Design and Analysis of Algorithms

Algorithm:

'Algos' — Greek — 'pain'

'Algor' — Latin — 'to be cold'

(x not root words)

Abu Abdullah Muhammad ibn Musa al-khwarizmi
(780-850)

"Al-khwarizmi" — Father of Algebra

0
1
2
:
9

Hindu-Arabic
(4th century)

Book published :- "on the calculation with Hindu Numerals"

↓ translated

Latin : "Algoritmi de numero Indorum"

↳ from the name of "Al-khwarizmi"

Computational Problems :- solved by computing device in its language

Eg:- Sorting Problem

Input : A list $a = (a_1, \ldots a_n)$ of n numbers

Output :- A permutation $a_1', a_2', a_3', \ldots a_n'$ of a such that

$$a_1' \leq a_2' \leq \ldots \leq a_n'$$

n : Input size Non-decreasing order

Algorithm :- step by step procedure for solving a well defined problem

(or)

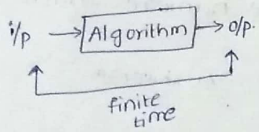
A sequence of steps which solves computational problem in finite time

"Donald E. knuth"
Father of Analysis of Algorithms

1. Input
2. Output
3. Finiteness
4. Definiteness (no Ambiguity)
5. Effectiveness (Simple Instruction)

no compound Instructions

Algo :-
step 1
step n



Computational Model : RAM model (Random Access Machine)

⇒ Arithmetic / Logical operations

Assumption :- 1. Under RAM model computational device, all Arithmetic/Logical operations take same/equal amount of time

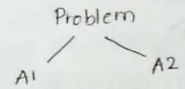
2. No parallel execution, only sequential execution of Instructions.

3. Infinite Memory available and random access for data

4. Loops are not considered as single operations

Finiteness :- total no of operations for an algorithm are finite.

correctness of Algorithm :- Algorithm which gives correct output for all specified Inputs



Algorithm → Time complexity
→ storage complexity

* Input size :- n

Analysis of Algorithm : Estimating / Predicting the resources required by Algorithm

step	operation	opcount
1.		$f(n)$
		$t_1(n)$
2.		$t_2(n)$
3		
:		
last		$t_{last}(n)$

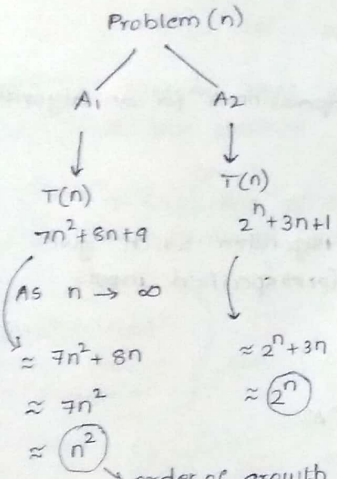
$$\sum_{i=1}^{last} t_i(n) = T(n)$$

$$T(n) = \sum_{i=1}^{last} t_i(n).$$

↓ operation count

"Running time"

* Execution time is machine specific.



↳ order of growth (or) rate of growth

1) Find running time
2) Drop lower order terms
3) Drop constant coefficients of highest order terms.

Algorithm having low order of growth is better.

** Algorithm complexity cannot be better than problem complexity.

i.e., problem complexity acts as lower bound for Algorithm complexity to that problem.

Basic operation : Which is repeated more number of times.

Algorithm Linear_Search (A, n, key)

1. i ← 1
2. while i ≤ n and A[i] ≠ key
3. i ← i + 1
4. if i ≤ n return i
5. else return -1

Analysis :-

1. n : Parameter Indicating Input size
2. comparison (A[i] ≠ key) : Basic operation
3. T(n) : no. of times basic operation is performed when 'n' is input size

↳ if it only depends on input size
(or) it also depends on actual input

↳ yes, Worst-case RT
Best-case RT
Average case RT

4. ? Recurrence relation.

Best case RT

$T_{best}(n) = T(n) = 1$

$\approx n^0$

Worst case RT

$T(n) = n$

Average case running time (RT)

Assumptions :-

1. Search is successful
2. Each position is equally likely.

$x =$	1	2	----	n
x_i				
P_i	$\frac{1}{n}$	$\frac{1}{n}$		$\frac{1}{n}$: Probability

$E[x] = \sum_{i=1}^{n} P_i x_i$ (Expectation)

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

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

if search success probability 'p'

$0 \le p \le 1$

$x =$	x_1	x_2	----	x_n	n
	1	2		n	
$P =$	$\frac{p}{n}$	$\frac{p}{n}$	----	$\frac{p}{n}$	$(1-p)$

$E[x] = \sum_{i=1}^{n} P_i x_i$

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

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

Avg case RT $= \frac{p}{n} \frac{n(n+1)}{2} + (1-p)n$

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

Note :-

* Most of the time, average case running time will be closer to worst case time

Except for quick sort Algorithm

in which average case RT $\approx$ Best case RT

* This type of Analysis is called asymptotic Analysis

$f(n)$ $g(n)$ → Monotonically increasing functions

$n \in \{0, 1, 2, \ldots\}$

if $n_1 \le n_2$ then $f(n_1) \le f(n_2)$

i.e.,

Asymptotic Notations

1. Big-oh (O)
2. Big-omega (Ω)
3. Theta (Θ)
4. Little-oh (o)
5. Little-omega (ω)

Big-oh (O)

$f(n)$ & $g(n)$ are functions

$O(g(n)) = \{ f(n) : OOG\ f(n) \le OOG\ g(n) \}$

set

OOG → order of growth

Eg: $O(n^2) = \{ n, 1, 2, \ldots, \log n, n \log n, \frac{n(n+1)}{2}, 2n^2 + 8n\log n + 20$

$n^2, 3n^2, \frac{1}{n}, \ldots \}$

$\log n \in O(n^2)$

$n^{1.5} \in O(n^2)$, $\sum_{i=1}^{n} 1 = O(n^2)$

$2^{n+3} = O(2^n)$ ✓

$2^{2n+3} = O(2^n)$ ✗ } ∵ they cannot be expressed as constant times 2^n or $n!$

$(n+1)! = O(n!)$ ✗

2. Big-omega (Ω)

$\Omega(g(n)) = \{ f(n) : \text{OOG } f(n) \geq \text{OOG } g(n) \}$

$\Omega(n^2) = \{ n!, 8n^3+9n+4, 2^n, (1.1)^n+7n+9, \frac{n(n+1)}{2}, \ldots$

↓ base should be more than 1

Eg:

$n^2 \log n \in \Omega(n^2)$ ✓

$n^2 \in \Omega(n^2)$ ✓

$6n+7 \in \Omega(n)$ ✓

$\sum_{i=1}^{n} i = \Omega(n^2)$ ✓

$\sum_{i=1}^{n} 1 = \Omega(n^2)$ ✗

$n! = \Omega(2^n)$ ✓

$n^n = \Omega(n!)$ ✓

$7\log_3 n + 8 = \Omega(\log_2 n)$ ✓

↓

$(\log_2 n \cdot \log_3 2)$

Irrespective of base for logarithmic rate of growths

$2^{n+3} = \Omega(2^n)$ ✓

$2^{2n+3} = \Omega(2^n)$ ✓

$(n+1)! = \Omega(n!)$ ✓

3. Theta (Θ)

$\Theta(g(n)) = \{ f(n) : \text{OOG } f(n) = \text{OOG } g(n) \}$

Ex: $\Theta(n^2) = \{ n^2, 7n^2+8n+9, \frac{n(n-1)}{2}, 2^{\log_{\sqrt{2}} n}$

$n^2 \in \Theta(n^2)$

$\sum_{i=1}^{n} i = \Theta(n^2)$ ✓

$\sum_{i=1}^{n} 1 = \Theta(n^2)$ ✗

$2^{n+3} = \Theta(2^n)$ ✓

** $\log(n!) = \Theta(n \log n)$ ✓ $= \Omega(n \log n) = O(n \log n)$

$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$ → stirling's approximate formula.

