Tentative Schedule (FJP-2) {DSA - level (1)}

Approximately 35 lectures remaining

1. Conceptual Building Blocks → Complete
2. Recursion & Backtracking → Complete
3. Searching & Sorting, Time & Space Complexity → 3 lectures
4. Dynamic Programming → 7 lectures
5. Linked list → 4 lectures
6. Stack & Queue, OOPS → 4 lectures
7. Generic Tree → 3 lectures
8. Binary Tree & BST → 4 lectures
9. Hashmap & Heap → 4 lectures
10. Graphs → 5 lectures

M, T, W ⇒ DSA → 8:30 to 12:00

Thursday, Sunday ⇒ Dev

Friday & Saturday → LLD + OOPS

Bi-weekly DSA contest + Hackathons

Tentative Schedule (FJP-2) {DSA - level (1)}

Approximately 35 lectures remaining

1. Conceptual Building Blocks → Complete
2. Recursion & Backtracking → Complete
3. Searching & Sorting, Time & Space Complexity → 3 lectures
4. Dynamic Programming → 7 lectures
5. Linked list → 4 lectures
6. Stack & Queue, OOPS → 4 lectures
7. Generic Tree → 3 lectures
8. Binary Tree & BST → 4 lectures
9. Hashmap & Heap → 4 lectures
10. Graphs → 5 lectures

M, T, W ⇒ DSA → 8:30 to 12:00

Thursday, Sunday ⇒ Dev

Friday & Saturday → LLD + OOPS

Bi-weekly DSA contest + Hackathons

Time & Space Complexity

TIME COMPLEXITY

→ Best case, Average case, Worst case
- Best case → omega (Ω or lower bound)
- Average case → theta (Θ or tight bound)
- Worst case → Big O (O or upper bound)

→ Asymptotic Analysis
- Ignoring constant/smaller terms.
  - eg { $O(n^3)$ vs $O(n^3)$ vs $O(2^n)$ }
- Ignoring small input trend; {Always look for large input}

→ Graph (Time vs Input)

→ Run-time vs Compile Time vs Asymptotic Time vs Amortized Time → Vector/Array list design

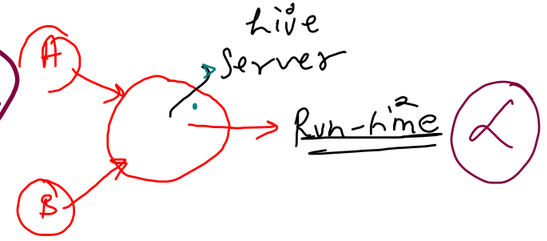
Architecture Neutrality

Time Complexity

{200 ms, 250 ms, 800 ms, 1 ms} → Run-time → Execution-time

Machine dependent → Architecture → 32bit, 64bit

Program & Process (RAM)



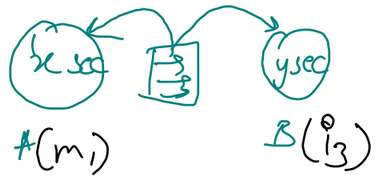
#Asymptotic Analysis → Algorithm & input size

ignore rest of the factors → machine independent

Runtime

$f(N)$ ⇒ function (input size)

