DSA

BUBBLE SORT

(also Sinking Sort / Exchanging Sort / In place Sorting Algo)

eg:- 3 1 5 4 2

Pass1:→ 1 3 5 4 2

1 3 4 5 2

1 3 4 2 5 → Largest element at last position

Pass2:- 1 3 4 2 5

1 3 2 4 5 → 2nd largest element at 2nd last position

& So on....

Pass1:→ i=0

i	j			
0	1	2	3	4
3	1	5	4	2
1	3	5	4	2
1	3	4	5	2
1	3	4	2	5

{ check whether element at j < i-1

j → internal loop

{ i ⇒ Counter }

Pass2:→ i=1

i	j			
0	1	2	3	4
1	3	4	2	5
1	3	2	4	5

Pass3:→ i=2

i		j		
0	1	2	3	4
1	3	2	4	5

j → < length - i
or
<= length - i - 1

(5-1=4), (5-2=3)

→ as there is no need of comparing elements these are already sorted

Q:- Why Bubble Sort ?

→ with the 1st Pass through the array, the largest element came to the end.

with 2nd Pass through the array, 2nd largest element is at the 2nd last position

and, So on.....

Space Complexity = O(1) ⇒ Constant → means No Extra Space required

i.e; no copying of the array required

Time Complexity :

Best Case :→ O(N) → Sorted already

Worst Case :→ O(N²) → Sorted in Opposite { eg:- array given in Descending Order → Have to make sort in Ascending Order

(No. of Comparisons / checks)

As the size of array growing → No of Comparisons also growing

Best Case :→

i	j			
0	1	2	3	4
1	2	3	4	5

Best Case Comparisons = N-1

In time complexity → Constants are ignored

When j is never sorted, (coherent) j never gets swapped → it means Sorted array

Worst Case :→

i	j			
0	1	2	3	4
5	4	3	2	1
4	5	3	2	1
4	3	5	2	1
4	3	2	5	1
4	3	2	1	5

→ First Pass (i=0)

⇒ N-1 Swaps (5-1=4)

j				
0	1	2	3	4
4	3	2	1	5
3	4	2	1	5

No need to compare this

→ Second Pass (i=1)

Total Comparisons = $(N-1)+(N-2)+(N-3)+(N-4)$

$= 4N - (1+2+3+4)$

$= 4N - \left(\frac{N(N+1)}{2}\right)$ { $\frac{4\times5}{2} = 10$ } (N=4)

$= 4N - \frac{N^2+N}{2}$

$= \frac{8N - N^2 - N}{2} = \frac{7N-N^2}{2}$

$O\left(\frac{7N-N^2}{2}\right) \equiv O(N^2)$ { In Big O-Notation → Const. are ignored }

3 2 4 i 5

3 2 1 4 5 → (N-2 Comparisons made) (5-2 = 3)

i

3 2 1 4 5 → Third Pass (i = 2)

i

2 3 1 4 5

2 1 3 4 5 → (N-3 Comparisons made) (5-3 = 2)

i

2 1 3 4 5 → Fourth Pass (i = 3)

1 2 3 4 5 → N-4 Comparisons made (5-4 = 1)

↓ no need to compare for i = 4

Stable Sorting Algo:→ when the original order is maintained for values that are equal

Unstable Sorting Algo:→ when the original order is not maintained for values that are equal

eg:→ (10) (20) (20) (30) (10)

↓ Sort

(10) (10) (20) (20) (30) ⇒ Stable

In original array, black ball of 10 no. before red ball of 10 and in the sorted, this order is maintained

(10) (20) (20) (30) (10)

↓ Sort

(10) (10) (20) (20) (30) ⇒ Unstable

here black 10 no. ball is after red 10 no. ball, order not maintained

Selection Sort

(It performs well on small lists/arrays)

Select the largest element (or smallest) & put it at its correct index

eg:- N-1 Comp^{n} (4) — 0 1 2 3 4 — 4 5 1 2 3 — (i=0)

{swap with correct index}

N-2 Comp^{n} (3) — 4 3 1 2 5 — (i=1)

N-3 Comp^{n} (2) — 2 3 1 4 5 — (i=2)

N-4 Comp^{n} (1) — 2 1 3 4 5 — (i=3)

0 Comp^{n} — 1 2 3 4 5 — (i=4) ⟹ Sorted

Total Comparisons = $0+1+2+\dots+(n-1)$

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

$O\left(\frac{n^2-n}{2}\right) \equiv O(n^2)$ → Time Complexity

Worst Case : $O(n^2)$

Best Case : $O(n^2)$

Stable : No

loop runs for → (N-i-1) times

↳ < N-i-1

→ { already at correct positions; Ignore future steps }

Insertion Sort

eg:- i=0 | 5 | 3 | 4 | 1 | 2 | (1 Comp^n)
(Sort)

After 1st Pass :→ i=1 | 3 | 5 | 4 | 1 | 2 | (2 Comp^n)

After 2nd Pass :→ i=2 | 3 | 4 | 5 | 1 | 2 | (3 Comp^n)

After 3rd Pass :→ i=3 | 1 | 3 | 4 | 5 | 2 | (4 Comp^n)

After 4th Pass → | 1 | 2 | 3 | 4 | 5 |

For every index element
put it at the correct index of LHS

i ≤ N-1 or i ≤ N-2
j > 0

```
for(int i=0; i<=N-2; i++){
    for(int j=i+1; j>0; j--){
        if(arr[j] < arr[j-1]){
            int temp = arr[j];
            arr[j] = arr[j-1];
            arr[j-1] = temp;
        } else {
            break;
        }
    }
}
```

→ else 'j' will be out of bound (length of array)

Q. Why Insertion Sort?

→ It is adaptive : Steps get reduced if array is sorted
No. of swaps reduced as compared to Bubble Sort

→ Stable Sorting algo

→ Used for smaller value of N → works good for which array is partially sorted
↳ It takes part in Hybrid Sorting algo

* Time Complexity :-

Worst Case: O(N²) { Using order sorted eg:- 5,4,3,2,1 }
↳ no of elements

Best Case: O(N) { Already Sorted eg:- 1,2,3,4,5 }
N-1 Comparisons
O(N-1) ≈ O(N)

Total Comparisons = 1 + 2 + 3 + ... + N-1

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

$O(\frac{N^2-N}{2}) \equiv O(N^2)$ → Time Complexity

Cycle Sort

should be continuous in any order

NOTE:→ when given numbers from range [1, N] → Use Cycle Sort

index = element - 1

↓ as index starts from 0.

eg:- 0 1 2 3 4

3 5 2 1 4

N=5

After Sorting :→ 0 1 2 3 4

1 2 3 4 5

(eg) → Worst Case :→ 4 swaps made + 5 comparisons made (swaps)

= [(N-1) + N] Swaps

= (2N-1) Swaps

O(2N-1) ≡ O(N) Also for Best Case

→ 3 5 2 1 4

↓ swap

2 5 3 1 4

↓ swap

5 2 3 1 4

↓ swap

4 2 3 1 5

↓ swap

1 2 3 4 5

(i++) loop over → ans

Strings & String Builder in java

Q. What are Strings?

→ Strings are datatype → stores the character of values → Strings acts the same as an array of characters in Java in Java → every character is stored in 16-bits

It is a class of Java

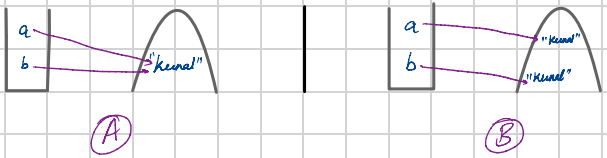
String name = "Hello World"

String → Datatype; name → Reference Variable; "Hello World" → Object

→ Internal working of String :-

String a = "Kunal"

String b = "Kunal"



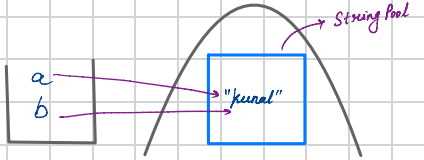
(A) (B)

Which is correct?
How is it stored?
A or B

→ Concepts :-

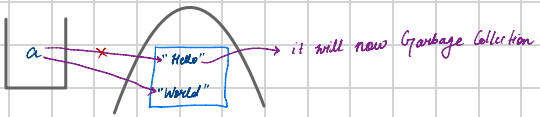
1) String Pool → separate memory structure inside the Heap memory

2) Immutability → Strings are immutable in Java (for security reasons)
   → cannot change / modify an object



String a = "Hello"

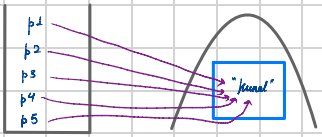
a = "World"



It will print "World" (didn't changed the object → created a new object)

Q7 Why we can't modify String objects?

