Subject:_____

Date:_____

Algoritum complexity

worst case: the function defined by the maximum no. of steps taken on any instance of size n.

Best case: The fun' defined by the minimum no. of steps taken on any instance of size n.

Average case: average no. of steps.

upper bound $O(g(N))$

lower " $\Omega(g(N))$

Tight " $\Theta(g(N))$

Subject:.....

Date :.....

$f(n) = 3n+2$

$f(n) \leq c \cdot g(n)$

$3n+2 \leq 3n+2n$

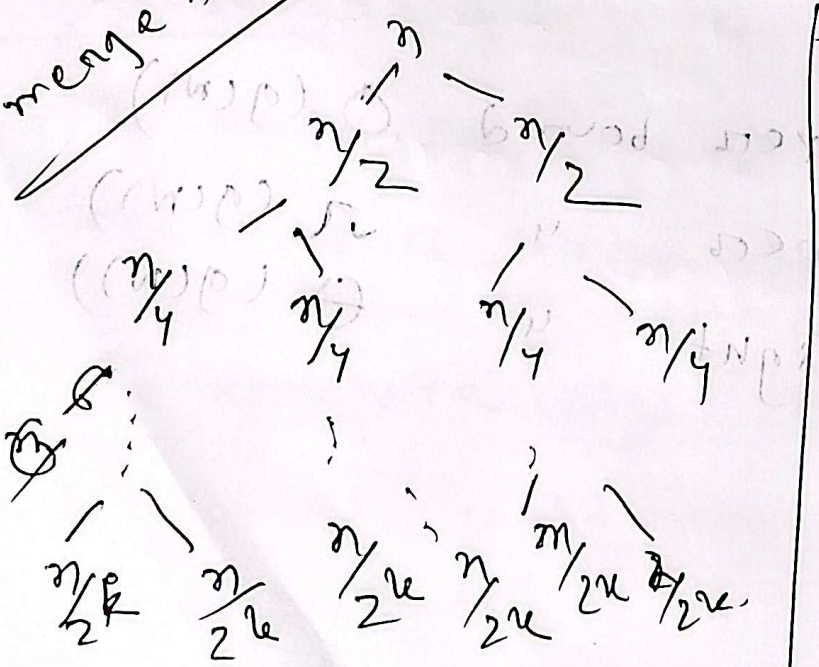
Ex: $f(n) = 5n^2+2n+1 = O(n^2)$

$g(n) = n^2$

$c = 8$, $f(n) \leq 8n^2$ $n \geq 1$

$n = 1$

merge sort



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

$2^k = n$

$\log 2^k = \log n$

$k \log 2 = \log n$

$k = \log n$

$nk = n \log n$

Subject:.....

Date:.....

Recurrence relation (1/2)

A recurrence relation is an eqn which is defined in terms of itself with smaller value.

why need?

many natural func are easily expressed as recurrences like:

recursive condition, base condition

$a_n = a_{n-1} + 1, \quad a_1 = 1 \longrightarrow a_n = n$ (polynomial)

$a_n = 2 a_{n-1}, \quad a_1 = 1 \longrightarrow a_n = 2^n$ (exponential)

$a_n = n a_{n-1}, \quad a_1 = 1 \longrightarrow a_n = n!$ (weird func)

more eqn:

$T(n) = 2 * T(n/2) + 1 \longrightarrow$ recursive condition

$T(1) = 1 \longrightarrow$ Base case

$T(n) = T(n-1) + n$
$T(1) = 1$
} selection sort.

Subject:.....

Date :.....

$T(n) = 2 * T(n/2) + n$
$T(1) = 1$
} merge, Quick sort

$T(n) = 2 * T(n/2) + \log n$
$T(1) = 1$
} Heap sort

$T(n) = T(n/2) + 1$
$T(1) = 0$
} binary search

Method for solving recursion:

1. Iteration method
2. Substitution method
3. Recursion Tree
4. master method

Subject:.....

Date:.....

(1) Iteration method:

* $T(n) = c + T(n/2)$, $T(1) = 1$

Now, $T(n) = c + c + T(n/4)$

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

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

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

Assume, $n = 2^k$

$T(n) = \underbrace{c + c + c + \dots + c}_{k \text{ times}} + T(1)$

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

$= \Theta(\log n)$

Subject:.....

Date :.....

