) → false

else

{

for K=4 to 4

b[5] = a[4]

b[5] = 11;

i=6;

}

for K=4 to 5

a[4] = 6

a[5] = 11

IC : Merge (1,3,5)

{

h=1 ; i=1 ; mid=3, j=4, high=5.

while (1≤3 && 4≤5) → true

if (a[1] ≤ a[4]) → if(10≤11) → true.

{

b[2] = a[1]

b[2] = 10;

h = 2;

i = 3;

while (2 ≤ 3 and 5 ≤ 5) → True

if (a[2] ≤ a[5] → if (15 ≤ 11) → false

else

b[3] = a[5]

b[3] = 11

j = 6

i = 4;

while (3 ≤ 3) and (6 ≤ 5) → false.

if (2 > 3) → false

else

for k = 2 to 3

{

b[4] = a[2]

b[4] = 15;

i = 5,

k = 3

b[5] = a[3]

b[5] = 21.

for k = 1 to 5,

a[1] = 6

a[2] = 10

a[3] = 11

a[4] = 15

a[5] = 21

II

Mergesort (6, 10)

(6 < 10) true

mid = $\frac{6+10}{2}$ = 8

Mergesort (6, 8) ⇒ IIA

Mergesort (9, 10) → IIB

Merge (6, 8, 10) → IIC.

IIA mergesort (6, 8)

6 < 8 → True

mid = $\frac{6+8}{2}$ = 7

mergesort (6, 7) → IIAa

Mergesort (8, 8) → IIAb

merge (6, 7, 8) → IIAC

IIAa - mergesort (6, 7)

6 < 7 → true

mid = $\frac{13}{2}$ = 6.5 = 6.

Mergesort (6, 6) → IIAa1

Mergesort (7, 7) → IIAa2

Merge (6, 6, 7) → IIAa3

IIAa1 : if (6 < 6) → false

IIAa2 : if (7 < 7) → false

IIAa3 : merge (6, 6, 7)

{

h=6 ; i=6 , mid = 6, j=7

while (6 ≤ 6 and 7 ≤ 7) → True

if (a[6] ≤ a[7]) → if (5 ≤ 23) → True

{

b[6] = a[6]

b[6] = 5

h = 7

i = 7

}

while (7 ≤ 6 and 7 ≤ 7) → false

if (7 > 6) → true

for K = 7 to 7

{

b[7] = a[7]

b[7] = 23

}

for K = 6 to 7

a[6] = 5

a[7] = 23

IIAb : Mergesort (8,8)

8<8 → false

IIAc :

Merge (6,7,8)

{

h=6, i=6, mid 7, j=8, high = 8.

while (6≤7 and 8≤8) → true

if (a[6]≤a[8]) → if (5≤8) → true

{

b[6] = a[6] ⇒ b[6] = 5

h=7,

i=7,

}

while (7≤7) and (8≤8) → true

if (a[7]≤a[8]) if (23≤8) false

else

{

b[7] = a[8]

b[7] = 8,

j=9

i=8

}

while (7≤7 and 9≤8) false

if (7 > 7) false

else

for k=7 to 7

{

b[8] = a[7]

b[8] = 23

for k=6 to 8

a[6] = 5

a[7] = 8

a[8] = 23

IIB : mergesort (9,10)

if (9<10) → true

mid = $\frac{19}{2}$ = 9

Mergesort (9,9) = IIBa

Mergesort (10,10) → IIBb
Merge (9,9,10) → IIBC.

IIBa : Mergesort (9,9)
if (9<9) false.

IIBb : Mergesort (10,10)
if (10<10) false

IIBC : Merge (9,9,10)
{
h=9, i=9, mid=9, high=10, j=10
while (9≤9 and 10≤10) true
{
if (a[9]≤a[10] → if (9≤13) true
{
b[9]=a[9]
b[9]=9
h=10,
i=10
}
while (10≤9 and 10≤10) false
if (10>9) true
for k=10 to 10
{
b[10]=a[10]
b[10]=13
}
for k=9 to 10
a[9]=9
a[10]=13

IIC
Mergesort (6,8,10)
{
h=8, i=6, j=9, mid=8, high=10
while (6≤8 and 9≤10) true
{
if (a[6]≤a[9]) → if (5<9) true
{
b[6]=a[6]
b[6]=5,

h=7,
i=7,

}

while (7≤8 and 9≤10) true

{

if (a[7]≤a[9] → if (8≤9)

{

b[7]=a[7]

b[7]=8

h=8,

i=8,

}

while (8≤8 and 9≤10) true

{

if (a[8]≤a[9]) & if(23≤9) false

else

{

b[8]=a[9]

b[8]=9

j=10,

i=9

while (8≤8 and 10≤10) true

{

if (a[8]≤a[10] → if(23≤13) false

else

b[9]=a[10]

b[9]=13

j=11,

i=10,

while (8≤8 and 11≤10) false

if(8>8) false

else

for k=8 to 8

{

b[10]=a[8]

b[10]=23

for k=6 to 10

a[6]=5
a[7]=8
a[8]=9
a[9]=13
a[10]=23

III : merge(1,5,10)