Why Immutability?



→ if any person tries to change their name → it will get changed for everyone in database → if Mutable

↳ that's why Immutability

→ String Comparison :

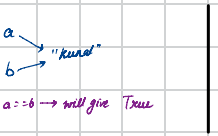
1) == Method (Comparator)

↳ check if ref variables are pointing to same object

a → "kunal"

b → "kunal"

a==b → will give False



a==b → will give True

String a = "kunal"

String b = "kunal"

a==b → will give True

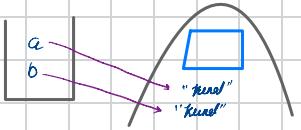
Q7 How to create different object of same value?

String a = new String("kunall")

String b = new String("kunall")

↳ creating these values outside the pool but it will be inside heap

a==b → will give False



→ a==b → will give False

To check only value → use .equals method

a.equals(b) → will give True

* Rounding off Float :

float a = 453.12f;

if 4 → then 453.12

if 5 → then 453.13

System.out.printf("formatted number is %.2f", a);

o/p → Formatted number is 453.13

* Placeholders : (There are many such placeholders ; you can search)

System.out.printf("Hello my name is %s and I am %s", "Shubham", "Cool");

o/p → Hello my name is Shubham and I am cool

RECURSION

→ a function that calls itself

(N) → while the function is not finished executing, it will remain in stack. When a function finishes executing, it is removed from the stack and the flow of program is restored to where the function was called

→ a simple if condition

Base Condition in Recursion → Condition where our recursion will stop making new calls

No Base Condition :- function calls will keep happening, stack will be filled again & again. Every call of function will take some memory. Memory of computer will exceed the limit → Stack Overflow Error

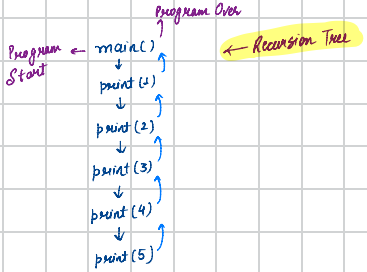
Q) Why Recursion ?

→ It helps in solving bigger/complex problems in a simple way. It helps in breaking down bigger problems into smaller problems.

You can convert recursion sol^n into iteration & vice-versa. (Loops)

Space Complexity is not constant because of Recursive Calls

→ Visualising Recursion :-



← Recursion Tree

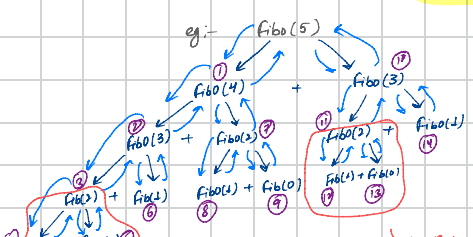
Q) Find n^{th} Fibonacci Number

0th 1st 2nd 3rd 4th 5th 6th 7th 8th

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

→ Fibo(N) = Fibo(N-1) + Fibo(N-2) → Recurrence Rel^n → Linear R.R.

eg :-



⇒ Recursive Tree

O → order they are called

→ Breaks it down into smaller problems

→ Base condition is represented by ans we already have

↳ Repeating function calls → to solve this problem ⇒ We use Dynamic Programming → it means if in the Recursion calls, if two or more Recursion calls are doing same work don't compute it again & again

{ → Not a Tail Recursion

↳ In Tail Recursion → just return with +, - etc

eg :- return print(n+1); → Direct return

Made with Goodnotes

(N) → How to understand & approach a problem?

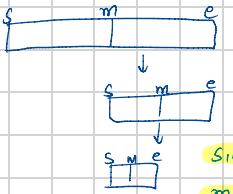
1. Identify if you can break down problem in to smaller problems
2. Write the Recurrence Rel^n if needed
3. Draw the recursive tree
4. About the tree :-
   - → See the flow of functions, how they are getting in stack
   - → Identify & focus on left tree calls & right tree calls
   - → Draw the tree & pointer again & again using pen & paper
   - → Use a debugger to see the flow
5. See how the values & what type of values (int, string, etc) are returned at each step
   See where the function call will come out
   In the end, you will come out of the main function

(N) → Variables :→ 1. Arguments
2. Return Type
3. Body of Function

(Tip) → Don't Overthink
→ if there are few variables that need to pass in the future function calls ⇒ Put it inside Arguments

Q: Binary Search with Recursion

1. Complexity → O(1)
2. Dividing into two halves
   → $F(N) = O(1) + F(\frac{N}{2})$ → Recurrence Rel^n → Divide & Conquer R.R.
   ↓ for comparison ↓ dividing in half



S.I.C → using in future calls → Put inside Arguments
m → not beneficial for future calls → Put in Body function

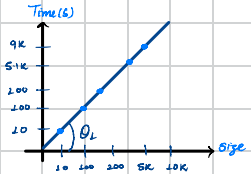
TIME & SPACE COMPLEXITY

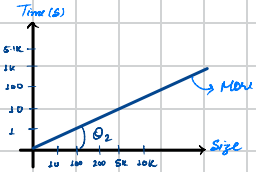
Q. What is Time Complexity?

→ Function that gives us the relationship about how the time will grow as the input grows.

Time Complexity ≠ Time Taken

eg:- For Linear Search a target that does not exists in the array: ∴ Graph for Linear Search →



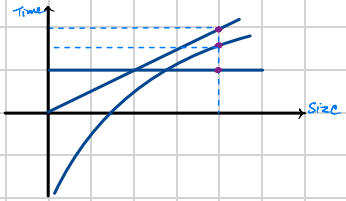


$\theta_1 > \theta_2$

Old Compⁿ → Takes 10s

New Compⁿ → Takes 1s

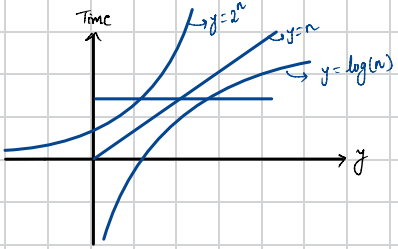
→ It does not mean → Time Complexity of New Compⁿ is better than Old Compⁿ



Better ←

$O(1) < O(\log N) < O(N)$ {Compare for bigger sizes not smalls}

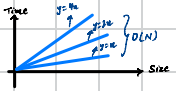
↓ Constt. ↓ For B.S ↓ For L.S.



$O(1) < O(\log(N)) < O(n) < O(2^n)$

↓ Constt. ↓ B.S ↓ L.S ↓ Recursive Fibonacci Series

Q. What do we consider when thinking about complexity?

1. Always look for worst case complexity
2. Always look at complexity for large / ∞ data
3. 

→ Even though the value of actual time different, they are all growing linearly.