$5n^2 + 3n + 6$



↳ growth of Runtime wrt input size

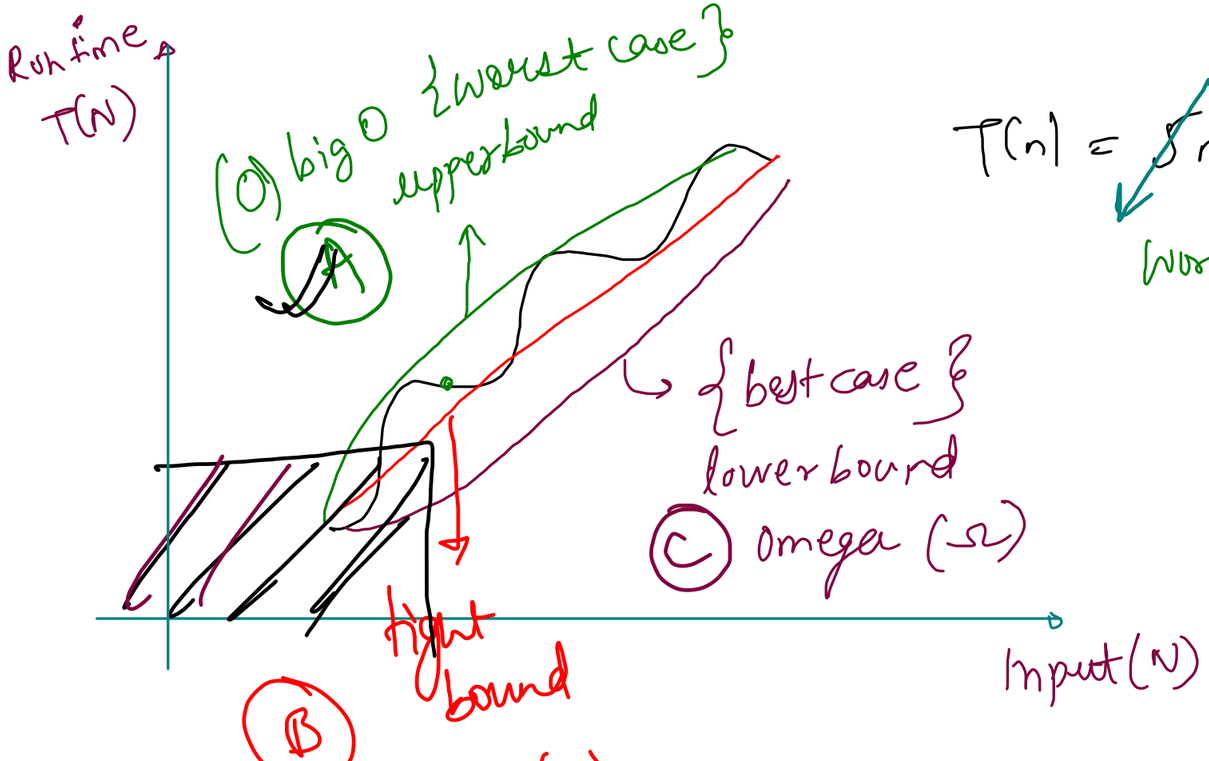
N=100 ele ⇒ Runtime
- ×10 → N = 1000 ele ⇒ Runtime ×10
- A → 100ms, B → 200ms
- A → 1000 ms, B → 2000ms

