UNIT-2
DIVIDE AND CONQUER

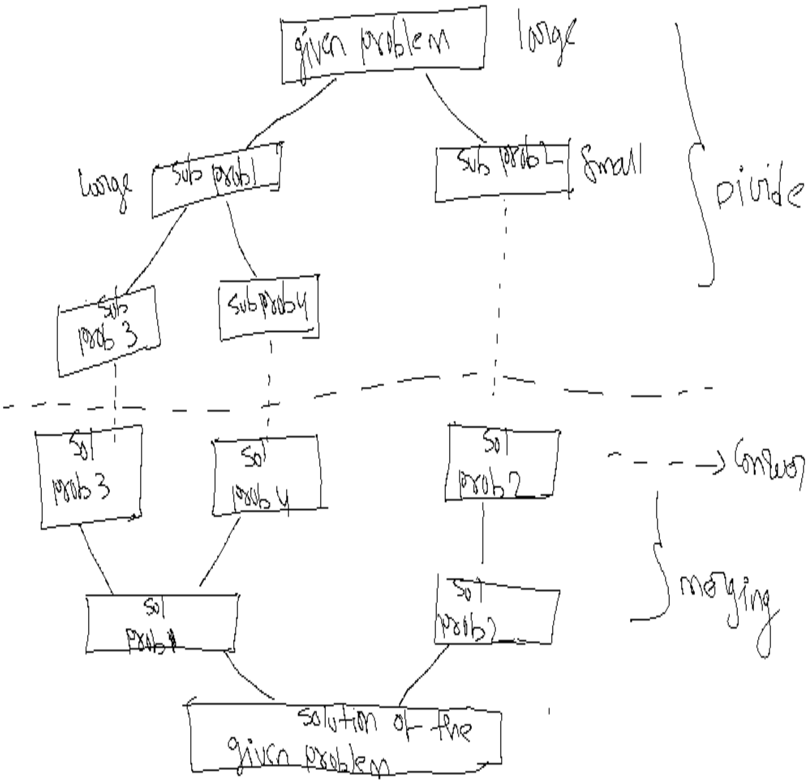
If the problem is large then we can divide the problem in to small sub problems (same size)
This process is continues until the problem becomes small. After than we will recursively find out the solutions of each and every sub problem?

Finally combine the solutions of each and every sub problem then we will get the solution of the given problem.

The principle behind the divide and conquer strategy is that it is easier to solve several small problems than one large problem.

The divide and conquer strategy involves 3 steps
1.divide
2.conquer
3.combine

1.divide: divide the problem in to small pieces(sub-problems) of same size
2.conquer: finding the solutions of a each and every sub-problems recursively.
3.combine: combining the solutions of each and every sub-problmes. Then we will get the solution of a given problem.



Algorithm:

Algorithm dc (p)
{
If problem is small then
Return the solution of a given problem
Else
{
Divide (p) and obtain p1,p2,p3.....pn
Where n>=1
Apply conquer(dc) to find the solutions each sub-problem
Return combine(dc(p1),dc(p2),dc(p3)....dc(pn))
}
}

The computing time of above procedure of divide and conquer strategy is given by the recurrence relation.

T(n)= g(n)--------------------------if problem is small
T(n1)+T(n2)+T(n3)+......+T(nr)+f(n)------when problem is large

Where T(n) is the time for divide and conquer size n.
G(n) is the computing time required to solve small problem
F(n) is the time required in dividing the problem and combing the solutions of a sub-problems.

21-12-2021
Recurrence Relation

In order to analyse the time complexity of recursive function we have 3 methods
1.Back substitution method
2.recursive(decision) tree method
3.master theorem method

Ex:
Fun()
{
Stat1
Stat2
....
;;;
Fun()
}

Steps1: in order to analyse the recursive program we need to write recurrence relation
Step2: by solving that recurrence relation we will the time complexity of the recurrence function

Ex:

```
void test(n)----------------------T(n)
{
If(n>0)-------------------------------1
{
Print("%d",n)---------------------1
Test(n-1);--------------------------T(n-1)
}
}
```

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

Sol:

Step1: write down the recurrence relation , the recurrence relation is the sum of all the lines

T(n)=T(n-1)+1   n>0
1   n=0

step2: solve the recurrence relation by using back substitution method

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

Substitute equation 2 in equation 1
T(n)=T(n-1)+1
=[T(n-2)+1]+1
T(n)=T(n-2)+2------------------------------4
Substitute equation 3 in equation 4

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

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

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

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

....

....

$= T(n-k)+k$

When it will stop the $n-k=0$

$N=k$

Step1:- $T(n) = \begin{cases} T(n-1)+1 & n>0 \\ 1 & n=0 \end{cases}$

Step2:- solve the Recurrence Relation by Back substitution method

$T(n) = T(n-1)+1$ —— (1)

$T(n-1) = T(n-2)+1$ —— (2)

$T(n-2) = T(n-3)+1$ —— (3)

substitute equation (2) in equation (1)

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

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

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

$T(n) = T(n-2)+2$ —— (4)

substitute equation (3) in equation (4)

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

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

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

$\vdots$

$= T(n-k)+k$

when it will stop $n-k=0$

$n=k$ [substitute $n=k$]

$= T(n-n)+n$

$= T(0)+n$

$T(n) = 1+n \Rightarrow \therefore$ Time complexity $= O(1+n)$

$= O(n)$

Ex2:

```
Void Test(n)--------------------------------T(n)
{
If(n>0)----------------------------------------1
{
For(i=0;i<n;i++)-----------------------------n+1
{
Printf("%d",n)--------------------------------n
}
Test(n-1)-------------------------------------T(n-1)
}
}
```

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

Sol:

Step1: we need to find out the recurrence relation for the program

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

Step2: solve the recurrence relation using back substitution method

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

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

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

Substitute equation 2 in equation 1

$T(n)=T(n-1)+n$ ------------- we know T(n-1) values

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

$=T(n-2)+(n-1)+n$------------------------4

Substitute equation 3 in equation 4

$T(n)=T(n-2)+(n-1)+n$ ------------- we know T(n-2) value

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

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

....

,.............................if it is k time k=3

Assume k=3 , 2=k-1, 1=k-2

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

When it is stop the condition n-k=0-----→n=k

Now k=n is substituted in the above equation

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

$=T(0)+(n-n+1)+(n-n+2)+n$

$T(n) \quad =1+1+2+n$

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

$= 1+\underbrace{1+2+3 \cdots +(n-1)+n}_{\to \frac{n(n+1)}{2}}$

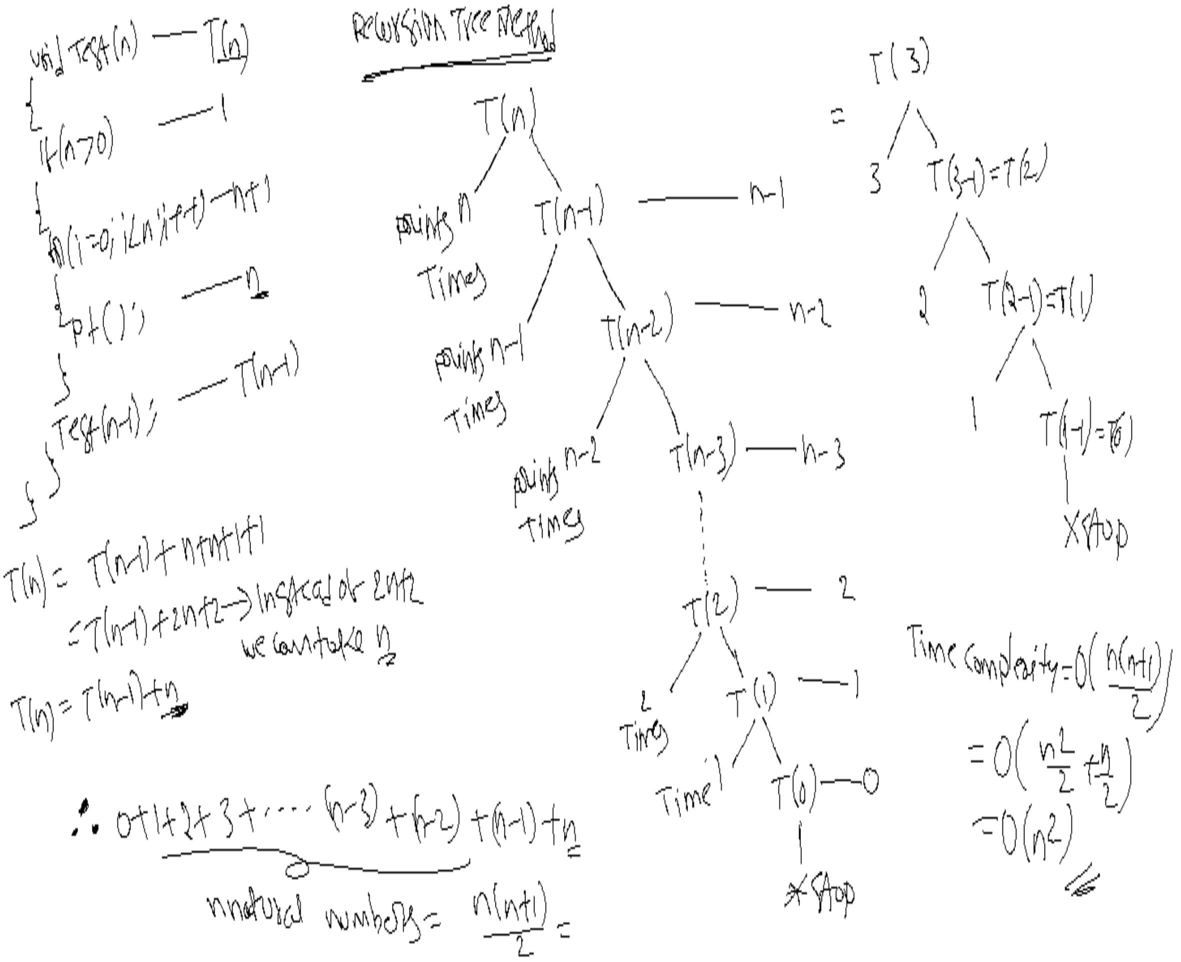
$T(n)= 1+\frac{n(n+1)}{2}$

$\therefore$ Time Complexity $= O\left(1+\frac{n(n+1)}{2}\right)$

$= O\left(\frac{n^2}{2}+\frac{n}{2}\right)$

$= O(n^2)$

Recursive Tree method:



Ex3:

```
Void Test(n)----------------------------T(n)
{
If(n>0)------------------------------------1
{
Printf("%d",n)--------------------------1
Test(n/2);--------------------------------T(n/2)
}
}
```

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

=T(n/2)+2

T(n)=T(n/2)+1 where n>0 problem is large

1 N=0 problem is small

Sol:

Step1: we need to find out the recurrence relation for the recursive function

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

Step2: using back substitution method we can solve the recurrence relation

T(n)=T(n/2)+1---------------------------------1
T(n/2)=T(n/4)+1-------------------------------2
T(n/4)=T(n/8)+1-------------------------------3

Substitute equation 2 in equation 1

T(n)=T(n/2)+1 --------------we know T(n/2) values
[T(n/4)+1]+1
=T(n/4)+2---------------------------------4

Substitute equation 3 in equation 4

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

T(n)=[T(n/8)+1]+2
=T(n/8)+3

$T(n) = T(n/8) + 3$

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

⋮

if it's K times

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

let $2^K = n \Rightarrow K = \underline{\log_2 n}$

$T(n) = T(n/n) + \log_2 n$

$T(n) = T(1) + \log_2 n \Rightarrow 1 + \log_2 n$

$\therefore$ Time complexity $= O(1 + \log_2 n)$

$= O(\log_2 n)$

Recursion Tree method:

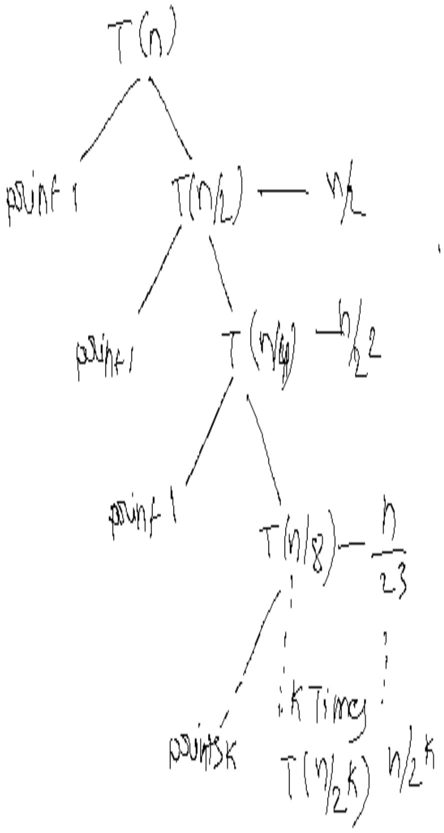
Recursion Tree method

```
void Test(n)        — T(n)
{
  if(n>1)           — 1
  {
    print("%d",n)   — 1
    Test(n/2)       — T(n/2)
  }
}
```

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

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

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



when it will stops the condition

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

$\Rightarrow \therefore k = \log_2 n$

it will stop at k level

$\therefore$ Time Complexity $= 1 \times k$ Times

$= 1 \times \log_2 n$

$= O(\log_2 n)$

24-12-2021

Ex:

```
Void Test(n)------------------------------------------T(n)
{
If(n>1)-----------------------------------------------1
{
For(i=0;i<n;i++)--------------------------------------n+1
{
Printf("%d",n)----------------------------------------n
}
Test(n/2)---------------------------------------------T(n/2)
Test(n/2)---------------------------------------------T(n/2)
}
}
```

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

Instead of 2n+2 we can take n

$$T(n)=\begin{cases} 2T(n/2)+n & \text{where } n>1 \text{ large problem} \\ 1 & n=1 \text{ small problem} \end{cases}$$

Sol:

Step1:we need to find out the recurrence relation for the above program

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

Step2: solve the recurrence relation using back substitution method

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

$T(n/2)=2T(n/4)+n/2$--------------------------------2

$T(n/4)=2T(n/8)+n/4$--------------------------------3

Substitute equation 2 in equation 1

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

$T(n) = 2T(n/2) + n$ —— 1

$T(n/2) = 2T(n/4) + n/2$ —— 2

$T(n/4) = 2T(n/8) + n/4$ —— 3

Substitute equation 2 in equation 1

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

$T(n) = 2\left[2T(n/4) + n/2\right] + n$

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

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

$T(n) = 4T(n/4) + 2n$ —— 4

Substitute equation 3 in equation 4

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

$= 4\left[2T(n/8) + n/4\right] + 2n$

$= 8T(n/8) + \cancel{4} \times \frac{n}{\cancel{4}} + 2n$

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

$T(n) = 8T(n/8) + 3n$

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

$\vdots$

If it k times $K = 3$

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

let $2^K = n$

$K = \log_2 n$

Substitute $2^K = n$

$T(n) = n\,T(n/n) + Kn$

$= n\,T(1) + n \log_2 n$

$= n(1) + n \log_2 n$

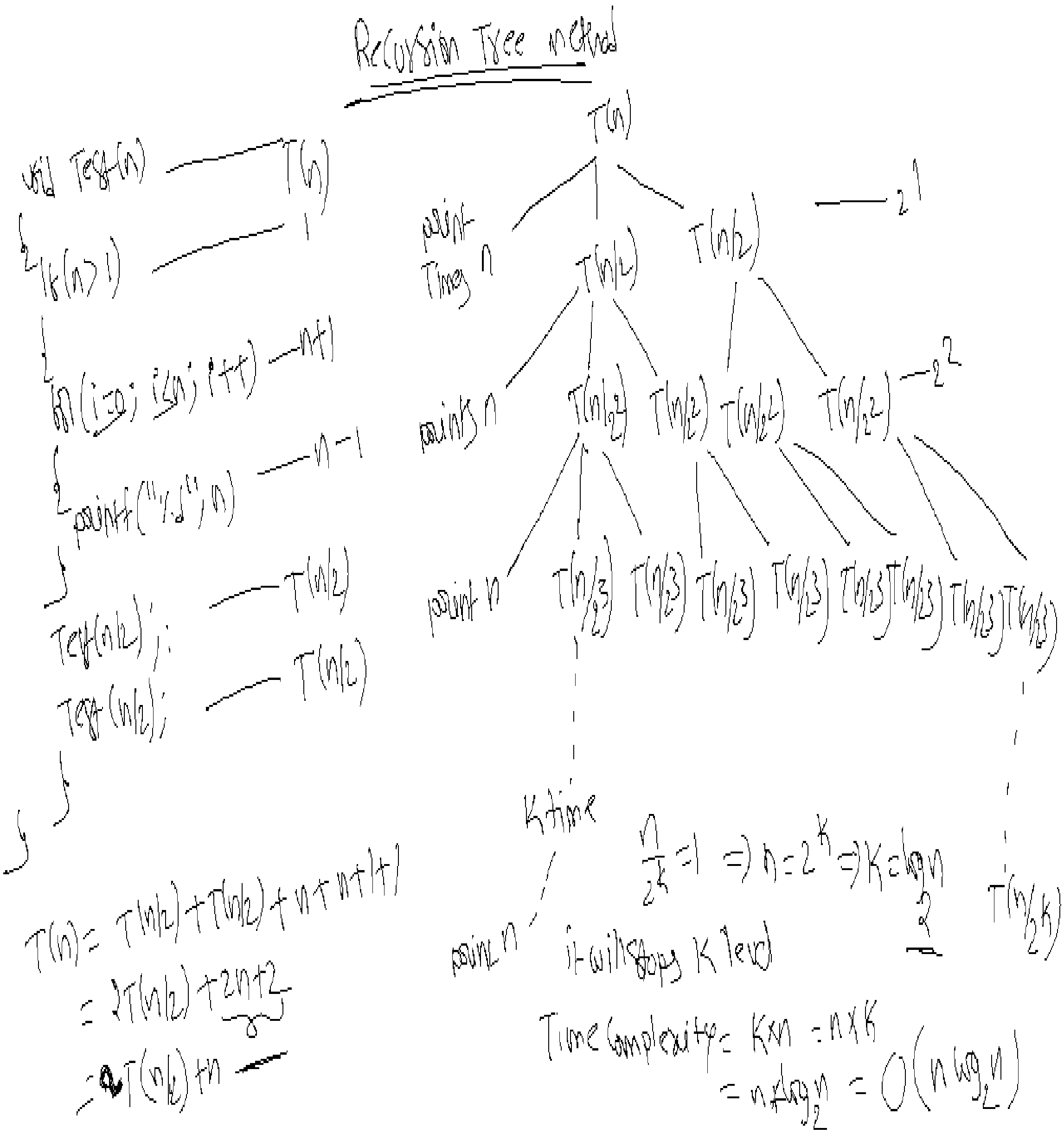
$= n + n \log_2 n$

$\therefore$ Time Complexity =

$O(n + n \log_2 n)$

$O(n \log_2 n)$

Recursion Tree Method



Ex:

```
Void Test(n)---------------------------------T(n)
{
If (n>0)-----------------------------------------1
{
For(i=0;i<n;i=i*2)-----------------------------n+1
{
Printf(%d",n)------------------------------------logn
}
Test(n-1)-----------------------------------------T(n-1)
}
}
```