→ We don't care about actual time ⇒ This is why, we ignore all constants

eg:- $y = 2n + 6$ → Still Linear

4. Always ignore less dominating terms.

eg:- $O(N^3 + \log N)$

From pt ② → let size of array = $1M = 10^6$

$\therefore (1M)^3 + \log(1M) = [(1M)^3 \text{ and } 6\text{sec}]$

(6 sec: very small, Hence, Ignore)

eg:- $O(3N^3 + 4N^2 + 5N + 6)$

→ $O(N^3 + N^2 + N) = O(N^3)$

* Big-Oh Notation :

↳ it says about "Upper Bound" → Word Definition

eg:- $O(N^3)$ → the graph of this cannot exceed N^3

↳ Upper Bound

Mathematically :

$f(N) = O(g(N))$

$\lim_{N \to \infty} \frac{f(N)}{g(N)} < \infty$

eg:→ $O(N^3) = O(6N^3 + 3N + 5)$

$g(N)$ $f(N)$

$\lim_{N \to \infty} \frac{f(N)}{g(N)} = \lim_{N \to \infty} \frac{6N^3 + 3N + 5}{N^3} = 6 < \infty$ (finite value)

* Big-Omega Notation :

↳ opp of Big-Oh Notation → it say about the "Lower Bound"

Word Definition :

eg:→ $\Omega(N^3)$ → the graph of this will be greater than N^3.

↳ Lower Bound

Mathematically :

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

$\lim_{N \to \infty} \frac{f(N)}{g(N)} > 0$

Q: What if an algo has LB & UB as N^2?

→ $O(N^2)$ & $\Omega(N^2)$

* Theta Notation :- (combining both O & Ω)

Word Def^n:- The graph will be b/w UB & LB

eg:- $\Theta(N^2)$:- UB & LB $= N^2$

Mathematically :-

$$0 < \lim_{N \to \infty} \frac{f(N)}{g(N)} < \infty$$

* Little o Notation :-

This also gives "UB".

Word Def^n:- Not a strict UB → Loose UB

Big - Oh	Little o
$f = O(g)$	$f = o(g)$
means → the growth of 'f' is not faster than 'g'	$f < g$ → Strictly slower
$f \leq g$	

Mathematically :-

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

eg:- $f = N^2$, $g = N^3$

$$\lim_{N \to \infty} \frac{f}{g} = \lim_{N \to \infty} \frac{N^2}{N^3} = \lim_{N \to \infty} \frac{1}{N} = 0$$

* Little omega Notation :- (ω)

Word Def^n:- Not a strict LB

Mathematically :-

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

eg:- $\lim_{N \to \infty} \frac{N^3}{N^2} = \lim_{N \to \infty} N = \infty$

Big Ω	Little ω
$f = \Omega(g)$	$f = \omega(g)$
$f \geq g$	$f > g$

Space Complexity :-

Auxiliary Space :→ Extra Space or temporary space used by an algo.

Space Complexity :→ of an algo. is total space taken by the algo. with respect to the input size.

↳ Auxiliary Space + Space used by input

Q.
```
for (i=1; i<=N; i++) {
    for (j=1; j<=K; j++) {
        // some op^n that
           take time t
    }
}
i = i+K
```
Find Time Complexity ?

→ Inner loop :- O(Kt) time

Ans → $O(Kt * \underbrace{\text{times outer loop is taking}}_{?})$ → So, $O\left(Kt * \left(\frac{N-1}{K}\right)\right) = O(t * N) = O(tN)$

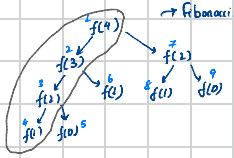
$i = 1, 1+K, 1+2K, 1+3K, \ldots, 1+xK$

$1+xK \leq N$

$\Rightarrow xK \leq N-1$

$\Rightarrow \boxed{x \leq \frac{N-1}{K}}$ → times the outer loop is running

* Recursive Algorithms :



→ Fibonacci Series

(N) → Only calls that are interlinked will be in the stack at same time

↕ → O(N)

Space Complexity → The longest chain or Height of Recursive Tree

↳ max^m recursive space at a time

* Types of Recursion: → 1) Linear eg:- F(N) = F(N-1) + F(N-2)

2) Divide & Conquer eg:- $F(N) = F(\frac{N}{2}) + O(1)$

* Divide & Conquer Recursion :

Form :

$$T(x) = a_1 T(b_1 x + \Sigma_1(x)) + a_2 T(b_2 x + \Sigma_2(x)) + \dots + a_k T(b_k x + \Sigma_k(x)) + g(x)$$

for $x \geq x_0$ ↳ some const. — $g(x)$ → when you get the ans from this + what you are doing takes how much time

eg:- $T(N) = T(\frac{N}{2}) + C$, $T(N) = 9T(\frac{N}{3}) + \frac{4}{5}T(\frac{5}{6}N) + 4N^3$

$a_1 = 1$, $b_1 = \frac{1}{2}$, $\Sigma_1(x) = 0$, $g(x) = C$

Q1. How to actually solve to get complexity?

1) Plug & Chug eg:- $F(N) = F(\frac{N}{2}) + C$

2) Master's Theorem

3) Akra Bazzi (1996)

* Akra Bazzi :

$$T(x) = \Theta\left(x^p + x^p \int_1^x \frac{g(u)}{u^{p+1}}\, du\right)$$

$p = ? \rightarrow \quad a_1 b_1^p + a_2 b_2^p + \ldots = 1$

$$\sum_{i=1}^{k} a_i b_i^p = 1$$

eg. $\rightarrow \quad T(N) = 2T\left(\frac{N}{2}\right) + (N-1)$

$a_1 = 2 \;,\; b_1 = \frac{1}{2} \;,\; g(x) = x-1 \equiv g(u)$

$\therefore \; 2 \times \left(\frac{1}{2}\right)^p = 1 \Rightarrow p = 1$

$$T(x) = \Theta\left(x^1 + x^1 \int_1^x \frac{u-1}{u^2}\, du\right)$$

$$= \Theta\left(x + x \int_1^x \left(\frac{1}{u} - \frac{1}{u^2}\right) du\right) = \Theta\left(x + x\left[\log(u) + \frac{1}{u}\right]_1^x\right)$$

$$= \Theta\left(x + x\left[\log x + \frac{1}{x} - \log 1 - \frac{1}{1}\right]\right)$$

$$= \Theta\left(x + x\left[\log x + \frac{1}{x} - 1\right]\right)$$

$$= \Theta\left(x + x\log x + 1 - x\right)$$

$$= \Theta\left(x \log x + 1\right) \rightarrow \text{ignore const.}$$

$$T(x) = \Theta(x \log x) \Longrightarrow \text{Time Complexity}$$

So, for coocay of size N : Merge Sort Complexity $= O(N \log N)$

Q: what if you can't find the value of p?

$$T(x) = 3T\left(\frac{x}{3}\right) + 4T\left(\frac{x}{4}\right) + \underbrace{x^2}_{g(x)}$$

$\rightarrow$ let $p = 1$:

$$3 \times \left(\frac{1}{3}\right)^1 + 4 \times \left(\frac{1}{4}\right)^1 = 1$$

$\Rightarrow \quad 1 + 1 = 2 \neq 1 \rightarrow 2 > 1 \rightarrow$ Increase the denominator $\rightarrow$ Increase $p \Longrightarrow p > 1$

Let $p = 2 : T$

$3 \times \frac{1}{9} + 4 \times \frac{1}{16} = 1$

$\Rightarrow \frac{1}{3} + \frac{1}{4} = \frac{7}{12} \neq 1 \rightarrow \frac{7}{12} < 1 \longrightarrow$ decrease the denominator $\rightarrow$ decrease $p \Longrightarrow p < 2$

