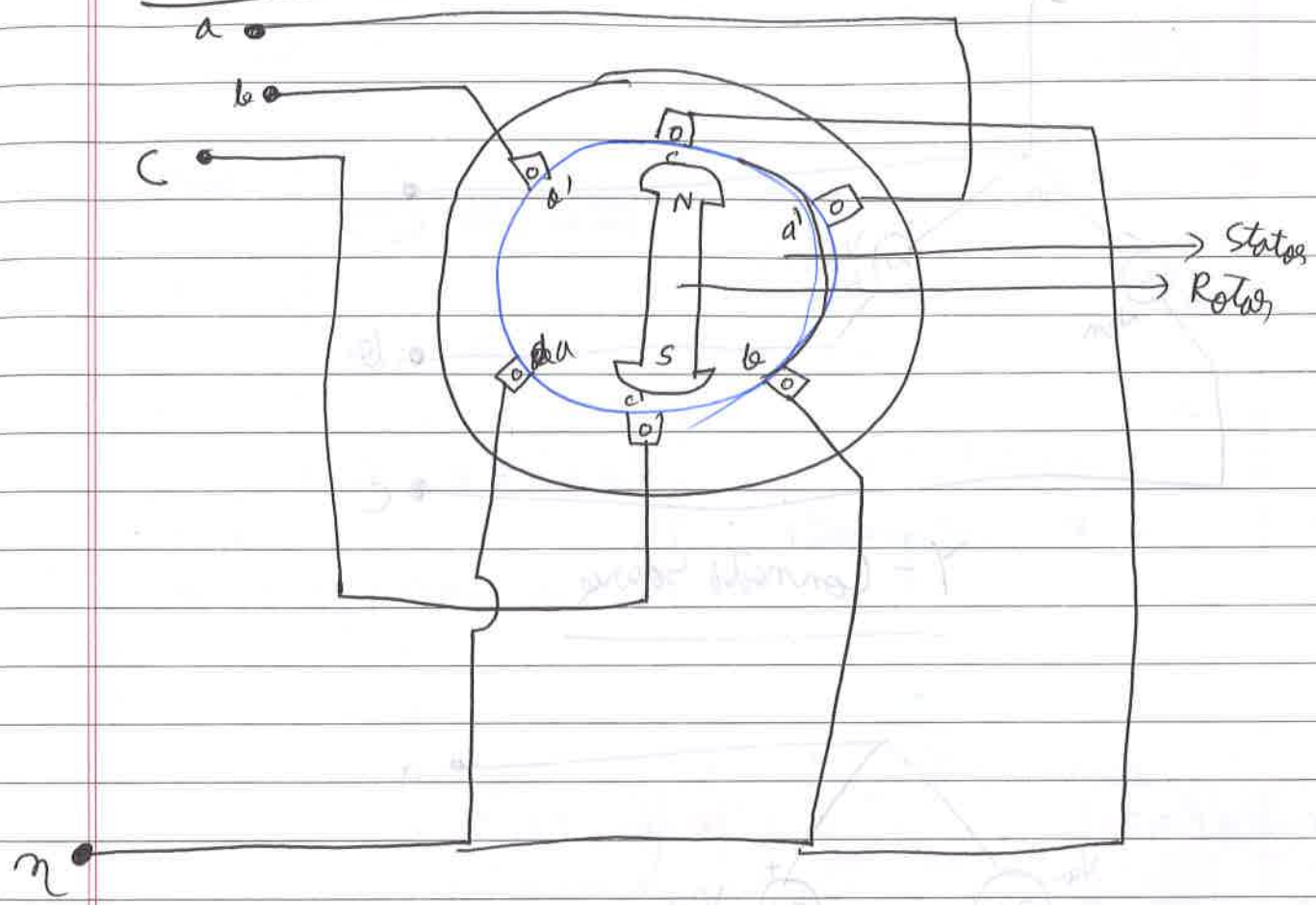
→ Three Phase Circuits -



Three phase generator

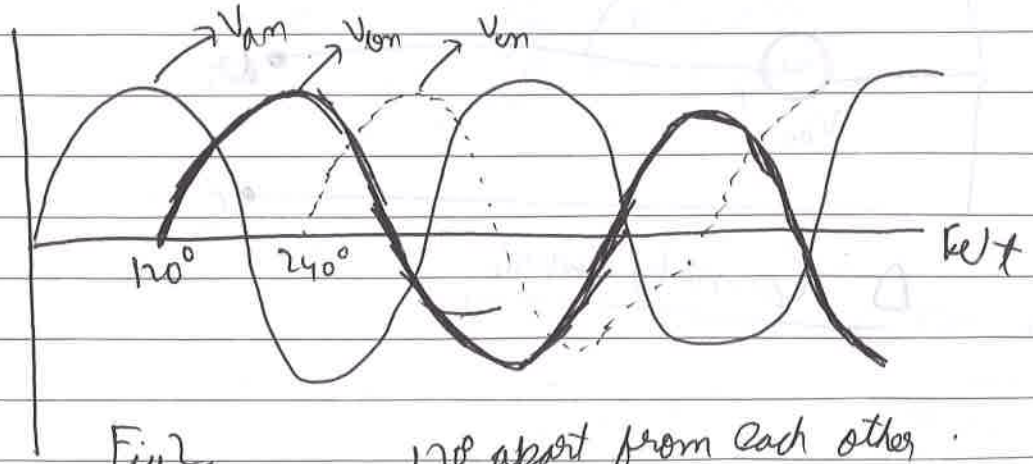
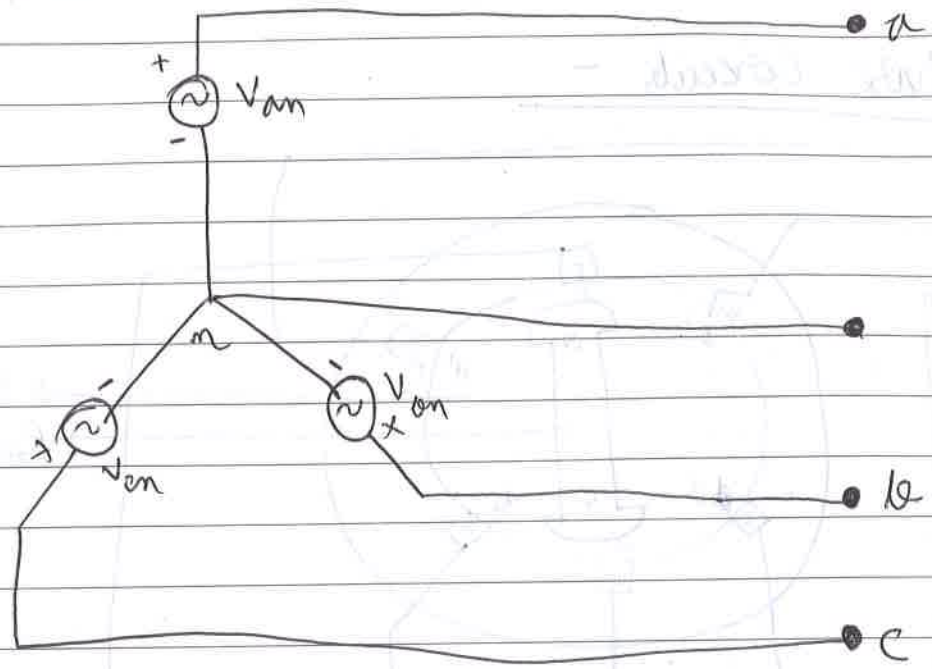
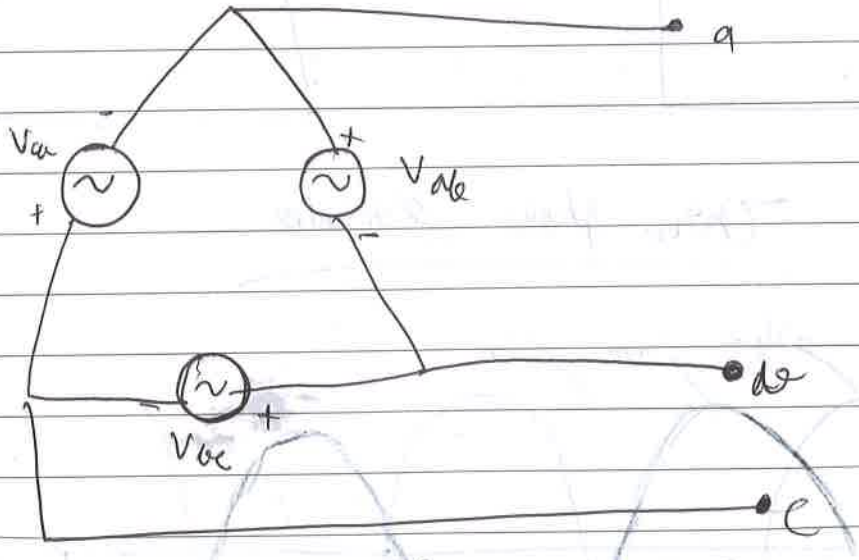


Fig 2 120° apart from each other.