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

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

T(n)=T(n-1)+ log n where n>0 large problem

1 where n=0

Sol:

Step1: we need to find out the recurrence relation for the above program

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

Step2: solve the recurrence relation using back substitution method

$T(n) = T(n-1) + \log n$ —— ①

$T(n-1) = T(n-2) + \log(n-1)$ —— ②

$T(n-2) = T(n-3) + \log(n-2)$ —— ③

Substitute equation ② in equation ①

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

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

$= T(n-2) + \log(n-1) + \log n$ —— ④

Substitute equation ③ in equation ④

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

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

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

⋮

k times $k=3,\ 2 = k-1,\ 1 = k-2$

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

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

when it will stops the condition $n-k=0$

$n = k$ Substitute $k = n$ value

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

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

$= 1 + \log 1 + \log 2 + \log n$ $\quad \therefore T(n) = O(n\log n)$

$= 1 + \log[1+2+\cdots n]$

$= 1 + \log n! \Rightarrow 1 + \log n^n \Rightarrow 1 + n\log n$

Recursion Tree method:

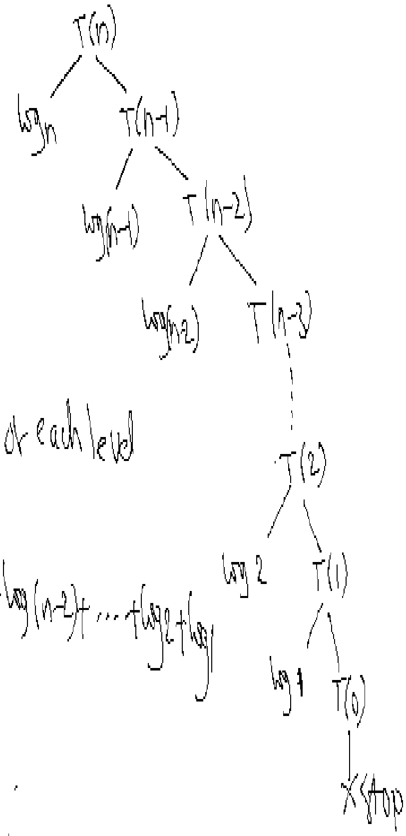
Recursion Tree method

```
void Test(n)              —— T(n)
{
  if(n>1)                 —— 1
  {
    for(i=0;i<n;i=i*2)    —— n+1
    {
      printf("Hello");    —— log n
    }
    Test(n-1)             —— T(n-1)
  }
}
```

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

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

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



∴ Total time is sum of each level

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

$\quad = \log n!$

$\quad = \log n^n$

∴ Time complexity $= O(n \log n)$

27-12-2021
Master Theorem

Master theorem can be used find the time complexity for the recurrence relation.

```
begin T(n)
{
 if(n>1)
 {
   printf("Hello");
   printf("Hi");      } f(n)

   T(n/b)
   T(n/b)             } 3T(n/b)
   T(n/b)
 }
}
```

The Recurrence Relation for the above program

$T(n) = 3T(n/b) + f(n)$

$T(n) = aT(n/b) + f(n)$ where $a \geq 1, b > 1$

where

$n \rightarrow$ is the size of the problem

$a \rightarrow$ is the no. of sub-problem

$n/b \rightarrow$ size of the sub-problem

$f(n) \rightarrow$ cost of the workdone other than Recursive calls

Master Theorem:

<u>master theorem</u>

$$T(n) = aT(n/b) + \theta(n^k \log^p n)$$

for $a \geq 1$, $b > 1$, $k \geq 0$ and p is Real number

Case 1:- $a > b^k$

then $T(n) = \theta(n^{\log_b a})$

Case 2:- $a = b^k$

a) If $p > -1$ then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$

b) If $p = -1$ then $T(n) = \theta(n^{\log_b a} \log(\log n))$

c) If $p < -1$ then $T(n) = \theta(n^{\log_b a})$

Case 3 $a < b^k$

a) If $p \geq 0$ then $T(n) = \theta(n^k \log^p n)$

b) If $p < 0$ then $T(n) = \theta(n^k)$

<u>Note</u>:- All the Recurrence Relations can't be solving by using master theorem. It is applicable when $a \geq 1$, $b > 1$ and $f(n)$ is a positive function.

Ex1: $T(n)=9T(n/3)+n$ find the time complexity using master theorem

Sol:

Ex $T(n)=9T(n/3)+n$

Sol:- check whether $a \geq 1, b > 1$

$a=9, b=3, K=1, P=0$

$9 \geq 1, 3 > 1$ all the conditions are true

Now we can solve by using master theorem

case 1:- $a > b^K$

$9 > 3^1 \Rightarrow 9 > 3$ — True

then $T(n) = \Theta(n^{\log_b a})$

$\therefore$ Time Complexity $= \Theta(n^{\log_3 9})$

$= \Theta(n^{\log_3 3^2})$ $\left[\because \log_a a^b \Rightarrow b \log_b a\right]$

$= \Theta(n^{2 \log_3 3})$ $\left[\because \log_a a = 1\right]$

$\therefore T(n) = \Theta(n^{2 \times 1})$

$T(n) = \Theta(n^2)$

$\therefore$ Time Complexity $= O(n^2)$

Ex2: T(n)=3T(n/4)+cn2 find the time complexity using master theorem

Sol: a=3, b=4, k=2 and p=0

Check whether a>=1, b>1,K>=0,p>=0

3>=1,4>1,2>=0 all the conditions are true so we can solve the recurrence relation using master theorem.

Q $T(n) = 3T(n/4) + cn^2$

Sol $a=3, b=4, K=2, p=0$

check whether $a \geq 1, b > 1, K \geq 0, p \geq 0$

$3 \geq 1, 4 > 1, 2 \geq 0$ all the conditions are True

now we can solve by using master theorem

case1:- $a > b^K$

$3 > 4^2 \Rightarrow 3 > 16$ — False

case2:- $a = b^K$

substitute a, b value here

$3 = 4^2 \Rightarrow 3 = 16$ — False

case3:- $a < b^K$

substitute a, b, k values here

$3 < 4^2 \Rightarrow 3 < 16$ — True

a) If $p \geq 0$ then $T(n) = \theta(n^K \log_n^p)$

$0 \geq 0$ True

$\therefore$ Time complexity $= \theta(n^K \log_n^p)$

$= \theta(n^2 \log_n^0) \Rightarrow \theta(n^2 \times 1) \Rightarrow \theta(n^2)$

$\therefore$ Time complexity $= \theta(n2)$

Ex4: T(n)= 3T(n/4)+n log n find the time complexity using master theorem.

Sol: a=3, b=4, k=1, p=1 check whether the a>=1, b>1

3>=1, 4>1, 1>=0 all the conditions are True now we can solve by using master theorem.

Q1 $T(n) = 3T(n/4) + n\log n$

Sol $a=3, b=4, K=1, p=1$

$3 \geq 1, 4 > 1$ True

Case1:- $a > b^K$

$3 > 4^1 \Rightarrow 3 > 4$ — False

Case2:- $a = b^K$

$3 = 4^1 \Rightarrow 3 = 4$ — False

Case3:- $a < b^K$

$3 < 4^1 \Rightarrow 3 < 4$ — True

a) If $p \geq 0$ then $T(n) = \Theta(n^K \log n^p)$

$1 \geq 0$ True

$\therefore T(n) = \Theta(n^1 \log n^1)$

$= \Theta(n \log n)$

$\therefore$ Time Complexity $= \Theta(n\log n)$

Find the Time Complexity by using Master Theorem

Ex4: T(n)=T(n/2)+1
Ex5:T(n)=T(2n/3)+1
Ex6:T(n)=T(n/2)-nlog n
Ex7:T(n)=0.5T(n/2)+n2
Ex8:T(n)=3T(n/2)+n2
Ex9:T(n)=4T(n/2)+n pow 2 log n
Ex10:T(n)=5T(n/2)+n Log n pow2
Ex11: T(n)=3T(n/2)-n2 log n2
Ex12:T(n)=1/4 T(n/2)+n

Applications:

1.Binary Search
2.Quick Sort
3.Merger Sort
4.Strassen's matrix multiplication

1. Binary Search:

We can search the element in the sorted array either in assending order or in decending order. First we can check whether the problem is small or larger.

If the problem is large we can divide the problem in to two sub-problems of equal size. Then two sub-problems are created.

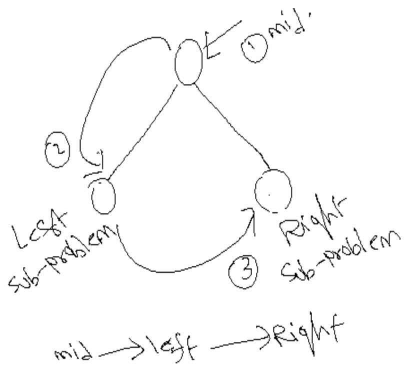
First we will check the searching element is in the left hand side Using **if(x<a[mid])** searching goes to left hand side.