So, p lies b/w $1 \& 2 \rightarrow p \in (1, 2)$

(N) → solve $p <$ power of $g(x)$

Ans $\Longrightarrow g(x)$

here, $g(x) = x^2 \rightarrow$ power of $g(x) = 2 \rightarrow p < 2$ by solving

Ans $= O(g(x)) = O(x^2)$

* Solving Linear Recurrences : T

Form : T

$f(x) = a_1 f(x-1) + a_2 f(x-2) + a_3 f(x-3) + \ldots\ldots + a_n f(x-n)$

$f(x) = \sum_{i=1}^{n} a_i f(x-i)$, for a_i, n is found $\quad n \rightarrow$ order of Recurrence

eg:- $F(N) = F(N-1) + F(N-2) \longrightarrow$ Solving for Fibonacci Series

Steps : T

1) Put $f(n) = \alpha^n$, for some const α

$\alpha^n = \alpha^{n-1} + \alpha^{n-2}$

$\Rightarrow \alpha^n - \alpha^{n-1} - \alpha^{n-2} = 0$

$\Rightarrow \boxed{\alpha^2 - \alpha - 1 = 0} \Longrightarrow$ Characteristic eqn of Recurrence

$\Rightarrow \alpha = \frac{+1 \pm \sqrt{1+4}}{2} = \frac{+1 \pm \sqrt{5}}{2}$

$\alpha_1 = \frac{1+\sqrt{5}}{2}, \alpha_2 = \frac{1-\sqrt{5}}{2}$

2nd. $f(n) = C_1\alpha_1^n + C_2\alpha_2^n$ → Sol^n for Fibonacci Series

$\equiv f(n-1) + f(n-2)$

$f(n) = C_1\left(\frac{1+\sqrt{5}}{2}\right)^n + C_2\left(\frac{1-\sqrt{5}}{2}\right)^n$

3rd. Fact:- No of roots = No of Cns you have already

here, we have two roots already → α_1 & α_2

Hence, we should have 2 Cns already

i.e., → $f(0)=0$ & $f(1)=1$

$\therefore f(0) = 0 \Rightarrow C_1 + C_2 = 1 \Rightarrow \boxed{C_1 = -C_2}$

$\therefore f(1) = 1 \Rightarrow C_1\left(\frac{1+\sqrt{5}}{2}\right) + C_2\left(\frac{1-\sqrt{5}}{2}\right) = 1$

$\Rightarrow C_1\left(\frac{1+\sqrt{5}}{2}\right) - C_1\left(\frac{-\sqrt{5}}{2}\right) = 1$

$\Rightarrow \boxed{C_1 = \frac{1}{\sqrt{5}}}$ & $\boxed{C_2 = \frac{-1}{\sqrt{5}}}$

So, $\left[f(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n\right]$ ⇒ Formula for n^{th} Fibonacci Number

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

↓ ignore const.

↳ as $n \to \infty$ ⇒ this will tends to 0
So, this is less dominating term
Hence, ignore

→ can also use $\left(\frac{1+\sqrt{5}}{2}\right)^n$

Time Complexity = $O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$ → for n^{th} Fibonacci Number

↓ ↳ Golden Ratio

$T(n) = O(1.6180^n)$

Q7. Equal Roots :-

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

1> $f(n) = \alpha^n$

$\alpha^n = 2\alpha^{n-1} + \alpha^{n-2}$

$\Rightarrow \alpha^n - 2\alpha^{n-1} - \alpha^{n-2} = 0$

$\Rightarrow \alpha^2 - 2\alpha - 1 = 0$

$\Rightarrow (\alpha-1)^2 = 0 \Rightarrow \alpha = 1, 1$ → Double roots

→ General Case :→ If α is repeated 'n' times, then

$\alpha^0, n\alpha^0, n^2\alpha^1, \ldots, n^{n-1}\alpha^n$ are all

sol^n of recurrence

Take two roots as :→ $1, n\alpha^n$

$f(n) = C_1\alpha^n + C_2 n\alpha^n$

$= C_1 + C_2 n$

$f(0) = 0$ & $f(1) = 1$

$f(0) = 0 \Rightarrow C_1 + 0 = 0 \Rightarrow C_1 = 0$

$f(1) = 1 \Rightarrow \overset{0}{C_1} + C_2 = 1 \Rightarrow C_2 = 1$

Ans $\Rightarrow f(N) = n$ → Time Complexity = $O(N)$

→ Homogeneous Eq^n :→ when $g(x) = 0$ {$g(x)$ is not present in eq^n ?}

eg :- $F(N) = F(N-1) + F(N-2)$

→ Non - Homogeneous Linear Recurrences :-

$f(n) = a_1 f(n-1) + a_2 f(n-2) + a_3 f(n-3) + \ldots + a_d f(n-d) + g(x)$

when this extra f^n is there, it is Non-Homogeneous Recurrence

→ How to Solve :-

1. Replace $g(n)$ by 0 & Solve usually

$f(n) = 4f(n-1) + 3^n$, $f(1) = 1$

$f(n) = 4f(n-1)$

$\Rightarrow \alpha^n = 4\alpha^{n-1}$

$\Rightarrow \alpha^n - 4\alpha^{n-1} = 0 \Rightarrow \alpha - 4 = 0 \Rightarrow \alpha = 4$

Homogn Sol^n → $f(n) = C_1\alpha^n = C_1 4^n$

2. Take $g(n)$ on one side and find Particular Sol^n

$\therefore f(n) - 4f(n-1) = 3^n$

Guess Something i.e, similar to $g(n)$

If $g(n) = x^2$, guess a polynomial of deg. 2

eg:- Guess :- $f(n) = C3^n$

So, $C3^n - 4C3^{n-1} = 3^n$

$\Rightarrow C3^n - 3^n - 4C3^{n-1} = 0$

$\Rightarrow 3C + 3 - 4C = 0$

$\Rightarrow \boxed{C = -3}$

Particular Sol^n :→ $f(n) = -3^{n+1}$

Now, General Sol^n → add both Homog^n Sol + Particular Sol^n

$f(n) = C_1 4^n - 3^{n+1}$

$f(1) = 1 \Rightarrow C_1 4 - 3^2 = 1$

$\Rightarrow C_1 = \frac{5}{2}$

Ans :→ $f(n) = \frac{5}{4} 4^n - 3^{n+1}$

→ How to guess Particular Sol^n?

→ If $g(n)$ is exponential, guess of same type

eg:- $g(n) = 2^n + 3^n$ → Guess $f(n) = a2^n + b3^n$

→ If $g(n)$ is Polynomial, guess of Same Degree

eg:→ $g(n) = n^2 + 1$ → Deg. of 2 → $f(n) = ax^2 + bx + c$ → Guess

eg:→ $g(n) = 2^n + n$ → $f(n) = a2^n + (bn + c)$ → Guess

→ What if Guess fails?

eg:- $f(n) = a2^n$ → Fails

Then → try $(an + b)2^n$ → If this also fails

Increase deg.