⇒ Algorithm is linearly dependent on input size

Compilation + Linking + loading

Package → Header files

Ignore



$T(n) = 5n^2 + 3n + 6$

Worstcase ⇒ $10n^2$

$T(n) = 15n^2$ $O(n^2)$

machine

$15n^2 \geq 5n^2 + 3n - 16$

$10n^2 \geq 3n + 6$

$n > 0$

A $2n^2 = 15n^2 = n^2$

B

Abdul Rani

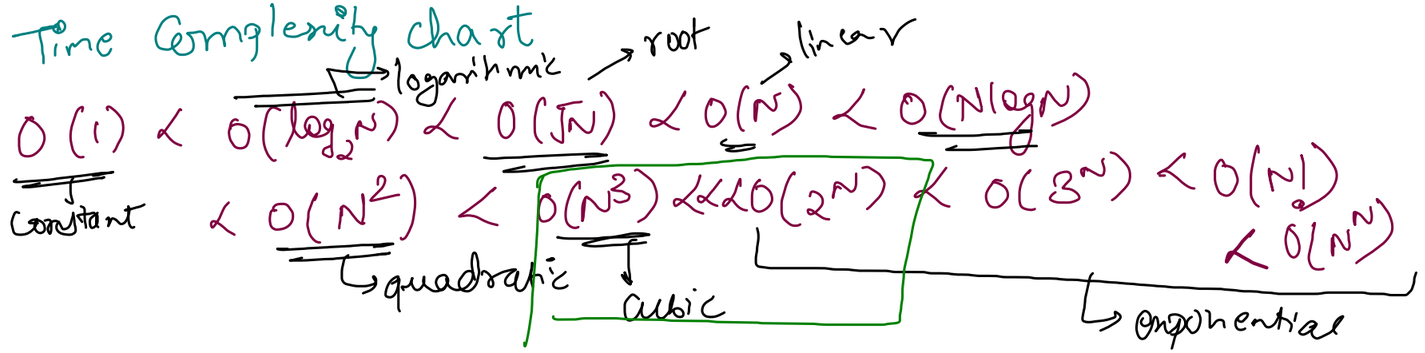
Bigo ≠ Worstcase

Omega ≠ BestCase

best case : $T(n) = 4n^2$

$\Omega(n^2)$

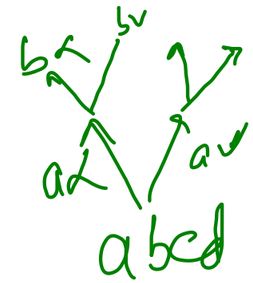
Time Complexity chart

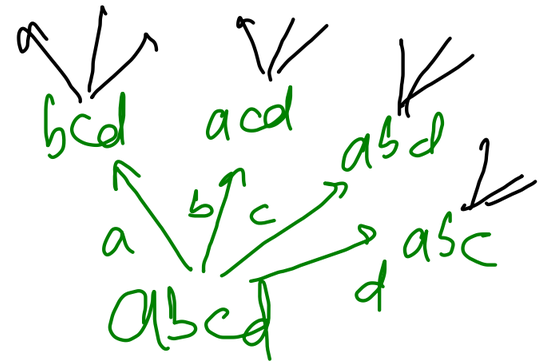


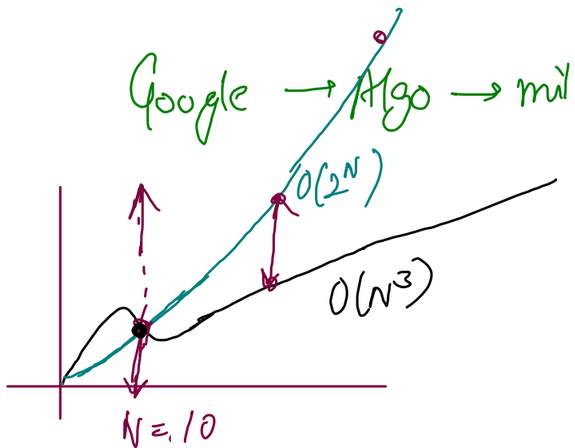
Constant < Logarithmic < Polynomial < Exponential

$O(N^3) < O(2^N)$

$n=2$	$O(2^3)=O(8)$	≮	$O(2^2)=O(4)$
$n=3$	$O(3^3)=O(27)$	≮	$O(2^3)=O(8)$
$n=4$	$O(4^3)=O(64)$	≮	$O(2^4)=O(16)$







N > Intersection pt

Large Input

Base case ignore

$T(n) = 4n^3 + 3n^2 \log n + 5n + 10$

$O(n^3)$

upperbound(n) $= C * n^3 \geq T(n)$

Runtime $T(n) = 8n^2 + 12n^2 + 20n^2 + 10n + 30n$

worst case $O(n^2)$

For loops time complexity { Code → Time complexity }

(1)

```
for (int i = 0; i < n; i++) {
        System.out.println(i);
}
```

n * k ms → k ms → O(1)

O(n)

(2)

```
for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
                System.out.println(i + " " + j);
        }
}
```

(3)

```
for (int i = 0; i < n; i++) {
        for (int j = n - 1; j >= 0; j--) {

        }
}
```

(4)

```
for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {

        }
}
```

(5)

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

}
```

(6)