Phase Sequence a - b - c



Y - Connected Source



Δ - Connected Source

$n \rightarrow$ neutral line.

$V_{an} \Rightarrow$ Voltage b/w line a and n
$V_{bn} \Rightarrow$ " " " b and n
$V_{cn} \Rightarrow$ " " " c and n

$\left.\begin{matrix} V_{an} \\ V_{bn} \\ V_{cn} \end{matrix}\right] \Rightarrow$ are called phase voltages.

$x = 1$; $n = 5$, m

$5! = (1 \times 2 \times 3 \times 4 \times 5) \% m$

$x = 1$

1) $x = (1 \times 2) \% m$
2) $x = ([(1 \times 2) \% m] \times 3) \% m$
3)

$(a/b) \% m$

$\downarrow$

$(a \% m \times \frac{1}{b} \% m) \% m$

$(\frac{1}{b} \% m) = \lambda$ means

$(b \times \lambda) \% (\mu m) = 1$

$(a \times b) \% m = ((a \% m) \times b) \% m$
$= (a \% m \times b \% m) \% m$

$(7 \times 6) \% 5$
$= 2$

$(7 + 6) \% 5$
3

$(2 \times 6) \% 5 = 2$
$2 \times 1 = 2$

$(2 + 6) \% 5 = 3$
$(2 + 1) \% = 3$

$2 + 5 = 3$

$S = a + ak + ak^2 + \ldots + b$
$Sk = ak \ldots + bk$
$Sk - S = bk - a$
$S = \frac{bk - a}{k - 1}$

$\frac{2^n - 1}{2 - 1}$

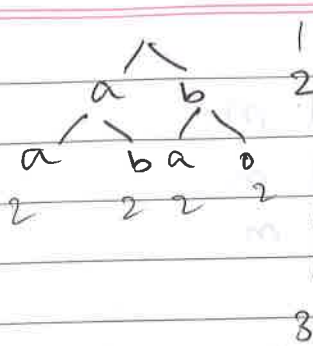
1	1	1
1	0	0
0	1	1
0	0	1

0 -

$\cancel{1 + \sqrt{5} = 1 + \sqrt{5}}$
$\cancel{2\sqrt{5}}$

1

$n \quad \sqrt{n} \quad \log n$



$2^n - 1 = O(2^n)$

n
$n/2$

$1 + 2 + 3 \quad \frac{(n)(n+1)}{2} = n^2$

5

1, 2, 4
5, 2, 1, 0

10, 5, 2, 1, 0

$2^n, (\sqrt{n})^3, n\log n, n^{\log n}$ (3) (4)

100, 50, 25, 12, 6, 3
$\log n$

$n^{\log n}, n\log n, n^{3/2}$
4 3 2

$\frac{n}{2}$

$n + \frac{n}{2} + \frac{n}{4} + \dots +$

~~1+3+~~

$S = n + \frac{n}{2} + \frac{n}{4} + \dots + 1$

$r = \frac{1}{2}$ $\quad \frac{bk - a}{k-1} \quad \frac{1(\frac{1}{2}) - n}{\frac{1}{2} - 1} \quad \frac{\frac{1}{2} - \frac{2n}{2}}{-\frac{1}{2}}$

$2n - 1 = n$

$N, \frac{N}{2}, \frac{N}{4}$

$S = n + \frac{n}{2} + \frac{n}{4} + \dots + 1$

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

$\cancel{-\frac{1}{2}} S = \cancel{-\frac{1}{2}}(1 - 2n) \quad \boxed{n} \quad 2n - 1$

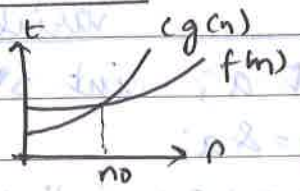
Time Complexity

Notations

→ 'O' Notation (Asymptotic Notation

- Upper bound of rate of growth of time
- $O(g(n)) = \{ f(n) : \exists\ c, n_0 \text{ st } \boxed{f(n) \le c\,g(n)},\ n \ge n_0 \}$

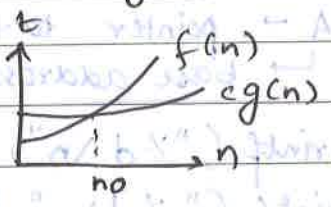
Eg:- $f(n) = 2n^2 + 2n + 1$, $g(n) = n^2$



→ 'Ω' Notation

- Lower bound of rate of growth of time
- $\Omega(g(n)) = \{ f(n) : \exists\ c, n_0 \text{ st } \boxed{f(n) \ge c\,g(n)},\ n \ge n_0 \}$

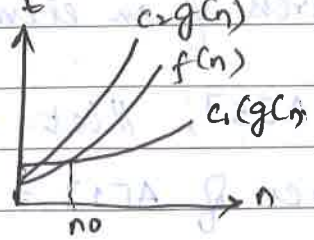
Eg:- $f(n) = 2n^2 + 2n + 1$, $g(n) = n^2$



→ 'Θ' Notation

- Hybrid of both 'O' Notation & 'Ω' Notation
- $\Theta(g(n)) = \{ f(n) : \exists\ c_1, c_2, n_0 \text{ st } \boxed{c_1 g(n) \le f(n) \le c_2 g(n)},\ n \ge n_0 \}$

Eg:- $f(n) = 2n^2 + 2n + 1$, $g(n) = n^2$



Time complexity Analysis

↳ Calculated for large input & worst case scenario

↳ Polynomials → $an^k + bn^{k-1} + cn^{k-2} + \dots$

→ Time complexity = $O(n^k)$

→ logarithm + polynomial → $T(n) = \Theta(n) + \log n = O(n)$
(as $\log(n)$ vanishes as $n \to \infty$)

Arrays

→ Pointers :- Variables that store address of another variable

```
int a;   int p*;
p = &a;
*p = 48;    // a = 48
```

→ Array :- sequenced collection of same datatype
- Array memory is arranged sequentially
- int A[5]; // array declaration

A → pointer to the first element of array
↳ base address

```
printf("%d\n", A); // prints base address
printf("%d\n", *A); // prints A[0]
printf("%d\n", A+1); // prints address of A[1]
printf("%d\n", *(A+1)); // prints A[1]
```

Address of an element = Base address + Offset

Eg:-
int A[5]; // let A = 100

address of A[1] = 100 + 4 // as A is int array

$n > m$

$n = n \% m$

$m \Longleftrightarrow n$

~~n=m, m~~ m

$n_1 = m_0, \quad m_1 = n_0 \% m_0$ $\qquad m_1 < m_0 < n_0$

$n_2 = n_0 \% m_0, \quad m_2 = m_0 \% (n_0 \% m_0)$ $\qquad m_1 < n_1$

$n_3 = m_0 \% (n_0 \% m_0) \quad m_3 = (n_0 \% m_0) \% (m_0 \% (n_0 \% m_0))$

$n_4 =$ $\qquad m_4 = n_3 \% m_3$

$(n_0 \% m_0) \% (m_0 \% (n_0 \% m_0))$

$(m_0 \% (n_0 \% m_0)) \% (n_0$

$a \% b < b$

$b \% (a \% b) < a \% b$

0 1 2 3 4 $\qquad 5/2 = 2$

1, 2, 3, 4, 5

$O(n)$

1 2 3 4 5 6 $\rightarrow 6 \% 2 = 3$

0 1 2 3 4 5

$10! =$

$10 \times 9 \times 8$

$5 \% 2 \qquad 4 \% 2$

3

3, 4

3, 4

1 2 3 4 5 6

0 1 2 3 4 5

temp2 temp

3628800

$3.6 \times 10^6 \times 10$

3.6×10^7

1234 5675

classmate

Date

Page

Module 'O'

Aug 16th

Module 'O'

I. A Bit of CS fundamentals (Time & Space Complexity)

TIME COMPLEXITY

Big 'O' Notation:- (Asymptotic Notation)

$f = O(g)$ if $\exists\, c > 0$ s.t. $f(n) \leq c \cdot g(n)$ for large n

$\Downarrow$ (alternate definition)

$f = O(g)$ if $\longrightarrow \lim_{n \to \infty} \frac{g(n)}{f(n)} = C,\ C > 0$

Classes of Running times →

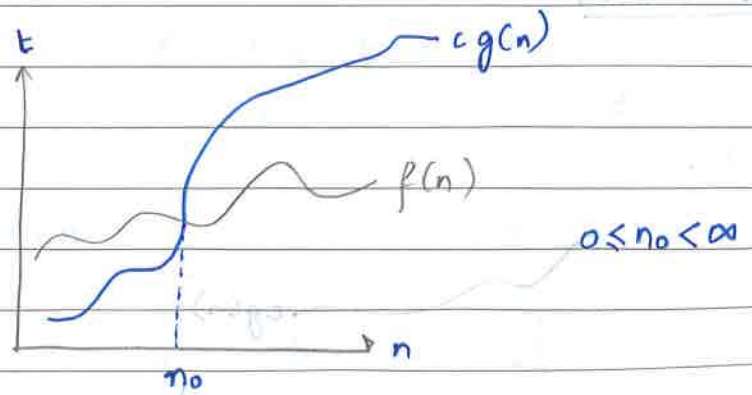
Best ↓ worst

- $O(1)$] constant
- Logarithmic [$O(\log N)$
- Linear [$O(N)$
- Linearithmic [$O(N \log N)$
- Polynomial [$O(N^2)$, $O(N^3)$, ⋮
- Exponential [$O(2^N)$, $O(3^N)$, ⋮

Note:- • focus on worst case *

• Time complexity is required to make code Efficient.