→ $(an^2 + bn + c)2^n$ → If this also → Increase Deg. ⇒ keep Trying

eg:→ $f(n) = 2f(n-1) + 2^n$, $f(0) = 1$

1. Put $2^n = 0$ → $f(n) = 2f(n-1)$

$f(n) = \alpha^n \Rightarrow 2\alpha^{n-1} = \alpha^n \Rightarrow \alpha^n - 2\alpha^{n-1} = 0 \Rightarrow \alpha = 2$

2. Guess Part^n Sol^n :-

$g(n) = 2^n$ → Guess → $f(n) = a2^n \Rightarrow a2^n = 2a2^{n-1} + 2^n$

$\Rightarrow a = a + 1$ → X Wrong

So, guess → $f(n) = (an + b)2^n \Rightarrow (an + b)2^n = 2[a(n-1) + b]2^{n-1} + 2^n$

$\Rightarrow an + b = an - a + b + 1$

$\Rightarrow \boxed{a = 1}$ → Discard b (2)

$f(n) = n \cdot 2^n$ → Particular Sol^n

General Sol^n :→ Homog^n Sol^n + Part^n Sol^n

$f(n) = C_1 2^n + n2^n$

$\because f(0) = 1 \Rightarrow \boxed{C_1 = 1}$

Ans :→ $f(n) = 2^n + n2^n$ → Time Complexity = $O(n2^n)$

Merge Sort

arr = [8 , 3 , 4 , 12 , 5 , 6]

[3 4 8] [5 6 12] → Comparisons are like → which is smaller

After Sorted → Merge them

⇒ [3 4 5 6 8 12]

3 or 5 → 3

4 or 5 → 4

8 or 5 → 5

8 or 6 → 6

8 or 12 → 8

remaining will be added after this (12)

→ Steps:-

1. Divide array into 2 parts
2. Get both parts sorted via Recursion
3. Merge the Sorted parts

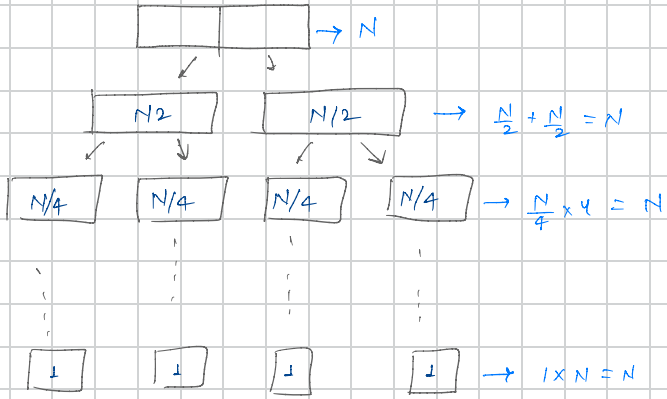
arr1 = [3 5 9 19] → remaining → put at end of array

arr2 = [4 6 8]

New array = [3 4 5 6 8 9 19]

{size → arr1 + arr2}

→ Time Complexity :-



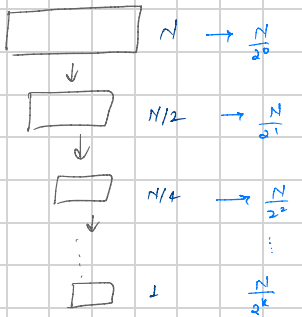
→ N

$\frac{N}{2} + \frac{N}{2} = N$

$\frac{N}{4} \times 4 = N$

$1 \times N = N$

At every level, N-elements are merged



$\frac{N}{2^0}$

$\frac{N}{2^1}$

$\frac{N}{2^2}$

$\frac{N}{2^k}$ → at k^{th} level

$\therefore 1 = \frac{N}{2^k} \Rightarrow 2^k = N$

$\Rightarrow k \log 2 = \log N$

$\Rightarrow k = \log_2 N$

Time Complexity :- $O(N * \log(N))$

- N → total comparisons
- log N → total levels
  - it gives comparisons at every levels

→ Space Complexity :-

→ Auxiliary Space Complexity :→ O(N)

→ $T(N) = T(\frac{N}{2}) + T(\frac{N}{2}) + (N-1)$ ← $(N-1)$ comparisons made — { Recurrence Rel^n

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

$\therefore \quad 2 \times \frac{1}{2^p} = 1 \Rightarrow p = 1$

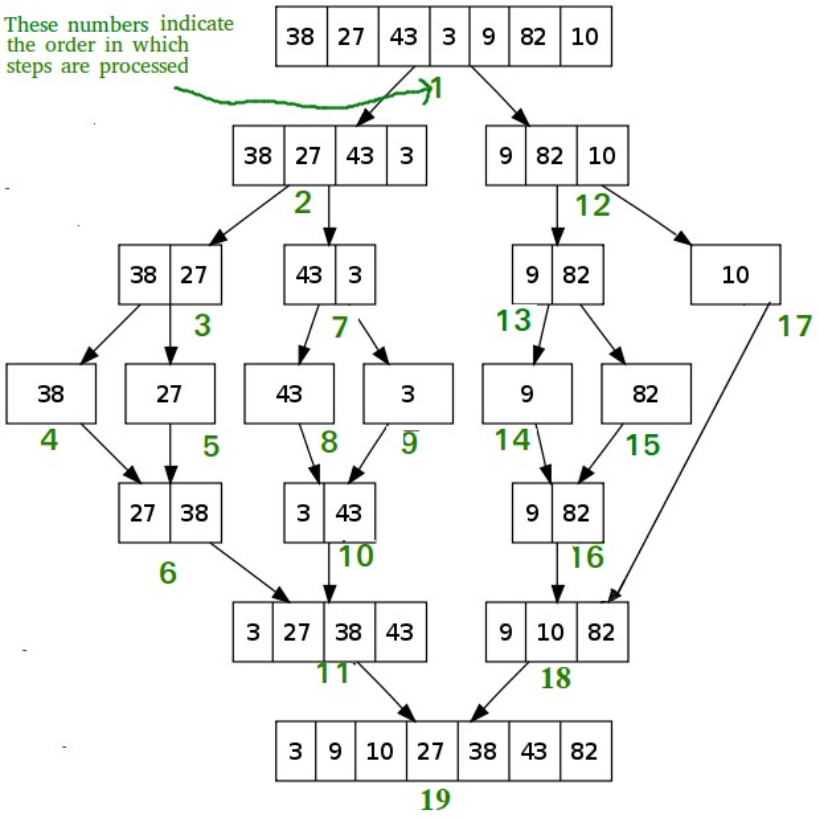
$T(N) = x + x\int_1^x \frac{u-1}{u^2}\,du$

$= x + x\left[\log u + \frac{1}{u}\right]_1^x = x + x\left[\log x + \frac{1}{x} - \log(1) - \frac{1}{1}\right]$

$= x + x\left[\log x + \frac{1}{x} - 1\right] = x + x\log x + 1 - x$

$= x\log x + 1$

→ $O(x\log x) \Rightarrow O(N\log N)$ → Space Complexity



Quick Sort

→ Pivot :→ choose any element → after first pass

all the elements < p → will be on LHS of p

all the elements > p → will be on RHS of p

arr = [5 (4) 3 2 1] ← let this be pivot

↓

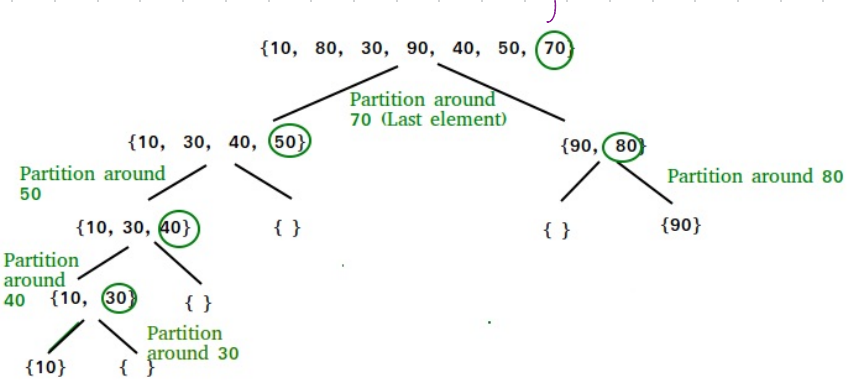
[1 (3) 2 (4) 5] (pivot) this gets to its correct index after every pass

[1 2 3] [5]

⇓

[1 2 3 4 5] → Sorted

here taking pivot as last element



Q. How to put pivot at correct position ?

arr = [5 (4) 3 2 1] p=4 (let) s=5, e=1

Swap

[1 4 3 2 5] p=4 s=4, e=2

(s<e)

[1 2 3 (4) 5] p=4 low=1, s=3, e=2, high=5

pivot is at correct position

(low < e)

(s > high)

Condition :→ while (a[s] < p): s++

while (a[e] > p): e--

low & end → tells us which part of array you are working on

s & e → just used for swapping

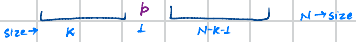
Q. How to pick pivot ?