4. Little-oh (o)

$o(g(n)) = \{ f(n) : \text{OOG } f(n) < \text{OOG } g(n) \}$

Eg: $o(n^2) = \{ \log n, n\log n, 1, 2, 7n^{1.5}+6n+100, (0.9)^n+6n+5, \ldots$

$\frac{1}{n}, \sqrt{n}, \ldots \}$

$\log n! = o(n \log n)$ ✗ , $n! = o(n^n)$ ✓ , $2^n = o(n^{9999})$ ✗

$2^n = o(n!)$ ✓ , $(\log n)^{99999} = o(n^{0.00001})$ ✓, $n^{1000} = o(2^{0.0001n})$ ✓

5. Little-omega (ω)

$\omega(g(n)) = \{ f(n) : \text{OOG } f(n) > \text{OOG } g(n) \}$

Ex: $\omega(n^2) = \{ n^3, 7n^{2.01}+8n+9, n^2\log n, 2^{n+3}, n^n, n!, 2^{2^n}, 2^n+6n^2+9, \ldots \}$

$2^{2n+1} = \omega(2^n)$ ✓

$2^{n+1} = \omega(2^n)$ ✗

$(n+1)! = \omega(n!)$ ✓

$n! = \omega(2^n)$ ✓

$n^{0.001} = \omega((\log n)^{900})$ ✓

$3^{0.001n} = \omega(n^{99999})$ ✓

* upper bound should be as close as possible. (small)

* Lower bound should be as high as possible

* Worst case R.T is noted by Big-oh
  General R.T of that algo Eg: Linear search O(n)

* Best case R.T is noted by Big-omega
  General R.T of that algo
  Eg: Linear search Ω(n)

Note:

$7(\log n)^2 + 8(\log n) + 9$ → poly logarithmic

$7n^2 + 8n + 9$ → polynomial

$(\log n)^a$ n^b $b>0$

OOG is very small for polylogarithmic compared to polynomial

Note:-
* Order of growth of exponential function is very much greater than order of growth of polynomial function, provided base of exponential function must be greater than 1

Note:

n^2 $n^{2+\cos n}$ → $[-1\ 1]$ → n, n^3

∴ two functions $f(n)$ & $g(n)$ cannot be always be classified as $f(n) = O(g(n))$ or $f(n) = \Omega(g(n))$.

Formal definitions of Asymptotic Notations

$f(n)$ $g(n)$ → Input size

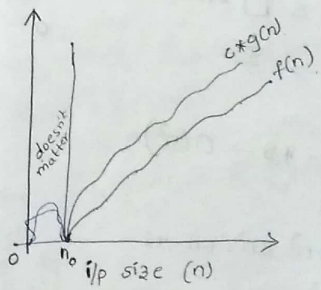
if $n_1 \leq n_2$ $n \in \{0, 1, 2, \ldots\}$

then $f(n_1) \leq f(n_2)$

(i) $f(n) = O(g(n))$

iff $f(n) \leq c * g(n)$, $\forall\ n \geq n_0$

c, n_0 are positive



O gives upper bound that need not be tight

Ex:- show that $\frac{n(n+1)}{2} = O(n^2)$

Proof :-

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

$g(n) = n^2$

$\frac{1}{2}n^2 + \frac{1}{2}n \leq \square^{?} * n^2, \quad \forall n \geq \square^{?}$

$\square^{?}$ = c, $\square^{?}$ = n_0

Find c, n_0

c = sum of coefficients } → For positive coefficients

n_0 is any positive numbers

Ex :- show that $\frac{n(n-1)}{2} = O(n^2)$

find c & n_0

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

$g(n) = n^2$

$\frac{1}{2}n^2 - \frac{1}{2}n \leq \square * n^2, \quad \forall n \geq \square$

$\square$ = $\frac{1}{2}$, $\square$ = 0

Ex :- $7n^3 - 8n^2 + 6n - 48 = O(n^3)$

find c & n_0

$f(n) = 7n^3 - 8n^2 + 6n - 48$

$g(n) = n^3$

$7n^3 - 8n^2 + 6n - 48 \leq \square * n^3, \quad \forall n \geq \square$

$\square$ = 13, $\square$ = 0

4. show that $\log n! = O(n \log n)$

$\log n! = \log(1 * 2 * \cdots * (n-1) * n)$

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

$\leq n \log n$

$1 * n \log n, \quad \forall n \geq ?$

$\Rightarrow c = 1, \quad n_0 = 1$

Theorem : If $f(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0$

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

Proof :- $f(n) = \sum_{i=0}^{m} a_i n^i$

$\leq \sum_{i=0}^{m} |a_i| n^i$

$= n^m \sum_{i=0}^{m} |a_i| n^{i-m}$ → max value is 1

$\leq n^m \sum_{i=0}^{m} |a_i|$

Theorem :- show that

If Algorithm has two parts

$f_1(n) = O(g_1(n))$

$f_2(n) = O(g_2(n))$

then $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$

$O(g_1(n))$

$f_1(n)$

$f_2(n)$

$= O(g_2(n))$

Proof:- Given $f_1(n) = O(g_1(n))$ & $f_2(n) = O(g_2(n))$

↳ $f_1(n) \leq c_1 * g_1(n)$, $\forall\ n \geq n_1 \rightarrow (1)$

↳ $f_2(n) \leq c_2 * g_2(n)$, $\forall\ n \geq n_2 \rightarrow (2)$

(1) + (2)

$f_1(n) + f_2(n) \leq c_1 * g_1(n) + c_2 * g_2(n)$, $\forall\ n \geq \max(n_1, n_2)$ → n_3

$\leq c_3 * g_1(n) + c_3 * g_2(n)$, $\forall\ n \geq n_3$

$= c_3\,(g_1(n) + g_2(n))$, $\forall\ n \geq n_3$

$\therefore f_1(n) + f_2(n) \leq c_3 * 2 * \max(g_1(n), g_2(n))$, $\forall\ n \geq n_3$

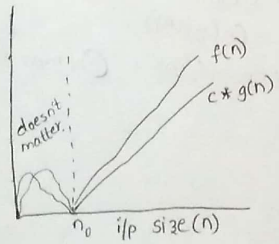
$f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$

2. Ω

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

iff $\exists\ c$ & n_0 such that

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



Ex

Ex:- show that $\frac{n(n+1)}{2} = \Omega(n^2)$

$\frac{1}{2}n^2 + \frac{1}{2}n \geq \square * n^2$, $\forall\ n \geq \square$

$c = \frac{1}{2}$, $n_0 = 0$

Ex:- show that $\frac{n(n-1)}{2} = \Omega(n^2)$

$\frac{1}{2}n^2 - \frac{1}{2}n \geq \square * n^2$, $\forall\ n \geq \square$

$c = \frac{1}{4}$, $n_0 = 2$

Ex:- show that $7n^2 - 6n - 200 = \Omega(n^2)$

$7n^2 - 6n - 200 \geq \square * n^2$, $\forall\ n \geq \square$

$c = 1$, $n_0 = 100$

Ex:- If $f(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0$, then $f(n) = \Omega(n^m)$ & $a_m > 0$

Ex: show that $\log n! = \Omega(n \log n)$

3. Θ

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

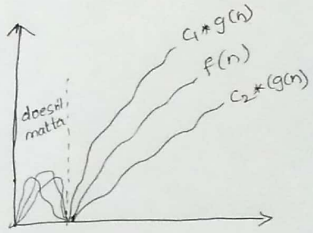
iff

$f(n) = O(g(n))$ $\quad f(n) = \Omega(g(n))$

$f(n) \leq c_1 * g(n), \forall n \geq n_1$ $\quad f(n) \geq c_2 * g(n), \forall n \geq n_2$

$\rightarrow (1)$ $\quad \rightarrow (2)$

$$c_2 * g(n) \leq f(n) \leq c_1 * g(n), \forall n \geq n_0$$



Ex: show that

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

$$\frac{1}{4} n^2 \leq \frac{n(n-1)}{2} \leq \frac{1}{2} n^2, \forall n \geq 2$$

$$c_2 = \frac{1}{4}, \quad c_1 = \frac{1}{2}, \quad n_0 = 2$$

Show that

(i) $8n^3 - 9n^2 + 6n + 20 = \theta(n^3)$

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

Ex: show that

if $f(n) = \sum_{i=0}^{m} a_i n^i$, then $f(n) = \Theta(n^m)$ & $a_m > 0$

4) o

$f(n) = o(g(n))$

iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Ex: $7n^3 + 9n^2 + 20n + 60 = o(n^4)$

$$\lim_{n\to\infty} \frac{7n^3+9n^2+20n+60}{n^4}$$

$$= 0$$

Ex: $n^a = o(b^n)$, where $b > 1$

$$\lim_{n\to\infty} \frac{n^a}{b^n} = 0$$

Ex: $(\log n)^a = o(b^{\log n})$, where $b > 1$

$b^{\log n} \rightarrow n^{\log b}$.

$$\lim_{n\to\infty} \frac{(\log n)^a}{n^{\log b}} = 0.$$

$(\log n)^a = o(n^b)$, $b > 0$

5) ω

$f(n) = \omega(g(n))$

iff

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

Ex: $7n^3 + 9n^2 + 20n + 60 = \omega(n^2)$

Ex: $b^n = \omega(n^a)$, where $b > 1$

$$\lim_{n\to\infty} \frac{n^a}{b^n} = 0.$$

OOG :-

Exponential > Polynomial > Polylogarithmic

Reflexive

1) $f(n) = O(f(n))$ ✓

$= \Omega(f(n))$ ✓

$= \Theta(f(n))$ ✓

$= o(f(n))$ ✗

$= \omega(f(n))$ ✗

2) If $f(n) = O(g(n))$ then $g(n) = \Omega(f(n))$ ✓

3) If $f(n) = \Omega(g(n))$ then $f(n) = \omega(g(n))$ ✗

4) Transitive

If $f(n) = \Omega(g(n))$, $g(n) = \Omega(h(n))$ then $f(n) = \Omega(h(n))$

O ✓

ω ✓

o ✓

Θ ✓

Algorithm

Isprime(n)

```
1. for i ← 2 to n-1
2.     if n % i = 0
3.         print "Not prime"
4.         return;
5. print "prime"
```

i/p size :- no. of bits required (i.e, memory)

$m = \lfloor \log_2 n \rfloor + 1$ [∵ Represented in binary]

(i/p size)

Worst-case RT:

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

$T(m) = 2^{m-1} - 2$

$OOG = 2^m$ (exponential)

For sorting problem, also

i/p size is $m = \lceil \log_2 (\text{max element in array}) \rceil \times n$.