Else searching goes to right hand side using the condition
If(x>a[mid])

Else searching goes to mid point by using the condition
If(x=a[mid]) means searching is in the mid point.
There are two types of binary search
1.Recursive Binary Search
2.Iterative Binary Search

1.Recursive Binary Search: in the recursive binary search first searching goes to middle point. Next searching goes to left hand side. Finally searching goes to right hand side.

Mid-----→left------→right



Algorithm:

Algorithm binarysearch(a,i,low,high,x)
{

If(low=high) ------------------**problem is small**
{
If(x=a[i]) then return i;
Else return 0
}
Else --------------------------------**problem is large**
{
//reduce the problem in to small sub-problems
Mid:=[(low+high)/2]

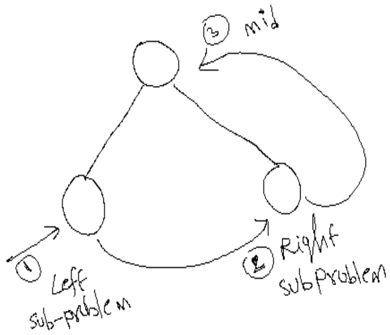
// search goes to middle point
If(x=a[mid]) then return mid;

//searching goes to left hand side
Else if(x<a[mid]) then
    Return binarsearch(a,i,mid-1,x)

//searching goes to right hand side
Else if(x>a[mid]) then
 Return binarysearch(a,i,mid+1,x)
}
}

2.Iterative Binary Search Algorithm: in iterative binary search first searching goes to left sub-problem, next searching goes to right sub-problem, finally searching goes to mid point.

Left----------→right------------→mid



Algorithm:

Algorithm binarysearch(a[],i,low,mid,high,x)
{
Low:=0, high:=n
While(low<=high)
{
//divide the problem in to two sub-problems
Mid:=(low+high)/2;

//searching goes to left sub-problem
If(x<a[mid]) then high:=mid-1
Retrun binarysearch(a,i,low,mid-1,x);

//searching goes to right sub-problem
Else if(x>a[mid]) then low:=mid+1
Return binarysearch(a,i,mid+1,high,x)

//searching goes to mid
Else Return mid;
}
Return 0;
}

Ex: array contains a=

1	3	4	6	7

Searching the element 3 in an array using Iterative binary search algorithm.

Sol:

Sol:
Algorithm BS(a,i,low,mid,x)
{ low=0; high=n
while(low<high)
{ mid=$\frac{low+high}{2}$;
if(x<a[mid]) then high=mid-1
return BS(a,i,low,mid-1,x)
else if(x>a[mid]) then low=mid+1;
return BS(a,i,mid+1,high,x)
else return mid.
}
return 0;
}

Sol [1|3|4|6|7] x=3
0 1 2 3 4

(1) low=0, high=4
while(0 ≤ 4) — True
{ mid= $\frac{0+4}{2}=\frac{4}{2}=2$
If(x<a[mid])
If(3<a[2]) => 3<4 True then high=2-1=1
means we can consider only left sub-problem
[1|3]
0 1

(2) low=0, high=1, x=3
-> while(0 ≤ 1) True
mid= $\frac{0+1}{2}=\frac{1}{2}=0.5=0$

-> If(x<a[mid])
If(3<a[0])
If(3<1) False
-> next else If(x>a[mid])
If(3>a[0])
If(3>1) True
then low=mid+1
=0+1=1
means we can consider only Right sub problem
[3]
1

$\boxed{3}$
1

(3) low = 1, high = 1, x = 3

→ while(low ≤ high)

(1 ≤ 1) — True

→ next mid = $\left(\frac{low+high}{2}\right) = \frac{1+1}{2} = \frac{2}{2} = 1$

$\boxed{3}$
1
↑mid

→ If (x < a[mid])

(3 < a[1]) → 3 < 3 — False

→ else If (x > a[mid])

(3 > a[1]) → 3 > 3 — False

→ else return mid → return 3

∴ the Searching element is found at 1st index in an Array.

Ex2: 40,11,33,37,42,45,99,100 are the elements in an array and searching the element 99 in an array using Recursive Binary Search algorithm.

Sol:

Sol $A = \boxed{40 \mid 11 \mid 33 \mid 37 \mid 42 \mid 45 \mid 99 \mid 100}$

0 1 2 3 4 5 6 7 low = 0, high = 7, x = 99

① If (low = high)

(0 = 7) False means problem is large

it executes else block

→ $mid = \left(\frac{low + high}{2}\right) = \frac{0+7}{2} = \frac{7}{2} = 3.5 = 3$

→ If (x = a[mid])

x = a[3] ⇒ 99 = 37 False

→ else If (x < a[mid])

If (99 < a[3] ⇒ If (99 < 37) False

→ else If (x > a[mid])

If (99 > a[3]) ⇒ 99 > 37 True

means we can consider only Right sub-problem

$\boxed{42 \mid 45 \mid 99 \mid 100}$

4 5 6 7

low high

② If (low = high) → (4 = 7) False means problem is large

→ $mid = \left(\frac{low + high}{2}\right) = \frac{4+7}{2} = \frac{11}{2} = 5.5 = 5$

$\boxed{42 \mid 45 \mid 99 \mid 100}$

4 5 6 7

low mid high

42	45	99	100
4	5	6	7

low ↑4   mid ↑5   ↑high 7

→ If ($x = a[mid]$)

$(99 = a[5]) \Rightarrow (99 = 45)$ False

→ else If ($x < a[mid]$)

If $(99 < a[5]) \Rightarrow$ If $(99 < 45)$ False

→ else If ($x > a[mid]$)

If $(99 > a[5]) \Rightarrow$ If $(99 > 45) =$ True

means element is in Right sub-problem

we can consider only Right sub-problem

99	100
6	7

low 6   7 ↑high

③ low = 6, high = 7   x = 99

If (low = high)

If (6 = 7) False problem is large

→ $mid = \left(\frac{low+high}{2}\right) = \left(\frac{6+7}{2}\right) = \frac{13}{2} = 6.5 = 6$

99	100
6	7

low ↑6 ↑mid   7 ↑high

→ If ($x = a[mid]$)

If $(99 = a[6]) \Rightarrow$ If $(99 = 99)$ TRUE

∴ we are finding the element in 6th Index in an array

31-12-2021

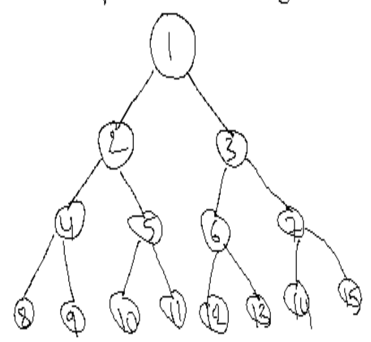
Analysis of Binary Search Algorithm:

We need to find out the time complexity of an algorithm using recurrence relation.

→ If the problem is large then we can divide the problem into two sub-problem (Left and Right)

→ If the element is less than mid point then consider left sub-problem again apply Binary Search

→ If the element is greater than mid then consider Right sub-problem again apply Binary Search



Recurrence Relation for Binary Search algorithm

$T(n) = T(n/2) + 1$ —— (1)

$T(n/2) = T(n/4) + 1$ —— (2)

$T(n/4) = T(n/8) + 1$ —— (3)

Substitute equation (2) in equation (1)

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

$T(n) = [T(n/4) + 1] + 1$

$T(n) = T(n/4) + 2$ —— (4)

Substitute equation (3) in equation (4)

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

$T(n) = [T(n/8) + 1] + 2$

$T(n) = T(n/8) + 3$

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

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

If it is K times

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

The process is continues until problem becomes small

$$\frac{n}{2^K} = 1 \Rightarrow n = 2^K \Rightarrow 2^K = n \Rightarrow K = \log_2 n$$

$$T(n) = T(n/n) + \log_2 n$$

$$T(n) = T(1) + \log_2 n \Rightarrow 1 + \log_2 n$$

$$\therefore \text{Time Complexity} = O(1 + \log_2 n)$$
$$= O(\log_2 n)$$

Quick Sort Algorithm:

Quick sort using divide and conquer strategy. In this algorithm division is dynamically carried out. Is contains three parts.

1.Divide: if the problem is large then we can divide the problem in to two sub-problems. The process is continues until problem becomes small.

2.conquer: after dividing the problem we can recursively sort the two sub-problems. Recursively find the solution for each and every sub-problems.

3.combine: combine all the solutions of the sub-problems.

Quick sort will check 3 conditions. And we can take 2 pointers i, j, and we can select pivot element in an array.

1.while(a[i]<=pivot) then i=i++

2.while(a[j]>=pivot) then j=j--

3.if(i<j) the swap a[i] with a[j]

else swap a[j] with pivot element

*the first element we can consider as pivot element,
*i= low in an array
*j=high in an array
*i pointer always moving from left hand side right hand side
*j pointer always moving from right hand side to left hand side
* after first portioning the elements which is less then pivot element consider as one sub-problem which is placed at left hand side of the pivot element.
* the elements which is greater than the pivot elements those are placing at right hand side.

