Page No. 6
Date : 14/02/16

# Chapter 2 : Divide & Conqueor

Solving big problems by dividing it into smaller ones.

= Recursion
Recurrance Relation & its relation

1. **Recursion** → A function is calling itself to solve a particular problem.

Ex 1 $f(6) = 6 \times f(5)$
$\Rightarrow 5 \times f(4) \Rightarrow f(3) \times 4 \Rightarrow 3 \times f(2) \Rightarrow 2 \times f(1)$

No. of returns are no. of functions called.
f(6) will be at bottom of stack f(1) at the top.
Recursion is nothing but solving the big problems in terms of smaller ones.

Why stack is used not queue?
bcoz of LIFO property.
To execute the recursive program, we are using stack data structure.
A recursion must have a terminating condition. Otherwise program will go to infinite loop. At end, the error "Stack Overflow".
Parameter value keep on changing of the function. no. of parameters, function name, code, parameter name will not change.
- Recursive program contains two steps -
  How to stop?
  How to continue?
- Drawback - more stack space.
- Benefit to the programmar. not for computer.



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

- For every recursive program, equal and non-recursive program is possible.
- Logic is only that defines time. (complexity).

∴ Recurrance relation for factorial →

$$f(n) = \begin{cases} 1 & \text{if } n=1 \\ n \times f(n-1) & \text{if } n>1 \end{cases}$$

TC = n

Ex 2. Write a recursive C program & recurrance relation to multiply two +ve no.s m & n where m, n > 0.

```
mul (n)
{
if (n==0) return mult;
else {
   mult = mult+m;
   mul (n-1) }
}
```

mult = 0

```
mul(m,n) {
    if (m==0 || n==0)
        return 0;
    else return (n+mul.(m-1,n))
}
```

SC = m-sloi

TC = O(m)

Major contribution is of else statements - 'if' is needed for termination purpose.

Recurrance -

$$mul(m,n) = \begin{cases} 0 & \text{if } m=0 \text{ or } n=0 \\ n+mul(m-1,n) & \text{Otherwise} \end{cases}$$



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

Ex 3 Write recursive program & recurrance relation to find nth fibonacci number.

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

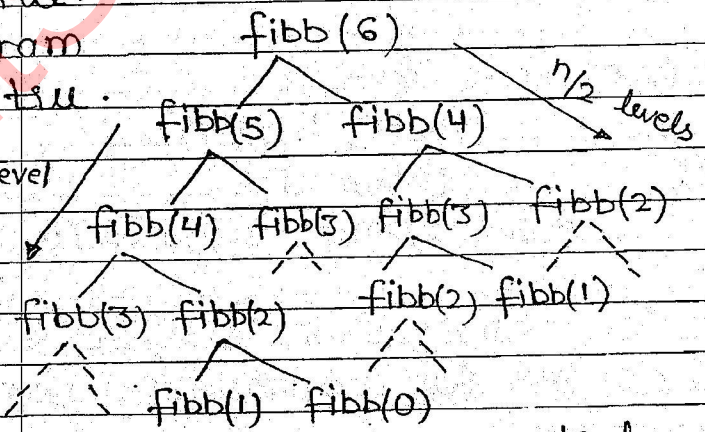
```
fibb(n) {
  if(n==1) return 1; if(n==0) return 0;
  else
    return fibb(n-1)+fibb(n-2);
  }
```

Recurrance

$$fibb(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ fibb(n-1) + fibb(n-2) & \text{otherwise} \end{cases}$$

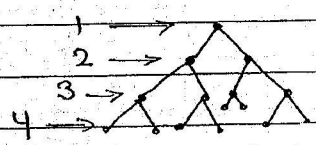
($n$ level, $n/2$ level)

Space complexity of a recurrance is no. of stack slots i.e. no. of levels in tree.
Every recursive program can be modelled as tree.




fibbonacci of 6 will generate 6 level tree. So, SC = 6.
[n]

Every time 2 fn called; so binary tree generated.
for time complexity of recursion, no. of functions called in the program.