② Substitution method:

→ Guess a soln

$$T(n) = O(g(n))$$

Induction goal: apply the defination ~~notation~~ of the asymptotic notation.

$T(n) \leq d\, g(n)$, for some $d > 0$ and $n \geq n_0$

Induction hypothesis : $T(k) \leq d\, g(k)$ for all $k < n$.

prove the induction goal case the induction hypothesis.

Subject:.....

Date:.....

Ex: Binary search

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

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

→ Induction goal: $T(n) \leq d \log n$, for some d and $n \geq n_0$

→ Induction hypothesis: $T(n/2) \leq d \log(n/2)$

proof of induction goal:

$T(n) = T(n/2) + c \leq d \log(n/2) + c$

$= d \log(n) - d + c \leq d \log(n)$

if $-d + c \leq 0$, $d \geq c$

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

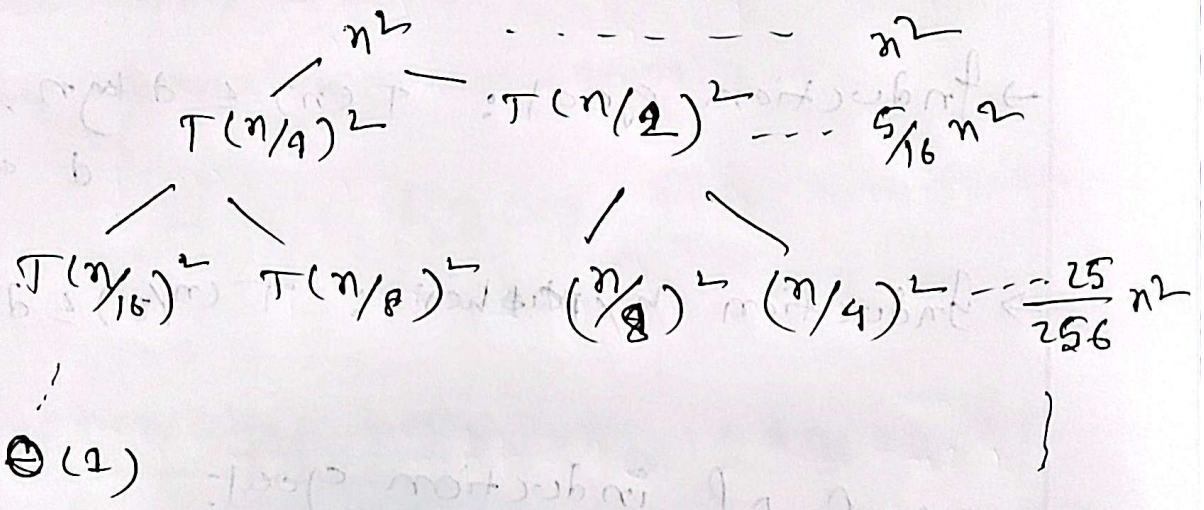
Subject:.....

Date :.....

(3) Recursion Tree :

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

Now,



$$\therefore \text{total} = n^2 \left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots \right)$$

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

→ geometric series

Subject:.....

Date:.....

(4) Master Method

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

→ $a \geq 1$, $b > 1$ are constant

→ $f(n)$ is asymptotically positive

→ n/b may not be an integer, but we ignore floors and ceilings.

Three cases:

① $f(n) < (n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a})$

~~② $f(n) = \Theta(n^{\log_b a})$~~

② $f(n) = n^{\log_b a}$ then $T(n) = \Theta(n^{\log_b a} \log n)$

③ $f(n) > n^{\log_b a}$ then $T(n) = \Theta(f(n))$

Subject:.....

Date :.....

Ex: $T(n) = 16T(n/4) + n$

let, $a = 16, b = 4,$

Now, $n^{\log_b^a} = n^{\log_4^{16}} = n^2$

$\therefore f(n) = n$

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

$\therefore$ Time complexity $= \Theta(n^2)$

Ex: $T(n) = T(3n/7) + 1$

here, $a = 1, b = \frac{7}{3}$

$\therefore n^{\log_{7/3}^1} = n^0 = 1$

$\therefore f(n) = 1$ which is equal then $n^{\log_b^a}$

$\therefore Te = \Theta(\log n)$

Subject:.....
Date:.....

Recursion Tree:

A recursion tree is a diagram used to visualize how a recursive algorithm breaks a problem into subproblems.