{
h=1, i=1, mid=5, j=6, high=10
while((1≤5) and (6≤10)) true
{
if (a[1]≤a[6]) ⇒ if (6≤5) → false
Else
{
b[1]=a[6]
b[1]=5
j=7
i=2,
while (1≤5 and 7≤10) true
{
if (a[1]≤a[7]) → if (6≤8) true
{
b[2]=a[1]
b[2]=6
h=2
i=3
while (2≤5 and 7≤10) true
{
if (a[2]≤a[7]) → if (10≤8) false
else
{
b[3]=a[7]
b[3]=8
j=8
i=4
}
while (2≤5 and 8≤10) true
if (a[2]≤a[8]) ⇒ if (10≤9) false
Else
{

b[4] = a[8]
b[4] = 9
j = 9
i = 5

}

while (2 ≤ 5 and 9 ≤ 10) true

{ if (a[2] ≤ a[9]) → if (10 ≤ 13) true

{

b[5] = a[2]
b[5] = 10
h = 3
i = 6

}

while ((3 ≤ 5) and (9 ≤ 10)) true

if (a[3] ≤ a[9] ⇒ if (11 ≤ 13) true

{

b[6] = a[3]
b[6] = 11
h = 4,
i = 7

}

while (4 ≤ 5 and 9 ≤ 10) true

{ if (a[4] ≤ a[9]) → if (15 ≤ 13) false

else

{

b[7] = a[9]
b[7] = 13
j = 10,
i = 8

while (4 ≤ 5 and 10 ≤ 10) true

{

if (a[4] ≤ a[10]) → if (15 ≤ 23) true

{

b[8] = a[4]
b[8] = 15
h = 5,
i = 9.

}

while (5≤5 and 10≤10) true

{ if (a[5] ≤ a[10]) → if (21≤23) true

b[9] = a[5]

b[9] = 21,

h = 6

i = 10,

}

while(6≤5 and 10≤10) false

if (6>5) true

for k=10 to 10

{

b[10] = a[10]

b[10] = 23

for k=1 to 10

a[1] = 5

a[2] = 6

a[3] = 8

a[4] = 9

a[5] = 10

a[6] = 11

a[7] = 13

a[8] = 15

a[9] = 21

a[10] = 23.

Strassen's Matrix multiplication:

General method:

In order to multiply the two matrices, the no.of columns of the first matrix is equal to the no.of rows of the second matrix. The resultant matrix order is the no.of rows of the first matrix and the no.of columns of the second matrix.

## Algorithm and Time Complexity:

$A_{2\times3} \; ; \; B_{3\times4} \Rightarrow C_{2\times4}$

```
for i=1 to n
    for j=1 to n
        C[i][j]=0
            for k=1 to n
                C[i][j]=C[i][j]+a[i][k]*b[k][j]
```

Time Complexity of matrix multiplication is $O(n^3)$

⇒ To reduce the time Complexity we use divide and conquer technique.

## Divide and Conquer:

$n \times n$

→ n as the power of 2.

2x2, 4x4, 8x8, 16x16 ---

Ex: 4x4

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right]_{4\times4}$$

$n \times n$ ⇒ $n/2 \times n/2$

$$B = \left[\begin{array}{cc|cc} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \hline b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{array}\right]_{4\times4}$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}\times b_{11} + a_{12}\times b_{21} & a_{11}\times b_{12} + a_{11}\times b_{12} \\ a_{21}\times b_{21} + a_{22}\times b_{22} & a_{21}\times b_{12} + a_{22}\times b_{11} \end{bmatrix}$$

$$T(n) = \begin{cases} 8T(n/2) + O(n^2) & n \geq 2 \\ \text{Constant} & n \leq 2 \end{cases}$$

$T(n) = 8T(n/2)$

$= 8 \cdot 8 (T(n/4))$

$= 8^2 \cdot 8 (T)(n/8)$

$= 8^3 T(n/2^3)$

$\vdots$

$= 8^k T(n/2^k)$

$= 8 \log_2 n \; C = n \log_2 8 = n \log_2 2^3$

$= n 3 \log_2 2$

$= n^3$

$T(n/2) = 8T(n/4)$

$T(n/4) = 8T(n/8)$

$a^{\log_c b} = b^{\log_c a}$

$n = 2^k$

$k = \log_2 n$

Stressen's method :

$P = (A_{11} + A_{22})(B_{11} + B_{22})$

$Q = (A_{21} + A_{22})(B_{11})$

$R = A_{11}(B_{12} - B_{22})$

$S = A_{22}(B_{21} - B_{11})$

$T = (A_{11} + A_{12}) B_{22}$

$U = (A_{21} - A_{11})(B_{11} + B_{12})$

$V = (A_{12} - A_{22})(B_{21} + B_{22})$

Ex: $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

$= \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$

→ multiplications are 7

Here we are using divide and Conquer method along with strassen's methods.

$T(n) = 7T(n/2) + O(n^2)$

$T(n) = 7T(n/2)$

$= n \log_2 7$

$= n^{2.8}$

$= O(n^{2.8})$