```
for (int i = 0; i * i < n; i++) {

}
```

$n^3 * O(n \log n) = O(n^4 \log n)$

$f(n)\ O(g(n)) = O(f(n) * g(n))$

(2)

```
for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
                System.out.println(i + " " + j);
        }
}
```

→ k ms

$n = 5$

n times

i=0 → j = 0,1,2,3,4 → (k+k+k+k+k) = O(n)

i=1 → j = 0,1,2,3,4 → (k+k+k+k+k) = O(n)

i=2 → j = 0,1,2,3,4 → (5k) = O(n)

i=3 → j = 0,1,2,3,4 → (5k) = O(n)

i=4 → j = 0,1,2,3,4 → (5k) = O(n)

n times

O(n) + O(n) + O(n) + ⋮ n times

$= O(n * n)$

$= O(n^2)$

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

③

```
for (int i = 0; i < n; i++) {
        for (int j = n - 1; j >= 0; j--) {
                sysa(i+j)
        }
}
```

N = 5

i = 0 j = 4, 3, 2, 1, 0

i = 1 j = 4, 3, 2, 1, 0

i = 2 j = 4, 3, 2, 1, 0

i = 3 j = 4, 3, 2, 1, 0

i = 4 j = 4, 3, 2, 1, 0

k + k + k + k + k = ~~O(n)~~ → k * 25 ⇒ O(n²)

k + k + k + k + k + k = ~~O(n)~~

k * 5 = O(n)

k * 5 = O(n)

k * 5 = O(n)

O(n) + O(n) + ... n times = O(n²)

(4)

```
for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
                syso(i+j)
        }
}
```

N = 5

for ke for is not always $O(n^2)$

i=0 j=0

i=1 j=0, 1

i=2 j=0, 1, 2

i=3 j=0, 1, 2, 3

i=4 j=0, 1, 2, 3, 4

k = 1*k

k+k = 2*k

k+k+k = 3*k

k+k+k+k = 4*k

k+k+k+k+k = 5*k

1k + 2k + 3k + 4k + 5k

= (1+2+3+...n)*k

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

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

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

$= O(n^2)$

(5)

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

N → input size

$N = (Gy)$

$\log_2 N \neq \frac{N}{2}$ (29)

$N = 2^{30}$

i	k
1	1
2	2
4	3
8	4
16	5
32	6
64	k

no of terms = x

x * k

1, 2, 4, 8, ... n

a = 1; r = 2

$ar^{x-1} = n$

$1 * (2)^{x-1} = n$

$\log_2 (2)^{x-1} = \log_2 n$

$\Rightarrow x - 1 = \log_2 n$

$\Rightarrow x = \log_2 n + 1$

$x * k \Rightarrow O(\log_2 n)$

(6)

```
for (int i = 0; i * i < n; i++) {

}
```

← (64)

i=0 ✓ 0*0 < 64

i=1 ✓ 1*1 < 64

i=2 ✓ 2*2 < 64

i=3 ✓ 3*3 < 64

i=4 ✓ 4*4 < 64

i=5 ✓ 5*5 < 64

i=6 ✓ 6*6 < 64

i=7 ✓ 7*7 < 64

} no of terms = (8)

↳ $O(\sqrt{n})$

(7)

```
for (int i = 0; i < n; i++) {


}
for (int j = 0; j < m; j++) {


}
```

O(n)

O(m)

O(n+m)

(9)

```
for (int i = 0; i < n; i++) {
        for (int j = 0; j < 4; j++) {


        }
}
```

4+4+4+4 n times

= 4n

= O(n)

(8)

```
for (int i = 0; i < n; i += 2) {


}
```

N1 = 20

i = 0, 2, 4, 6, 8, 10, 12, 14, 16, 18

↳ N/2 times

O(N/2) = O(0.5n) = O(n)

(10)