→ Can pick → Random Element

↳ Corner Elements

↳ Middle Element

→ Complexity Comparison of Pivot Positions



→ $T(N) = T(K) + T(N-K-1) + O(N)$ → { Recurrence Rel^n for Quick Sort }

- $T(K)$ ↓ time to sort K elements
- $T(N-K-1)$ ↓ time to sort N-K-1 elements
- $O(N)$ ↓ time to put pivot at correct position

→ Worst Case → when pivot is Smallest/Largest element

↳ when $K = 0$ → when one part of array is Empty

arr = [3 5 8 20 34 18 (66)]

(n-1) elements ↓

Recursion Step that reduced is very little

So, $T(N) = T(0) + T(N-1) + O(N)$ → Linear R.R. (Solve it)

$\Rightarrow T(N) = T(N-1) + O(N)$

→ $O(N^2)$ → Time Complexity

→ Best Case → when gets divided into two halves

↳ $K = \frac{N}{2}$

→ $T(N) = T(\frac{N}{2}) + T(\frac{N}{2}) + O(N)$

$\Rightarrow T(N) = 2T(\frac{N}{2}) + O(N)$ { it same RR for Merge Sort }

→ $O(N \log N)$ → Time Complexity

(N) → Not Stable

- In-place i.e why preferred for arrays instead of Merge Sort

  Merge Sort takes O(N) extra space (Auxiliary)

- Merge Sort is better in linked list due to memory allocation not continuous

* Hybrid Sorting Algorithms (Tim Sort)

→ Merge Sort + Insertion Sort

works well with partially sorted data

(Collection Framework) = STL

Subsets :-

(Permutations & Combinations) (or Subsequences)

↳ non-adjacent collections

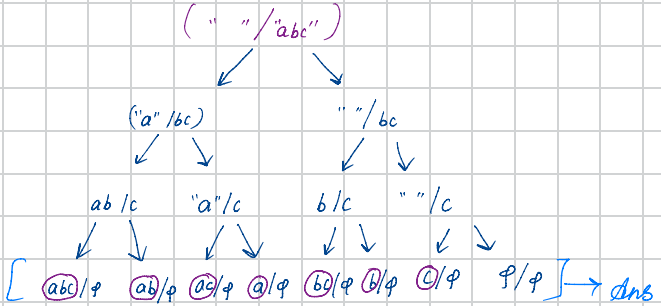
eg:- [3,5,9] → [3], [3,5], [3,9], [3,5,9], [5,9], [5], [9], [ ] (2^n)

↳ Recursion & Iteration :-

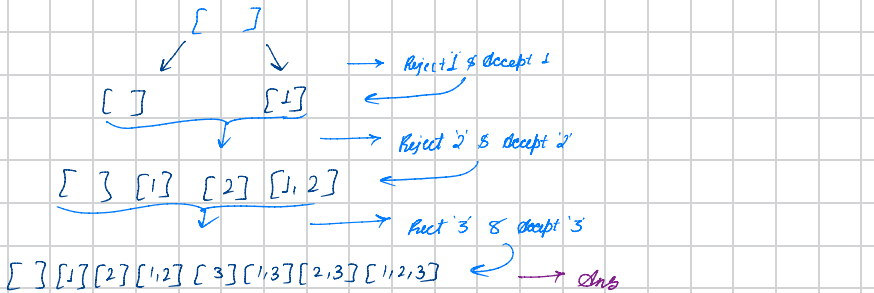
Str = "abc"

ans = ["a", "b", "c", "ab", "ac", "bc", "abc"]

↳ This pattern of taking some elements and removing some elements is known as → Subset Pattern



ans = [1 2 3]



Time Complexity :→ $O(N * 2^N)$

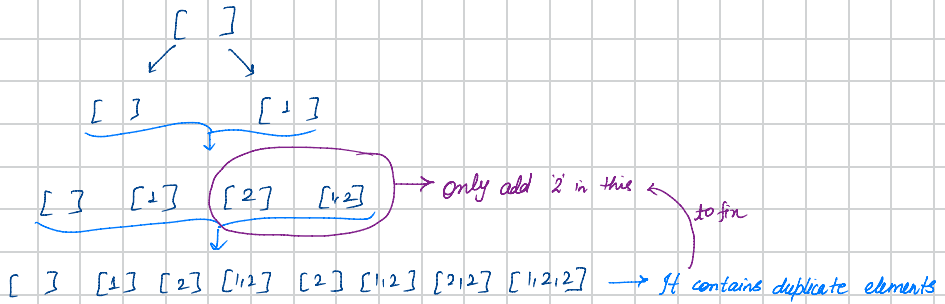
- time taken at each level (for copying)
- No. of subsets at each level

Space Complexity :→ $O(2^N * N)$

- Total no. of subsets
- Space taken by each subset ↳ O(N)

- Subsets with duplicate elements :-

arr = [1 2 2]



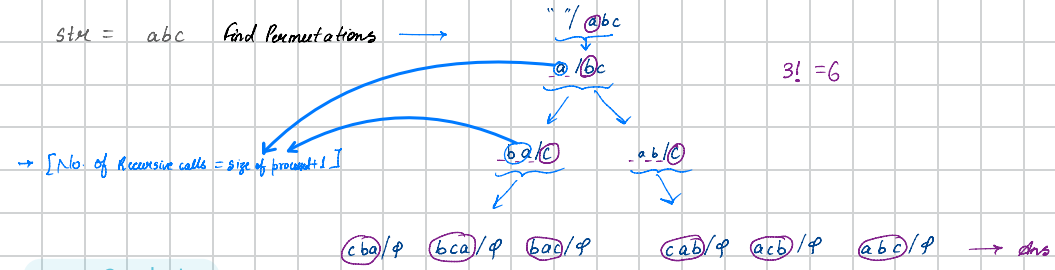
→ when you find a duplicate element → only add it in the newly created subset of previous step

→ if arr = [2,1,2] → so because of above point → duplicate have to adjacent

↳ to fix this → Sort the Array

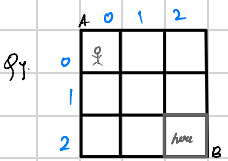
* Permutations :-

str = abc Find Permutations →

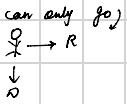


Backtracking

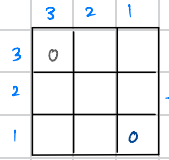
* Maze Problems :-

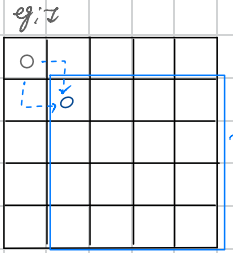


Ans: → RRDD
DDRR
RDDR
RDRD
DRDR



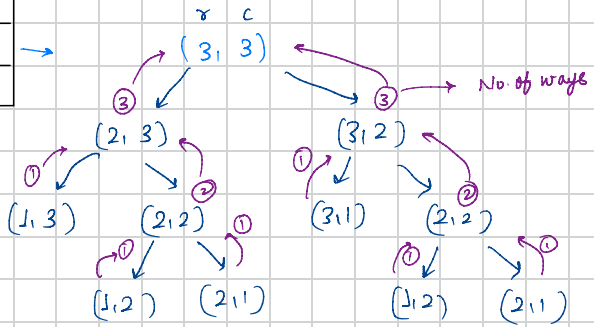
(0,0) → (2,2) → Ways=?





