Algorithm

is a finite set of inst^n that can be used to perform certain task.

Algo: Step by step procedure to perform a task

prog - Step by step procedure to perform a task.

What is the difference ?

Algo	program
is written at design phase	Implementation
write in any lang	prog lang.

Algo must satisfy the following criteria

1. i/p → algo must take 0 or more i/p.
2) o/p → it must generate 1 o/p
3) Definiteness – unambigious & clear
4) finiteness – it must terminate after some steps.
5) effectiveness – every inst^n is related to prob. avoid unnesary steps.

prob stmt → Algo → prog → ip → result < Correct / Incorrect

How to Analyze Algo :- it refer, how much processing time & storage an algo requires.

(1) Time Complexity
(2) Space Complexity

Time Complexity - amt of time taken by algo to perform certain task.

Space Complexity - amt of memory needed to perform certain task

Everbody is interested in reducing time complexity of algo & the reason behind this is - cost of mem ↓ very much in last 25 year where as cost of processing unit is not.

Performance Analysis of an Algo :-

① Algo swap(a,b)

```
{ temp = a ;      — 1
  a = b ;         — 1
  b = temp ;      — 1
}
```

f(n) = 3

time	space
	a - 1
	b = 1
	temp = 1
	S(n) = 3

3 is constant & constant is rep. as O(1)

— Time required for executing it once

→ No of times the stmt is executed.

Time Complexity — is the no. of dominating opⁿ executed by the Algo

ex:-

② `i = 1` — 1

`for (i <= n)` — n+1

`{ i = i+1 ;` — n

`}`

1 + n + 1 + n

2 + 2n

O(n)

③ `i = 1` — 1

`for (i <= n)` — n+1 / n/2+1

`{ i = i+2` — n/2

`}`

$1 + \frac{n}{2} + \frac{n}{2} + 1$

$2 + n = O(n)$

4)

Algo Sum (A,n)	time	space
{ s=0	1	s=1
for(i=1 to n do	n+1	i=1
{ s=s+a[i]	n	n=1
} return s;	1	a[]=n
}	$f(n)=2n+3$	$s(n)=n+3$
	$=O(n)$	$O(n)$

5) 2D Array.

Algo Sum (a[],m,n)	time	space
{ for i=1 to n do	n+1	i=1
for J=1 to m do	n.(m+1)	n=1
s = s + a[i][J]	n.m	m=1
} return s;	1	J=1
}		s=1
		$a=n^2$
		a is 2D array;
		$s(n)=n^2+5$
		$=n^2$
		$n \times n$
		or $n \times m$

$n+1+n.(m+1)+n.m+1$

$n+1+n.m+n+n.m+1$

$2n+2+2nm$

$2(nm+n+1)$

$\because n=m$

$O(n^2)$

Matrix Multiplication

(6)

	time	space
`for (i=0; i<n; i++)`	$n+1$	$i=1$, $n=1$, $J=1$, $K=1$, $A()=n^2$, $B()=n^2$, $C()=n^2$
`{ for(J=0; J<n; J++)`	$n.(n+1)$	
`C[i,J] = 0`	$n.n$	
`for (K=0; K<n; K++)`	$n.n(n+1)$	
`{ C[i,J] = C[i,J] + A[i,K] * B[K,J]`	$n.n.n$	
`}`		
`}`		$3n^2+4$
		$O(n^2)$

n
n

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

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

$= O(n^3)$

(7)