It shows:

→ How many subproblems each call generates
→ The size of each subproblem
→ The cost at each level of recursion
→ total cost by summing all levels.

It is mainly used to solve recurrence relations & analyze time complexity (especially Divide & conquer algorithm)

Why used:

→ To understand how recursion expands
→ To calculate total work done at all levels
→ To find Ø. Big-O complexity of recursive algorithms.

Subject:
Date:

Divide & conquer :

Divide & conquer is an algorithm design technique where a problem is solved by

1. Divide - breaking the main problem into smaller subproblems
2. conquer: solving each subproblem (recursively)
3. combine: merging the soln of subproblem to form the final ans

Solving a big problem by splitting it into smaller ones ~~bit~~ until they become simple enough

ex: merge, quick, Binary search, Strassen's Matrix multiplication

Subject:.....

Date:.....

Ex:01

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

Sol^n: Given, $T(n) = 2T\left(\frac{n}{2}\right) + n$

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

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

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

$$\vdots$$

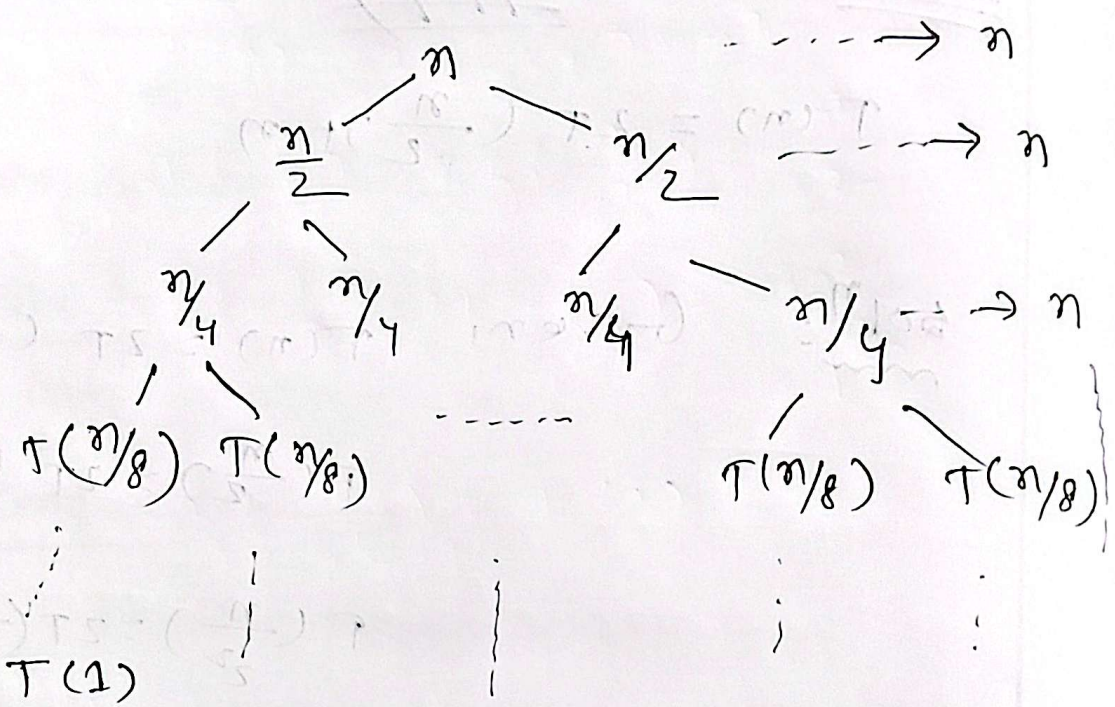
$$T\left(\frac{n}{2^{k-1}}\right) = 2T\left(\frac{n}{2^k}\right) + \frac{n}{2^{k-1}}$$

and, $T\left(\frac{n}{2^k}\right) = O(1)\ T(1)$

Subject:.....

Date:.....

make Tree:



continue expanding until the problem size reduces to 1

Total cost = cost of leaf nodes + cost of Internal nodes

∴ total cost = $L_c + I_c$

now, $T\left(\frac{n}{2^k}\right) = T(1)$

$\Rightarrow n = 2^k$

$\Rightarrow k = \log n$

Subject:.....

Date:.....