Algorithm for partition:

```
Algorithm partition(a,low,high)
{
//first element in the array is assumed as pivot element
Pivot:=a[low]
I:=low
J:=high
While(low<high)
{
While(a[i]<=pivot) then i=i++
While(a[j]>=pivot) then j=j--
If(i<j) the
Swap a[i] with a[j]
Else swap a[j] with pivot
}
A[low]=a[high]
A[high]=pivot
Return high
}
```

Algorithm for Quicksort:

Algorithm for quicksort(a,low,high)
{
If(low<high)
{
mid:=partition(a,low, high)
//recursively sort the sub-arrays
Quicksort(low,mid-1)
quicksort(mid+1,high)
}
}

EX: 26,5,37,1,61,11,59,15,48,19 sort the elements using quick sort
Sol:

26,5,37,1,61,11,59,15,48,19

Sol

26	5	37	1	61	11	59	15	48	19
0	1	2	3	4	5	6	7	8	9

pivot ↑low ↑high

pivot=26, i=0, j=9

→ while(low<high)
(0<9) True

① while (a[i]<=pivot)
(a[0]<=26) => (26<=26) True
then i=i+1=0+1=1

26	5	37	1	61	11	59	15	48	19
0	1	2	3	4	5	6	7	8	9

pivot ↑i=1 ↑j

i=1, j=9

Again same while loop is executed until condition becomes false

① while(a[i]<=pivot)
(a[1]<=26) => (5<=26) True
then i=i+1=1+1=2

26	5	37	1	61	11	59	15	48	19
0	1	2	3	4	5	6	7	8	9

pivot ↑i ↑j

i=2, j=9

① while(a[i]<=pivot)
(a[2]<=26) => (37<=26) False
Control goes to next while statement

② while(a[j]>=pivot)
(a[9]>=26) => (19>=26) False
Condition false means control goes to next statement

③ if(i<j)

if(2<9) True then

swap a[i] with a[j]

26	5	37	1	61	11	59	15	48	19
0	1	2	3	4	5	6	7	8	9
↑ pivot		↑ i							↑ j

26	5	19	1	61	11	59	15	48	37
0	1	2	3	4	5	6	7	8	9
↑ pivot		↑ i							↑ j

i=2, j=9

① while(a[i]<=pivot)

(a[2]<=26) => (19<=26) True

then i=i++ => i+1 => 2+1=3

26	5	37	1	61	11	59	15	48	19
0	1	2	3	4	5	6	7	8	9
↑ pivot			↑ i						↑ j

i=3, j=9

① while(a[i]<=pivot)

(a[3]<=26) => (1<=26) True

then i=i++ = i+1 = 3+1=4

26	5	37	1	61	11	59	15	48	19
0	1	2	3	4	5	6	7	8	9
↑ pivot				↑ i					↑ j

① while(a[i]<=pivot)

(a[4]<=26) => (61<=26) False

then control goes to next while block

i=4, j=9

② while(a[j]>=pivot)

(a[9]>=26) => (37>=26) True

j=j-1=9-1=8

26	5	19	1	61	11	59	15	48	37
0	1	2	3	4	5	6	7	8	9
↑p				↑i				↑j	

② while (a[j]>=pivot)

(a[8]>=26) => (48>=26) True

then j=j-1=8-1=7

26	5	19	1	61	11	59	15	48	37
0	1	2	3	4	5	6	7	8	9
↑p				↑i			↑j		

② while(a[j]>=pivot)

(a[7]>=26) => (15>=26) False

Control goes next condition

③ if(i<j) => (4<7) True

then Swap a[i] with a[j]

26	5	19	1	61	11	59	15	48	37
0	1	2	3	4	5	6	7	8	9
↑p				↑i			↑j		

26	5	19	1	15	11	59	61	48	37
0	1	2	3	4	5	6	7	8	9
↑p				↑i			↑j		

① while(a[i]<=pivot)

(a[4]<=26) =>

(15<=26) True then

i=i+1=4+1=5

26	5	19	1	15	11	59	61	48	37
0	1	2	3	4	5	6	7	8	9
					↑i	↑j			

① while (a[i]<=pivot)

(a[5]<=26) => (11<=26) True

then i=i+1=5+1=6

26	5	19	1	15	11	59	61	48	37
0	1	2	3	4	5	6	7	8	9
↑p						↑i	↑j		

① while(a[6]<=26)

(59<=26) False

Control goes to next

② while(a[j]>=pivot)

(a[7]>=26)

(61>=26) True

then j=j-1=7-1=6

26	5	19	1	15	11	59	61	48	37
0	1	2	3	4	5	6	7	8	9
↑p						↑i ↑j			

② while(a[6]>=26)

(59>=26) True

j=j-1=6-1=5

26	5	19	1	15	11	59	61	48	37
0	1	2	3	4	5 ↑j	6 ↑i	7	8	9

② while(a[j]>=pivot)

(a[5]>=26) => (11>=26)

False

③ if(i<j)

(6<5) False

Swap a[j] with pivot

26	5	19	1	15	11	59	61	48	37
0	1	2	3	4	5	6	7	8	9
↑pivot					↑j	↑i			

11	5	19	1	15	26	59	61	48	37
0	1	2	3	4	5	6	7	8	9

Left subproblem

Right

Left sub-problem

0	1	2	3	4
11	5	19	1	15
pivot → ↑p				↑j

→ while(low<high)

(0<4) True

① while(a[i]<=pivot)

(a[0]<=11)

(11<=11) True

then i=i+1=0+1=1

0	1	2	3	4
11	5	19	1	15
p↑	↑i			↑j

① while(a[i]<=pivot)

(a[1]<=11)

(5<=11) True

then i=i+1=1+1=2

0	1	2	3	4
11	5	19	1	15
↑p		↑i		↑j

① while(a[i]<=pivot)

(a[2]<=11)

(19<=11) False

Control goes to next block

② while(a[j]>=pivot)

(a[4]>=11)

(15>=11) True

then j=j-1=4-1=3

0	1	2	3	4
11	5	19	1	15
↑p		↑i	↑j	

② while(a[j]>=pivot)

(a[3]>=11)

(1>=11) False

③ if(i<j)

(2<3) True

then swap a[i] with a[j]

0	1	2	3	4
11	5	19	1	15
↑p		↑i	↑j	

0	1	2	3	4
11	5	1	19	15
↑p		↑i	↑j	

① while(a[i]<=pivot)

(a[2]<=11)

(1<=11) True

then i=i+1=2+1=3

0	1	2	3	4
11	5	1	19	15
↑p			↑i j	

① while(a[i]<=pivot)

(a[3]<=11)

(19<=11) False

Control goes to next while statement

② while(a[j]>=pivot)

(a[3]>=11)

(19>=11) True

j = j-1 = 3-1 = 2

11	5	1	19	15
0	1	2	3	4
↑p		↑j	↑i	

② while(a[j]>=pivot)

(a[2]>=11)

(1>=11) False

③ if(i<j)

if(3<2) False

then swap a[j] with pivot

11	5	1	19	15
0	1	2	3	4
↑p		↑j	↑i	

1	5	11	19	15
0	1	2	3	4

Left Pivot Right

Consider Right sub-problem

19	15
3	4
pivot ↑i	↑j

→ while(low<high)

(3<4) True

① while(a[i]<=pivot)

(a[3]<=19)

(19<=19) True

then i=i+1=3+1=4

19	15
3	4
↑p	↑i ↑j

① while(a[i]<=pivot)

(a[4]<=19)

(15<=19) True

② while(a[j]>=pivot)

(a(4)>=19)

(15>=19) False

③ if(i<j)

if(4<4) False

swap a[j] with pivot

19	15
3	4
↑p	↑j ↑i

15	19

Combine

1	5	11	15	19
0	1	2	3	4

Analysis of Quick Sort:

We need to find out the time complexity of recurrence relation using back substitution method.

Recurrence relation for quick sort is

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

Sol Recurrence Relation for Quick Sort

$T(n) = 2T(n/2) + cn$ —— 1

$T(n/2) = 2T(n/4) + c(n/2)$ —— 2

$T(n/4) = 2T(n/8) + c(n/4)$ —— 3

Substitute equation 2 in equation 1

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

$T(n) = 2\left[2T(n/4) + c\left(\frac{n}{2}\right)\right] + cn$

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

$T(n) = 4T(n/4) + 2cn$ —— 4

Substitute equation 3 in equation 4

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

$= 4\left[2T(n/8) + c\left(\frac{n}{4}\right)\right] + 2cn$

$= 8T(n/8) + \cancel{4} * \frac{cn}{\cancel{4}} + 2cn$

$T(n) = 8T(n/8) + 3cn$

$T(n) = 2^3 T(n/2^3) + 3cn$

K times ⇒ K = 3

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

it will stops when $\frac{n}{2^K} = 1 \Rightarrow n = 2^K \Rightarrow K = \log_2 n$

$T(n) = n T(\cancel{n}/\cancel{n}) + \log_2 n \, cn$

$= nT(1) + cn\log_2 n \quad [\because T(1) = 1]$

$= (n + cn\log_2 n)$ | ∴ Time Complexity = $\boxed{n\log_2 n}$

04-01-2022

Strassen's Matrix Multiplication

Suppose we want to multiply two matrices A and B of each size N

$C = A \times B$

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

multiplication gives

$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$

$C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$

$C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$

$C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}$

To multiply 2X2 matrix there are Total 8 multiplications and 4 Addition

Algorithm for matrix multiplication

```
Algorithm mul(A,B,C,i,j,n,k)
{
  for i := 1 to n do          ———— n+1
    for j := 1 to n do        ———— n(n+1)
      C[i,j] = 0              ———— n^2
      for k := 1 to n do      ———— n^2(n+1)
      C[i,j] := A[i,j] * B[i,j];  ———— nxnxn
}
```