```
for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j *= 2) {


        }
}
```

infinite

j = 0*2

= 0*2

= 0*2

= 0

N = 5

O(inf)

↳ non-terminating

Algorithms - Competitive Programming Chart!

(constraint) Input Size (N)	Time Complexity (Worst case)	(Idea) Hint of Algorithm
≤ 10	$O(n!)$ or $O(2^n)$	Backtracking {eg permutation, subsets}
≤ 18	$O(2^n \cdot n)$	Travelling Salesman
≤ 22	$O(2^n \cdot n^2)$	DP with Bitmasking
$\leq 10^2$	$O(n^4)$	Quadruplets / 4 Nested loops (4D DP)
$\leq 4*10^2$	$O(n^3)$	3 nested loops, Floyd Warshall, (3D DP)
$\leq 2*10^3$	$O(N^2 \log N)$	Nested loops & Binary Search
$\leq 10^4$	$O(N^2)$	2 Nested loops (Bubble, Insert, Select) (2D DP)
$\leq 10^5 - 10^6$	$O(N \log N)$	Sorting (QS, MS), BS on Answer, Greedy, etc.
$\leq 10^8$	$O(N)$	Array & Strings, (1D DP)
$\leq 10^{18}$	$O(\log_2 N)$ or $O(1)$	mathematical formula {Bit manipulation}

N

leetcode
codeforces
codechef

go!

Brute force / Naive Approach

1s ⟹ 10^8 operations

$N = 10^4$

$O(N) = 10^4$ ✓

$O(N^2) = 10^8$ ✓

TLE $O(N^3) = 10^{12}$ ✗

Recursion Generic Formula

99%

worstcase TC = (No of calls)^{height} + (preorder + postorder) * height

if else || &&

→ Recursion Basics (Factorial, Fibo, First & Last Index, max, Print I/D/ID)

→ Tower of Hanoi

→ Power - I, II, III

→ Get & Print → maze Paths
→ Stair Paths
→ Subsets
→ Keypad Encoding
→ Permutations

→ Backtracking → Flood fill
→ N Queen
→ Knight's Tour

++, --, *, /, +, - arithmetic operations

Output

Input

(logical) if/else op

Constant Time Complexity?

Return

Base Case

NP-Hard & NP-Complete Problems.

eg Subset Sum
Hamiltonian Path & Cycle
Travelling Salesman
Graph colouring

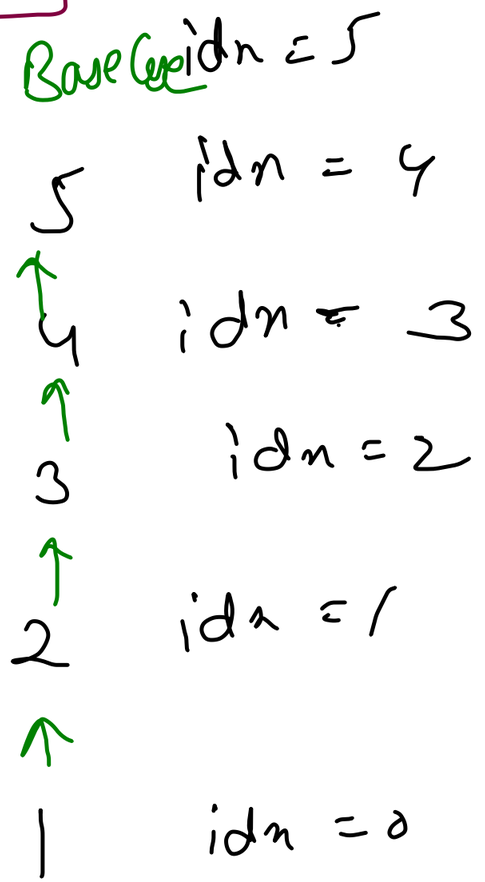
```java
public static void main(String[] args) throws Exception {
    // write your code here
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for(int i=0; i<n; i++)
        arr[i] = scn.nextInt();

    displayArr(arr, 0);
}

public static void displayArr(int[] arr, int idx){
    // Base Case
    if(idx == arr.length) return;

    // 1. Print arr of idx
    System.out.println(arr[idx]);