$\therefore L_e = 2^k = 2^{\log n} = n$

and, I_e (internal node cost) $= k \cdot n$

$\Rightarrow I_e = n \log n$

$\therefore$ total cost $= L_e + I_e$

$= n + n \log n$

$= O(n \log n)$

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

Subject:.....

Date :.....

Ex-02:

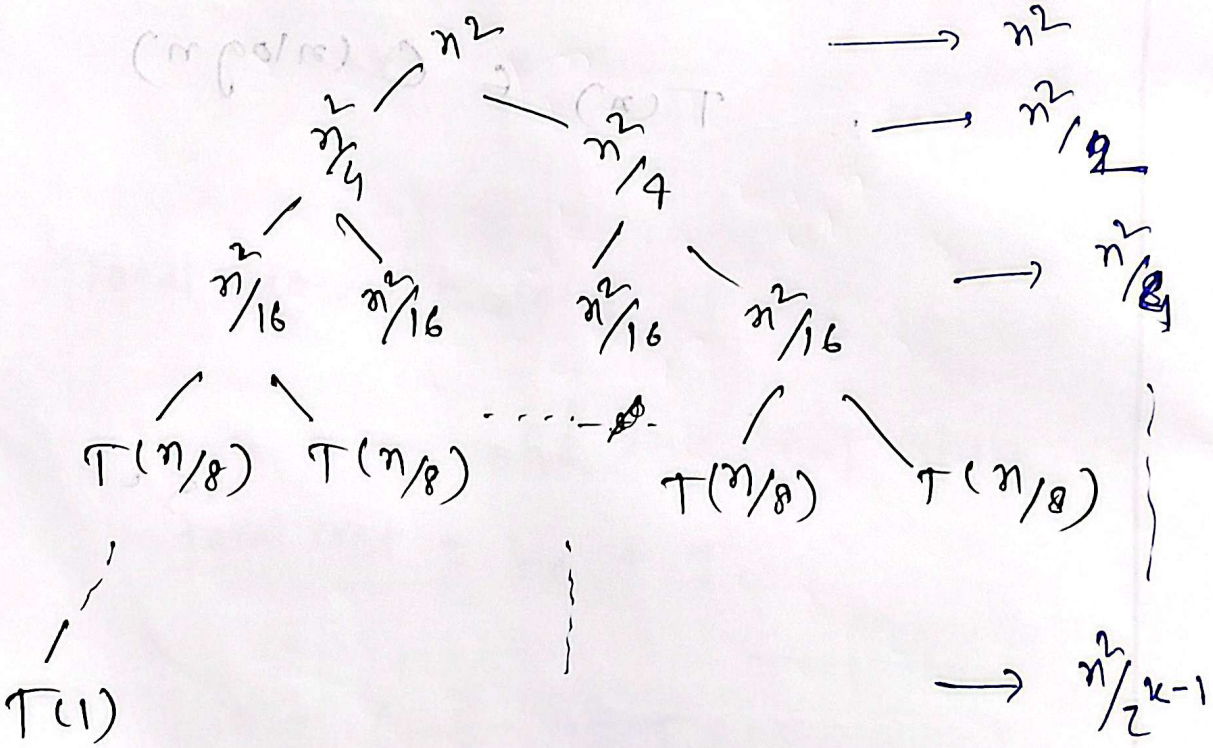
$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

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

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

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

make tree:



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

$\Rightarrow n = 2^k$

Subject:.....

Date:.....

$\therefore k = \log n$

here, $L_c = 2^k = 2^{\log n} = n$

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

$$= n^2 \left(\frac{1}{1 - \frac{1}{2}} \right) \rightarrow \left[S = \frac{a}{1-r} \right]$$

$$I_c = 2n^2$$

$\therefore$ total cost $T(n) = L_c + I_c$

$$= n + 2n^2$$

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

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

Subject:.....

Date :.....