(ceil)

$m \simeq n$.

primes is in P → refer AKS primality test

Insertion Sort

Algorithm

1. for $j \leftarrow 2$ to n
2. for $i \leftarrow j-1$ and key = A[j]
3. while $i > 0$ and $A[i] > key$
4. $A[i+1] \leftarrow A[i]$
5. $i \leftarrow i-1$
6. $A[i+1] \leftarrow key$

Eg:

10	25	30	12	
			↑	key=12
10	25	30	30	
10	25	25	30	
10	12	25	30	

To prove Correctness of algorithm

Method :- "Loop Invariant"

Elements A[1----j-1] are in sorted order

1. Initialization: $j=1 \Rightarrow$ array has one elements so always in sorted order

2. Maintenance: if LIV is true before the iteration starts it must be true before the start of next iteration

Here it is maintained

3. Termination : Loop finishes after finite no. of steps

Running time:

1. Input size = n
2. basic operation : comparison
3. T(n)
   ↓
   no. of times basic operation is performed

Check T(n) if it depends on n or actual values in input

∵ it depends on actual values of input also

Best-case :-

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

Best case running time of Insertion sort is : $O(n)$
: $\Omega(n)$
: $\Theta(n)$

Worst-case :

(reverse sorted order)

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

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

OOG of $T(n) : n^2$

Worst case running time of Insertion sort : $O(n^2)$
: $\Omega(n^2)$
: $\Theta(n^2)$

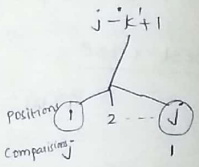
Average Case Running Time

Assumption: All possible Inputs are Equally likely

C_j → average no. of comparisons to insert an element in a[j] in already sorted a[1] --- a[j-1]

$$T(n) = \sum_{j=2}^{n} C_j$$

a[j] to a[k]



$$C_j = (1 + 2 + \cdots + j)/j$$

$$C_j = \frac{j(j+1)}{2}/j$$

$$C_j = \frac{j+1}{2}$$

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

$$= \frac{n(n+1)}{4} + \frac{n-1}{2} - 1$$

$$= \frac{n^2}{4} + \frac{3n}{4} - \frac{3}{2}$$

O of $T(n)$: n^2

Inversion : $i<j$ & $A[i] > A[j]$

if there are n elements

max. inversions : $\frac{n(n-1)}{2}$

min. inversions : 0

$$0 \le f(n) \le \frac{n(n-1)}{2}$$

$T(n) =$

$n + f(n)$

$\Theta(f(n))$

$\Theta(\max\{n, f(n)\})$

$\Omega(f(n))$

$T(n) = O(n^2), \Omega(n)$

Running time of Insertion sort

Recursive algorithm

GCD :- $x, y \in \mathbb{Z}$, $x > y$

$GCD(x,y)$ = Greatest positive Integer which divides x & y without leaving any remainder

$GCD(x,y) = GCD(-x,y)$

i.e., $GCD(x,y) = GCD(|x|,|y|)$

Euclid's Algorithm for finding GCD

** $\boxed{GCD(x,y) = GCD(y, x \bmod y)}$

$GCD(x,0) = |x|$

Algorithm : Euclid - GCD(x,y)

1. if $y = 0$
2. return $|x|$;
3. else return $(y, x \bmod y)$

Correctness of Algorithm :-

⇒ GCD Theorem :-

For any non-negative integer 'a' and any positive integer b, $GCD(a,b) = GCD(b, a \bmod b)$

Proof :-

Let $d = gcd(a,b)$,

If d/a and d/b ($d/a \Rightarrow$ d divides a)

$a = b\left\lfloor \frac{a}{b} \right\rfloor + a \bmod b$ (quotient, remainder)

$a \bmod b = a - \left\lfloor \frac{a}{b} \right\rfloor b \longrightarrow (1)$

$\Rightarrow d/a \bmod b$ [∵ If d/a & d/b then $d/(ax+by)$]

& d/b

$\hookrightarrow d/gcd(b, a \bmod b)$ (if c/a, c/b then $c/gcd(a,b)$)

$= gcd(a,b) / gcd(b, a \bmod b)$

similarly $gcd(b, a \bmod b) / gcd(a,b)$

$\Rightarrow gcd(a,b) = gcd(b, a \bmod b)$

Running Time :-

Input size = x & y [no. of bits in binary representation of x]

No. of recursive calls

R.T depends not only on the input size but also on the actual values of input

(i) Best case :- constant

$O(1)$

(ii) Worst case :-

1845

Lame's Theorem :-

Let k = no. of steps in Euclids algorithm for the input x, y where $x > y$.

Then

$y \geq F_k$, F_k is k^{th} Fibonacci number

Proof :- (x_k, y_k)

{ k steps

$(x, 0)$

$\gcd(-,-)$

claim :- If (x_{k+1}, y_{k+1}), (x_k, y_k), (x_{k-1}, y_{k-1}) } 3 consecutive pairs in Euclids way of finding gcd

then $y_{k+1} \geq y_k + y_{k-1}$

Proof of claim :-

$x_k = y_{k+1}$

$y_k = x_{k+1} \bmod y_{k+1}$

$x_k = (q)\, y_k + x_k \bmod y_k$

$x_k \geq y_k + y_{k-1}$

$y_{k+1} \geq y_k + y_{k-1}$

$y_k \;) x_k \;(q$

$x_k \bmod y_k$

Hence claim is proved.

continuation of actual proof :-

Mathematical Induction k

Base case : $k = 1$:

$y \geq F_1$

Inductive Hypothesis : $y \geq F_k$ $(\leq k)$

Inductive step : $k+1$ such that

$y_{k+1} \geq y_k + y_{k-1}$

$y_{k+1} \geq F_k + F_{k-1}$

$y_{k+1} \geq F_{k+1}$

Hence prooved Lame's theorem.

$$F_k = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2}\right)^k - \left(\frac{1-\sqrt{5}}{2}\right)^k \right]$$

ϕ ($1.6180\ldots$) , $(1-\phi)$ (-0.618)

$\hat{\phi} = 1 - \phi$

$$F_k = \frac{1}{\sqrt{5}}\left[\phi^k - \hat{\phi}^k\right]$$

$$\simeq \frac{1}{\sqrt{5}}\phi^k \quad \text{for very large } k$$

$$y \geq \frac{1}{\sqrt{5}}\phi^k$$

$$\phi^k \leq y\sqrt{5}$$

$$k \leq \log_{\left(\frac{1+\sqrt{5}}{2}\right)} y\sqrt{5}$$

$$K = O(\log y)$$

$$k \leq \square + \square * \log_2 y$$

Worst case running time : $O\left(\log_2 \min(x, y)\right)$

ch-31 in coremen Section-2 (31.2)

2 consecutive fibonacci numbers

② Max Finding Algorithm

Algorithm Find_max (A, n)

1. max ← -1
2. for i ← 1 to n
3. if max < A[i]
4. max ← A[i]
5. return max

if considered as base step → R.T = O(n)

1. n
2. Assignment
3. T(n)

?? Average no. of times Assignment operation is performed.

Distribution :- → {1, 2, - - - n}
All values in input are distinct (A[i])

$S_{n,k}$ - all permutations where no. of assignment operations are exactly 'k'

k = 1 to n.