4 level tree
Nodes = $2^4 - 1 = 15$
Complete binary



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

fibb(n) = n-level complete tree (assume)
$2^n - 1$ nodes
TC = $O(2^n)$ [upper bound]
fibb(n) = n/2 level complete tree
$2^{n/2} - 1$ nodes
TC = $\Omega(2^{n/2})$ [lower bound]
Ternary tree → No. of nodes are $3^n - 1$.

f(1)
f(2)
f(3)
f(4)
f(5)
f(6)

Ex.4 Write recursive program & recurrance rel$^n$ to find GCD (Greatest Common Divisor) of (m,n)
gcd(10,20) = 10 gcd(5,7) = 1 gcd(10,0) = 10
gcd(0,0) = undefined
gcd(13,17) ⇒ 13) 17 (1
13
4
⇒ gcd(4,13) ⇒ 4) 13 (3
12
1
gcd(0,1) ⇐ 1) 4 (4 ⇐ gcd(1,4) ⇐
4
0

Recurrance relation

$$gcd(m,n) = \begin{cases} \infty & \text{if } m=0 \text{ \&\& } n=0 \\ m & \text{if } n=0 \\ n & \text{if } m=0 \\ gcd(n\%m, m) & \text{otherwise} \end{cases}$$

```
gcd(m,n) {
  if (m==0 && n==0) return(-1); (signal).
  if (m==0) return n;
  if (n==0) return (m);
  else
    return gcd(n%m, m);
  }
```

Best case TC = O(1)
Worst TC = $\log_2 n$
Average n > m
SC = same as i/f$^n$ called.



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

## Recurrance Relation Solving →

1. Substitution
2. Recursive Tree method
3. Master theorem

→ Substituting the given $f^n$ repeatedly until given function is removed.

Ex1 $T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n-1)+n & \text{if } n>1 \end{cases}$

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

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

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

$= \vdots^{n-1}$

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

$= T(1)+1+2+3+\cdots n = \frac{n(n+1)}{2} = O(n^2)$

Ex2. $T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n-1)\cdot n & \text{if } n>1 \end{cases}$

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

$= T(n-2)(n-1)n$ ] 3 times sub., 3 terms

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

$= \vdots$

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

$= n! = O(n^n) = \Theta(n!)$

$T(n-K)\times(n-(K-1))\times(n-(K-2))\times\cdots n$

Ex3 $T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n-1)+1/n & \text{if } n\geq 1 \end{cases}$

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

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

$=$



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

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

$= T(n-K) + \frac{1}{n-(K-1)} + \frac{1}{n-(K-2)} + \cdots + \frac{1}{n}$ ↓ $n-1$

$= T(1) + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}$

$= O(\log n)$

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

5 $T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-2)+n^2 & \text{if } n>0 \end{cases}$

6 $T(n) = \begin{cases} 1 & \text{if } n=2 \\ 8T(n/2)+n^2 & \text{if } n>2 \end{cases}$

7 $T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2T(n/2)+n\log n & \text{if } n>1 \end{cases}$

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

$= T(n/4) + n/2 + n$ $\quad T(n/2^3) + n/2^2 + n/2 +$

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

$T(n/K) + n/K/2 + n/K/4 + \cdots n$ $\quad T(n/2^K) + \frac{n}{2^{K-1}} + \cdots$