Graph of Big 'O' Notation



when $n \geq n_0$, $f(n) \leq c\,g(n)$

Example :- $f(n) = N^2 + N$

$g(n)$ can be both N^2 & N^3

$f(n) = O(N^2)$ & $O(N^3)$ $\rightarrow$ both valid

* but we focus on the most tight bound

hence,

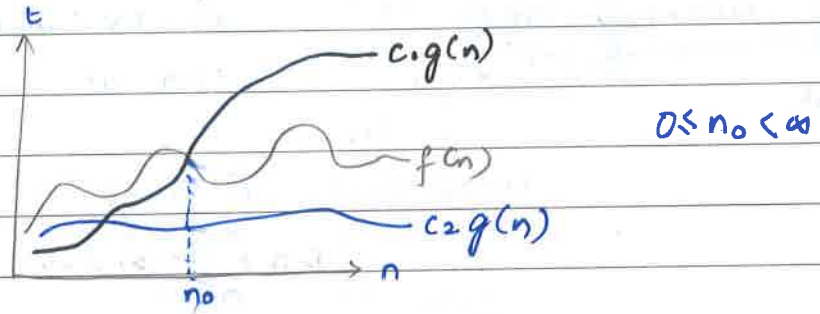
$$\boxed{f(n) = O(N^2)}$$

Theta Notation :- $f = \Theta(g)$ if $\exists\ c_1, c_2 > 0$ st

$f(n) \leq c_1 g(n)$ & $f(n) \geq c_2 g(n)$

for large n

Graph of Theta Notation



$0 \leq n_0 < \infty$

when $n \geq n_0$, $f(n) \leq c_1 g(n)$ & $f(n) \geq c_2 g(n)$

Example :- $f(n) = N^2 + N$

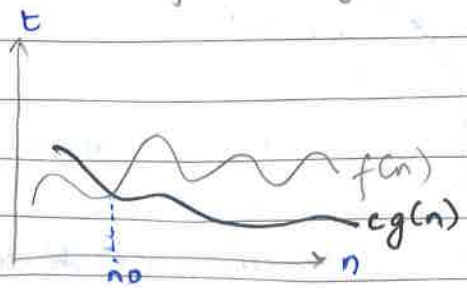
$g(n)$ can only be N^2

$\therefore$ $\boxed{f(n) = \Theta(N^2)}$

Omega Notation :- $f = \Omega(g)$ if $\exists\ c > 0$ st

$f(n) \geq c g(n)$ for large n

Graph of Omega Notation



$0 \leq n_0 < \infty$

when $n \geq n_0$, $f(n) \geq c g(n)$

Example :- $f(n) = N^2 + N$

$g(n)$ can be both N^2 & N

$f(n) = \Omega(N^2)$ & $\Omega(N) \rightarrow$ both valid

* but we focus on the most tight bound

hence

$$\boxed{f(n) = \Omega(N^2)}$$

Note:- $O \rightarrow$ upper bound $\rightarrow$ we take the smallest upper bound

$\Omega \rightarrow$ lower bound $\rightarrow$ we take the largest lower bound

$\theta \rightarrow$ exact class $\rightarrow$ sandwich of 'O' & 'Ω'

$\rightarrow$ When loops are independently NESTED $\rightarrow$ Multiply their individual time complexities

$\rightarrow$ When there are CONSECUTIVE LOOPS $\rightarrow$ Add their individual time complexities

$\rightarrow$ $O(N^2) + O(N) = O(N^2)$ [as N^2 is dominating when N is large]

Note:- If Time limit is 1 sec & constraint is $1 \leq n \leq 10^k$.

Suppose, the code is of $O(N^m)$

If, on replacing N with 10^k

$(10^k)^m \leq 10^8 \rightarrow$ code would work

$\leq 5 \times 10^7 \rightarrow$ code would definately work

$> 10^9 \rightarrow$ wont work

bounds	possible time complexity when time limit is 1 sec
$n \leq 10$	$O(n!)$, $O(n^7)$, $O(n^6)$
$n \leq 20$	$O(n.2^n)$, $O(n^5)$
$n \leq 80$	$O(n^4)$
$n \leq 400$	$O(n^3)$
$n \leq 7500$	$O(n^2)$
$n \leq 10^5$	$O(N\sqrt{N})$
$n \leq 5 \times 10^5$	$O(n.\log(n))$
$n \leq 5 \times 10^6$	$O(N)$
$n \leq 10^{12}$	$O(\sqrt{n}.\log(n))$, $O(\sqrt{N})$
$n \leq 10^{18}$	$O(\log^2(n))$, $O(1)$, $O(\log N)$

* The above possibilities are calculated by checking if the worst case is $\leq 10^8$ | Imp

Eg:- $n \leq 10^5 \rightarrow O(n\sqrt{n})$ is allowed as

$10^5\sqrt{10^5} \simeq 10^5 \times 317 \leq 10^8$

worst case

Amortization :- • When nested loops are not independent i.e they are interdependent, the order is not exactly multiplication of the # independent order of the loops. Such a case is called Amortization

Eg:
```
while (i<N){
    while(i<N & arr[i] != v) i++;
    if (i<N){
        i++;
    }
}
```

→ Ideally this has to be $O(n^2)$ but as loops are not independent, it is actually $O(n)$

SPACE COMPLEXITY

Eg:- arr[n][n] $\longrightarrow O(N^2)$

We use similar notation as of time complexity

Note:- • While implementing a problem, use
$\boxed{\text{int arr}[10^8]}$
to stay within general memory constraints

IMP

It is again safe to use
$\boxed{\text{int arr}[5 \times 10^7]}$

If it is an array of higher order, say $O(N^k)$, ensure that while defining the array, the worst space complexity stays $\boxed{\leq 5 \times 10^7}$

Eg:- arr[N][N] $\longrightarrow O(N^2)$
for worst case, $N^2 \leq 5 \times 10^7$ for the code to run

Time complexity of various classes

$f(n) = O(2^n + \log(n) + N^2 + N^N + C)$

In the above example, we see that the terms are of different classes. In such a case, we just keep the $\boxed{\text{WORST CLASS}}$ ie the class taking majority time & eleminate the others

$f(n) = O(N^N)$

NP-Hard

→ Some algorithms are such that we don't know if they are bounded by a polynomial

→ NP-Hard or NP-complete are a class of such problems

→ NP-Complete problems are reducible to one another, i.e if we solve 1 NP-hard problem in polynomial time, others get solved in polynomial time as well

IMP

→ But these are yet not proved to be solved in polynomial time

Example :- Travelling Salesman problem

NP → Non Polynomial
↳ Class of problems not solvable in polynomial time i.e. they dont have a solution of $O(n^c)$

IMP [Note :- • It is advantageous to know NP-hard problems
• It reduces energy & time wasted on finding optimal solution, as it won't at all exist

IMP [→ Such problems are solved using "Brute force" method mostly. Hence, their order is very high. The time complexity can be as high as $O(2^n)$ or $O(n^n)$

Master Theorem

Used for "Recursive Problems" mostly

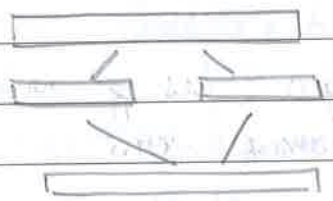
Master theorem can be used when the recurrence relation is of the form

$$\boxed{T(n) = a\,T\left(\frac{n}{b}\right) + O(f(n))}$$

→ a, b are constants,

→ T(n) is the time taken by the algorithm when input size is n

Example :- Merge Sort



Input n

Recursively divides in 2 parts of size $\frac{n}{2}$ each

Merges in O(n) time

Recurrence relation :- $T(n) = 2\,T\left(\frac{n}{2}\right) + O(n)$

The above can be solved using master theorem

Step 1 :- Note down a, b

Step 2 :- Calculate $n^{\log_b a}$ & let $\boxed{c = O(f(n))}$

Step 3 :- If $O(n^{\log_b a}) = c$, Ans = $O(f(n) \cdot \log n)$

If " $> c$, Ans = $O(n^{\log_b a})$

If " $< c$, Ans = $O(f(n))$

Thus, the time complexity of merge ~~sot~~ sort is $O(n \log n)$

Imp

Note: Master theorem is not applicable when $O(f(n))$ has the form

$$f(n) = \frac{n^k}{\log n}$$ i.e $\log n$ having -ve powers

We can get an approximate upper bound for such cases by ignoring the $\log n$ in the denominator & then applying Master theorem

Eg:- $T(n) = 8\,T(\frac{n}{2}) + \frac{n^3}{\log n}$

$n^{\log_b a} = n^{\log_2 8} = n^3$

$O(f(n)) = \frac{n^3}{\log n} \simeq n^3$

$T(n) \simeq O(n^3 \log n)$ using master theorem as $n^{\log_b a} = O(f(n))$ approx.