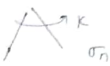
↓ classified to n classes

class 1 = no. of assignm operations is 1

$$P_{n,k} = \frac{S_{n,k}}{n!}$$

$$E[x] = \sum_{k=1}^{n} k \cdot P_{n,k} = \sum_{k=1}^{n} k \cdot \frac{S_{n,k}}{n!}$$

$S_{n,k}$: $(\sigma_1, \sigma_2, \ldots, \sigma_n)$



$\sigma_n = n$ $\quad$ $\sigma_n \neq n$

$(\sigma_1, \ldots, \sigma_{n-1})$ $\quad$ $(n-1) \cdot S_{n-1,k}$

$k-1$ Assignment

$S_{n-1,\,k-1}$

$$\boxed{S_{n,k} = S_{n-1,k-1} + (n-1) \cdot S_{n-1,k}} \rightarrow (1)$$

↓ Generating Function

$$S_n(x) = \sum_{k=0}^{\infty} S_{n,k} \, x^k$$

↓ $S_{n,0}$, $S_{n,k}$ $k>n = 0$.

$$S_n(x) = \sum_{k=1}^{n} S_{n,k} \cdot x^k$$

From Eq (1) $\times x^k$

& summation $\sum_{k=1}^{n}$

$$\sum_{k=1}^{n} S_{n,k} \cdot x^k = \sum_{k=1}^{n} S_{n-1,k-1} \, x^{k-1} + \sum_{k=1}^{n} (n-1) \, S_{n-1,k} \, x^k$$

$$S_n(x) = x \, S_{n-1}(x) + (n-1) \, S_{n-1}(x)$$

$$S_n(x) = (x+n-1) \, S_{n-1}(x) \rightarrow (3)$$

$$S_1(x) = x \cdot S_0(x)$$
$$= x$$

$$S_2(x) = (x+2-1) \, S_1(x)$$
$$= (x+1) \, x$$

$$S_3(x) = (x+3-1) \, S_2(x)$$
$$= (x+2)(x+1)(x)$$

$$\vdots$$

$$S_n(x) = x(x+1)(x+2)\ldots(x+n-1)$$
$$= \prod_{j=0}^{n-1} (x+j)$$

Apply logarithm

$$\log(S_n(x)) = \log x + \log(x+1) + \ldots + \log(x+n-1)$$

derivative on both side

$$\frac{S_n'(x)}{S_n(x)} = \frac{1}{x} + \frac{1}{x+1} + \ldots + \frac{1}{x+n-1}$$

$$\frac{S_n'(1)}{S_n(1)} = \frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{n}$$

$$= H_n \quad \text{[Harmonic mean]}$$

$$\frac{S_n'(1)}{S_n(1)} = H_n \rightarrow (4)$$

$$E[x] = \sum_{k=1}^{n} k \cdot \frac{S_{n,k}}{n!} = \frac{S_n'(1)}{S_n(1)} \quad [\text{from } (5)]$$

$$S_n(x) = \sum_{k=1}^{n} S_{n,k} \cdot x^k$$

$$S_n'(x) = \sum_{k=1}^{n} S_{n,k} \cdot k \cdot x^{k-1}$$

$$S_n'(1) = \sum_{k=1}^{n} k \cdot S_{n,k} \rightarrow (5)$$

$$E[x] = \frac{S_n'(1)}{S_n(1)} = H_n$$

$$H_n = \sum_{i=1}^{n} \frac{1}{i}$$

if $f(k)$ is monotonically decreasing function

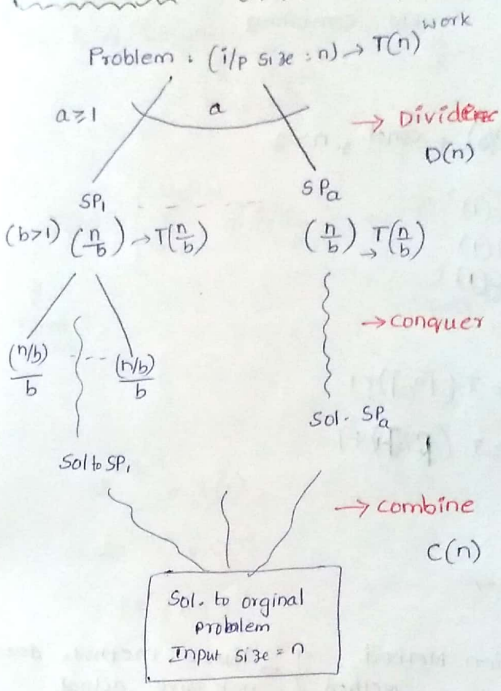
then $\int_{m}^{n+1} f(x)\,dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m-1}^{n} f(x)\,dx$

$$\int_{1}^{n+1} \frac{1}{x}\,dx \leq \sum_{k=1}^{n} \frac{1}{k} \leq \int_{1-1}^{n} \frac{1}{x}\,dx$$

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

Design

DIVIDE AND CONQUER



$$T(n) = D(n) + a\,T\left(\frac{n}{b}\right) + C(n)$$

$$T(n) = \begin{cases} a\,T\left(\frac{n}{b}\right) + f(n), & n > \square \\ \ldots & n \leq ? - 1 \end{cases}$$

$a \geq 1$, $b > 1$ and $f(n) > 0$

In binary search ✓ Dividing – Finding mid element
✓ conquering
No combining

$T(n) \leq T(n/2) + \text{const}, \quad n \geq 2$

$T(1) = \Theta(1)$
$O(1)$
$\Omega(1)$

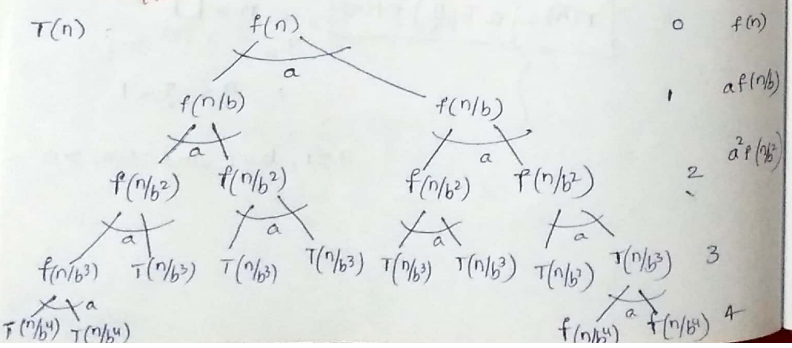
$T(n) \leq T(\lceil n/2 \rceil) + 1$

$T(n) \geq T(\lfloor n/2 \rfloor + 1) + 1$

/* refer ch-4 of coremen */

substitution Method
Recursion tree Method
Master Method
} → These methods does not give actual value of T(n) but they give orders of growth.

Recursion Tree



Level	Work done
0	$f(n)$
1	$af(n/b)$
2	$a^2 f(n/b^2)$
3	
4	

For ith level → $T\left(\frac{n}{b^i}\right)$

if h is height

$h = \log_b n$

$\frac{n}{b^h} = 1$

$$T(n) = \left[\sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \right] + \Theta\left(n^{\log_b a}\right) \rightarrow (1)$$

↓ Total work

↓ Last level work

$$a^{\log_b n} * T(1)$$
$$= a^{\log_b n} * \Theta(1)$$
$$= \Theta\left(n^{\log_b a}\right)$$

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right) \rightarrow (2)$$

$$T(n) = g(n) + \Theta\left(n^{\log_b a}\right)$$

Case :- 1. $f(n) = O\left(n^{\log_b a - \epsilon}\right)$, for some $\epsilon > 0$

$$f(n) < n^{\log_b a}$$

$$f\left(n/b^j\right) = O\left(\left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}\right) \text{ for some } \epsilon > 0$$

$$g(n) = O\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}\right)$$

$$\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon} = n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} \frac{a^j \, b^{j\epsilon}}{(b^j)^{\log_b a}}$$

$$= n^{\log_b a - \epsilon} \cdot 1 \cdot \frac{(b^\epsilon)^{\log_b n} - 1}{b^\epsilon - 1}$$

$$= n^{\log_b a - \epsilon} \cdot \left(\frac{n^\epsilon - 1}{b^\epsilon - 1}\right)$$

$$= n^{\log_b a}$$

$$g(n) = O\left(n^{\log_b a}\right)$$

$$T(n) = g(n) + \Theta\left(n^{\log_b a}\right)$$

$$T(n) = O\left(n^{\log_b a}\right) + \Theta\left(n^{\log_b a}\right)$$

$$\boxed{T(n) = \Theta\left(n^{\log_b a}\right)}$$

Case 2 : If $f(n) = \Theta\left(n^{\log_b a}\right)$

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

$$= \sum_{j=0}^{\log_b n - 1} a^j \, \Theta\left(\left(\frac{n}{b^j}\right)^{\log_b a}\right)$$

$$= \Theta\left(\sum_{j=0}^{\log_b n - 1}\right)$$

$$= \Theta\left(n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \left(a^j \left(\frac{1}{b^j}\right)^{\log_b a}\right)\right.$$

$$= \Theta\left(n^{\log_b a} \cdot \log_b n\right)$$

$$T(n) = \Theta\left(n^{\log_b a} \cdot \log_b n\right) + \Theta\left(n^{\log_b a}\right)$$

$$\boxed{T(n) = \Theta\left(n^{\log_b a} \cdot \log_b n\right)}$$

$T(n) = a\,T(n/b) + f(n)$, if $n > ?$

$T(n) =$ ___ , if $n \leq ? - 1$

$$T(n) = \underbrace{\sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)}_{g(n)} + \theta\left(n^{\log_b a}\right) \longrightarrow (1)$$