$= T(1) + 2 + 4 + 8 + \cdots \frac{n}{2} + n$ $\quad n = 2^K = 2^{\log}$

$= \frac{1(2^n - 1)}{2-1}$ $\quad 1 + [2^1 + 2^2 + 2^3 + \cdots 2^{\log_2 n}]$

$= 2^n - 1 \leq O(2^n)$ $\quad 1 + \frac{2(2^{\log_2 n} - 1)}{2-1}$

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



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

Avoid solving $2^{\log_2 n - 1}$

5] $T(n) = T(n-2) + n^2$

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

$= T(n-6) + (n-4)^2 + (n-2)^2 + n^2$

$= T(n-2k) + (n-(2k-2))^2 + (n-(2k-4))^2 + \dots n^2$

$= T(0) + 2^2 + 4^2 + 6^2 + \dots n^2$

$= 2^0 + (2\times1)^2 + (2\times2)^2 + (2\times3)^2 + \dots (2\times n/2)^2$

$= 1 + 2^2 [1^2 + 2^2 + 3^2 + \dots (n/2)^2] = 1 + 2^2 \left(\frac{\frac{n}{2}(\frac{n}{2}+1)(n+1)}{2}\right)$

$= O(n^3)$

6] $T(n) = 8T(n/2) + n^2$

$= 8\left[8T\left(\frac{n}{2^2}\right) + \left(\frac{n}{2}\right)^2\right] + n^2 = \boxed{8^2 T\left(\frac{n}{2^2}\right) + 2n^2 + n^2}$

$= 8\left[8\left[T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right)^2\right] + \left(\frac{n}{2}\right)^2\right] + n^2 = \boxed{8^3 T\left(\frac{n}{2^3}\right) + 2^2 n^2 + 2n^2 + n^2}$

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

$n = 2^{k+1}$

$\log n = k+1$

$k = \log_2 n - 1$

$= 8\left[8\left[\dots 8\left[T\left(\frac{n}{2^k}\right) + \left(\frac{n}{2^{k-1}}\right)^2\right] + \left(\frac{n}{2^{k+1}}\right)^2 + \dots\right] + n^2$

$= 8^{\log_2 n - 1} T\left(\frac{n}{2^{\log_2 n - 1}}\right) + n^2 \left[2^0 + 2^1 + 2^2 + \dots 2^{\log_2 n - 2}\right]$

$= n^3/8\, T(2) + n^2 \left[\frac{2^{\log n - 1} - 1}{2-1}\right] = O(n^3)$

7] ~~X~~ $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$

$= 2\left[T\left(\frac{n}{2^2}\right) + \frac{n}{2}\log\frac{n}{2}\right] + n \log n$

$= 2\left[2\left[T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\log\frac{n}{2^2}\right] + \frac{n}{2}\log\frac{n}{2}\right] + n \log n$



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

$$2\left[2\left[\cdots T\left(\frac{n}{K}\right)+\frac{n}{K/2}\log\frac{n}{K/2}\right]+\frac{n}{K/4}\log\frac{n}{K/4}\right]-n\log$$

$$2\left[2\left[\cdots T(1)+2n\log(2n)\right]+\cdots\right.$$

$$2\left[2\left[2(1+2n\log(2n))+4n\log 4n\right]+8n\log 8n\right]$$

7. $T(n) = 2T(n/2) + n\log n$

$$= 2\left[2T\left(\frac{n}{2^2}\right)+\frac{n}{2}\log\frac{n}{2}\right]+n\log n$$

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

$$= 2^2\left[2T\left(\frac{n}{2^3}\right)+\frac{n}{2^2}\log\frac{n}{2^2}\right]+n\log\frac{n}{2}+n\log n$$

$$= 2^3T\left(\frac{n}{2^3}\right)+n\log\frac{n}{2^2}+n\log\frac{n}{2^1}+n\log\frac{n}{2^0}$$

K times.

$$= 2^KT\left(\frac{n}{2^K}\right)+n\left[\log\frac{n}{2^{K-1}}+\log\frac{n}{2^{K-2}}+\cdots\log\frac{r}{2}\right.$$

$$= 2^{\log_2 n}T\left(\frac{n}{2^{\log_2 n}}\right)+n\left[\lg\frac{n}{2^0}+\log\frac{n}{2^1}+\cdots\log\frac{n}{2^{\log_2 n}}\right.$$

$$= n+n\left[(\log_2 n-\log_2 2^0)+(\log_2 n-\log_2 2)+\cdots\right.$$
$$\cdots(\log_2 n)-\log_2 2^{\log_2 n-1}\Big]$$

$$= n+n\left[(\log_2 n-0)+(\log_2 n-1)+\cdots\cdots 2+1\right]$$ sum

$$= n+n\left[\frac{(\log_2 n)(\log_2 n+1)}{2}\right] = n+n[\log_2 n]^2$$

$$= O(n(\log_2 n)^2)$$



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

3. $T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n/2)+c & \text{if } n>1 \end{cases}$