$i = 1$ — 1

`for (i<=n)` —

`{ i = i * 2`

`}`

i	0
1	2
$2 = 2^1$	
4	2^2
8	2^3
16	2^4
32	2^5
:	:
	$2^k = n$

$n = 5^0$

$i = 1$
$i = 2$
$i = 4$
$i = 8$

i
1
$2 = 2^1$
$4 = 2^2$
$8 = 2^3$
$16 = 2^4$
:
2^k
$i = 2^k$

Condition will terminate when $i \geq n$

& $i = 2^k$

$2^k \geq n$

$2^k = n$

$k = \log_2 n$

$2^k = n$

$\log_2 k = \log n$

$k = \log_2 n$

$\{ 2^3 = 8 \quad \log_2(8) = 3 \}$

(7)

```
for(i=1; i<n; i++)
{
  stmt
}
```

$i = 1+1+1+1+\cdots n$

$k = n$

```
for (i=1; i<n; i=i*2)
{
  stmt
}
```

$i = 1 \times 2 \times 2 \times 2 \times \cdots = n$

$2^k = n$

---

(8)

```
for(i=n; i>=1; i=i/2)
{
  stmt.
}
```

i
n
$n/2$
$n/4 = n/2^2$
$n/8 = n/2^3$
⋮
$n/2^k$

$n=8$

$i=8$

8	n
4	$n/2$
2	$n/2^2$
1	$n/2^3$
	⋮
	$n/2^k$

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

stmt will terminate or when $i < 1$

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

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

$n = 2^k$

$k = \log_2 n$

Complexity Values.

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 \ldots < n^k < \ldots < 2^n < 3^n < \ldots$

exponential

$\log n$	n	n^2	2^n
0	1	1	2
1	2	4	4
2	4	16	16
3	8	64	256

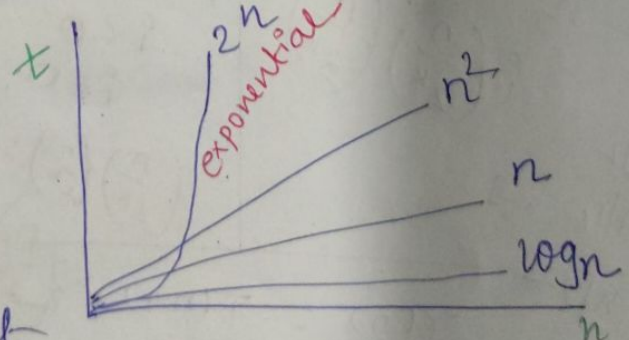
(n^k) $n^{100} < 2^n$ for higher values of n

$n^k \longrightarrow$ for larger value of n

$2^{100} < 2^2$

n^{100} will be smaller than 2^n

n can be any no. $(0 \text{ to } \infty)$



$O(1)$ - constant

$O(\log n)$ - logarithmic

$O(n)$ - linear

$O(n^2)$ - Quadratic

$O(n^3)$ - Cubic

$O(2^n)$ - Exponential

Mathematical way of representing time complexity.

→ A-priori Analysis

Asymptotic Notation :— approaching a value

It is used to discribe running time of an algo.

running time complexity is measured as fastest possible, slowest possible & average possible.

3 Notations are used for this

O , (upper bound)

Ω (lower bound)

θ (Average bound) tight bound

Big O Notation — longest amount of time taken by algo → Ceiling of growth for a given func

bigO of a function gives us "rate of growth of the step count function f(n) in terms of simple fun g(n)

Def - f(n) & g(n) be 2 non (-)ve function

f(n) = O(g(n))

f(n) is big O of g(n) if and only if there exist a (+)ve constant c and no such that

f(n) ≤ c * g(n) for all n and n ≥ n0

n denote all i/p value

no denote 1 i/p from n

Big-Oh. $f(n) \leq c * g(n)$.

for $\forall\ n > n_0$ $c > 0$ & $n_0 \geq 1$.

then $f(n) = O(n)$

$f(n) = 2n+3$

$2n+3 \leq$ ——

i can write anything which is

< greater than $2n+3$.

but we can not write multip

Coefficient.

$2n+3 \leq 10n$ ✓ $f(n)$

$2n+3 \leq 5n$ ✓✓ $c = 5$ $g(n) = n$

$2+3 = 5$ put $c = 5$.

How can i write $2n+3 \leq 5n^2$ ✓ Yes $g(n) = n^2$

$f(n) = O(n^2)$ ✓✓

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \ldots < n^k < 2^n < 2^3 <$

$f(n)$ belongs to n class

It means all those fun. which r greater than n becomes Θ (upper bound.
all this funct^n $\leq n$ becomes lower bound
& n becomes average bound.

try to write closest funct^n so $f(n) = O(n)$

Omega $f(n) \geq c * g(n)$, $n \geq n_0$.

then $f(n) = \Omega\, g(n)$.