Master theorem.

case 1 :- If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$, for some $\epsilon > 0$

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

case 2 :- If $f(n) = \theta\left(n^{\log_b a}\right)$ then

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

case 3 :- If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some $\epsilon > 0$

and $a\,f\left(\frac{n}{b}\right) \leq c\,f(n)$ for some $c < 1$

then $T(n) = \theta(f(n))$

$$f\left(\frac{n}{b}\right) \leq \frac{c}{a} f(n)$$

$$f\left(\frac{n}{b^2}\right) \leq \frac{c}{a} f\left(\frac{n}{b}\right)$$

$$\Rightarrow f\left(\frac{n}{b^2}\right) \leq \left(\frac{c}{a}\right)^2 f(n)$$

$$f\left(\frac{n}{b^3}\right) \leq \left(\frac{c}{a}\right)^3 f(n)$$

$$\vdots$$

$$f\left(\frac{n}{b^j}\right) \leq \left(\frac{c}{a}\right)^j f(n)$$

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

$$\leq \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{c}{a^j}\right) f(n)$$

$$= f(n) \sum_{j=0}^{\log_b n - 1} c^j$$

$$= f(n)\,\frac{\left(c^{\log_b n} - 1\right)}{c - 1}$$

$$< f(n)\,\frac{\left(n^{\log_b c} - 1\right)}{c - 1}$$

$g(n) = O(f(n))$ ← constant

$\Rightarrow T(n) = \theta(f(n))$

/* 4.6 correctness of master theorem */

polynomially lesser

" Equal

" higher

Eg :- 1. $T(n) = 4\,T(n/2) + 2n^3$

$f(n) = 2n^3$

if $f(n)$ is polynomial by default

$a\,f(n/b) \leq c\,f(n)$ ← this condition will be true

$4\,f(n/2) \leq c \cdot 2n^3$

$4 * \left(\frac{n}{2}\right)^3 \leq c \cdot 2n^3$

$c \geq 1/2$ ⇒ less than 1 ⇒ $T(n) = \theta(n^3)$

2. $T(n) = 8T\left(\frac{n}{2}\right) + n^2$, $n > ?$

$= \text{const}$, $n \leq ? - 1$

$f(n) = n^2$

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

$a = 8$

$b = 2$

$f(n) = n^2$

$\underline{f(n)}$ $\qquad$ $\underline{n^{\log_b a}}$

n^2 $\qquad$ $n^{\log_2 8} = n^3$

$n^2 = O(n^{3-0.5})$, for some $\epsilon = 0.5$

∴ By case 1 of master theorem

$T(n) = \theta(n^3)$

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

$a = 1$

$b = 2$

$f(n) = 1$

$\underline{f(n)}$ $\qquad$ $\underline{n^{\log_b a}}$

$1 = n^0$ $\qquad$ $n^{\log_2 1} = n^0$

$1 = \theta\left(n^{\log_2 1}\right)$

∴ By case 2 of Master's theorem

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

4) $T(n) \leq T\left(\frac{n}{2}\right) + 1$

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

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

$a = 2$

$b = \frac{3}{2}$

$f(n) = \theta(n)$

$\underline{f(n)}$ $\qquad$ $\underline{\log_{3/2} 2} > 1$

∴ By case 1 of Master theorem

$T(n) = \theta\left(n^{\log_{3/2} 2}\right)$

6) $T(n) = 4T\left(\frac{n}{2}\right) - n^2$ → ∵ $f(n) < 0$

7) $T(n) = nT\left(\frac{n}{2}\right) + 1$ → ∵ a is non constant

} Master theorem not applicable

8) $T(n) = T(\sqrt{n}) + 1$,

Master theorem not applicable directly

change of variables

$n = 2^k$, $k \geq 0$

$T(2^k) = T\left(2^{k/2}\right) + 1$

Let $S(k) = T(2^k)$

$S(k) = S(k/2) + 1$

$S(k) = \theta(\log k)$

$T(2^k) = \theta(\log k)$

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

9. $T(n) = 2T(\sqrt{n}) + 1$

10. $T(n) = T(\sqrt{n}) + \log n$

Let $n = 2^k$, $k \geq 0$

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

Let $S(k) = T(2^k)$

$S^k = S(k/2) + k$

$f(n)$	$\log_b^a$
k	$k^0 = 1$

By 3rd case

$S(n) = \theta(k)$

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

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

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

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

↳ by third case of Master theorem

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

$T(n) = \Omega(n^2) \rightarrow (2)$

From (1) & (2)

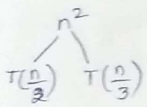
$T(n) = \theta(n^2)$

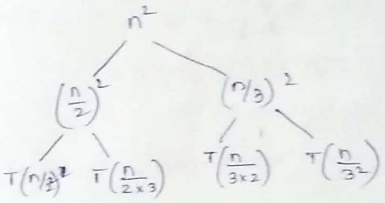
11) Without Master theorem

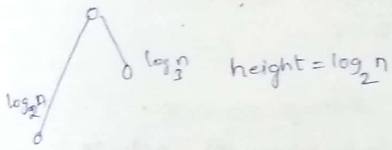
solve $T(n) = T(\frac{n}{3}) + T(\frac{n}{2}) + n^2$

Recursion tree Method:

$T(n)$:







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

$T(\frac{n}{2}) = T(\frac{n}{2^2}) + T(\frac{n}{3\times2}) + (\frac{n}{2})^2$

$\leq 2 \cdot (\frac{n}{2})^2$

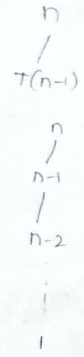
Recursion tree Method helps to guess on order of growth.

But bounds may not be tight.

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

No master theorem

$T(n)$:



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

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

Master theorem is not a way of solving recurrence relation. It just gives the highest ordered term.

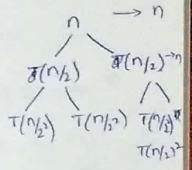
Substitution Method :

guess OOG & prove.

1) Make a good guess

2) show that your guess is correct : Mathematical Induction.

Ex:- $T(n) = \begin{cases} 1, & \text{if } n=1 \\ 2T(\frac{n}{2}) + n, & \text{if } n \geq 2 \end{cases}$



Sol:- 1. Guess: $T(n) = n\log n + n$

Amount of work = $n \times \log n$.

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

2. Guess is correct :

Base case: $n=1$:

$T(1) = 1\log(1) + 1$ ✓

Inductive Hypothesis

$T(n) = n\log n + n$ ✓

Inductive step:

$n = k+1$

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

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

$= n\log n - n\log_2 2 + n + n$

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

Guess $T(n) = O(n \log n)$ → from recurrence tree.

$\leq c * n \log n, \quad \forall n \geq d$

10.

$$T(n) = 2T(\tfrac{n}{2}) + n$$

$$\leq 2\left(d \tfrac{n}{2} \log n/2\right) + n$$

$$= cn \log n/2 + n$$

$$= cn \log n - cn + n$$

$$= cn \log n + (1-c)n \qquad (c \geq 1)$$

$$T(n) \leq cn \log n$$

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

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

$$\geq c * n \log n$$

$$T(n) \geq 2\left(d \tfrac{n}{2} \log \tfrac{n}{2}\right) + n$$

$$\geq 2dn \log n/2 + n$$

$$\geq dn \log n - dn + n$$

$$\geq dn \log n + (1-d)n$$

$d < 1$.