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

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

$= T\left(\frac{n}{2^3}\right)+c+c+c$

$= T\left(\frac{n}{2^k}\right)+kc.$

$k = \log_2 n$

$= T\left(\frac{n}{2^{\log_2 n}}\right)+[\log_2 n]c.$

$= 1+(\log_2 n)c. = O(\log_2 n)$

4. $T(n) = \begin{cases} 1 & \text{if } n=1 \\ 7T(n/2)+n^2 & \text{if } n>1 \end{cases}$

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

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

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

$= 7^2\left[7T\left(\frac{n}{2^3}\right)+\left(\frac{n}{2^2}\right)^2\right]+7\left(\frac{n}{2}\right)^2+n^2$

$= 7^3 T\left(\frac{n}{2^3}\right)+7^2\left(\frac{n}{2^2}\right)^2+7\left(\frac{n}{2}\right)^2+7^0\left(\frac{n}{2^0}\right)^2$

$\vdots$ K times

$= 7^k T\left(\frac{n}{2^k}\right)+7^{k-1}\left(\frac{n}{2^{k-1}}\right)^2+7^{k-2}\left(\frac{n}{2^{k-2}}\right)^2+\cdots 7^0$



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

Page No. 1
Date : 20/02/16

20 feb 16

$K = \log_2 n$

$= 7^{\log_2 n} T(1) + 7^{\log_2 n - 1}\left(\frac{n}{2^{\log_2 n - 1}}\right)^2 + \cdots\cdots 7^0\left(\frac{n}{2^0}\right)^2$

$= n^{\log_2 7} + n^2\left[\left(\frac{7}{4}\right)^{\log_2 n - 1} + \left(\frac{7}{4}\right)^{\log_2 n - 2} + \cdots\cdots \frac{7^0}{4^0}\right]$

$= n^{\log_2 7} + n^2\left[\frac{1\,(7/4)^{\log_2 n} - 1}{(7/4) - 1}\right]$

$= n^{\log_2 7} + n^2 \cdot \frac{n^{\log_2 7/4} \cdot (7)^{\log_2 n}}{(4)^{\log_2 n}}$

$= n^{\log_2 7} + n^2 \cdot \frac{n^{\log_2 7}}{n^2} = O(n^{\log_2 7}) = O(n^{2.81})$

10. $T(n) = \begin{cases} 2 & \text{if } n = 2 \\ \sqrt{n}\,T(\sqrt{n}) + n & \text{if } n > 2 \end{cases}$

$T(n) = \sqrt{n}\,T(\sqrt{n}) + n$ → Ist sub.

$= \sqrt{n}\left[\sqrt{\sqrt{n}}\,T(\sqrt{\sqrt{n}}) + \sqrt{n}\right]$

$= n^{3/4} T(n^{1/4}) + n + n$ → 2nd sub.

$= n^{3/4}\left[n^{1/8} T(n^{1/8}) + n^{1/4}\right]$

$= n^{7/8} T(n^{1/8}) + n + n + n$ → 3rd sub.

⋮ K times.