$\therefore$ Frequency Count $= n+1+n^2+n+n^2+n^3+n^2+n^3$

$= 2n^3 + 3n^2 + 2n + 1$

$\therefore$ Time Complexity $= O(n^3)$

* Strassen's showed that 2X2 matrix multiplication can Reduced to 7 multiplication and 10 additions (8) substractions

* it is following Divide and Conquer Strategy

1. Divide :- If the problem is large then we can divide the problem into two sub-problems until problem becomes small

2. Conquer :- After dividing, we can Recursively finding the solutions of all sub-problems

3. Combine :- Combining the solutions of All the sub-problems.

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

$S_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$

$S_2 = (A_{21} + A_{22}) \times B_{11}$

$S_3 = A_{11} \times (B_{12} - B_{22})$

$S_4 = A_{22} \times (B_{21} - B_{11})$

$S_5 = (A_{11} + A_{12}) \times B_{22}$

$S_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$

$S_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$

$C_{11} = S_1 + S_4 - S_5 + S_7$

$C_{12} = S_3 + S_5$

$C_{21} = S_2 + S_4$

$C_{22} = S_1 + S_3 - S_2 + S_6$

* Now we can compare Strassen's matrix multiplication with Traditional matrix multiplication

$C_{11} = S_1 + S_4 - S_5 + S_7$

$= (A_{11}+A_{22}) \times (B_{11}+B_{22}) + A_{22} \times (B_{21}-B_{11}) - (A_{11}+A_{12}) B_{22} + (A_{12}-A_{22})(B_{21}+B_{22})$

$= A_{11}B_{11} + \cancel{A_{11}B_{22}} + \cancel{A_{22}B_{11}} + \cancel{A_{22}B_{22}} + \cancel{A_{22}B_{21}} - \cancel{A_{22}B_{11}} - \cancel{A_{11}B_{22}} - \cancel{A_{12}B_{22}} + A_{12}B_{21} + \cancel{A_{12}B_{22}} - \cancel{A_{22}B_{21}} - \cancel{A_{22}B_{22}}$

$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$

$C_{12} = S_3 + S_5$

$= A_{11} \times (B_{12}-B_{22}) + (A_{11}+A_{12}) B_{22}$

$= A_{11}B_{12} - \cancel{A_{11}B_{22}} + \cancel{A_{11}B_{22}} + A_{12}B_{22}$

$C_{12} = A_{11} \times B_{12} + A_{12}B_{22}$

$C_{21} = S_2 + S_4$

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

$= A_{21} \times B_{11} + \cancel{A_{22}B_{11}} + A_{22}B_{21} - \cancel{A_{22}B_{11}}$

$C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$

$C_{22} = S_1 + S_3 - S_2 + S_6$

$C_{22} = (A_{11}+A_{22})(B_{11}+B_{22}) + A_{11}(B_{12}-B_{22}) - (A_{21}+A_{22}) B_{11} + (A_{21}-A_{11})(B_{11}+B_{12})$

$= \cancel{A_{11}B_{11}} + \cancel{A_{11}B_{22}} + \cancel{A_{22}B_{11}} + A_{22}B_{22} + \cancel{A_{11}B_{12}} - \cancel{A_{11}B_{22}} - \cancel{A_{21}B_{11}} - \cancel{A_{22}B_{11}} + \cancel{A_{21}B_{11}} + A_{21}B_{12} - \cancel{A_{11}B_{11}} - \cancel{A_{11}B_{12}}$

$\therefore C_{22} = A_{22}B_{22} + A_{21}B_{12} \Rightarrow A_{21}B_{12} + A_{22}B_{22}$

Ex $A=\begin{bmatrix}1&1\\1&1\end{bmatrix}$ $B=\begin{bmatrix}0&0\\1&1\end{bmatrix}$ multiply using Strassen's matrix multiplication

$A_{11}=1, A_{12}=1, A_{21}=1, A_{22}=1$

Sol $B_{11}=0, B_{12}=0, B_{21}=1, B_{22}=1$

Traditional matrix multiplication

$C = A \times B$

$= \begin{bmatrix}1&1\\1&1\end{bmatrix} \times \begin{bmatrix}0&0\\1&1\end{bmatrix}$

$C = \begin{bmatrix}1\times0+1\times1 & 1\times0+1\times1\\ 1\times0+1\times1 & 1\times0+1\times1\end{bmatrix}$

$C = \begin{bmatrix}0+1 & 0+1\\ 0+1 & 0+1\end{bmatrix}$

$C = \begin{bmatrix}1&1\\1&1\end{bmatrix}$

Now we can solve using Strassen's

$S_1 = (A_{11}+A_{22})(B_{11}+B_{22})$

$S_1 = (1+1)(0+1) = 2\times1 = 2$

$S_2 = (A_{21}+A_{22})\times B_{11}$

$S_2 = (1+1)0 \Rightarrow 2\times0 = 0$

$S_3 = A_{11}\times(B_{12}-B_{22})$

$S_3 = 1\times(0-1) \Rightarrow 1\times1 = -1$

$S_4 = A_{22}(B_{21}-B_{11})$

$S_4 = 1(1-0) = 1\times1 = 1$

$S_5 = (A_{11}+A_{12})\times B_{22}$

$S_5 = (1+1)\times1 = 2\times1 = 2$

$S_6 = (A_{21}-A_{11})(B_{11}+B_{12})$

$S_6 = (1-1)(0+0) = 0\times0 = 0$

$S_7 = (A_{12}-A_{22})(B_{21}+B_{22})$

$S_7 = (1-1)(1+1) = 0\times2 = 0$

$C_{11} = S_1+S_4-S_5+S_7$

$= \cancel{2}+1-\cancel{2}+0$

$C_{11} = 1$

$C_{12} = S_3+S_5$

$C_{12} = -1+2 = 1$

$C_{21} = S_2+S_4$

$C_{21} = 0+1 = 1$

$C_{22} = S_1+S_3-S_2+S_6$

$C_{22} = 2-1-0+0 = 1$

$\therefore \begin{bmatrix}C_{11}&C_{12}\\C_{21}&C_{22}\end{bmatrix} = \begin{bmatrix}1&1\\1&1\end{bmatrix}$

Analysis of strassen's matrix multiplication

We need to find the time complexity of an algorithm using recurrence relation

Recurrence Relation

$T(n) = 7T(n/2) + cn^2$ — (1) [∵ c is the constant amount of time]

$T(n/2) = 7T(n/4) + c\left(\frac{n}{2}\right)^2$ — (2)

$T(n/4) = 7T(n/8) + c\left(\frac{n}{4}\right)^2$ — (3)

Substitute equation (2) in equation (1)

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

$= 7\left[7T(n/4) + c\left(\frac{n}{2}\right)^2\right] + cn^2$

$T(n) = 49T(n/4) + 7c\frac{n^2}{4} + cn^2$ — (4)

Substitute equation (3) in equation (4)

$T(n) = 49T(n/4) + 7c\frac{n^2}{4} + cn^2$

$T(n) = 49\left[7T(n/8) + c\left(\frac{n}{4}\right)^2\right] + 7c\left(\frac{n}{2}\right)^2 + cn^2$

$T(n) = 343T(n/8) + 49\frac{cn^2}{16} + 7\frac{cn^2}{4} + cn^2$

$= 7^3T(n/2^3) + 7^2\frac{cn^2}{16} + 7\frac{cn^2}{4} + cn^2$

$T(n) = 7^3T(n/2^3) + cn^2\left[(7/4)^2 + 7/4 + 1\right]$

⋮

it is k times $k=3$

$= 7^kT(n/2^k) + cn^2\left[(7/4)^{k-1} + (7/4)^{k-2} + \dots + (7/4)^0\right]$

$= 7^kT(n/2^k) + cn^2(7/4)^k$ ∴ let

when it will stop the condition

let $\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2 n$

$T(n) = 7^k T(n/2^k) + cn^2 (7/4)^k$

when it will stop the condition

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2 n$

$T(n) = 7^{\log_2 n} T(n/n) + cn^2 (7/4)^{\log_2 n}$

$= 7^{\log_2 n} T(1) + cn^2 \frac{7^{\log_2 n}}{4^{\log_2 n}}$ $\qquad [\because a^{\log_b c} \Rightarrow c^{\log_b a}]$

$= 7^{\log_2 n} \times 1 + cn^2 \frac{n^{\log_2 7}}{n^{\log_2 4}}$

$= 7^{\log_2 n} + cn^2 \frac{n^{\log_2 7}}{n^{\log_2 2^2}}$

$= 7^{\log_2 n} + c\cancel{n^2} \frac{n^{\log_2 7}}{\cancel{n^2}}$ $\qquad [\because \log_2 2^2 \Rightarrow 2]$

$= \underline{7^{\log_2 n}} + c\, n^{\log_2 7}$ $\qquad [\because a^{\log_b c} \Rightarrow c^{\log_b a}]$

$\Rightarrow n^{\log_2 7} + c\, n^{\log_2 7}$

$\Rightarrow n^{\log_2 7}[1 + c]$

$\therefore$ Time Complexity $= O\left(n^{\log_2 7}\underline{(1+c)}\right)$

$= O\left(n^{\log_2 7}\right)$ $\qquad [\because \log_2 7 = 2.81]$

(or)