Practice Questions

Practice Problems 2 --- Question 1

Q.
```
for(int i = 1; i <= N; i++){
    for( int j = 1; j <= i; j = j x 2){
        //O(1)
    }
}
```

Ans $O(n \log n)$

Practice Problems 2 - Question 2

Q.
```
for( int i = 1; i <= N; i++){
    for( int j = i; j <= N; j = j + i){
        //O(1)
    }
}
```

Ans $O(n \log n)$ → as $1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{N} < \log N$

Note :- Practical Tips

Read Problem → Think of solution → & Calculate time complexity
↓
Check if it is inside $\leq 10^7 \times 5$ → If yes, then code → else optimize

→ Coding out problems that exceed Time limit is generally useless unless there is partial points or brute force points which is rare

→ In case of TLE, check for worst case test cases that might be causing it. Optimize the code accordingly. There might be mistake in code or calculation of time complexity of the code.

→ A good practice is to comment out the complexity of each chunck while coding itself as it helps to debug in case of TLE

II Getting started with C++

Data-Types

1. Int

int → range = ± 2×10^9

long long → range = ± 9×10^{18}

→ for problems having larger numbers, use (%m) after each step like "x%=m" along with "long long"

→ If only positive numbers + 0 are a part of the program, "unsigned long long" can also be used

→ type casting to long long → multiply with "1LL"

2. Float & Double

float < double < long double < long long double → range

↳ generally used

Note :-

long long int & long long double (and other data-types with high ranges) may cause memory to be exhausted.

Hence they have to be used only when absolutely necessary

—x—

Getline function

→ When we input string, we use cin generally. But, cin considers blank space and newline as end of the string.

Imp [→ This is when getline is used. It is used to take input along with space until newline is encountered

Common mistake :- If we use cin and then getline, input is just taken as '\0' as the newline character after cin is not eliminated.

IMP [Hence, use a dummy getline between actual getline & cin when cin is used before getline

Syntax:- getline (cin, string_variable);

Note :- int ~ long < long long hence, if int overflows, long would also overflow

Note :- "endl" takes a lot of time, hence use '\n' at all places

Important tips for coding

→ To use the value of pi, define
long double PI = acos (-1.0)

Practice Questions

C++ Solved Examples P2 → Problem 5

Q. Given 2 numbers A & B, Print floor, ceil & round of A/B.

Ans. floor = A/B
ceil = if A%B=0 then A/B else (A/B+1)
round = (A/B) + (1/2) → $\left(\frac{2A+B}{2B}\right)$ → correct
↳ incorrect a floor first

min( ) & max( ) functions

Syntax :- min(a, b) , min(a, (min(b, c))),
min({a, b, c})

Returns :- minimum of all the numbers

similar syntax for max

ASCII Values

Characters	-	ASCII Values
0 - 9	-	48 - 57
A - Z	-	65 - 90
a - z	-	97 - 123

Practice Question

C++ Solved Examples P3 - Problem 10

Q. ~~Sl~~ Given A, B, C, D, check if $A^B > C^D$
$(1 \le A, C \le 10^7)$ $(1 \le B, D \le 10^{12})$

→ Use log on both sides. The problem transforms to $\boxed{B \log A > D \log C}$. As log is an increasing function, sign does not change.

For comparision use $\boxed{B \log A - D \log C < \text{epsilon}}$ to check equality. This is to avoid floating point error

epsilon can be $\boxed{1e-9}$ or $\boxed{1e-12}$.

Note :- Timelimit is given for a testfile (i.e. T number of testcases) and not for a single test case.

~~Suppose we have T test cases with $N_1, N_2 \ldots N_T$ array size.~~
~~Limit:~~

Understanding Online Judge

1) Suppose we have a question with the following constraints

→ $T \leq 10^5$
$N \leq 10^6$

ΣN across test cases $\leq 10^6$

In such a case we first think that the rough estimation of total number of cases would be $\boxed{N \times T \leq 10^9}$ → wrong

This is actually a wrong estimation due to the last line. In the correct estimation is $\boxed{N_1 + N_2 + \ldots + N_T \leq 10^6}$ → correct

2) Most imp

TLE → Time limit ~~Error~~ Exceeded → Inefficient → ∞ loops
WA → Wrong Answer → wrong logic → misunderstand problem; → presentation issue → overflow
RE / RTE → Runtime Error → out of bounds → divide by '0'; → pointer problem → ∞ loops → improper input
CE → Compilation Error → wrong syntax → fun^n not defined
MLE → Memory limit ~~Error~~ Exceeded → using large memory; → allocation in wrong scope

Least imp

generally seen in DP problems

#Note

→ In interviews use 'int' and ask if there is any limit to data
→ In coding tests use 'long long' at all places generally
↳ (not a good practice but works)

Note :-

```
int main(){
int arr[1000 000 00];
}
```

→ Gives MLE as this declaration uses stack hence overflow

→ use int x = new arr[1000 000 00]; or declare array globally to avoid this problem. Such declarations use memory from heap hence no overflow

heap can handle large memory chunks easily, stack cannot.

Declarare arrays of $\boxed{size \geq 10^6}$ in global scope or using pointers only

bracket every operation when dealing with binary operators.

To avoid declaring unwanted array of large size, use arr[100100]. This avoids confusion in zeros

Tricks :-

→ To use resizeable arrays, use vectors.

→ memset(arr, -1, sizeof(arr)); // O(N)
this || sets the whole array || to -1 or ~~any~~ ~~number~~ 0 ~~specified~~ only

→ Global arrays → preset to '0'
local " → " " garbage

strings end with '\0' character

Suppose s is a string.

for (int i=0; i<n; i++){ s=s+'a'} //O(n²)	for (int i=0; i<n; i++){ s+='a'} //O(n)
↳ not optimal	↳ optimal
add and copy hence $1+2+\dots = \frac{n(n+1)}{2} = O(n^2)$	only append so O(n)

String functions → O(n) funtions

reverse (s.begin()+i, s.begin()+j+1) → reverses from i^{th} index to j^{th} index

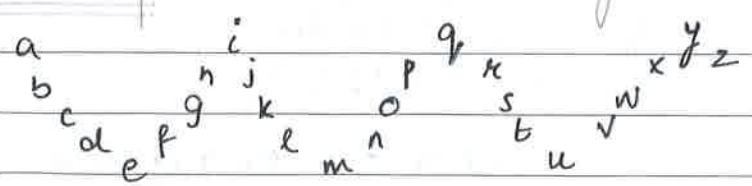
s.substr (1, 3) → returns substring from index 1 and of size 3 (starting from 0)

s.substr(1) → returns substring from index 1 to end

Stoi(s) → string to int
stoll(s) → " " long long
stod(s) → " " decimal double
stold(s) → " " long "
to_string(x) → converts any datatype to string

Pattern Questions

→ Use loops and not array



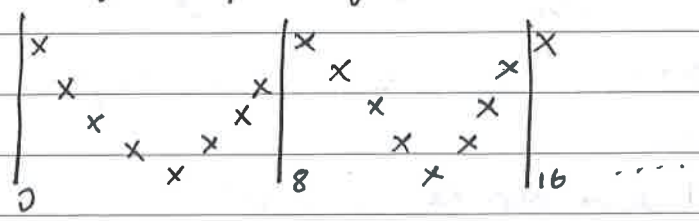
→ eg

→ any cell → space / → char → which character

we need to build a function that gives the character/space present at each cell given the position. Suppose it is f(i,j)