$= n^{\frac{2^K - 1}{2^K}} T(n^{1/2^K}) + Kn$

$= n^{\log\log n - \frac{1}{2^K}} T(n^{1/2^K}) + Kn$

$= \frac{n}{n^{1/2^K}} T(2) + \log\log(n)\,n$

$= \frac{n}{2}(2) + n\log\log n = O(n\log\log n)$

$n^{\frac{1}{2^K}} = 2.$

$\frac{1}{2^K}\log n = 1.$

$\log n = 2^K$

$K = \log(\log$



Download all Notes and papers from StudentSuvidha.com

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

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

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

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

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

$= 2^3T(n-3)+2n-4+2n+n-2$

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

$\vdots$

$2^kT(n-k)+2^{k-1}(n-(k-1))+2^{k-2}(n-(k-2))+\cdots n$

$k = n-1$

$2^{n-1}T(1)+2[2+3+4+\cdots n-1]$

$2+2(2+3+4+\cdots n-1)$

$2[1+2+3+4+\cdots n-1] = (n-1)n = O(n^2)$

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

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

Combination of AP and GP series.

Multiply the series by 2 i.e. 2 (converting to GP series)

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

$T(n)-2T(n) = n-2-2^2-2^3-2^4\cdots-2^{n-2}-2^{n-1}-2^n$

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

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

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

$T(n) = 2^n-n = O(2^n)$

How recursive program looks good inside the computer ?



Download all Notes and papers from StudentSuvidha.com

```
main() {
    int n = 5;        declaration
printf("%d", fact(n));
}
fact(int n) {
    int a,b;
if (n==1) return (1)
else {
a = fact(n-1);
b = n*a;
return(b); }}
```

n → 2 bytes (1000, 1001)

3000
$2^8 = 256$
$2^5 = 32$
1100
10010 1100
8 7 6 5 4 3 2 1 0

fact(n) ⇒ n level tree ⇒ (n-1) fun ⇒ n(6B) ⇒ 6nB ⇒ O(n)

Total space ⇒ i/p + extra

Space → 2B ⇓ 6nB ⇒ 6nB ⇒ O(n) ⇓ O(n)

Space complexity → no. of levels → 2 bytes

n a b

fact(n) → Recursion
T(n) = n
S(n) = n

Non-recursion
T(n) = n
S(n) = const.

## Divide and Conqueor -

Divide the problems into some sub-problems

Conqueor the sub problems by calling recursively until we will get subproblem solutions.

Combine the sub-problem solutions so that we get final problem solution.



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

DAC - abstract algorithm

```
DAC (a, p, q)
{
if ( small (a, p, q))
   return ( solution (a, p, q))
else
  {
  m = Divide (a, p, q)
  C = DAC ( a, p, m)
  D = DAC (a, m+1, q)        } Conqueor
  E = combine (C, D)
  return E ;
  }
}
```

Abstract code, not universal code

Applications

How to find time complexity of any problem if is solved using divide and conqueor ?

$$T(n) = \begin{cases} O(1) & \text{if } n \text{ is small} \\ \text{Divide} \Rightarrow f_1 \\ T(n/2) \quad T(n/2) \\ \text{Combine} \Rightarrow f_2 & \text{if } n \text{ is big} \end{cases}$$

$$= f_1 + 2T\left(\frac{n}{2}\right) + f_2$$

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

↳ Divide + Combine Time

for K- subproblems

$KT(n/k)$ = conqueor time



Download all Notes and papers from StudentSuvidha.com

In general

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

- $a$ → no. of sub-problems
- $b$ → size of sub-problem

$T(n) = T(n/2) + C$ | Binary Search / Recurrance

↳ stack space $\log n$ ( how many times you are substituting ).