$\therefore$ Time Complexity $= \underline{\underline{O\left(n^{2.81}\right)}}$

EX:

$$A=\left[\begin{array}{cc|cc}1&1&1&1\\1&1&1&1\\\hline 1&1&1&1\\1&1&1&1\end{array}\right]\quad B=\left[\begin{array}{cc|cc}1&1&1&1\\1&1&1&1\\\hline 1&1&1&1\\1&1&1&1\end{array}\right]$$ multiply strassen's

Traditional matrix multiplication

$$A_{11}=\begin{bmatrix}1&1\\1&1\end{bmatrix}\quad A_{12}=\begin{bmatrix}1&1\\1&1\end{bmatrix}\quad A_{21}=\begin{bmatrix}1&1\\1&1\end{bmatrix}\quad A_{22}=\begin{bmatrix}1&1\\1&1\end{bmatrix}$$

$$B_{11}=\begin{bmatrix}1&1\\1&1\end{bmatrix}\quad B_{12}=\begin{bmatrix}1&1\\1&1\end{bmatrix}\quad B_{21}=\begin{bmatrix}1&1\\1&1\end{bmatrix}\quad B_{22}=\begin{bmatrix}1&1\\1&1\end{bmatrix}$$

$$C=A\times B\quad \begin{bmatrix}C_{11}&C_{12}\\C_{21}&C_{22}\end{bmatrix}=\begin{bmatrix}A_{11}&A_{12}\\A_{21}&A_{22}\end{bmatrix}\times\begin{bmatrix}B_{11}&B_{12}\\B_{21}&B_{22}\end{bmatrix}$$

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

$$C_{11}=A_{11}\times B_{11}+A_{12}\times B_{21}$$

$$C_{12}=A_{11}\times B_{12}+A_{12}\times B_{22}\qquad C_{22}=A_{21}\times B_{12}+A_{22}\times B_{22}$$

$$C_{21}=A_{21}\times B_{11}+A_{22}\times B_{21}$$

$$C_{11}=\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}+\begin{bmatrix}1&1\\1&1\end{bmatrix}\begin{bmatrix}1&1\\1&1\end{bmatrix}$$

$$C_{11}=\begin{bmatrix}1\times1+1\times1 & 1\times1+1\times1\\1\times1+1\times1 & 1\times1+1\times1\end{bmatrix}+\begin{bmatrix}1\times1+1\times1 & 1\times1+1\times1\\1\times1+1\times1 & 1\times1+1\times1\end{bmatrix}$$

$$C_{11}=\begin{bmatrix}1+1&1+1\\1+1&1+1\end{bmatrix}+\begin{bmatrix}1+1&1+1\\1+1&1+1\end{bmatrix}\Rightarrow\begin{bmatrix}2&2\\2&2\end{bmatrix}+\begin{bmatrix}2&2\\2&2\end{bmatrix}=\begin{bmatrix}4&4\\4&4\end{bmatrix}$$

$$C_{12}=\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}+\begin{bmatrix}1&1\\1&1\end{bmatrix}\begin{bmatrix}1&1\\1&1\end{bmatrix}\Rightarrow\begin{bmatrix}2&2\\2&2\end{bmatrix}+\begin{bmatrix}2&2\\2&2\end{bmatrix}=\begin{bmatrix}4&4\\4&4\end{bmatrix}$$

$$C_{21}=\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}+\begin{bmatrix}1&1\\1&1\end{bmatrix}\begin{bmatrix}1&1\\1&1\end{bmatrix}\Rightarrow\begin{bmatrix}2&2\\2&2\end{bmatrix}+\begin{bmatrix}2&2\\2&2\end{bmatrix}=\begin{bmatrix}4&4\\4&4\end{bmatrix}$$

$$C_{22}=\begin{bmatrix}4&4\\4&4\end{bmatrix}$$

$$C=\begin{bmatrix}C_{11}&C_{12}\\C_{21}&C_{22}\end{bmatrix}=\begin{bmatrix}4&4&4&4\\4&4&4&4\\4&4&4&4\\4&4&4&4\end{bmatrix}$$

Strassen's

$$S_1=(A_{11}+A_{22})\times(B_{11}+B_{22})$$
$$=\left(\begin{bmatrix}1&1\\1&1\end{bmatrix}+\begin{bmatrix}1&1\\1&1\end{bmatrix}\right)\times\left(\begin{bmatrix}1&1\\1&1\end{bmatrix}+\begin{bmatrix}1&1\\1&1\end{bmatrix}\right)$$
$$S_1=\begin{bmatrix}2&2\\2&2\end{bmatrix}\times\begin{bmatrix}2&2\\2&2\end{bmatrix}=\begin{bmatrix}8&8\\8&8\end{bmatrix}$$

$$S_2=(A_{21}+A_{22})\times B_{11}$$
$$=\left(\begin{bmatrix}1&1\\1&1\end{bmatrix}+\begin{bmatrix}1&1\\1&1\end{bmatrix}\right)\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$$
$$S_2=\begin{bmatrix}2&2\\2&2\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}\Rightarrow\begin{bmatrix}2+2&2+2\\2+2&2+2\end{bmatrix}=\begin{bmatrix}4&4\\4&4\end{bmatrix}$$

$$S_3=A_{11}\times(B_{12}-B_{22})$$
$$=\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\left(\begin{bmatrix}1&1\\1&1\end{bmatrix}-\begin{bmatrix}1&1\\1&1\end{bmatrix}\right)\Rightarrow\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}\Rightarrow\begin{bmatrix}0&0\\0&0\end{bmatrix}$$

$$S_4=A_{22}\times(B_{21}-B_{11})=\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\left(\begin{bmatrix}1&1\\1&1\end{bmatrix}-\begin{bmatrix}1&1\\1&1\end{bmatrix}\right)\Rightarrow\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}=\begin{bmatrix}0&0\\0&0\end{bmatrix}$$

$$S_5=(A_{11}+A_{12})\times B_{22}=\left(\begin{bmatrix}1&1\\1&1\end{bmatrix}+\begin{bmatrix}1&1\\1&1\end{bmatrix}\right)\times\begin{bmatrix}1&1\\1&1\end{bmatrix}=\begin{bmatrix}2&2\\2&2\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}=\begin{bmatrix}4&4\\4&4\end{bmatrix}$$

$$S_6=(A_{21}-A_{11})\times(B_{11}+B_{12})\Rightarrow\left(\begin{bmatrix}1&1\\1&1\end{bmatrix}-\begin{bmatrix}1&1\\1&1\end{bmatrix}\right)\times\left(\begin{bmatrix}1&1\\1&1\end{bmatrix}+\begin{bmatrix}1&1\\1&1\end{bmatrix}\right)$$
$$S_6=\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}2&2\\2&2\end{bmatrix}\Rightarrow\begin{bmatrix}0&0\\0&0\end{bmatrix}$$

$$S_7=(A_{12}-A_{22})\times(B_{21}+B_{22})\Rightarrow\left(\begin{bmatrix}1&1\\1&1\end{bmatrix}-\begin{bmatrix}1&1\\1&1\end{bmatrix}\right)\times\left(\begin{bmatrix}1&1\\1&1\end{bmatrix}+\begin{bmatrix}1&1\\1&1\end{bmatrix}\right)$$
$$S_7=\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}2&2\\2&2\end{bmatrix}=\begin{bmatrix}0&0\\0&0\end{bmatrix}$$

$$C_{11}=S_1+S_4-S_5+S_7$$
$$=\begin{bmatrix}8&8\\8&8\end{bmatrix}+\begin{bmatrix}0&0\\0&0\end{bmatrix}-\begin{bmatrix}4&4\\4&4\end{bmatrix}+\begin{bmatrix}0&0\\0&0\end{bmatrix}$$
$$C_{11}=\begin{bmatrix}8&8\\8&8\end{bmatrix}-\begin{bmatrix}4&4\\4&4\end{bmatrix}=\begin{bmatrix}4&4\\4&4\end{bmatrix}$$

$$C_{12}=S_3+S_5$$
$$C_{12}=\begin{bmatrix}0&0\\0&0\end{bmatrix}+\begin{bmatrix}4&4\\4&4\end{bmatrix}=\begin{bmatrix}4&4\\4&4\end{bmatrix}$$

$$C_{21}=S_2+S_4$$
$$C_{21}=\begin{bmatrix}4&4\\4&4\end{bmatrix}+\begin{bmatrix}0&0\\0&0\end{bmatrix}=\begin{bmatrix}4&4\\4&4\end{bmatrix}$$

$C_{22} = S_1 + S_3 - S_2 + S_6$

$$C_{22} = \begin{bmatrix} 8 & 8 \\ 8 & 8 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$C_{22} = \begin{bmatrix} 8 & 8 \\ 8 & 8 \end{bmatrix} - \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$$

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} 44 & 44 \\ 44 & 44 \\ 44 & 44 \\ 44 & 44 \end{bmatrix}$$

$C = A \times B$

$\therefore$ Strassens matrix multiplications are satisfying Traditional matrix multiplication