$f(n) = 2n+3$

$2n+3 \geq$ —

$2n+3 \geq 1.n$ ✓ $\forall\, n \geq 1$.

c $g(n)$

$\therefore f(n) = \Omega(n)$

$2n+3 \geq 1.\log n$.

$2n+3 \geq (1.n^2)$ but if u are searching for lower bound go left side of series.

Θ $c_1 g(n) \leq f(n) \leq c_2 * g(n)$

$f(n) = 2n+3$

$c_1 * g(n) = 1.n$

$c_2 * g(n) = 5(n)$

$1.(n) \leq 2n+3 \leq 5(n)$

$f(n) = \Theta(n)$

i cannot use $f(n) = \Theta(n^2)$ ✗

Problem Solving Strategies: — strategy is an approach to design a prob.

① Divide & Conquer —
is suitable for a prob. where subproblems are of same type of original prob.

→ it divides the larger prob. in to subprob & solut^n of the original prob is obtained by combining solut^n of small subprob.

↝ This type of algo are basically used for parallel computation.

i) Divide
2) Solve
3) Combine

Conquer - dealing with strong/ large prob meaning effeciently

eg :- Map reduce in Hadoop, Binary Search etc.

(2) Greedy — it builds a solut^n to a prob in steps or phases
at each step we take the best, without regard for future consiquences

→ we hope that by choosing a local optimal at each step we will end up at a Global optimal.

— Grab the things which looks best at this moment

ex:- shortest path, scheduling, huffman coding etc

(3) Dynamic programming:—

method of solving complex prob by breaking it down into simple subproblem once and storing their result for future. Next time when the same subprob occour instead of recomputing its solut°, it simply looks up the previously computed solu°

→ saving computation time.

→ Technique of storing solut° of subprob is called as Memorization

ex:- search Engine recomadion Google Map

- concept of reusebility & overlapping subprob.

(4) Backtracking:- it does not allow fixed rules of computation it use trial & error to solve a prob.

(B) Branch & Bound :-

based on expro exploring the solut^n by applying bound.

~~Like in DFS~~

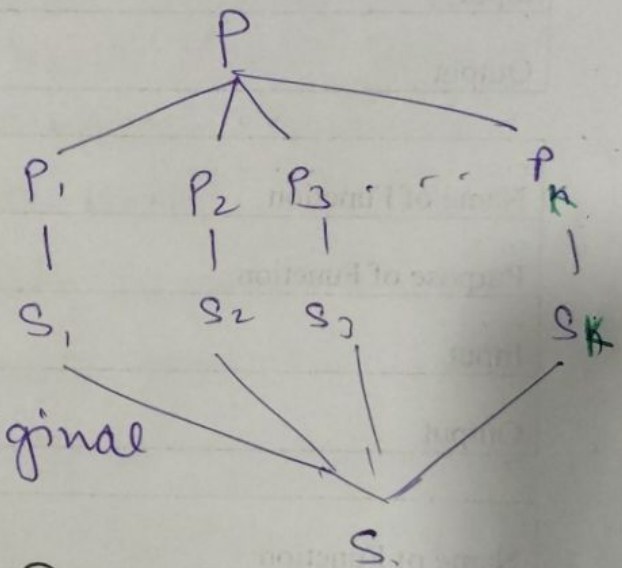
it build state space tree & find the optimal solut^n by pruning few branches which does not satisfy the bound

ex:-

TSP

Divide & Conquer →

Recursively solvable prob.



subprob. are of the same time as the original prob.

Algo – Control Abstraction for D&C

```
Algo D&C (P)
{ if Small (P) then return S(P);
  else
  { divide P in P1, P2 ... Pk ≥1
    Apply D&C on (P1, P2 ... Pk)
    return Combine (D&C(P1), D&C(P2), D&C(P3)
                    ... D&C(Pk) ...)
  }
}
```

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

$f(n)$ time for merging & deviding the list

a = no. of subprob

n/b = size of subprob

n = size of prob assume $n = b^k$

$T(1) = 0$ when there is only 1 element in the list. it require 0 time to perform op^n.