Ex-03:

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

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

$$T\left(\frac{n}{4^2}\right) = 3T\left(\frac{n}{4^3}\right) + \frac{n^2}{16^2}$$

$$\vdots$$

$$T\left(\frac{n}{4^k}\right) = T(1).$$

Now, $n = 4^k$

$\Rightarrow k = \log_4 n$

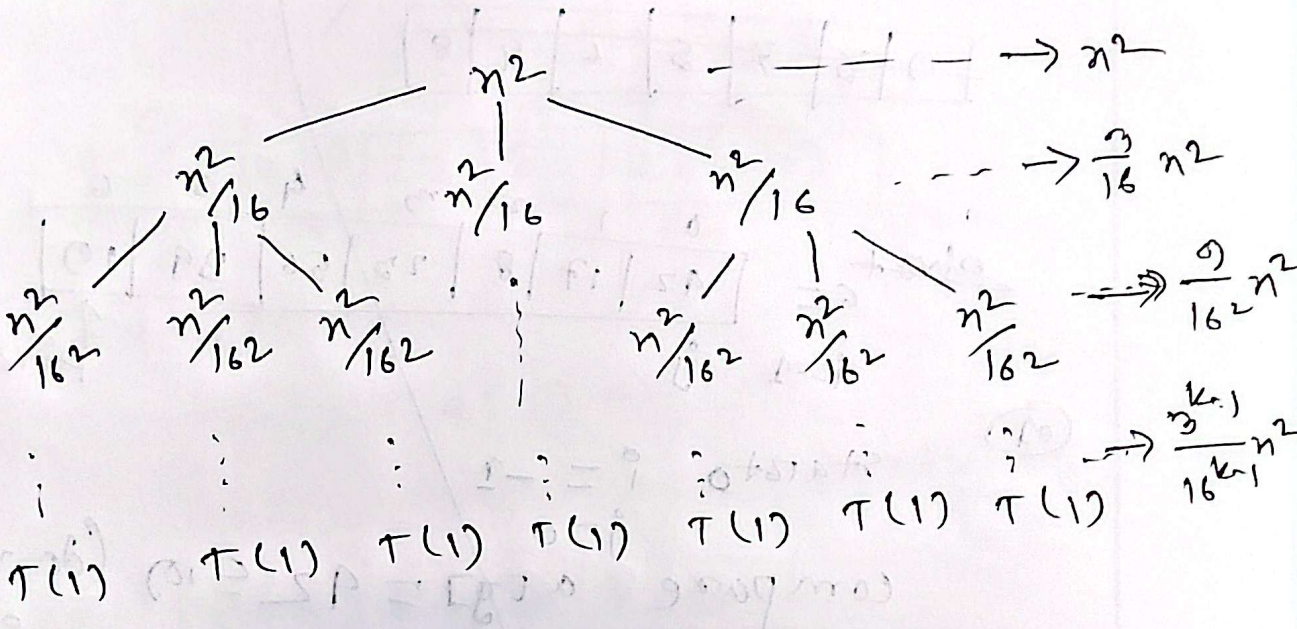
Here, $L_e = 3^k = 3^{\log_4 n}$

$\therefore L_e = n^{\log_4 3}$

Subject:.....

Date:.....

make tree:



$$T_c = n^2 \left[\left(\frac{3}{16}\right)^0 + \left(\frac{3}{16}\right)^1 + \left(\frac{3}{16}\right)^2 + \cdots + \left(\frac{3}{16}\right)^{k-1} \right]$$

$$= n^2 \left[\frac{1}{1 - \frac{3}{16}} \right]$$

$$= \frac{16}{13} n^2$$

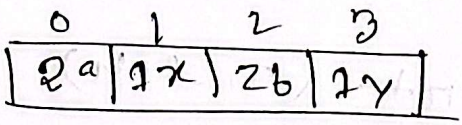
total cost $T(n) = n^{\log_4 3} + \frac{16}{13} n^2$

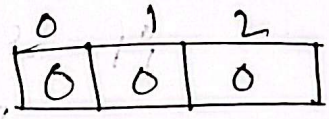
$\therefore T(n) \in O(n^2)$

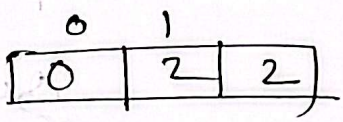
Subject:.....

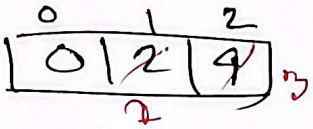
Date :.....

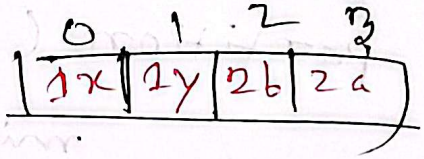
Why counting sort is more stable?











counting sort scans the input array from right to left when placing elements into the output array.

from this example we see that,

both 1's → x before y

both 2's → a before b

therefore right to left placement combined with prefix ensures that elements with equal values keep their original input order.