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

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

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

$T(n) \leq c * n \log n, \quad \forall n \geq 1.$ → (1)

$< n$

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

$= 2\left[d \tfrac{n}{2} \log n/2\right] + n$

$= dn \log n/2 + n$

$= dn \log n - dn \log 2 + n$

$= dn \log n - dn + n$

$= dn \log n - [n(d-1)]$

$T(n) \leq dn \log n$ $(d-1) \geq 0$

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

$nd - n > 0$

$n(d-1) > 0$

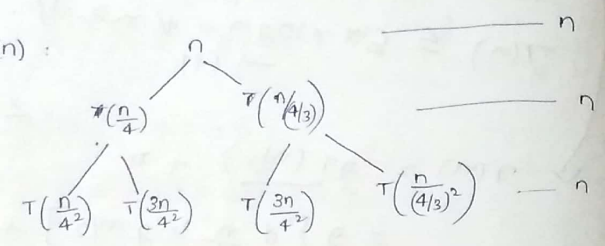
$n(d-1) \geq 0$

4/2/19

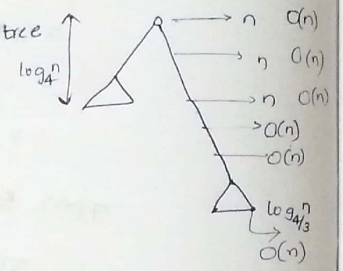
$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + \theta(n)$ → using case 1 of master theorem

$T(n) = O\left(n^{\log_{4/3} 2}\right)$

$T(n)$:



It is not complete binary tree



Total work done $= O(n \log n)$

minimum work done $= (1 + \log_4 n)\,\theta(n)$

$= \theta(n \log n)$

$T(n) \leq c * n\log n, \forall\ n \geq n_0$

↓

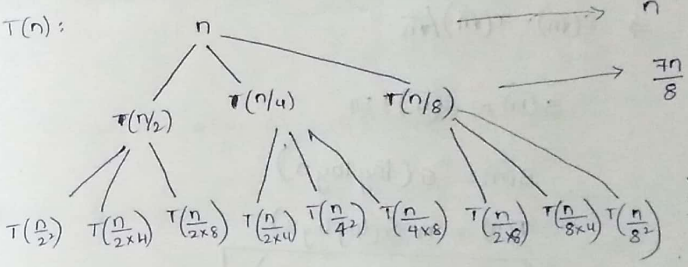
$T(n) \leq c*n\log n - ( )$

$( ) \geq 0$

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

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

$T(n)$:



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

Guess : $T(n) = O(n)$

$T(n) \leq c * \frac{n}{2} + c * \frac{n}{4} + c * \frac{n}{8} + n$

~~$T(n) \leq c*n, n \geq n_0$~~ ~~(1)~~

~~using Mathematical Induction~~

$T(n) \leq \frac{7c}{8} n + n$

$= n\left(1 + \frac{7c}{8}\right)$

$1 + \frac{7c}{8} = c$

$c = 8$

$T(n) \leq c*n$

③ $T(n) = \sqrt{n}\, T(\sqrt{n}) + 50n$

Sol:-

$$\frac{T(n)}{n} = \frac{T(\sqrt{n})}{\sqrt{n}} + 50$$

Let $S(n) = T(n)/n$

$\Rightarrow S(\sqrt{n}) = T(\sqrt{n})/\sqrt{n}$

$$S(n) = S(\sqrt{n}) + 50$$

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

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

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

④ $T(n) = T(n-2) + \log n$

$$\leq \log n + \log(n-2) + \cdots + \log 1$$

$$\leq \sum_{i=1}^{n} \log i$$

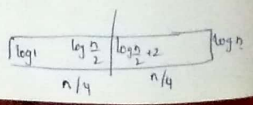
$$= O(n \log n)$$

show $T(n) = \Omega(n \log n)$

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

$$= \sum_{i=1}^{n/2} \log 2i$$

$$\geq \frac{n}{4} \log \frac{n}{4}$$



Maximum Subarray sum problem

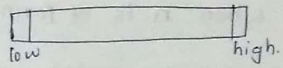
Input : $A[1 \ldots n]$; some are -ve

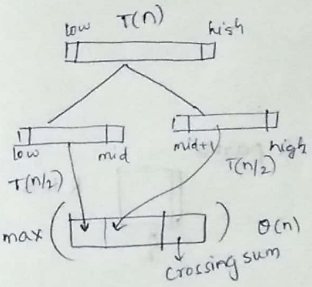
~~Input~~ output :- Indices i, j such that $A[i..j]$ has the greatest sum of any non-empty contiguous sub array of A, along with the sum of the values in $A[i \ldots j]$.

$\{-20, -50, 60, -20, -30, 10, 50, -60\}$

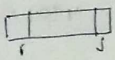
$n = high - low + 1$

Divide & conquering





A :



$low \leq i \leq high$

$i \leq j \leq high$,

Answer is in
i) only on left subarray
ii) " " right "
iii) crosses mid

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

dividing ↓ conquering ↓ combining ↓ max among three

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

by case 2 of master theorem

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

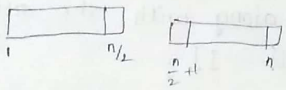
Problem complexity $O(n)$ (kadanes Algorithm)

Algorithmic ~~It~~ complexity cannot be better than problem complexity

Merge sort :-

$$T(n) = T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) + n-1 \qquad n \geq 2$$

↑ comparisons



i) min $\frac{n}{2}$

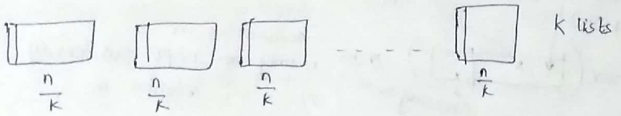
ii) max (alternative) n

When n is of form 2^k

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

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

if array is divided into k parts



Maintain a heap

1) Prove Pre/In/post Order traversal takes $O(n)$ timer.

Matrix Multiplication (using Divide & Conquer)

$A = (a_{ij})_{n \times n}$ $B = (b_{ij})_{n \times m}$

$C = AB$?

$( )_{r_1 \times c_2} = ( )_{r_1 \times c_1} \times (r_2 \times c_2)$

$=$

Total number of element Multiplication $= r_1 \times c_1 \times c_2$

(or)

$r_1 \times r_2 \times c_2$

Problem complexity $\Omega(n^2)$

Algorithmic Complexity $\Omega(n^2)$

if $n > 1$

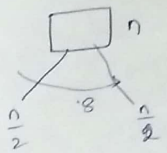
$$A = \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right)_{n \times n} \quad \text{(each block } \frac{n}{2} \times \frac{n}{2}\text{)}$$

$$B = \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right)_{n \times n}$$

$$C = \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right)_{n \times n} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}_{n \times n}$$

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

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



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

$$T(n) \begin{cases} = 8\,T\left(\frac{n}{2}\right) + \Theta(n^2), & n \geq 2 \\ = \Theta(1), & n = 1 \end{cases}$$

By case 1 of master theorem

$$T(n) = \Theta(n^3)$$

Problem complexity (PC) n^2

Algorithmic complexity (AC) n^3

$T(n) = 7\,T(n/2) + 18\left(\frac{n}{2}\right)^2$

Strassen's Matrix Multiplication

7 product matrices each of size $\frac{n}{2}$

$P_1 = A_{11} * S_1$

$P_2 = S_2 * B_{22}$

$P_3 = S_3 * B_{11}$

$P_4 = A_{22} * S_4$

$P_5 = S_5 * S_6$

$P_6 = S_7 * S_8$

$P_7 = S_9 * S_{10}$

$S_1 = B_{12} - B_{22}$

$S_2 = A_{11} + A_{12}$

$S_3 = A_{21} + A_{22}$

$S_4 = B_{21} - B_{11}$

$S_5 = A_{11} + A_{22}$

$S_6 = B_{11} + B_{22}$

$S_7 = A_{12} - A_{22}$

$S_8 = B_{21} + B_{22}$

$S_9 = A_{11} - A_{21}$

$S_{10} = B_{11} + B_{12}$

$C_{11} = P_5 + P_4 - P_2 + P_6$

$C_{12} = P_1 + P_2$

$C_{21} = P_3 + P_4$

$C_{22} = P_5 + P_1 - P_3 - P_4$

$T(n) = \Theta\left(n^{\log_2 7}\right)$

/*

If order is n $\quad 2^{k-1} < n < 2^k$

append $2^k - n$ rows & colums with 3010s

and solve

*/

If other than Square Matrix is given ?

/*

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

$n^{\log_4 a} < n^{\log_2 7}$

$a < 7^2$

$a < 49$

∴ At max 48 subproblems with $\frac{n}{4}$ size each.