$$T(n) = aT(n/b) + n$$ { let $f(n) = n$, $a=2$, $b=2$ & $n=b^k$ — for merging

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

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

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

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

$$= 8T\left(\frac{n}{8}\right) + 3n$$

$\vdots$

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

k is no. of time loop will run } k = max value

we can write this

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

$$= (2^k = n)$$

$$k = \log_2 n$$

$$= nT\left(\frac{n}{n}\right) + \log_2 n \cdot n$$

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

$$= n(0) + n\log_2 n$$

$$= n\log_2 n$$

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

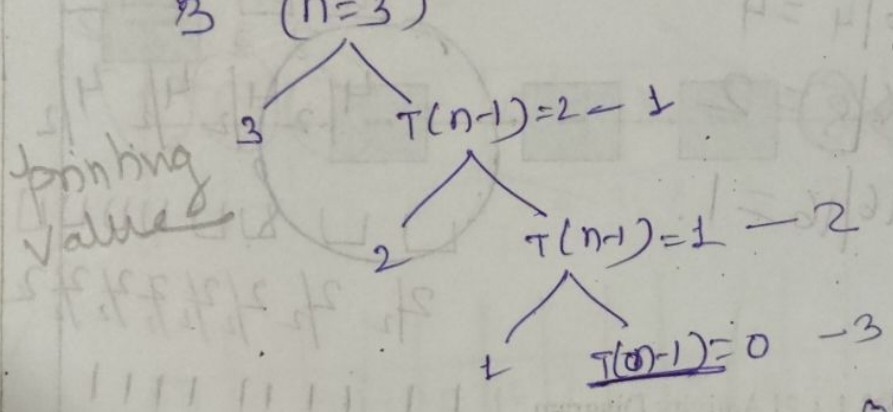
$T(n) = T(n-1)+1+1$

$T(n) = T(n-1)+2$ → constant

$= T(n-1)+1$

(n+1) call

if n = 3



$O(n)$

$T(n) = T(n-1)+1$ → recursive call

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

$= T(n-2)+2$

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

$= T(n-3)+3$

$= T(n-k)+k$ $\quad n=k$

$= T(n-n)+n$

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

```
void Test (int 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)
  }
}
```

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

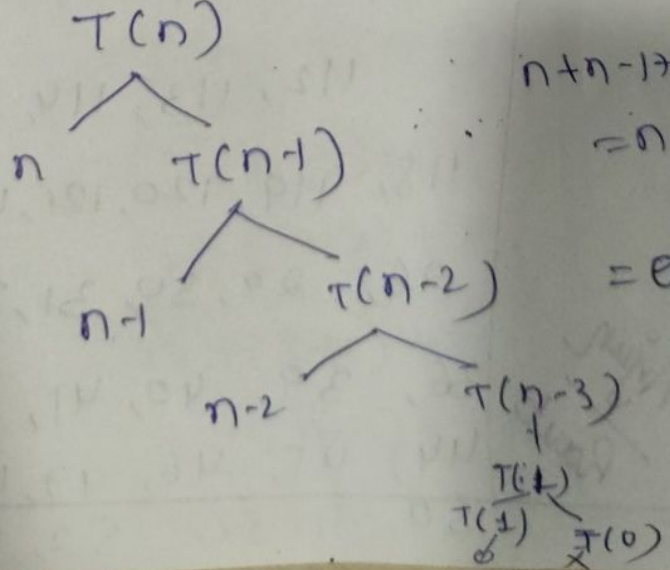
$= T(n-1)+2n+2$ → 3rd part ignore

$\underline{\underline{T(n) = T(n-1)+n}}$ {asymptotic}

↳ when $n>0$

Tree Method

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



$n+n-1+n-2+n-3$

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

$= \Theta(n^2)$

2nd method Substitution

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

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

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

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

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

: after k time

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

Assume $n-k=0$.