## Applications →

### I FIND MAX-MIN -

I/p : An array of elements

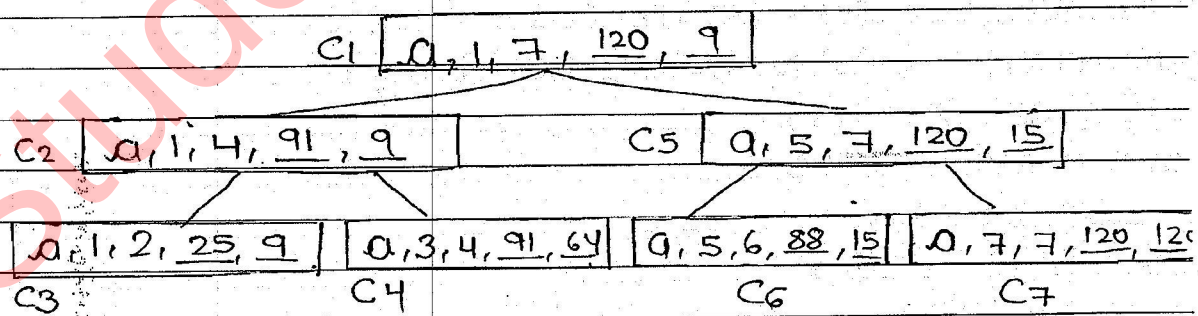
o/p : max, min return.

Small < 1 element ⇒ 0-C
2 element ⇒ 1-C.

Big — div by 2.

Example.

i/p A [ 25 (1) 9 (2) | 91 (3) 64 (4) | 15 (5) 88 (6) | 120 (7) ]




Post-order C3 C4 C2 C6 C7 C5 C1 Execution of functions

Pre-order function calling C1 C2 C3 C4 C5 C6 C7



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

| | | | |
|---|---|---|---|
| C3 | C4 | C6 | C7 |
| C2 | C5 | | |
| C1 | | | |

$3 \Rightarrow \lg_2 7$ units of stack

```
DACmaxmin (a,i,j)
 {
if (i==j) {
max = min = a[i]
return (max, min) }
if (i = j-1) {
if (a[i] > a[j]
max= a[j], min= a[j];
else
max= a[j], min= a[i];
return (min, max); }
else {
  mid = [(i+j)/2]
(max1, min1) = DACmaxmin (a, i, mid);
(max2, min2) = DACmaxmin (a, mid+1, j);
  if (max1 > max2)
max = max1 else
    max = max2
if (min1 > min2)
   min = min2 else
   min = min1 }
   } return (max, min)
    }
```



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

Let T(n) be the time complexity (no. of comparisons) of above algorith on n elements array, then recurrance relation-

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

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

$$= 2\left[2T\left(\frac{n}{2^2}\right) + 2\right] + 2$$

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

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + 2\right] + 2^2 + 2$$

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

$$= 2^K T\left(\frac{n}{2^K}\right) + 2^1 + 2^2 + \cdots 2^K$$

$\frac{n}{2^K} = 2$ ; $n = 2^{K+1}$ ; $K = \log_2 n - 1$

$$= 2^{\log_2 n - 1} T(2) + 2^1 + 2^2 + 2^3 + \cdots 2^{\log_2 n - 1}$$

$$= \frac{n}{2} + \left[\frac{2\left(2^{\log_2 n - 1} - 1\right)}{2-1}\right]$$

$$= \frac{n}{2} + n - 2 = \frac{3n}{2} - 2 = 1.5n - 2$$

(BC, WH AC)

∴ No. of comparis

for no. of subtractions

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

Space complexity — I/P + extra

Array + $\log_2 n \Rightarrow O(n)$

2nB (Int)



Download all Notes and papers from StudentSuvidha.com

## W BINARY SEARCH

### I Power of an element -

1. Make the problem clear
   i/p : 2-positive integers a & n
   o/p : return $a^n$
2. Example
   Divide - $a^n = a^{n/2} \cdot a^{n/2}$
   Combine - multiply

```
S=1
for(i=1 to i=n)
   S=S*a
) TC = O(n)
```

$2^{64} \Rightarrow 2^{32} \cdot 2^{32}$

bXb $(2^{16})(2^{16}) \Rightarrow 2^8 \cdot 2^8 \Rightarrow 2^4 \cdot 2^4 \Rightarrow 2^2 \cdot 2^2 \Rightarrow 2^1 \cdot 2^1$

Small problem if n=1

- If n is given, no. of multiplications = $\log_2 n$

$64 \Rightarrow 6$, $128 \Rightarrow 7$

Division in three parts $a^n = a^{n/3} \cdot a^{n/3} \cdot a^{n/3}$
But the no. of multiplications will be more.