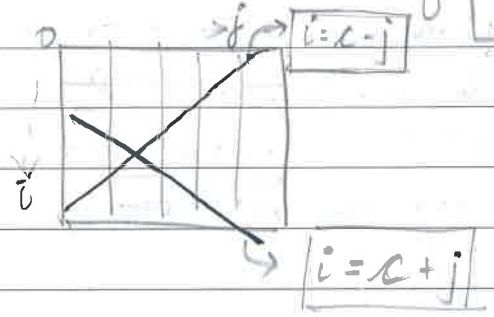
- We observe that, a, b, ... z are present in columns 0, 1 .... 25. Hence we print $\boxed{\text{char('a' + column)}}$

- the pattern from a to h repeats in units of 8 [check for repeating units]



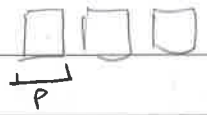
here we can use %8.
character exists if $\boxed{\text{column \%8 = row or (8 − row)}}$, else empty

rules

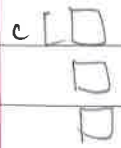


i = row index
j = column index
c = constant

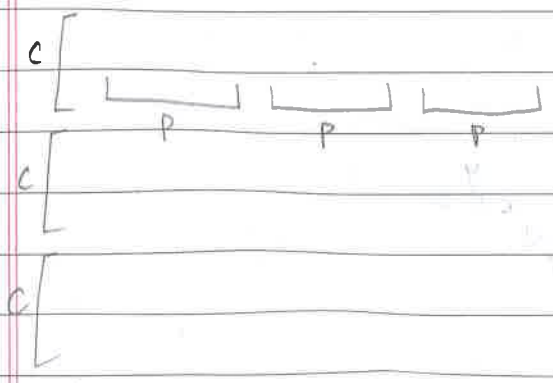
→ formulae for repeating pattern



→ pattern repeating horizontally, use $\boxed{\text{column \% p}}$



→ pattern repeating vertically, use $\boxed{\text{row \% c}}$



→ for pattern repeating horizontally by 'p' and vertically by 'c' units, use $\boxed{\text{row \% c}}$ & $\boxed{\text{column \% p}}$

call → char → which char
→ space

Solved using periodic logic, shifting & lines

simple observation

→ solution methodology

Problem Solving Paradigm

Commonly used terms

1) Recursion :- function calling itself through itself or some other function.

Along with the paradigms we need maths to solve problems

MATH
- → % Arithmetic
- → Combinatorics
- → Number theory
- → Geometry
  - → Mathematical
  - → computational

2) State :- Configuration that a problem is in right now.
Eg:- Rubics cube → to solve you move from state to state.
Fibonacci problem → initially n is the state of the function Fib(n) which moves backwards and hence the state changes.

Types of Paradigms

1) Brute force :- Exploring every possibility until you reach the final solution.
→ Code is generally a "backtracking recursive" code.
↳ (Moving from state to state to check for final solution and recursing back in the case of no solution in the path)
→ But it does not always use backtracking. it can be ITERATIVE as well.

2) Divide & Conquer :- Divide into 2 subparts, solve them and combine to get a complete solution
→ Done recursively until the subpart is undividable
→ Eg:- Merge sort, (Merging happens in O(N) time)

Total time taken → O(n log n)

Local optimal states are used to proceed towards global optimal states

3) Greedy :- Does some computation at every state and proceeds towards the most apt state.

Eg :- we have $S = \{1, 5, 6, 7, 2\}$. We need to find combination that sums to '14'

Solutions

- Brute-force :- check all combination
- Divide & conquer :- Divide into even & odd $\{1, 5, 7\}$ & $\{6, 2\}$. As the sum 14 is even, we need to take odd numbers in pairs only
- Greedy :- We take the largest number & recurse, We have $S' = \{1, 5, 6, 2\}$ and we need to find combination that sums to 7 $(14 - 7 = 7)$.

4) DP (Dynamic Programming) :- Brute force + Caching (memoization)

→ In brute force ~~many~~ many paths get repeatedly explored, this increases the computation cost & time.

→ Caching decreases this problem by saving the result of a state at each step in a path. When a path is revisited, the prior information that is ~~cho~~ cached is used and thus time & computation is reduced.

5) Graph Based Ideas :- There are 15-16 std. algos only.

- Brute-force → Heavy → need Constraint relaxation
- Greedy → light → work well with tight constraints
- DP → Heavy but have repeating subparts → work well with moderately light constraints
- Divide & Conquer → Only finite eg.

Note :- • Greedy algos. might be wrong for some examples as they don't explore all the paths.
• Brute-force & DP are generally right as all paths are explored. Hence if some question works well with all three, DP, Greedy & Brute-force, Prefer DP.

DP generally works when there is a constrainted variable in the problem

Difference between sub-array, sub-sequence & sub-set

→ Sub-array :- A continuous bit of the main array

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

→ N elements array

↳ N+1 dividers.

We choose any 2 dividers to get a sub-array b/w them. Thus, no. of sub-arrays = $\boxed{{}^{n+1}C_2}$

→ Sub-sequence & Sub-set :- Any sequence formed by deleting elements from the array. It is similar to sub-set, but the ordering matters here while is does not in sub-set. Thus, the total number of sub-sets / sub-sequences is $\boxed{2^n}$ including the empty set / sequence

Eg:- {6, 1} is a sub-set while {1, 6} is both but not sub-array

Note

Pruning → Removing bad options ~~paths~~ from any path/state.

Problem Solving Strategy

Steps to solve problem

1) Read (5 min on avg)
   - → Understand the problem
   - → ~~Check for Input/Output type & constraints~~
   - → Read & Understand the sample cases (atleast 1)

2) Ideate (Pen & paper) (25 min from scratch)
   - → Memory (to find similar questions)
     - → Long term (Interesting questions)
     - → Short term (Recent questions)
   - → Brute the idea
     - → Brute force
     - → Divide & conquer
     - → Greedy
     - → DP
     - → Maths Related → Graph
     - → Choose method using
       - → Past Experience
       - → Pattern
       - → Constraints
       - → Elimination (MOST IMP)
     - → Quick check with sample cases

3) Formulate → Decide coding structure to implement the logic (helps write short codes)
   (1-2 min)

4) Coding → Implementation (DO NOT USE TEMPLATES)
   (15-20 mins) → Read codes to improve your style

5) Debug
   (10 mins approx.)

(Timing is for ~~medium level~~ codes which are out of ~~comfort~~ comfort zone)

STL

III Applied learning of C++ (only coding questions)

IV Unleash the power of STL → Standard Template library

STL
- → Algorithms
- → Data structures / Containers

Algorithms :- Sort(), Binary search, etc.
DS :- Vector, stack, queue, priority queue, map, set, multiset, bitset, deque, pair, list etc.

Note :- The best time complexity for sorting is $O(n \log n)$. 'sort()' has the same time complexity.

cppreference.com (second best), cplusplus.com (best) → reference sites

learncpp.com (oops is good in this)

Vector :-
→ Array with a dynamic size.
→ Any number of elements can be added at the end of a vector. Deletion is also possible.
→ Just like array, we can access any element in between

IMP → Suppose we have ~~an~~ a vector of size 'n'. To add $n+1^{th}$ element, vector doubles its size to '2n'. To add $2n+1^{th}$ element, vector again doubles its size to '4n' and so on.

Eg :- V = | 3 (0) | 2 (1) | 1 (2) | 5 (3) |

V[2] = 5 ; // O(1), replaces 1 with 5, V = 3, 2, 5, 5
// RANDOM ACCESS

v. back(); // returns the last element 5

v. push_back(3); // adds 3 at the end, v = 3 2 5 5 3

IMP → v. size(); // returns 5 of type unsigned int

v. capacity(); // returns 8 due to doubling

v. pop_back(); // deletes last element, v = 3 2 5 5

Note :- • Capacity does not decrease after pop_back

• push_back & pop_back have average case O(1)

Amortized complexity for each time used

↓

average of doubling and push_back is O(1)

• Runtime error occurs if we access beyond index.

Syntax :- vector<int> v(3) // creates ~~empty~~ 0 vector of size 3

vector<int> v(n, 3); // creates n sized int vector with all entries as 3

vector<int> v = {1, 2, 3}; // creates v = 1 2 3

vector<int> v // creates empty v

Note :- • unsigned int x = 0;

cout << x - 1 << endl; // gives a very large number

// it is the max bound of unsigned int

IMP • typecast v.size() into (int) to perform any integer actions

• This is true for all ".size()" functions

Note :- • vectors work slower than arrays

• The dynamic STL datatypes work slower than their static counterparts

Stack :- (FILO or LIFO)

Syntax:- stack <int> st; // creates st integer stack
st.push(3) ; // st = 3
st.push(4) ; // st = 4 3
st.pop() ; // st = 3 , returns Nothing
st.top() ; // returns 3

Note :- → st.top() gives Run-time error if stack is empty. Hence, never use st.top() if stack is empty.
→ st.empty() returns True if empty else False.
→ st.size() returns size of the stack
→ All functions take O(1) time to execute once.
(worst case)

Note :- → In stack only top element can be accessed
→ The advantage of stack is that it is faster than vector functions. But, that hardly affects the Run-time.