$n=k$

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

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

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

$\Theta(n^2)$

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

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

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

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

$= T[n-3] + 3n - 3$

$= T[n-4] + n-3 + 3n-3$

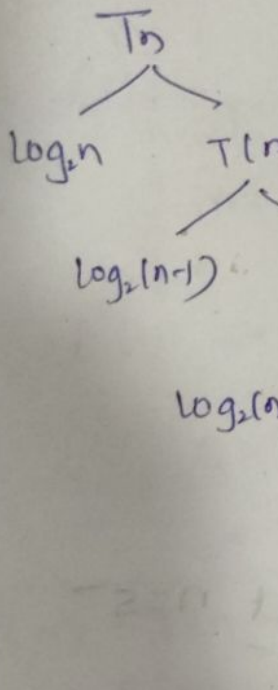
$= T[n-4] + 4n - 6$

$= T[n-5] + n-4 + 4n-6$

$= T[n-5] + 5n - 10$

```
void Test(int n)  — T(n)
{ if (n>0)
  {
    for(i=1; i<n; i=i*2)        } log2n
    { printf("%d", i);          }
    Test(n-1);   —— T(n-1)
  }
}
```

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + \log_2 n & n > 0 \end{cases}$$



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

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

$= T(n-2) + \log n-1 + \log n$ — (2)

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

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

k time

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

$n-k=0$, $n=k$

$= T(0) + \log n!$

$= 1 + \log n!$ $= O(n \log n)$

Master Theorem – for D & C

Not applicable for all prob.

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

Assume $f(n) = n^k \log^p n$

$a \geq 1$, $b > 1$, $k \geq 0$ & p is real no

Case 1

if $a > b^k$, then

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

Case 2 if $a = b^k$

(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 if $a < b^k$

(a) if $p \geq 0$ Then $T(n) = \theta(n^k \log^p n)$

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

$(\log n)^2$
$= \log n * \log n$
&
$\log^2 n^2$
$= \log \log n$
$\log n^2$
$2 \log n$

① $T(n) = 3T(\frac{n}{2}) + n^2$

$a=3, b=2, k=2, p=0$

$a \quad b^k$

$3 < 2^2$ case 3

if $p \geq 0$ then

$T(n) = \theta(n^k \log^p n)$

$= \theta(n^2 \log^0 n)$

$= \theta(n^2)$

② $T(n) = 4T(\frac{n}{2}) + n^2$

$a=4, b=2, k=2, p=0$

$a \quad b^k$

$4 = 2^2$

case 2 (a) $p > -1$

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

$= \theta(n^{\log_2 4} \log^1 n)$

$= \theta(n^2 \log n)$

③ $T(n) = 4T(\frac{n}{2}) + n$

$a=4, b=2, k=1, p=0$

$a \cdot b^k$

$4 \geq 2^1$

case 1 $T(n) = \theta(n^{\log_2 4})$

$= \theta(n^2)$

④ $T(n) = T(n/2) + n^2$

$a=1, b=2, k=2, p=0$

$a \quad b^k$

$1 < 2^2$ case 3

$p \geq 0$

$T(n) = \theta(n^k \log^p n)$

$= \theta(n^2 \log^0 n)$

$= \theta(n^2)$

⑤ $T(n) = 2^n T(\frac{n}{2}) + n^n$

Can not be solved using Master

a should be constant.

⑥ $T(n) = 16T(n/4) + n$

$a=16, b=4, k=1, p=0$

$a \quad b^k$

$16 > 4$ case 1

$T(n) = \theta(n^{\log_4 16})$

$= \theta(n^{\log_4 4^2})$

$= \theta(n^2)$

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

$a=2, b=2, k=1, p=1$

$a=b$ case 2(a)

$\theta(n^{\log_b a} \log^{p+1} n)$

$\theta(n^{\log_2^2} \log^2 n)$

$\theta(n \log^2 n)$

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

$a=2, b=2 \Rightarrow 2T(n/2) + n\log^{-1} n$

$k=1, p=-1$

$a = b^k$

$2 = 2$ case 2

$p = -1$ (b)

$T(n) = \theta(n \log_b^a \log\log n)$

$= \theta(n \log_2^2 \log\log n)$

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

(9) $T(n) = 2T(n/4) + n^{0.5}$

$a=2, b=4, p=0, k=0.5$

~~but~~ $k \geq 0$

$a \; b^k$

$2 < 4^{0.5}$ case 3 (a)

$T(n) = \theta(n^k \log^p n)$

$= \theta(n^{0.5} \log^0 n)$

$= \theta(n^{0.5})$

(10) $T(n) = 0.5T(\frac{n}{2}) + 1/2$

$a = 0.5$ but $a \geq 1$ so

can not solve this with Master.