TC

Subject:.....

Date :.....

① for (i=1 ; i<=n ; i = i*5)

i = 1, 5, 5^2, 5^3 ... 5^k, $5^k = n$

$\log_5 5^k = \log_5 n$

$\therefore k = \log_5 n$

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

② for (i=n ; i>=1 ; i = i/5)

⟶ i = n, $\frac{n}{5}$, $\frac{n}{5^2}$, $\frac{n}{5^3}$ ... $\frac{n}{5^k}$

$\therefore \frac{n}{5^k} = 1$

$\Rightarrow 5^k = n$

$\Rightarrow \log_5 5^k = \log_5 (n)$

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

Subject:.....

Date :.....

(3) for ($i = 2$; $i <= n$; $i = i^2$)

$i = 2, 2^2, 2^{2^2}, 2^{2^3} \cdots 2^{2^k}$

Now, $2^{2^k} = n$

$\log_2 2^{2^k} = \log_2 n$

$2^k = \log_2 n$

$\log_2 2^k = \log_2 \log_2 n$

$\therefore k = \log_2 \log_2 n$

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

(4) for($i = 1$; $i <= 2^n$; $i++$) →

$i = 1, 2, 3, 4 \cdots k$

$k = 2^n$

$Tc = O(2^n)$

Subject:.....

Date :.....

(5) for (i = n²; i >= 1; i = i/2)

$i = n^2, \frac{n^2}{2}, \frac{n^2}{2^2}, \frac{n^2}{2^3} \cdots \frac{n^2}{2^k}$

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

$$\Rightarrow 2^k = n^2$$

$$\Rightarrow k = 2 \log n$$

$$\therefore k = \log n = \Theta(\log n)$$

(6) for (i = 5^n; i >= 5; i = $\sqrt[5]{i}$)

$i = 5^n, (5^n)^{\frac{1}{5}}, (5^n)^{(\frac{1}{5})^2}, (5^n)^{(\frac{1}{5})^3} \cdots (5^n)^{(\frac{1}{5})^k}$

$((5^n)^{\frac{1}{5^k}} = 5$

$(5^n)^{(\frac{1}{5})^k} = 5$

$5^{(\frac{n}{5})^k} = 5$

$(\frac{n}{5})^k = \log 5$

$\frac{n}{5^k} = 1$

$\Rightarrow 5^k = n$

$\therefore k = \log n$

$\therefore T.C = \Theta(\log n)$

Subject:.....

Date :.....

⑦

```
for (i=1; i<=n; i++)          ——— n
    for(j=1; j<=n; j++)       ——— n
        for(k=1; k<=n; k++)   ——— n
            x=x+1
```

Tc : $\Theta(n^3)$

both best & worst case performance

⑧

```
for(i=3; i<=n; i=i*3)          ——— log_3 log_3 n
    for(j=n; j>=1; j=j/2)      ——— log_2 n
        for(k=1; k<=n; k=k*4)  ——— log_4 n
```

Tc = $\Theta(\log^2 n \cdot \log\log n)$

Subject:.....

Date:.....

* For (i=1; i<=n; i++) ——— O(n)

  for (j=1; j<=n; j=j+i)

    x = x+1

For i = 1

second loop run n times

For i = 2

2nd loop run n/2 times

For i = n

2nd loop run $\frac{n}{n} = 1$ times

$\therefore$ total $= n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}$

$= n\left[1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right]$

$= n \log n$

$\therefore$ TC $= \Theta(n \log n)$

Subject:.....

Date:.....

* for (i=1; i<=n; i++) ——— O(n)

for (j=1; j<=i; j++)

for(k=1; k<=j; k++)

NOW,

i=1

j=1 to 1

k=1 to 1

1 times

i=2

j=1 to 2

k=1 to 1 k=1 to 2

(1+2) times

i=3

j=1 to 3

k=1,1,1,2,1,3

(1+2+3) times

.... i=n

j=1 to n

(1+2+3+... +n) times.

total = 1+(1+2)+(1+2+3)+(1+2+3+...+n)

$= x_1 + x_2 + x_3 + \dots + x_n$

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

Subject:.....
Date:.....