Always check complexity before using functions.

Nesting

We can nest datatypes in STL.

Eg:- vector < vector <int> > arr;
stack < vector <int> > st ;

Pair :-

Syntax:- pair<int, int> x; // creates a pair x
x.first(); // returns the first element
x.second(); // " " second "
pair<int, string> y {5, 6}; // creates y

for strings, gives largest string lexographically

Priority Queue :-

Syntax:- priority-queue <int> p ; // creates priority queue
p.push (3); // inserts 3
p.push (5); // " 5
p.push (2); // " 2
p.top () ; // returns largest value i.e 5
p.pop () ; // deletes 5 from p
p.top () ; // returns 3

Note :- → All of the above functions except top() take O(logn) time
→ Priority queue is a heap based data structure that maintains the maximum value. (Heap is a tree type data structure)
→ top() takes O(1) time

Queue :- (FIFO)

Syntax:- queue <int> q; // creates q
q.push (3); // q = 3
q.push (5); // q = 3 5
q.front() ; // returns 3
q.back () ; // returns 5
q.pop () ; // returns nothing, q = 5

Note :- → All the above functions take O(1) time

Map :- (tree map / ordered map)

Syntax:- map <int, string> mp ; // creates a map whose ~~index~~ key is int and value is string.

mp[3] = "abc"; // node 3 contains "abc" // O(logn)
mp[5] = "a"; // " 5 " "a"
mp.erase (3) ; // erases node 3
it = mp.find(5) ; // returns iterator to 5. if 5 is not present, // returns mp.end().

// it is a pair, it.first = 5, it.second = "a" or
it -> first = 5, it -> second = "a"

Note:- it++ is Amortized O(1) but O(log N) worst case

→ Map is a datatype in which we can store values in nodes. They are implemented using RedBlack Trees. Maps are always sorted by keys

→ Map creates only those indices which are required, unlike arrays. Hence, it can handle large indices as well.

→ All operations of map is O(log N)
→ keys cannot be changed in anyway.
→ Map is a key, value pair.

Set:- Maintains distinct elements in sorted order (Bidirectional)

Syntax:-
```
set <int> s;
s.insert(2);   // s = {2}
s.insert(3);   // s = {2, 3}
s.insert(2);   // no duplicates allowed, hence s = {2,3}
s.erase(2);    // s = {3}  returns iterator to 3
```
O(1)
```
s.size();      // returns 1
s.begin();     // gives the smallest element
s.end();       // " " largest "
```

→ Set is again a tree based structure (R-B trees), hence insert & erase take O(log N) time on the worst case.
→ Construction takes O(n log n) time.
→ Index based access is not possible in set.

Multiset:- Same as set, but duplicates are allowed

Syntax:-
```
multiset <int> m;
m.count(5);   // gives frequency of 5
```

Note:- Clear always takes O(N) time.

Deque :- Hybrid of Stacks & queues →

- Can Insert & Delete from both ends.
- Any index in between can also be accessed.

Syntax :-
```
deque <int> d;
d.push_back(3); //d = 3
d.push_back(5); // d = 3 5
d.push_front(1); // d = 1 3 5
d.pop_back(); //d = 1 3
d.pop_front(); // d = 1
```

All the above functions take O(1) time.

Note :- Deque is slower than Vector

Template

Class / function that can be applied to different data types

Syntax :-
```
template <class T> T function_name (T param_1, T param_2 ...)
{
    // code
    return (val); // val has to be of type T.
    // template & class are key words
}
```

Eg :-
```
template <class T> T absmax (Ta, Tb) {
    return abs(a) > abs(b) ? a : b;
}
```

Allowed :- absmax (-1, -2);
absmax ('a', 'b');

not allowed :- absmax (1.5, 3); // datatype of a & b cannot differ
absmax ("a", "b"); // we cannot find absolute of a string

Sort — (Random Access Iterator)

Syntax :- sort (arr, arr+n); // sorts 0 to n-1 indices of array

Time complexity :- $O(n \log n)$

Note :- To sort other things, we need to define '<' operator or a compare function.

Eg :-
```
struct St{
string name;
double height;
};
bool operator < (St a, St b){
return a.height < b.height;
};
```

Eg :-
```
bool comp (St a, St b){
return a.height < b.height; };
sort (s, s+n, comp);
```

Note :- • To sort in reverse order, replace < with > inside curly brackets in both the functions
• We get error if comp returns true for both comp(x,y) & comp(y,x). This applies to the custom operator < as well
• You can also write the comp function inside sort.

nth_element → (Random Access Iterator)

Finds the nth element of array (indexing from 0) in case array is sorted from 0 to m.

Syntax :- nth_element (arr, arr + n, arr + m);

Order :- $O(n \log n)$ → worst case
$O(n)$ → Avg case

reverse → (Bidirectional Iterator)

Syntax :- reverse (arr+m, arr+n+1); // reverses from m to n
reverse (s.begin()+m, s.begin()+n+1) // same as above
reverse (s.begin(), s.end()) // reverses the whole string

Order :- $O(n)$ | s is string

Unique → (Forward Iterator)

Syntax :- int n = unique (s, s + strlen(s)) - s;
s[n] = 0;

Order :- $O(n)$

Unique clubs the ~~the~~ continuous non-unique elements.

Eg:- "aabbbccadd" will be converted to "abcad"

This function is generally used ~~to~~ with sort to ~~to~~ find unique elements of a string | array.

Note :- s.length(), strlen(s) give the same output
↳ string datatype ↳ char arrays

binary_search → (Forward Iterator)

Syntax :- binary_search (arr+l, arr+m+1, n); // searches if n exists in b/w l to m in arr

Order :- $O(\log n)$ → for random access iterator
$O(n)$ → for forward iterator

Returns :- bool

Note :- For binary_search to work the range ~~be~~ must be either sorted increasingly or partitioned wrt the value of n.

1 2 3 [5] 5 5 [6] 7

→ upper bound of 5 (>)

lower bound of 5 (≥)

lower_bound → (Forward Iterator)

Syntax :- lower_bound(arr+l, arr+n+1, n) // returns the left most element position ≥ n (iterator)

Order :- Same as binary search, hence the condition to work are also same

→ O(1) for random access iterator
O(n) " others

note :- distance(a, lower_bound(a, a+10, 5)) gives the number of values less than 5 in a. Or we can say that it gives the index of the left most element ≥ 5.

→ lower_bound gives the position where we can add the element without disturbing the sorting/partitioning.

upper_bound → (forward iterator)

→ It is same as lower_bound except that it returns the left most element position > n

In case all elements are < n, then (arr+size) is returned in both algorithms

next_permutation

Returns the lexographically ~~next~~ next greater permutation

Syntax :- next_permutation(arr, arr+len);

Order :- O(n) per call → O(1) Amortized

Note :- If we hit ~~if~~ lexographically the largest permutation, then false is returned & the array is converted to lexographically smallest permutation

String_functions

1. s.substr(n, m); // gives substring from n^{th} element and of length m
2. s.compare(t); // returns +ve number when s>t, -ve number when s<t & 0 when s=t
3. s.c_str(); // returns char array form of string

to use array functions on ~~string~~ string datatype, s.begin() , s.end() has to used in place of arr , arr+length . The same applies for vectors

Iterators

Pointers through containers (like vectors)

Eg :-

```
for( vector<int>::iterator it = a.begin(); it != a.end(); ++it){
    sum += *it;   //gives sum of all vector elements
}
```

note:- "vector<int>::iterator can" be replaced with "auto". This is used when we want to auto detect the datatype.

Eg:- auto it = upper_bound(a.begin(), a.end(), 10);

"*it" gives the value at the pointer.