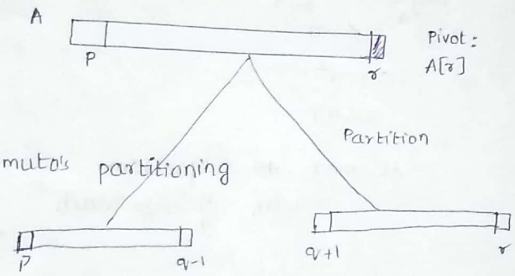
*/

Quick Sort:-

(Partition - exchange sort)

Tony Hoare 1959

Dividing part - more work
Combining - No work
Opposite of Merge sort

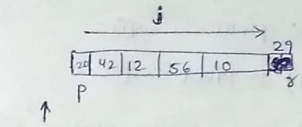


Algorithm

```
Quicksort(A,P,r)
1. if   P<r
2.      q ← PARTITION(A,P,r)
3.      QUICKSORT (A,P,q-1)
4.      QUICKSORT ( A,q+1,r)
```

```
PARTITION (A, P, r)
1. x ← A[r]
2. i ← p-1
3. for j ← p to r-1
4.      if A[j] ≤ x        → basic operation
5.          i ← i+1
6.          swap(A[i], A[j])
7. swap (A[i+1], A[r])
8. return i+1;
```

Lomuto's partitioning left → right

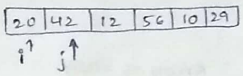


i = p-1

A[j] ≤ pivot ? if true

20 ≤ 29

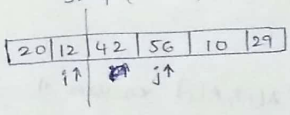
i++;
swap(A[i], A[j])



42 ≤ 29 ? X

12 ≤ 29 ✓

i++;
swap(A[i], A[j])



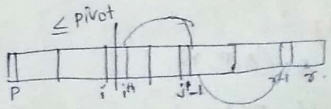
4 divisions of array

(p, i) → elements less than pivot

(i+1, j-1) → elements are higher than pivot

(j, r-1) → unseen

r → pivot



Pivot ← A[r]

A[p ... i] ≤ Pivot

A[i+1 ... j-1] > pivot

A[j ... r-1] not seen

Correctness of Partition Algorithm.

Loop Invariant: $A[p \ldots i] \leq$ pivot. $\rightarrow$①

$A[i+1 \ldots j-1] >$ pivot $\rightarrow$②

$A[j \ldots r-1]$ not known status $\rightarrow$③

$A[r]$ is pivot $\rightarrow$④

Initialisation: $i = p-1$

① ✓ ∵ Array is empty

② ✓ " "

③ " ✓

Maintainance:

swap($A[i], A[j]$) ensures it

Termination:

Running Time:

Algo

QS (A, P, r)

if $p < r$

$q \leftarrow$ partition (A, P, r)

QS $(A, P, q-1)$

QS $(A, q+1, r)$

partition Algo takes $\Theta(n)$ time

Given n elements $\Rightarrow n-1$ comparisons

I/p size:- n basic operation : comparison

Running time depends on the actual values of Input not only input size.

Worst-case:- → sorted order / Reverse sorted order

conquering & Dividing

$$T(n) = T(0) + T(n-1) + n-1 \quad , n \geq 2$$

↳atleast 2 elements in array

$T(1) =$ const

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

$$T(n) = \max_{0 \leq k \leq n-1} \{T(k) + T(n-1-k)\} + n-1$$

L	R
0	n-1
1	n-2
2	n-3
⋮	⋮
$T(k) \leftarrow k$	$n-1-k$ ↓ $T(n-1-k)$
⋮	⋮
$n-2$	1
$n-1$	0

Guess: $T(n) = O(n^2)$

show that guess is correct:

$$T(n) = \max_{0 \leq k \leq n-1} \{ ck^2 + c(n-1-k)^2 \} + n-1$$

$$= \max_{0 \leq k \leq n-1} \{ c(k^2 + (n-1)^2 + k^2 - 2(n-1)k \} + n-1.$$

$$\max_{0 \leq k \leq n-1} \{ c(2k^2 + (n-1)^2 - 2(n-1)k) \} + n-1$$

$k = 0$ or $k = n-1$

$$T(n) \leq c(n-1-0)^2 + (n-1)$$

$$= cn^2 - 2cn + c + n - 1$$

$$\leq cn^2$$

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

Best case:

$$T(n) = \min_{0 \leq k \leq n-1} \{ T(k) + T(n-1-k) \} + n - 1$$

$$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n), \quad n \geq 2$$

$$T(1) = \text{const}$$

By case 2 of Masters theorem

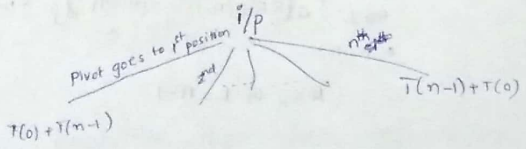
$$T(n) = O(n \log n)$$

Note:-

* If any sorting algorithm which uses comparison method to sort is found to take $\Omega(n \log n)$ complexity

Average-case:

Every chosen pivot has equal probability to go to any position.



$$T(n) = \frac{1}{n}\left[\sum_{k=0}^{n-1} T(k) + T(n-1-k) + (n-1)\right]$$

$$T(n) \approx 1.39\, n \log_2 n$$

Average case complexity of quick sort is just 39% more than Best case

Balanced partitioning

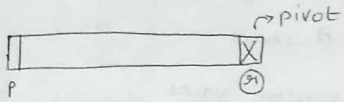
If array is partitioned in to $l : m$ ratio

$$T(n) = T\left(\frac{ln}{l+m}\right) + T\left(\frac{mn}{l+m}\right) + n - 1$$

↙ Average case / Best case

$$T(n) = T(9) + T(n-10) + n - 1, \quad n \geq 10.$$

↙ Worst case



$p, p+1, \ldots, r$

$p \leq i \leq r$

swap(A[i], A[r])

randomly chosing pivot

Randomised Quick Sort :

~~Randomised QS~~

```
Randomised_QS(A,P,r)
1. if p<r
2.    q ← Randomised_QS(A,p,r)
3.    Randomised_QS(A,p,q-1);
4.    Randomised_QS(A,q+1,r);
```

```
RANDOMISED_PARTITION(A,P,r)
1. i ← RANDOM(P,r)
2.    swap(A[i], A[r])
3.    PARTITION(A,P,r)
```

Indicator random Variable :

S → Sample space.

A → event

X → random Variable

$X_A = I\{A\}$

$E[X_A]$ = pr(occurs A) * 1 + pr(does not occur A) * 0

= pr(occurs A)

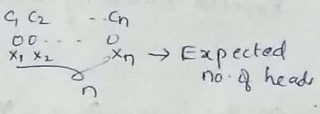
Q:- Determine the expected no. of heads when we flip a fair coin 1 time.

$X_A = I\{\text{getting head}\}$

$E[X_A]$ = probability of getting head

$= \frac{1}{2}$

Q:- Determine the expected no. of heads in 'n' - coin flips.



X

$E[X] = ?$

$X = \sum_{i=1}^{n} X_i$

$E[X] = \sum_{i=1}^{n} E[X_i]$

$= \sum_{i=1}^{n} Pr(\text{getting head on } i^{th} \text{ coin})$

$= \sum_{i=1}^{n} \frac{1}{2}$

$= \frac{n}{2}$

Expected running time of randomised Quick sort

calls to partition (atmost n) : $O(n)$

$x_1, x_2, x_3 \cdots, x_n$

$X \rightarrow$ In all calls to partition algo how many comparisons are performed.

$E[X] = ?$

Expected running time : $O(n+?)$

$z_1, z_2, \cdots z_n$ $\quad \forall\ 1 \le i \le n$, z_i is i^{th} smallest element

$z_1 : z_2, z_3, \cdots, z_n$

$z_2 : z_3, z_4, \cdots, z_n$

$z_3 : z_4, z_5, \cdots z_n$

$\vdots \quad \vdots$

$z_{n-1} : z_n$

event: Comparing one element with other

$X_{ij} = I\{ z_i \text{ is compared with } z_j \}$

$Z_{ij} = \{ z_i, z_{i+1}, \cdots, z_j \}$

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \rightarrow (1)$$ [By linearity of expectation

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr\{ z_i \text{ compared with } z_j \}$$

$Pr\{ z_i \text{ compared with } z_j \} = Pr\{ z_i \text{ pivot or } z_j \text{ pivot} \}$