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

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

$$= \frac{1}{2} \left\{ \frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} \right\}$$

$$= \frac{n(n+1)(n+2)}{6}$$

$\therefore Tc = \Theta(n^3)$

~~*~~ for (i=n; j=0; i>0; i/=2; j+=i)

iteration → 1: $i = n$, $j = 0$

$i = n/2$, $j = n/2$

$i = n/2^2$; $j = \frac{n}{2^2} + \frac{n}{2^2}$

$i = \frac{n}{2^3}$, $j = \frac{n}{2^2} + \frac{n}{2^2} + \frac{n}{2^3}$

$$\therefore Val(j) = n + \frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots + \frac{n}{2^{k-1}}$$

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

$k = \log_2 n + 1$

Subject:.....

Date :.....

Now,

$$Val(j) = n + \frac{n}{2} + \frac{n}{2^2} + \cdots + \frac{n}{2^{\log_2 n}}$$

$$= n\left(1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^{\log_2 n}}\right)$$

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

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

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

$$= 2n - \frac{1}{2n} \times 2n$$

$$= 2n - 1$$

$$\therefore T(n) = \Theta(n)$$

Subject:.....
Date:.....

* for (i=1; i<2n; i++) { — (i) $\Theta(n)$

for (j=n; j>1; j=j/2) — (ii) $\log n$

++p;

for (k=1; k<p; k=k*2) — (iii)

$\log \log n$

(ii) $j = n, \frac{n}{2}, \frac{n}{2^2} \cdots \frac{n}{2^k}$

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

$\therefore k = \log n$

(iii) $k = 1, 2, 2^2, 2^3 \cdots 2^k$

$2^k = \log n$

$k = \log \log n$

$\therefore$ total $= \Theta(n \log \log n)$

Subject:.....

Date :.....

* for (i=1; i<=n ; i++) {

for (j=1 ; j<n; j+=i)

i=1, j = 1, 2, 3 ... n times

i=2, j = 1, 3, 5 ... n/2

i=3, j = 1, 4, 7 ... n/3

$$\text{total} = n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}$$

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

$$= n \log n$$

$$\therefore TC = \Theta(n \log n)$$

Subject:.....

Date :.....

$F(n) = 4n^2 + 50n + 200$

For $n = 5$, total operation,

$F(5) = 4(5)^2 + 50 \times 5 + 200$

$= 550$ operations.

For, $n = 10$, total operation

$F(10) = 4(100)^2 + 50 \times 100 + 200$

$= 1100$ operation

Subject:.....
Date:.....

TC:

complexity:

complexity is the measure of time & memory an algorithm requires as the input size increases.

It helps compare the efficiency of different algorithm.

TC: TC is the number of operation needed to run an algorithm on large amount of input data. And the number of operations can be considered as time because the computers uses some time for each operation.

→ TC is a mathematical measure that describes how the running time of an algorithm grows as the input size increases.

Subject:.....
Date:.....

SC: SC measures how much extra memory an algorithm needs during execution, depending on input size.

why need complexity:

TC: → Program A takes n^2 operation to sort n names

→ progam B takes n log n opration

if we have 10 names (n=10)
A ≈ 100 operations
B ≈ 33 "

if we have 10000 names (n = ~~10000~~ 10000)
A ≈ 100000000 operation
B ≈ 138277

this is a massive difference.
program B is about 750 times faster for this input size.

Subject:.....
Date:.....

SC: ① Algorithm A (bubble sort) —
use O(1) extra space → only a few variables

② Algorithm B (merge sort) —
uses O(n) extra space → needs an extra array to merge

A is better use :

Asymptotic Notations:

Asymptotic Notation is a mathematical tool used to describe the efficiency (growth rate) of an algorithm as the input size become very large.

It ignore: constant factors and lower-order terms, to focus on the dominant behaviors of the algorithm

Subject:
Date:

LPS: longest ~~prefix~~ prefix & suffix

LPS is an array used in the kmp algorithm where each position stores the length of the longest proper prefix of the substring (0 to i) that is also a suffix of that string.

proper prefix means it cannot be equal to the whole string.

Ex:

A	B	A	B
0	0	1	2

Prefix: A prefix is a sequence of a characters taken from the start of the string

Ex: ABCD → A, AB, ABC, Be are prefix.

Subject:.....

Date:.....

Suffix: A suffix is a sequence of characters taken from the end of the string.

Ex: ABCD ⟶ D, CD, BCD are suffix.

When do prefix and suffix match?

∴ ABAB ⟶ P = AB

S = AB

both length = 2

∴ LPS value = 2