3. DACpower (a,n) {
   if (n==1) ] small problem
      return a;
   else {
   half = n/2; ] - Division
   b = DACpower (a, half) ] - Conquer
   return (b*b); ] - Combine
   } // end of else
   }

1. Recurrance relation -
   Let T(n) be the no. of multiplications required to find $a^n$ using above algorithm -



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

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

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

$T(n) = T(\frac{n}{4}) + 1 + 1$

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

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

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

How many substitution
= Stack size
= $\log_2 n$

for-3 parts.

$$T(n) = \begin{cases} 0 & \text{if } n=1 \\ T(n/3)+2 & \text{if } n>1 \end{cases}$$

$T(n) = 2\log_3 n = O(\log_3 n).$

| $a(n/2^{\log n-1})$ |
|---|
| ⋮ |
| $(a, n/2)$ |
| $(a, n)$ |

Local variables × $\log_2 n$ = Space.
↳ size of each slot

Space complexity = I/P + extra.
↓
3 + $\log_2 n$ B => $\log_2 n$ B => $O(\log$

for odd numbers

```
if (n%2 == 1)
{
  a1 = DAC power (a, n-1)
  return (a * a1)
}
```

```
for (i=1 to log2 n)
{
  a = a * a;
}
```

Non-recursive



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

# II BINARY SEARCH -

## Linear search

i/p - An array of $10^n$ elements, element x

o/p - position of x

example

| 25 | 50 | 60 | 15 | 45 | 90 | 80 |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| x=15 | x=80 | x=25 | x=55 |
|---|---|---|---|
| 4 | 7 | 1 | -1 |
| | Worst case | Best Case | |
| | O(n) | O(1) | |

Average case : $\frac{1+2+3+\cdots+n}{n} = \frac{n+1}{2} = O(n)$

↳ Exact number of comparisons.

## Binary search

i/p - an array (sorted), element x;

o/p - position of x.

example - A

| 10 | 20 | 30 | 40 | 50 | 60 | 70 |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

x = 40

Small problem - 1 element

```
DACsearch (A, x, i, j)
{
 if (i==j){ if(a[i]==x){ } stopping condition
    return(i); } else return(-1); }
 else {
   mid = (i+j)/2 ;                    O(1)
   if ( A[mid]==x)   } Best case.     (cons.)
     return mid;
   else if (A[mid]>x)                 1- comp.
```



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

j = mid-1; — 1 assign.
else
i = mid+1;
DAC search (A, x, i, j) — T(n/2).
}

Recurrance Relation - Let T(n) be the time complexity of above problem-

No one to one mapping b/w relation & algorithm.

$$T(n) = \begin{cases} O(1) & \text{if } n=1 \\ T\left(\frac{n}{2}\right)+c & \text{if } n>1 \end{cases}$$

↳ O(1)+O(1)+O(1)

Worst case & Average case

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

Total space - i/p + extra
⇓ ⇓
2n bytes + log n (stack space)

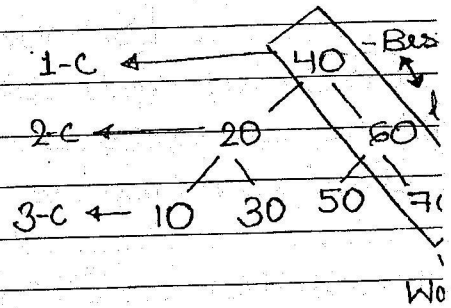
Best-case - O(1) - Time complexity

Average case -

$$= \frac{1+2+3+4+\cdots+\lg n}{\lg n}$$

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

$$= O(\log n)$$




If array not sorted, linear search will be better.

Ex 1.
A sorted array of n-distinct element is given. Find any element A[i] such that A[i] = i. find best algorithm with worst case.