$= Pr\{ z_i \text{ is pivot} \} + Pr\{ z_j \text{ is pivot} \}$

$= \frac{2}{j-i+1}$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{j=2}^{n-i+1} \frac{2}{j}$$

$$\le \sum_{i=1}^{n-1} \sum_{j=1}^{n} \frac{2}{j}$$

$$= \sum_{i=1}^{n-1} \log n$$

$E[X] = O(n \log n)$

Expected running time of randomised Quick sort

$O(n + n\log n)$

$= O(n \log n)$

Selection Problem

Input :- S of n distinct elements and i, $1 \le i \le n$

output :- $x \in S$, such that S has exactly $i-1$ elements smaller than x

i^{th} order statistic : i^{th} smallest element

$i = 1$: minimum

$i = n$: maximum

median : $i = \left(\frac{n+1}{2}\right)^{th}$ smallest (n : odd)

n : even, $i = \frac{n}{2}$: lower median

$i = \frac{n}{2} + 1$: upper median

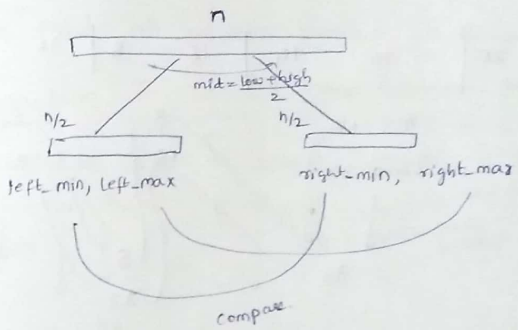
Problem complexity $O(n)$

Algorithmic Complexity $\Omega(n)$

Minimum :- $n-1$ comparison

Maximum:- $n-1$ comparision

Minimum and Maximum : $2(n-1)$ comparisons



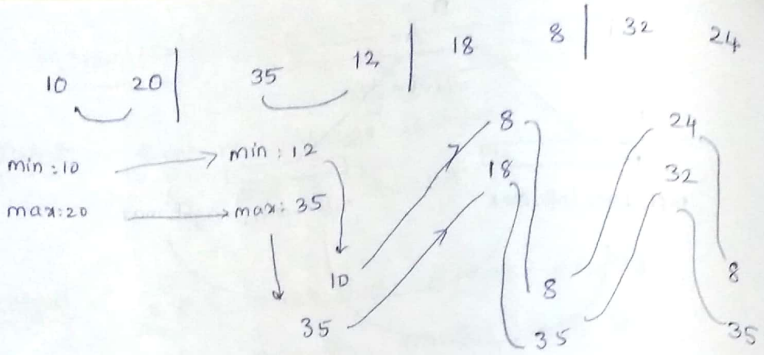
$T(n) = 2\,T\left(\frac{n}{2}\right) + 2 \quad , n \ge 2$

$T(1) = 0$

$T(2) = 2$

T

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



min = 8
max = 35

With this for every pair of elements 3 comparisons are done except for first pair (1 comparison)

n is even :-

Total comparisons $= 1 + \left(\frac{n-2}{2}\right)3$

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

if n is odd :- First element only considered as min & max.

Total comparisons $= \left(\frac{n-1}{2}\right)3$

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

$\therefore$ Total no. of comparisons $\leq 3\left\lceil \frac{n}{2} \right\rceil$

But All are asymptotically same.

to find ith smallest number in linear time.

Selection in Worst-Case Linear time :-

1) Divide into 5 element groups $O(n)$

$G_1, G_2, \ldots, G_{\lceil n/5 \rceil}$ (if n is not multiple of 5, n mod 5 no. of elements)

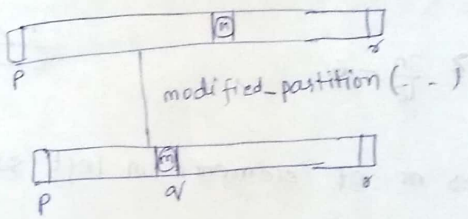
medians

2) $m_1, m_2, \ldots, m_{\lceil n/5 \rceil}$ $\lceil \frac{n}{5} \rceil O(1) = O(n)$

3) $m = \text{median}(m_1, m_2, \ldots, m_{\lceil n/5 \rceil})$, SELECTION $T\left(\frac{n}{5}\right)$

4) m : pivot : calling MODIFIED-PARTITION $O(n)$

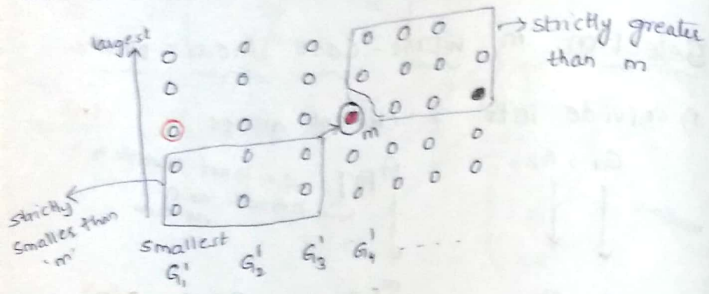
$1 \leq i \leq r-p+1$



Let $k = q - p + 1$

5) $i = k \Rightarrow$ m. (found)
$i < k \rightarrow$ SELECTION $(A, i, \ldots)$
$i > k \rightarrow$ SELECTION $(A, i-k, \ldots)$ $T\left(\frac{7n}{10}\right)$

Ex:-



All the groups are reorder such that median of a group is greater than medians of all the groups left of it

no. of elements in ~~left~~ right subarray

$\geq \left(\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil - 2\right) * 3$

$\geq \frac{3n}{10} - 6$

$\Rightarrow$ no. of elements in left subarray

$\leq n - \left(\frac{3n}{10} - 6\right)$

$= \frac{7n}{10} + 6$

time taken for step (5)

$\therefore$ select $(P, q-1, i \ldots\ldots)$ $T\left(\frac{7n}{10} + 6\right)$

$\therefore T(n) = O(n) + O(n) + T\left(\left\lceil\frac{n}{5}\right\rceil\right) + O(n) + T\left(\frac{7n}{10} + 6\right)$

$$T(n) \leq T\left(\left\lceil\frac{n}{5}\right\rceil\right) + T\left(\frac{7n}{10} + 6\right) + O(n), \quad n \geq 140$$

$$T(n) \leq O(n), \quad \text{if } n < 140$$

$$T(n) \leq T\left(\left\lceil\frac{n}{5}\right\rceil\right) + T\left(\frac{7n}{10} + 6\right) + an, \quad n \geq 140$$

Guess:- $T(n) = O(n)$

Substitution Method

i.e., $T(n) \leq c * n$, $\forall\ n \geq ?$

Inductive step : $T(n) \leq c\left\lceil\frac{n}{5}\right\rceil + c\left(\frac{7n}{10} + 6\right) + an$

Inductive Hypothesis.

$$\leq c\left(1 + \frac{n}{5}\right) + \frac{7cn}{10} + 6c + an$$

$$\leq \frac{9cn}{10} + 7c + an$$

$$\leq cn - \frac{cn}{10} + 7c + an$$

$T(n) \leq cn$

if $-\frac{cn}{10} + 7c + an \leq 0$

$\frac{cn}{10} - 7c - an \geq 0$

$c\left(\frac{n}{10} - 70\right) \geq an$

$c \geq \frac{10an}{(n-70)}$

$$c \geq 10a\left(\frac{n}{n-70}\right)$$

if $n \geq 140 \Rightarrow \frac{n}{n-70} \leq 2$

$$c \geq 20a$$

$\therefore T(n) \leq cn$ if $c \geq 20a$

$$\geq \left(\frac{1}{2}\left\lceil\frac{n}{7}\right\rceil - 2\right) * 4$$

$$\leq \frac{n-28}{7} 2$$

$$\frac{2n}{7} - 56$$

Large Integer Multiplication

$x = x_1 x_2 \cdots x_m$

$y = y_1 y_2 \cdots y_n$

No. of single digit multiplications $= m * n$

$x = x_L | x_R = x_L * 10^{n/2} + x_R$

$y = y_L | y_R = y_L * 10^{n/2} + y_R$

$$xy = (x_L * 10^{n/2} + x_R) * (y_L * 10^{n/2} + y_R)$$

$$xy = x_L y_L 10^n + (x_L y_R + x_R y_L) 10^{n/2} + x_R y_R \rightarrow (1)$$

Divide & conquer:

$$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n), \text{ if } n \geq 2$$

$T(1) = \text{constant}$

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

reduce number of subproblems from 4 to 3

$$T(n) = 3T\left(\frac{n}{2}\right) + \theta(n) \quad , \text{if } n \geq 2.$$

$$(x_L + x_R)(y_L + y_R) = x_L y_L + (x_L y_R + x_R y_L) + x_R y_R$$

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R \quad \rightarrow (2)$$

Substitute (2) in (1)

$$XY = x_L y_L 10^n + \left((x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R\right) 10^{n/2} + x_R y_R \quad \rightarrow (3)$$

$$T(n) = 3T\left(\frac{n}{2}\right) + \theta(n) \quad , \text{if } n \geq 2$$

$$T(n) = \theta\left(n^{\log_2 3}\right) = \theta\left(n^{1.585}\right)$$