→ search space reduced to new matrix

→ RD + ( )
DR + ( ) → recursion calls

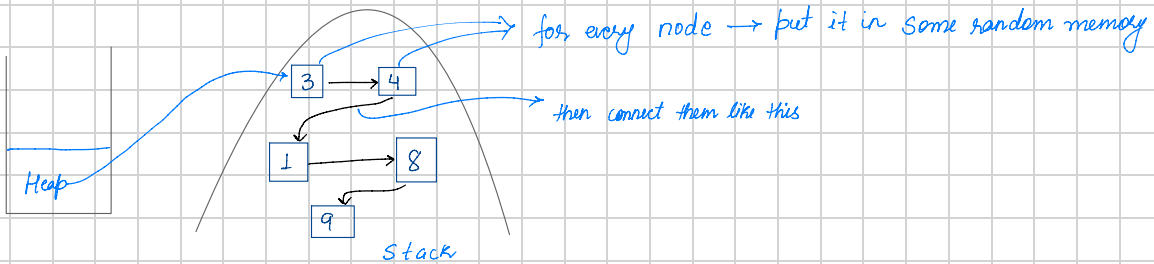


Total no. of ways = 3+3 = 6

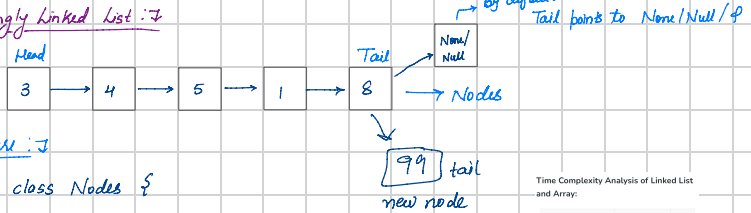
Linked List

Adavantages & Disadvantages

→ non continuous memory allocation



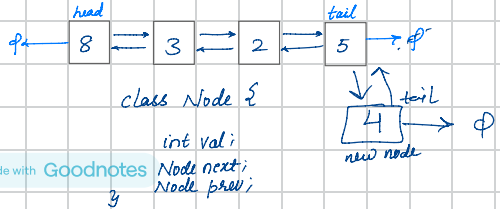
→ Singly Linked List :-



Structure :-

```
class Nodes {
    int val;
    Node next;
}
```

→ Doubly Linked List :-

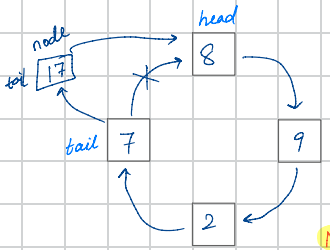


Time Complexity Analysis of Linked List and Array:

Operation	Linked list	Array
Random Access	O(N)	O(1)
Insertion and deletion at beginning	O(1)	O(N)
Insertion and deletion at end	O(N) (If we maintain only head)	O(1)
Insertion and deletion at a random position	O(N)	O(N)

Operation	Time Complexity	Auxiliary Space	Explanation
Insertion at Beginning	O(1)	O(1)	Constant-time pointer updates.
Insertion at End	O(n)	O(1)	Traversal required to find the last node.
Insertion at Position	O(n)	O(1)	Traversal to the desired position, then constant-time pointer updates.
Deletion at Beginning	O(1)	O(1)	Constant-time pointer update.
Deletion at End	O(n)	O(1)	Traversal required to find the second last node.
Deletion at Position	O(n)	O(1)	Traversal to the desired position, then constant-time pointer updates.
Searching in Linked list	O(n)	O(1)	Traversal through the list to find the desired value.

→ Circular Linked List :-




```
tail.next = node
node.next = head
tail = node
```

```
class Node {
    int val;
    Node next;
}
```

No null

```
if (head == null)
    head = node
    tail = node
```

→ hint

Linked List Data Structure

Last Updated : 04 Jan, 2025

A **linked list** is a fundamental data structure in computer science. It mainly allows efficient **insertion** and **deletion** operations compared to arrays. Like arrays, it is also used to implement other data structures like stack, queue and deque. Here's the comparison of Linked List vs Arrays

Linked List:

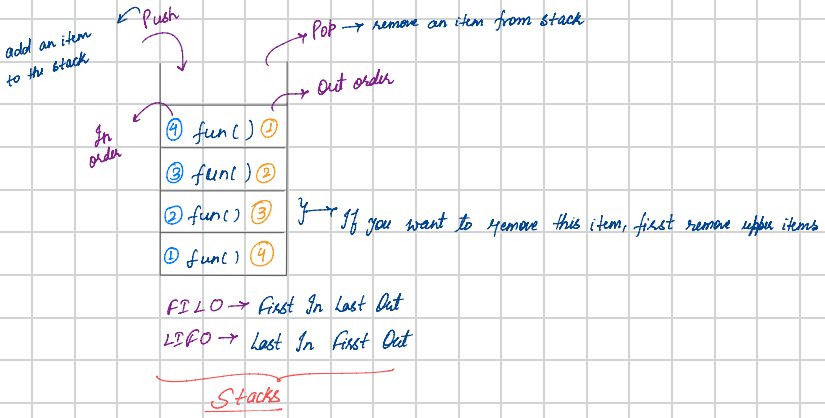
- **Data Structure:** Non-contiguous
- **Memory Allocation:** Typically allocated one by one to individual elements
- **Insertion/Deletion:** Efficient
- **Access:** Sequential

Array:

- **Data Structure:** Contiguous
- **Memory Allocation:** Typically allocated to the whole array
- **Insertion/Deletion:** Inefficient
- **Access:** Random

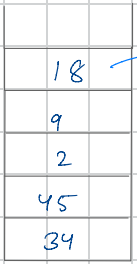
Stacks & Queues

→ Uses :→ when you want to store answers so far & put some results behind
↳ BT Traversals / DFS & BFS (Trees) / converting Recursion to Iteration



FILO → First In Last Out
LIFO → Last In First Out

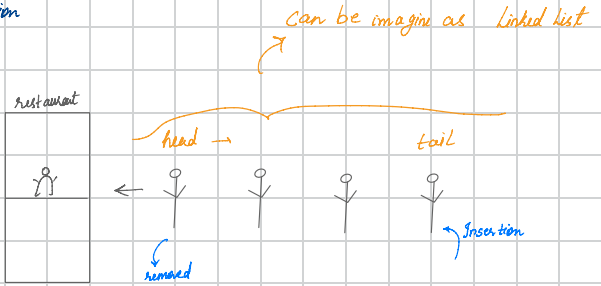
Stacks



Time Complexity :→ O(1)

0	1	2	3	4
34	45	2	9	18

uses Array internally



FIFO → First In First Out LIFO → Last In Last Out

Queues

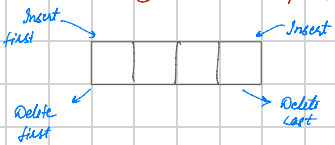
Queue<Integer> queue = new LinkedList<>();
using LinkedList class to create Queue object

→ (Deck) Dequeue / Doubly Ended Queue :— can remove/insert from both sides

Dequeue deck = new ArrayDequeue();
use this to create Dequeue

ArrayDequeue class → provides us to create an
basically it have resizable array object of the various methods

(N) :→ Faster than Queue & Linked List



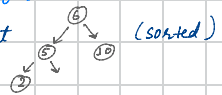
Trees

→ a type of Graph → Directed Acyclic Graph

→ Why Trees? (BT)

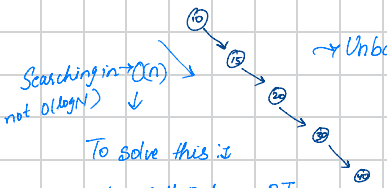
→ efficiently Add or Delete in → O(logN)

→ in BST → searching is efficient (sorted)



→ cost efficient

→ What are limitations?