11) $T(n) = 6T(n/3) + n^2 \log n$

$a = 6, b = 3, k = 2, p = 1$

a	b^k
6	$< 3^2$

$p \geq 0$

case 3 (a)

$T(n) = \theta(n^k \log^p n)$

$= \theta(n^2 \log n)$

(12) $T(n) = 64T(n/8) - n^2 \log n$ → can not solve this with master

13) $T(n) = 7T(n/3) + n^2$

$a = 7, b = 3, k = 2, p = 0$

a	b^k
7	$< 3^2$

case 3 & $p \geq 0$

3(a) $T(n) = \theta(n^k \log^p n)$

$= \theta(n^2 \log^0 n)$

$= \theta(n^2)$

(14) $T(n) = \sqrt{2}T(n/2) + \log n$

$a = \sqrt{2}, b = 2, k = 0, p = 1$

a	b^k
$\sqrt{2}$	$> 2^0$

case 1

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

$= \theta(n^{\log_2 \sqrt{2}})$

$= \theta(\sqrt{n})$

(15) $T(n) = 2T(n/2) + n\sqrt{n}$

$a = 2, b = 2, k = 1/2, p = 0$

a	b^k
2	$2^{1/2}$
$2 >$	$2^{1/2}$

case 1

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

$= \theta(n^{\log_2 2})$

$= \theta(n)$

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

$a = 3, b = 2, k = 1, p = 0$

a	b^k
3	$> 2^1$

case 1

$T(n) = \theta(n^{\log_2 3})$

(17) $T(n) = 3T(n/3) + n\sqrt{n}$

$a = 3, b = 3, k = 1/2, p = 0$

$3 > 3^{1/2}$ case 1

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

$= \theta(n^{\log_3 3})$

$= \theta(n)$

(18) $T(n) = 4T(n/2) + c \cdot n$

$a = 4, b = 2, k = 1, p = 0$

$a \quad b^k$

$4 > 2^1$ case 1

$T(n) = \theta(n \log_b^a)$

$= \theta(n \log_2^4)$

$= \theta(n^2)$

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

$a = 3, b = 4, k = 1, p = 1$

$a \quad b^k$

$3 < 4^1$ case 3, $p > 0$

3(a) $T(n) = \theta(n^k \log^p n)$

$= \theta(n^1 \log^1 n)$

$= \theta(n \log n)$

$2T(\frac{n}{2}) + n \quad a = 2, b = 2, k$

Large Number Multiplication algo

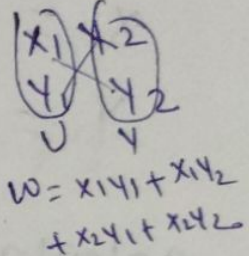
$\underline{X}\ \underline{Y}$ are Two large Numbers

Assume they are always in Power of 2

$$X = 10^{n/2} \times x_1 + x_2$$

$$Y = 10^{n/2} \times y_1 + y_2.$$

Calculate $Z = w - (u + v)$



where $u = x_1 y_1$

$v = x_2 y_2$

$w_1 = x_1 y_2 + x_2 y_1 + x_1 y_1 + x_2 y_2$

$$z = (x_1 y_2 + x_2 y_1 + x_1 y_1 + x_2 y_2) - x_1 y_1 - x_2 y_2$$

$$\boxed{z = x_1 y_2 + x_2 y_1}$$

final product $\boxed{p = 10^{n} u + 10^{n/2} z + v.}$

... Multiplication algo

Algo karatsuba (int x, int y)

{

if (x & y < 10)

then return x×y.

else

$x = x_1 \cdot 10^{n/2} + x_2$ $n = \max \text{length}(\text{len}(x), \text{len}(y))$

$y = y_1 \cdot 10^{n/2} + y_2$ $m = \text{floor}\lfloor n/2 \rfloor$

x_1 = high-part $(x, n/2)$

x_2 = low part $(x, n/2)$

y_1 = (high-part $(y, n/2)$

y_2 = low $(y, n/2)$

P_1 = karatsuba (x_1, y_1) } $T(n/2)$

P_2 = karatsuba (x_2, y_2) } $T(n/2)$

P_3 = karatsuba (x_1+x_2, y_1+y_2) } $T(n/2)$

$P_{cross} = P_3 - (P_1 + P_2)$

$\text{Final\_P} = P_1 \times 10^{2\times m} + 10^{m} \times P_{cross} + P_2$

return result.