    // 2. Faith
    displayArr(arr, idx + 1);
}
```



TC

$(1) + \{k+c\} * N$

$= k + kN$

$O(N)$

Recurrence Relations → without code → logic, Dry run, Recursion tree

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

① linear search, Power - linear

② Binary search (Divide & Conquer), Power - logarithmic

(Tower of Hanoi, Print zig zag)

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

③ Power - III

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

```java
public static int power(int x, int n){
    // Base Case: x^0 = 1
    if(n == 0) return 1;

    // 1. Calculate x ^ n-1 using Faith
    int xpn1 = power(x, n-1);

    // 2. Calculate x ^ n using x ^ n-1
    int xpn = xpn1 * x;

    // 3. Calculate x ^ n
    return xpn;
}
```

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

Expectation (x^n) = Faith (x^{n-1}) + meeting Expectation

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

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

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

$T(n-3) = T(n-4) + k$

$T(1) = T(0) + k$

base case

$T(n) = T(0) + n * k_1$

$T(n) = k_2 + n * k_1$

$O(n)$

```java
public static int power(int x, int n){
    // Base Case x^0 = 1
    if(n == 0) return 1;

    // 1. Faith : x ^ n/2
    int xpnb2 = power(x, n/2);

    // 2. x^n = x^n/2 * x^n/2
    int xpn = xpnb2 * xpnb2;

    // 3. If n is odd
    if(n % 2 == 1) xpn = xpn * x;

    // 4. Return
    return xpn;
}
```

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

$x^n = x^{n/2} + \text{constant}$

Exp, Faith, Meeting

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

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

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

$T(n/8) = T(n/16) + k$

$T(1) = T(0) + k$

$T(n) = T(0) + x * k$

$= T(0) + \log_2 N * k$

$= O(\log_2 N)$

```java
// Number of calls in power2 : Homework
//     public static int power2(int x, int n)
//     {
//         if(n == 0) return 1;