"*it" gives random value (garbage) if 'it' points at a.end(). Hence check before printing

2D vector

Syntax:- vector<vector<int>> a(n, vector<int>(m));

a[i][j]; //returns a[i][j]

→ vectors can be compared using lexographic rules

Eg:- a={1,2,3} , b={1,2,4}

a<=b; //returns 1

→ The same comparison is done for pairs too

List :- implemented using bidirectional linked list

Syntax :-
```
list<int> l{10,20};
l.push_back(30);
l.push_front(1);
l.front(); // return 1
l.back(); // return 30
auto it = l.begin(); // >0 10 20 30
++it; // 0 >10 20 30
O(1) ← it = l.erase(it); // 0 >20 30
++it; // 0 20 >30
O(1) ← it = l.insert(it, 25) // 0 20 >25 30
l.insert(l.end(), 40) // 0 20 >25 30 40
cout << *it; // print 25
```

→ l.rbegin() & l.rend() traverses from end to start

→ sort cannot be used for linked list. Instead use
```
l.sort(); // O(n log n)
```
Reason is that it is not random access type

	vector	list	deque
Random Access [ ]	O(1)	—	O(1)
push / pop front	O(n)	O(1)	O(1)
push / pop back	O(1)	O(1)	O(1)
Iterators	Random Access	Bidirectional	Random Access
Insert / erase at iterator	O(n)	O(1)	O(n)
Sort	O(n log n)	O(n log n)	O(n log n)
Binary Search	O(log n)	No	O(log n)
Contiguous Memory	Yes	NO	No

To find minimum of a set using priority queue, insert negative of all numbers. For strings, the following syntax can be used to declare. Then the pq will give minimum string.
```
priority_queue<string, vector<string>, greater<string>> pq;
```

If st is ~~it~~ a set, st.insert(x) returns pair<set<T>::iterator, bool>.
The ~~f~~ boolean is true, if the element is not inserted before and is false if the element is already existing in set x.

Custom structures with sets

Eg:-
```
struct A{
    string s;
    bool operator < (const A &o) const {
        return s[0] < o.s[0];   // important to define
    }
};
void main(){
set<A> st;
st.insert(A{"Apple"});
st.insert(A{"Banana"});
st.erase(A{"Banana"});
auto it = st.insert(A{"Appee"});
cout << it.first -> s << endl; // print Apple
```

# Set is a bidirectional iterator.
it += 5 ; // O(5) not O(1) as it can traverse front or back only

distance(it1, it2) // O(it2 - it1)

Unordered_set / unordered_map

Unordered versions of set / map implemented using hash tables.

Order :- Avg O(1), worst O(n)

Note:- (for unordered-set)

→ The elements here are not sorted

→ Binary search cannot be performed

→ begin() does not return the smallest element

→ Like vectors, once full capacity is reached, the ~~of~~ capacity of is doubled

Syntax:- unordered-set <int> us;
us.insert(20);
us.bucket-count(); //returns bucket size

→ int, double & pair are hashable, hence can be used in unordered-set. But pair is not, hence custom hash function needs to be defined

Ref: Module 0, Unleash ~~fe~~ the Power of STL, STL feature Rundown 2, 38.06

Iterator validity → iterator pointing to end() will be invalidated after decrease in size

1) set / map

```
st = {1, 2, 3};
it = st.begin();
st.erase(it);
cout << *it;
```

wrong as reallocation has not happend and the previous iterator is erased

```
it = st.erase(it);
cout << *it; //print 2
```

right

insert → All iterators remain valid
erase → Iterators to erased element are invalidated

2) unordered set/map

insert → All iterators are invalidated due to rehashing
erase → Iterators to erased element are invalidated

Note :- You cant compare iterators as < or >,
they can only be compared as == or !=

For set & multiset

→ sort, upperbound etc dont work normally as they do for arrays/ vectors. Tree based algos have separate functions for them

→ find_by_order( ) gives the value at index (starting with 0)

→ order_of_key( ) gives the index of the value

→ used in ordered_set, not used at all, Not imp

note.

- multimap does not support [ ] operator

- multiset is arranged acc. to the ascending order of the keys (first values). If 2 keys are same, then the values of those 2 keys are compared and the smaller one is arranged first.

- Default value of any in map is 0.

Note

- gcd(a,b) can be used to find gcd of a 2 integers. It returns an integer
  Time complexity : O(log(max(a,b)))

★ gcd cannot be found for -ve numbers

- Do NOT access indices of vector/deque without inserting using push back/front. Results in SEG FAULT.

Module 1 - Week 1

Making STL Your Humsafar

I STL Application Ideas

STL Applications 1

Q1) Checking if the bracket string is valid → String contains brackets of one type

⟹ let us use temp=0 variable

temp++ if '('

temp-- if ')'

In between we would encounter -ve temp if a bracket is closed before it is ~~opende~~ opened.

Also, at the end, temp value would be zero if all opened brackets are closed.

Hence, string is Invalid if → temp < 0 at any point

→ temp ≠ 0 at the end.

else the string is valid

Note:- • for (int i=0; i< s.length(); i++
{ v= s[i]; }

imp

can be easily written as

for (auto v: s){ }

• for(auto &v: s){ } is passing by reference

in this, any change in v will also reflect in s

Q1) a) extended:- Suppose we have a string of brackets, we need to check if a sub-string is valid.

⟹ let the input string be

(( ) (( )))

temp values - 0 1 2 1 2 3 2 1 0

A sub-string is valid if → temp value before encountering the sub-string is same as the temp value after the end of the sub-string

→ (checks equality of '(' & ')') &

→ temp value does not fall less than the initial value anywhere in the ~~string~~ sub-string
(checks if a bracket is closed only after it is opened)

else it is invalid

valid

(( ) (( )))

0 1 2 1 2 3 2 1 0

invalid

Q1) b) extended:- Checking if the bracket string is valid in case of multiple type of brackets

⟹ ~~Ult~~ Use the same logic as in Q1)a), i.e, ~~to~~ on encounter of a new type of opening bracket, check if the sub-string is valid till it get closed

Eg:- [({ })]

OR

use stack → easy method

Note:→ To reduce if-else statements, use maps
[→ 1, (→ 2, { → 3,] → 4,) → 5, } → 6
We ~~than~~ can check if top (3-bracket) ~~with~~ value on encountering a closing bracket.
→ Always use if (!st.empty() || st.top <....)
to avoid Runtime error.

Q2) Given :- N → size of array, k → window size & array of size N.
We need to find the minimum value of ~~arr~~ sub-array of size k.

Eg :- Suppose N=9, k = 3

Array ⇒ 3 2 1 5 8 7 6 1 2

Expected Ans :- 1 1 1 5 6 1 1 → (N - k + 1) No.s of values to be printed

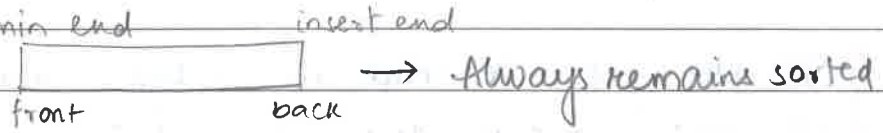
⇒ Solution 1 :-
(Using multiset & array.)

- Store the array from input ⇒ Arr = 3 2 1 5 8 7 6 1 2
- Traverse ~~of~~ over it, to put first k elements in multiset, multiset = 3 2 1
- Print the first element of multi-set (i.e 1), erase first element of Arr (i.e 3) from multi-set and insert next element of Arr (i.e 5) into the multi-set. ∴ multi-set = 2 1 5
- Repeat till the end of ~~arr~~ Arr is reached

Complexity :- O(n log k)

★★ IMP

Solution 2 :- (Monotone-deque) (Complexity O(n log k))
(Using deque)

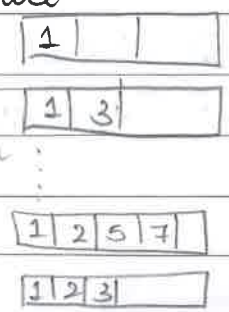


→ Always remains sorted

Construct a monotone deque, which maintains minimum at front end, and element can only be inserted from back.

Inserting rule :-

Eg :-



← 3 valid

← 2 invalid as 3 is greater than 2, so pop 3 & then insert 2

← 3, we pop 5 & 7, then insert 3

Popping-rule :-

1	2	3	

Suppose we want to pop j^{th} element of array. It is only popped if it is equal to the front element of the deque. Else, continued.

Printing-rule :-
The front element is printed

Algorithm :-
- Call Inserting-rule for first K elements
- Call Printing-rule and pop the $i-k^{th}$ element of array by calling Popping-rule
- Traverse till the end of array.

Logic :- A number is printed/minimum until it is removed or a ~~vate~~ another number lesser than it is encountered

STL Applications 2 # Note :- sort() function uses intro-sort which is a mix of 3 algos.

Q3) We are given N items in a shop with cost $a_1, a_2 \ldots a_n$
We are also given M ~~dt~~ budgets $b_1, b_2, \ldots b_m$.
We need to calculate the number of items we can ~~by~~ buy using each budget. (After each person buys, the items are replenished i.e. for every buyer, there is no option unavailable)

$N, M \leq 10^5$

$a_i, b_i \leq 10^9$

$O(n \log n + m \log n)$

⟹ Let arr = 4, 2, 7, 5 // array of items
sortarr = 2, 4, 5, 7 // O(n log n)
array ⟶ CC = 2, 2+4, 2+4+5, 2+4+5+7 // cumulative cost
= 2, 6, 11, 18 // O(n)

for each budget bi, find it = upper_bound (cc.begin(), cc.end(), bi) // O(m log n)

distance (cc.begin(), it) gives the number of items one can buy with bi amount of money // O(m × 1)

Eg:- • bi = 2, it points to 6, distance is 1, so with 2 money, we can buy 1 item
• bi = 7, it points to 11, distance is 2, so with 7 money, we can buy 2 items

Note:- The cumulative cost array (cc) is known as the 'prefix Sum' array.