Merge Sort

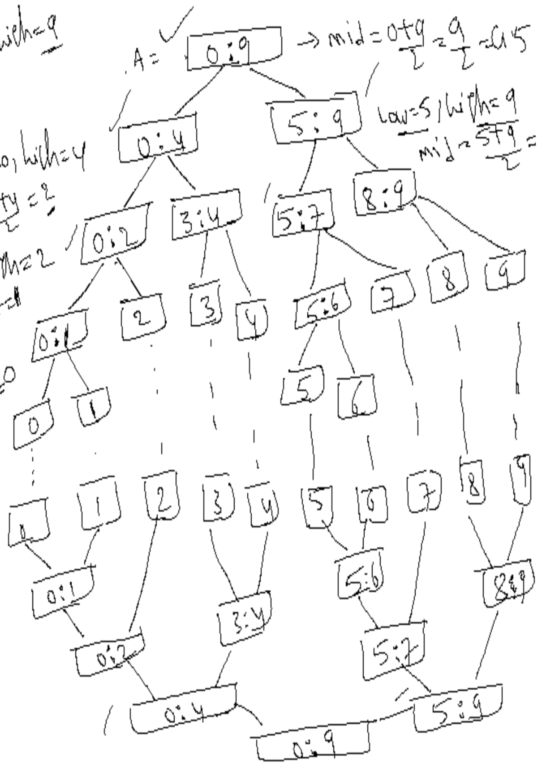
Merge sort is a sorting algorithm that uses divide and conquer strategy. It contains 3 steps to solve the problem

1.Divide: if the problem is large then we can divide the problem in to two sub-problems until problem becomes small. Small means problem contains only one element
2.Conquer: recursively find the solutions of all the sub-problems.
3.Combine: combine the solutions of all the sub-problems.

Flooring function:
X+y/2=2+3/2=5/2=2.5=2
Ceiling function:
X+y/2=2+3/2=5/2=2.5=3

low=0, high=9

mid = $\left(\frac{low+high}{2}\right)$ =

A = 

$\rightarrow$ mid = $\frac{0+9}{2} = \frac{9}{2} = 4.5$

low=5, high=9
mid = $\frac{5+9}{2} = \frac{14}{2} = 7$

Left = low $\rightarrow$ mid
Right = mid+1 $\rightarrow$ high

Divide
$\rightarrow$ low=0, high=4
mid = $\frac{0+4}{2} = 2$
$\rightarrow$ low=0, high=2
mid = $\frac{0+2}{2} = 1$
$\rightarrow$ low=0, high=1
mid = $\frac{0+1}{2} = 0.5 = 0$

$\rightarrow$ low=3, high=4
mid = $\frac{3+4}{2} = \frac{7}{2} = 3.5$

$\rightarrow$ low=5, high=7
mid = $\frac{5+7}{2} = \frac{12}{2} = 6$

$\rightarrow$ low=8, high=9
mid = $\frac{8+9}{2} = \frac{17}{2} = 8.5 = 8$

$\rightarrow$ low=5, high=6
mid = $\frac{5+6}{2} = \frac{11}{2} = 5.5 = 5$

Conquer
$\rightarrow$ low=2, high=2
small
$\rightarrow$ low=3, high=3
small

Combine
$\rightarrow$ low=5, high=5
small

K-way merge sort

Ex 2 8 4 6 7 10 11 18 K=2

[2,8] [4,6] [7,10] [11,18]

[2,4,8,16] [7,10,11,18]

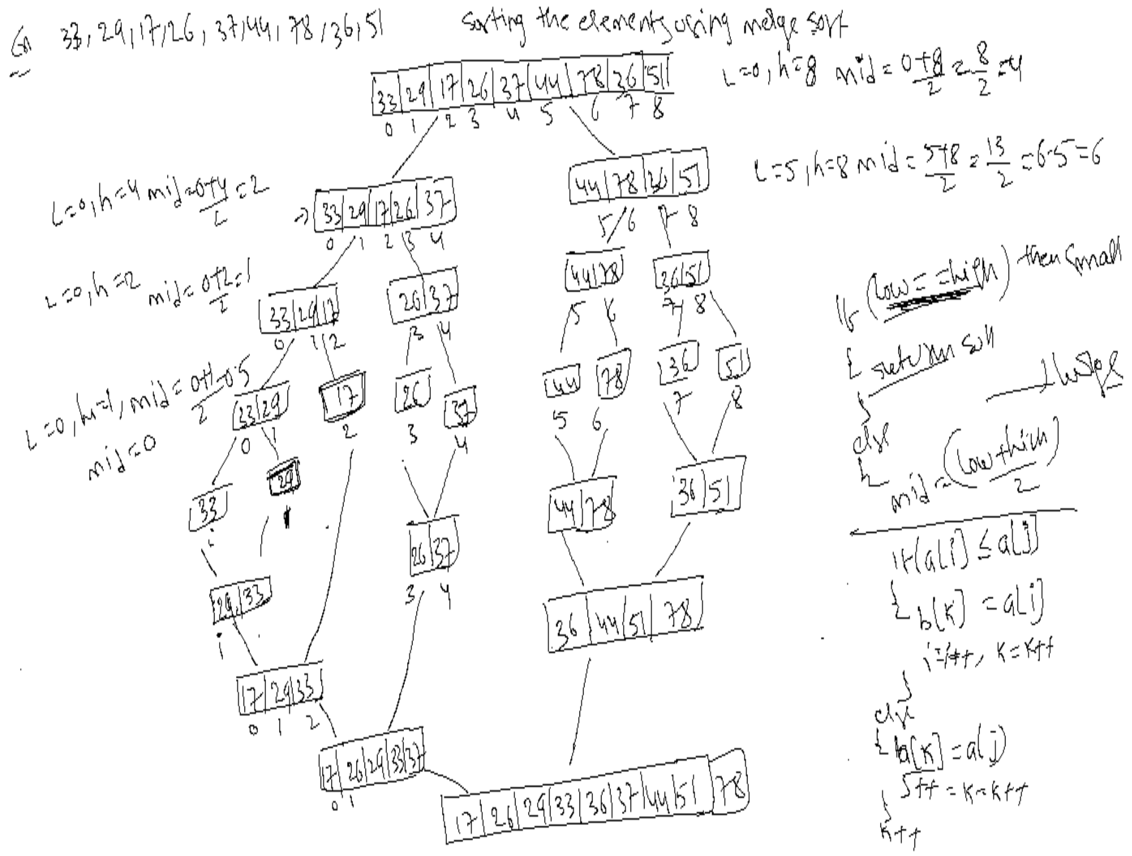
[2,4,7,8,10,11,16,18]

Ex 4 10 2 6 5 K=2

[4,10] [2,6] 5

[2,4,6,10] 5

[2,4,5,6,10]



Algorithm:

Algorithm mergesort(a,first,last)
{
If(low==high)
{
Return solution
}
Else
{
Mid:=(low+high)/2
Mergesort(a,low,mid)
Mergesort(a,mid+1,high)
Merge(a,low,mid,last)

```
}

Algorithm merge(a,low,mid,last)
{
I=low     //index in first sub-array a[low:mid]
J=mid+1   //index in second sub-array  a[mid+1,high]
K=low   //index in local array b[]

While(i<=mid&&j<=high)
{
If(a[i]<=a[j]
{
B[k]=a[i]
I=i+1
}
Else
{
B[k]=a[j]
J=j+1
}
K=k+1

While(i<mid)
{
b[k]=a[i]
I=i++
K=k+1
}
While(j<high)
{
B[k]=a[j]
J=j+1
K=k+1
}
For k= first to last
A[r]=b[r]
}
```

Analysis of Merge sort

We need to find out the time complexity of an algorithm using recurrence relation

Recurrence Relation

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

$T(n) = 2T(n/2) + cn \longrightarrow (1)$

$T(n/2) = 2T(n/4) + c\left(\frac{n}{2}\right) \text{ — } (2)$

$T(n/4) = 2T(n/8) + c\left(\frac{n}{4}\right) \text{ — } (3)$

Substitute eq (2) in eq (1)

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

$= 2\left[2T(n/4) + c\frac{n}{2}\right] + cn$

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

$T(n) = 4T(n/4) + 2cn \text{ — } (4)$

Substitute eq (3) in eq (4)

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

$= 4\left[2T(n/8) + c\frac{n}{4}\right] + 2cn$

$= 8T(n/8) + \cancel{4}\times\frac{cn}{\cancel{4}} + 2cn$

$T(n) = 8T(n/8) + 3cn$

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

If it is k times $k=3$

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

when it will stop the condition

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2 n$

$T(n) = n\,T(n/n) + cn\log_2 n$

$T(n) = n \times T(1) + cn\log_2 n$

$= n \times 1 + cn\log_2 n$

$T(n) = n + cn\log_2 n$

$\therefore$ Time complexity $=$

$= O(n + cn\log_2 n)$

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

$\therefore T.C = O(n\log_2 n)$

master theorem

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

Sol: $a=2,\ b=2,\ K=1,\ p=0$

$a \geq 1,\ b > 1$

$2 \geq 1 \rightarrow$ True, $2 > 1$ True

case1: $a > b^K$

$2 > 2^1 \Rightarrow 2 > 2$ False

case2: $a = b^K$

$2 = 2^1 \Rightarrow 2 = 2$ True

a) If $p > -1 \Rightarrow 0 > -1$ True

$T(n) = \theta\left(n^{\log_b a} \log n^{p+1}\right)$

$T(n) = \theta\left(n^{\log_2 2} \log n^{0+1}\right)$

$= \theta\left(n^1 \log n^1\right)$ $\left[\because \log_a a = 1\right]$

$= \theta\left(n \log n\right)$

$\therefore$ Time Complexity $= O\left(n \log n\right)$