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

Start every problem with linear search.
Apply binary search then.
Criteria is that check the middle element. If it is giving a logic / idea where to move either left or right, then binary search is applicable.

P - Position V - value

```
if (P<V)
    go left
else
    go right
```

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

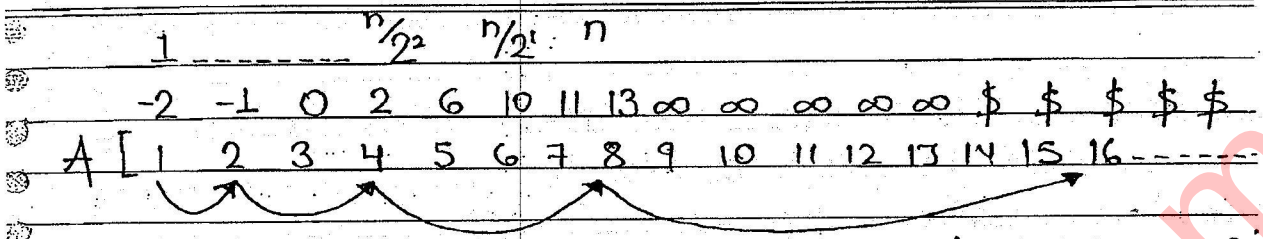
Special case → 1. If the elements are not distinct, then binary search not possible.
then linear search will apply. Answer is $O(n)$.
ii. If the elements are not sorted, then again linear search.

Ex 2. [No sorting, but still binary search]
I/P: An array of n- elements in which until some place are integers and all other ∞.
O/P: find position of Ist ∞.

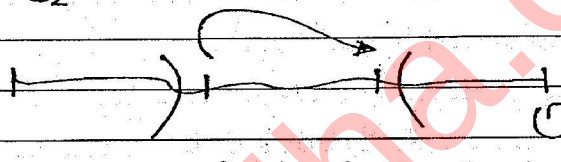
Linear search - $O(n)$
Binary Search - applicable (middle person is giving a logic where to move)
So $O(\log n)$
Sorted means according to problem (indirectly).
Special case → 1. Assume n is unknown
2. After array all are $ (dollar).
Linear search possible



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com




Every time multiply by 2. (reverse of division by 2)
then, apply binary search.
Answer is $O(\log_2 n)$

Ternary Search  $O(\log_3 n)$

Ex.3 i/p : A sorted array of n distinct elements.
o/p : Find any 2 elements a & b such that
$a+b = 1000$.

Solution → Consider
A [100 200 300 400 500 600 700 800]
1 2 3 4 5 6 7 8

Linear Search
fix as 'a' → Rotate as 'b' → No
fix as 'a' → Rotate as 'b' → Yes

```
for(i=1 to n) { for(j=i+1 to n) { -- }}   O(n²)
```

Binary Search
1 2 3 4 5 6 7 8
fix as 'a' → come to middle → No / Yes
fix as 'b' → come to middle

Time-complexity - $n \log n = O(n \log n)$



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

Page No. 7
Date :21/02/16

Special case → If a+b+c = 1000, then thru. loops
Have to fix two loops
Third one linear search $O(n^3)$
binary search $O(n^2 \log n)$

The best algorithm for this problem is grudy technique.

(i) a = Ist ; b = last
(ii) if (a+b == c)
return (a,b)
else
if (a+b > c) b = b-1
else a = a+1

a, b → sorted
maximum
n times
= O(n)

If array not sorted = sort & search = O(n log n)
n log n + n log n

Condition - Try for a+b > 1000. Answer - Take last two elements. O(1). [sorted]

1. linear search ($O(n^2)$), 2. Sort & search O(n log n) [unsorted]
3. find max min - O(n) [find + take]
4. Apply two passes of bubble sort & take (O(n))

for a+b+c > 1000, using grudy
fix one element, apply grudy to b & c
TC = $O(n^2)$.



StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com