//         int xpn = power2(x, n/2) * power2(x, n/2);
//         if(n % 2 == 1) xpn = xpn * 2;
//         return xpn;
//     }
```

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

x^n → Exp, $x^{n/2}$ → Faith, $O(1)$ → M.E

$T(n) = 2T(n/2) + k \quad \} * 2^0$

$\{T(n/2) = 2T(n/4) + k\} * 2 = 2^1$

$\{T(n/4) = 2T(n/8) + k\} * 4 = 2^2$

$\{T(n/8) = 2T(n/16) + k\} * 8 = 2^3$

$\{T(n/16) = 2T(n/32) + k\} * 32 = 2^4$

$\{T(1) = 2T(0) + k\} * 2^x$

$2^0 T(n) = 2^1 T(n/2) + k$

$2^1 T(n/2) = 2^2 T(n/4) + 2^1 k$

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

$2^3 T(n/8) = 2^4 T(n/16) + 2^3 k$

$2^x T(1) = 2^{x+1} T(0) + 2^x * k$

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

$\{x = \text{No of terms} = \log_2 N\}$

$= 2^{x+1} * T(0) + k\{2^0 + 2^1 + 2^2 + \cdots 2^x\}$

$= 2^{x+1} T(0) + k * (2^x)$ $\quad \{\sum GP = \frac{a(r^x - 1)}{r - 1}\}$

$= 2^{\log_2 N} k' + k * 2^{\log_2 N}$

$= N k' + k * N = O(N)$

4 → $O(n^2)$

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

5 → $O(n \log n)$

$T(n) = T(n-1) + O(\log_2 N)$

6 → $O(2^N)$

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

7 → $O(n \log n)$

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

8 → $O(\sqrt{N})$

$T(n) = T(\sqrt{N}) + O(1)$ {Root-function}

Recurrence Relation ?

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

$T(n^1) = T(n^{1/2}) + O(1)$

$T(n^{1/2}) = T(n^{1/4}) + O(1)$

$T(n^{1/4}) = T(n^{1/8}) + O(1)$

$T(n^{1/8}) = T(n^{1/16}) + O(1)$

$T(1) = T(0) + O(1)$

$T(n) = T(0) + k * x$ (no of cmps)

$n, n^{1/2}, n^{1/4}, n^{1/8}, \ldots 1$

256, 16, 4, 2, 1

$2^8, 2^4, 2^2, 2^1, 2^0$

No of bytes ∝ Input Size

Space Complexity

Input → LL, Tree → Input, Graph → Input

Recursion call stack Space

Auxiliary Space
- Extra
- Data structure

Input Space

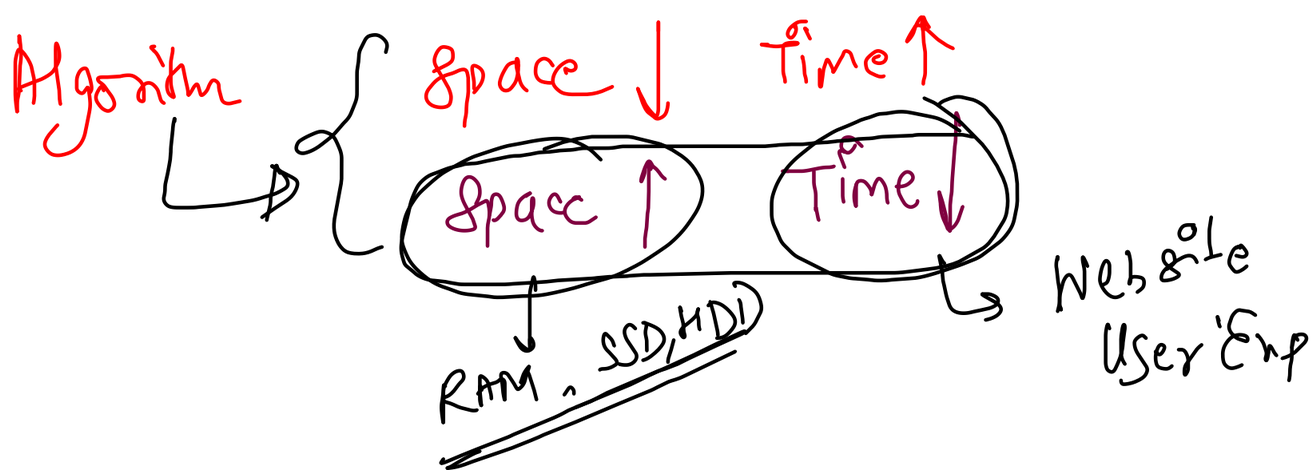
Output Space

Constant Extra space ?

int (4) int (4) 4*26 double

"Primitive datatypes"

Time vs Space Tradeoff !



Master Theorem

Decreasing Functions

Recurrence Relation

Master theorem for Decreasing Functions

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

$a > 0 \quad b > 0$ and $f(n) = O(n^k)$ where $k \geq 0$

Case

1. if $a < 1$ $O(n^k)$ / $O(f(n))$
2. if $a = 1$ $O(n^{k+1})$ / $O(n * f(n))$
3. if $a > 1$ $O(n^k a^{n/b})$

$T(n) = T(n-1) + 1 \rightarrow O(n)$

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

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

$T(n) = 2T(n-2) + 1 \rightarrow O(2^{n/2})$

$T(n) = 3T(n-1) + 1 \rightarrow O(3^n)$

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

Master Theorem

Dividing Function

Master Theorem for Dividing Functions

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

① $\log_b^a$

② k

$a \geq 1$, $b > 1$

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

Case 1: if $\log_b^a > k$ then $\Theta(n^{\log_b^a})$

Case 2: if $\log_b^a = k$

- if $p > -1$ $\Theta(n^k \log^{p+1} n)$
- if $p = -1$ $\Theta(n^k \log\log n)$
- if $p < -1$ $\Theta(n^k)$

Case 3: if $\log_b^a < k$

- if $p \geq 0$ $\Theta(n^k \log^p n)$
- if $p < 0$ $\Theta(n^k)$