$T(n) = 3T(n/2) + n$ → addition & subtract^n of $n/2$ digit no.

Solve this

$T(n) = n^{\log_2 3}$ Solve this

$= n^{1.5}$

x 1456

y 6533

n = 4 so m = 4/2 = 2.

$x_1 = 14$, $x_2 = 56$

$y_1 = 65$, $y_2 = 33$.

Recursive call for $14 \times 65 \Rightarrow P_1$

$x = 14$ $x_1 = 1, x_2 = 4$ 910

$y = 65$ $y_1 = 6$ $y_2 = 5$.

Recursive call algo for $P_2 = 1848$

56×33

$x_1 = 5, x_2 = 6$

$y_1 = 3, y_2 = 3$

Recursive call algo for (14+56, 65+33)

(70, 98)

$x_1 = 7, x_2 = 0$

$y_1 = 9$ $y_2 = 8$. = 6860

$P_{cross} = P_3 - (P_1 + P_2)$

$= 6860 -$

$= 4102$

$P = P_1 \times 10^{2m} + 10^{m} P_{cross} + P_2$

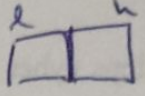
$= 910 \times 10^{4} + 10^{2} \times 4102 + 1848$

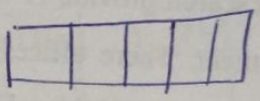
$= 9100000 + 410200 + 1848$

$= 95,12,048$

Algo RMinMax(arr, l, h, max, min)

```
if (l = h) then
    max = min = a[l]
else if (l = h-1) then
    { if a[l] > a[h]
        max = a[l];
        min = a[h];
      else
        max = a[h]
        min = a[l]
    }
else
    m = (l+h)/2
```

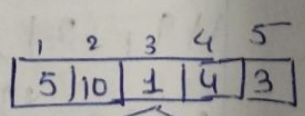


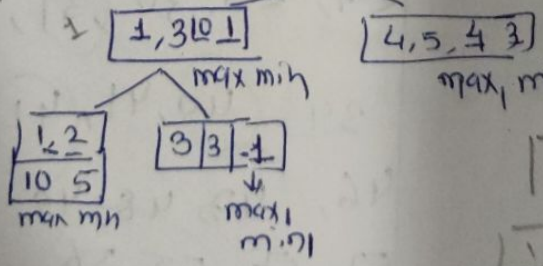


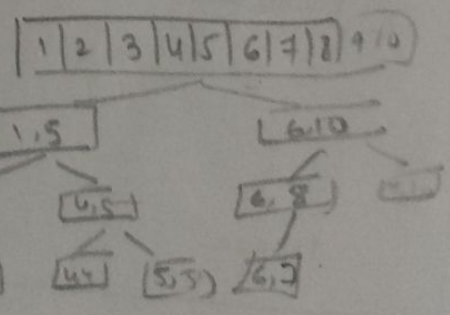
n/2 RMinMax(arr, l, mid, max, min)

n/2 RMinMax(arr, mid+1, h, max1, min1)

```
    if (max1 > max)        // combine
      then
        max = max1
2   if (min1 < min)        // combine
        min = min1
}
```







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

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

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

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

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

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

there will

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

recursion will stop at $\frac{n}{2^k} = 2$

or $n = 2.2^k$ or $2^k = \frac{n}{2}$

$= 2^{k+1}$

$= 2^k T(2) + \underbrace{2^k + 2^{k-1} + \cdots + 2}_{\text{sum of finite no}}$

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

$= 2^k + 2.2^k - 2$

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

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

$= \frac{3n}{2} - 2$ time complexity