→ Unbalanced BT

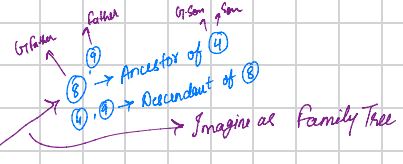
Searching in → O(n) not O(logN)

To solve this :→ make Self Balancing BT

→ not efficient for Sorted data

→ Where is it used?

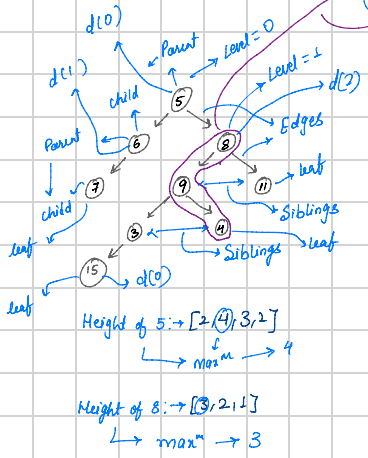
→ File System, Databases, Algo, Networking, Maths, DTs (ML) (Decision Trees), Compression of files (Huffman Coding), Future DS's (Data Struct.)



→ Structure :→

BT :→ Node :

```
int value;
int left;
int right;
```

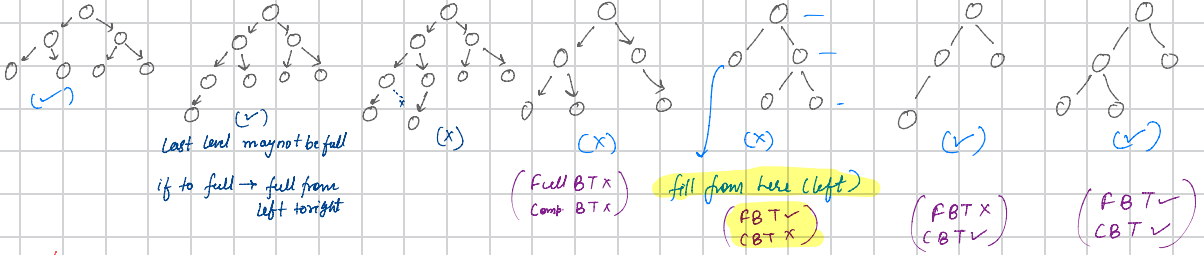


→ Properties :→

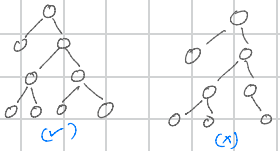
1. Size = Total number of Nodes
2. Child & Parent
3. Siblings :→ Node Having same Parent
4. Edge :→ Line connecting two Nodes
5. Height :→ Max^m no. of edges from the Node & leaf Node
6. Leaf Node :→ Bottom most Nodes
7. Level :→ Height of Root Node − Height of a Node OR No. of Parent Nodes b/w Node & Root Node
   (root = 0)
   for Root Level → Level = 0 {h_r − h_r = 0}
8. Ancestor & Descendant
9. Degree :→ No. of children of a Node | Degree of a Tree = Max^m degree (0, 1, 2)

→ Types of Binary Tree :↓

1. Complete Binary Tree :→ All the levels full
   → Last level may not be full
   → Last level filled from left to right

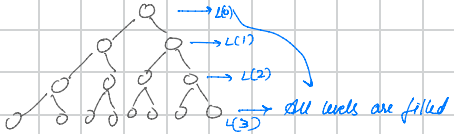


2. Full BT / Strict BT :→ Each Node have either 0 or 2 children



→ Use case :→ Compression, Segment Trees

3. Perfect BT :→ all the interal Nodes have two children
   &
   all the leaf Nodes are on same level
   ⇒ All Levels are full



N.→ Root Node is not Internal Node

4. Height Balanched BT :→ Avg. Height = O(log N)

5. Skewed BT :→ Every Node have only 1 child

Avg. Height = O(N) → Like Linked list (Bad thing)

6. Organised Binary Tree :→ Every Node has some property / condition that it follows
   eg:— BST

→ Properties that will help in some Question :-

1. Perfect BT → Height = h

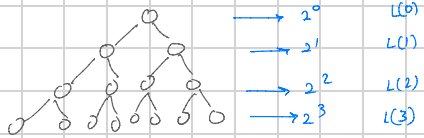
→ Total Nodes = $2^{(h+1)} - 1$

→ Total Nodes at i^{th} level = 2^i

→ Total Leaf Nodes = 2^h

→ Total internal nodes apart from leaf nodes

$= 2^{(h+1)} - 1 - 2^h = 2^h - 1$



$\therefore 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + \cdots + 2^n \rightarrow$ GP

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

2. N = no. of leaf nodes in BT

Least no. of levels = $\log N + 1$

N = no. of nodes in BT

Min^m no. of Levels = $\log(N+1)$

3. Strict BT :-

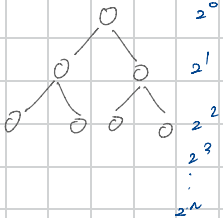
N = Leaf nodes

Internal Nodes = N-1

No. of Leaf Nodes = No. of internal nodes + 1

4. No. of Leaf Nodes = No. of internal nodes with 2 children + 1

(not including root node & leaf node)



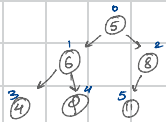
$N = 2^{h}$ ← height

$\Rightarrow \log_2 N = h \log_2 2$

$\Rightarrow \boxed{h = \log_2 N}$

→ Implementation :-

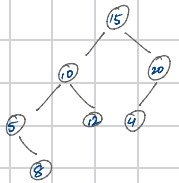
1. Linked Representation
2. Sequential → using array



0	1	2	3	4	5
5	6	8	4	9	11

→ Binary Search Tree :-

↳ sorted manner

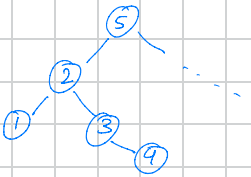


→ Start from root Node

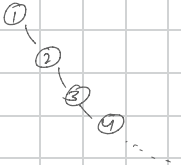
```
Node {
  int value;
  Node left;
  Node right;
  int height;
}
```

for sorted array :→ if start from 1 then → it will make skewed BT (Bad)

↳ so make root node → middle element

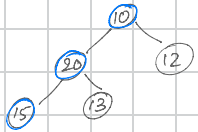


→ Time complexity :→ N log(N)



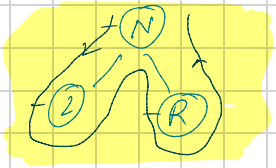
→ Traversal Methods :-

1. Pre-Order :→ N → L → R



10, 20, 15, 13, 12

N L R



→ uses :-

↳ for evaluating math expressions

↳ making a copy

↳ serialization from string/array

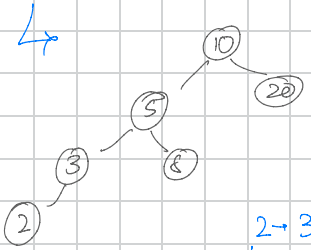
2. In Order :→ L → N → R

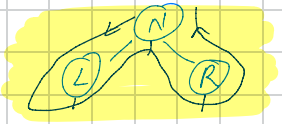
15 → 20 → 13 → 10 → 12

L N R

→ uses :-

↳ In BST → visit node in sorted manner

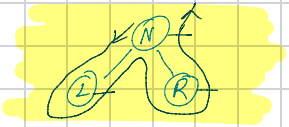




2 → 3 → 5 → 8 → 10 → 20 → Sorted Manner

L N R

3. Post Order :→ L → R → N



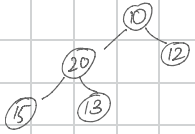
→ uses :-

↳ Deleting Binary Tree

↳ Bottom-up Cal^n

↳ Cal^n Height

↳ Cal^n diameter of a tree



15 → 13 → 20 → 12 → 10

L R N

* AVL Trees :-