It is created by adding the current value to the previous ~~sum~~ cumulative sum and updating the array value:- cc[0] = arr[0];
cc[i] = cc[i-1] + arr[i]; (i ∈ [1, n-1])

Prefix sum array is always sorted if all entries of the parent array are +ve.

Note:- distance (cc.begin(), it) can be replaced by $\boxed{\text{it} - \text{cc.begin()}}$ (this only works for vectors)

For maps, sets etc, we cannot subtract iterators.

1 2 4 4 4 5 7

upper-bound(4)

lower-bound(4)

classmate

Date

Page

imp

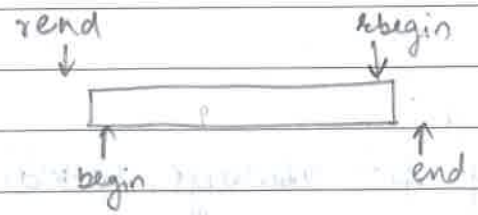
lower-bound()	vs	upper-bounds()
→ points to first value $\geq x$		→ points to first value $> x$
→ distance (a.begin(), it) gives the number of values in the array $< x$		→ distance (a.begin(), it) gives the number of values in the array $\leq x$

Algorithmic design

Creating a customized datastructure to support the required operations and perform them in minimum possible time complexity

Eg:- Create a structure where we can insert numbers and get the sum of all the numbers inserted so far at any given time.

→
```
struct bag {
    int cur_sum = 0;
    void insert (x)
        cur_sum += x;
    int Sum()
        return (cur_sum);
};
```



for any data-structure

Note:- • iterator + 3 only works in random access iterators.
• bidirectional iterators only support iterator ++ & iterator --

Note:- One can create ~~be~~ debug functions that prints certain output for each run which will help debug the code.

STL Application 3 (IMP)

Q4) We are given N cards with $\binom{A_1}{B_1}, \binom{A_2}{B_2} \dots \binom{A_n}{B_n}$ written on each card respectively. We are playing for player A and player B is the opponent. Suppose we pick i^{th} card, A_i is added to our score, and B_i automatically is unavailable for player B. Develop a strategy to win with maximum difference in score. (Note:- Cards are picked alternatively by A & B, and score is added for player B in the same way as player A)

⇒ Possible strategies

Picking card with
- max $(A_i - B_i)$
- max A_i
- max $(A_i + B_i)$
- max $(\max(A_i, B_i))$
- max $(B_i - A_i)$

Counter examples

$\binom{2}{7}\binom{3}{1}$

→ correct strategy

$\rightarrow \binom{8}{1}\binom{3}{7}$

$\rightarrow \binom{8}{5}\binom{2}{7}$

We can think along these lines:- The logic must be symmetric, i.e, if applied to player B, it should work. This elimenates 1st. & last option

Deriving the correct strategy

Differences

Suppose B has all cards initially → score A = 0, score B = $\sum_1^n B_i$] → $0 - \sum B_i = -\sum B_i$

We pick j^{th} card from B → score A = A_j, score B = $\sum_i B_i - B_j$] → $A_j - (\sum B_i - B_j) = A_j + B_j - \sum B_i$

$\Delta = A_j + B_j$ = final - initial

Picking one card has the potential to change the difference in scores by $(A_j + B_j)$. This is what we need to maximize.

for two cards, if $(A_i + B_i)$ is same, then if player A picks any of the cards, the difference would be same. Hence, it doesn't matter which card the player picks

Note:- Never use >= or <= in a comparator function, this would result in error. ~~when~~ Only use < or >.

Q5) We are given an array of size N and entries A_i, $i \in [0, n-1]$
We need to output the number of sub-arrays with sum X.

V. IMP.

$1 \leq N \leq 10^5$, $-10 \leq X \leq 10^6$, $-10^6 \leq A_i \leq 10$

Time constraint = 1 sec

$\Rightarrow$ Total no. of sub-arrays $= {}^{n+1}C_2$ (as we have n+1 partitions in an Array & we need to choose 2 to form a sub-array)

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

We cannot use Brute-force method, as the problem would then be of $O(n^2)$ and ~~will~~ would result in TLE

We can create a prefix sum array $\rightarrow O(n)$
Let Prefix sum array be P, left point L & right pointer R

So, the question is reformulated to outputting the number of pairs (L, R) such that

- $L \leq R$ &
- $P(R) - P(L-1) == X$

$P(R) - P(L-1)$ gives the sum of subarray from index L to index R

This is still $O(n^2)$ solution

We can tweak the second condition to get the following condition • $P(L-1) == P(R) - X$

We need to count the number of $P(L-1)$ such that $(L-1) < R$ & $P(L-1) == P(R) - X$. We need to do this in $\log n$ time. Hence we use a frequency map.

The pseudo code is as follows.

```
map<int, int> m;
int count = 0
for (i=0; i < size of P; i++){ //O(nlogn)
    count += m[P[i] - x]; //O(logn)
    insert P[i] or increment P[i]; //O(logn), insert if not present else increment
}
print sum;
```

So the whole code runs in $O(n\log n)$ time

Note:- • If $A = \quad -1 \quad 1 \quad 2 \quad -3 \quad 5$

$P = \boxed{0} \quad -1 \quad 0 \quad 2 \quad -1 \quad 4$

imp as we need to access $P(L-1)$ to get sum of A_i's from R to L

Hence $(\text{size of } A) + 1 == (\text{size of } P)$

• If we are asked to also print the values of L & r we can use a map<int, vector<int>> m where the vector stores all the indices of prefix sum array where ~~the~~ an element is present

* for code refer to M1 W1, STL Application Ideas, STL Application 3, Time stamp: 01:39:33

STL Application 4 - Logic, Code

Q6) We need to build a structure in which we can perform the following operations, each in < O(n) time.

→ insert( )
→ delete( )
→ find_mean( )
→ find_variance( )
→ find_median( )
→ find_mode( )

⟹ a) find_mean() → O(1)

→ maintain "sum", and "count" while insert & delete
→ return (sum / count)

b) find_variance() → O(1)

$$var = \frac{\Sigma(x_i-\mu)^2}{N} = \frac{\Sigma(x_i)^2}{N} + \frac{\Sigma\mu^2}{N} - \frac{\Sigma 2\mu x_i}{N}$$

$$= \frac{\Sigma(x_i)^2}{N} + \mu^2 - 2\mu \frac{\Sigma x_i}{N}^{\mu}$$

$$= \frac{\Sigma(x_i)^2}{N} - \mu^2$$

→ maintain an variable "elem_sq_sum" which is $\Sigma(x_i)^2$ and gets updated on insert & delete
→ return((elem_sq_sum / count) - ((sum / count) * (sum / count)))

c) find_median() → O(logn)

maintain 2 multisets such that,

→ 1 has smaller half elements & the other has larger half elements
→ size of both can differ by 1 at max (to accomodate odd no. of values)

i.e both have equal no. of value. In case of odd no. of values, the smaller half would accomodate one extra value.

→ • if total count is even, then return

$$\frac{\text{largest element of first set} + \text{smallest element of second set}}{2}$$

• if total count is odd, return the largest element of first set as it accomodates the odd ~~even value~~ number of values

d) find_mode() → O(logn)

maintain 1 multimap & 1 set of pairs, such that

1) m[element_value] = frequency
2) n(frequency, element_value)

Update both while inserting & deleting

Return, the second value of last pair.

STL Application 4.5 : Top K Sum

Note:- ~~Ay~~ • Anything declared in global has a '0' default value
• " " " a function " 'garbage' " "

Q7) Create a structure to find top k sum of all the inserted numbers

⇒ Maintain a -ve priority queue, which helps pop the smallest element on inserting new element

If size of priority queue == k+1, pop the top element

Simultaneously maintain a "sum" variable and update it with every push/pop.

Note :-

```
struct name{
 int temp;
 void init (int temp) {
    this→temp = temp; // assigns the passed value to
                         the variable of same name in
                         the structure.
 }

          // OR
 void init (int that){
    temp = that;      // this assignment also works
 }
```

Customizing priority queue to store smallest number on top

```
class prioritize {                    // globally
  public : bool operator () (int &p1, int &p2) {
                return p1 > p2;
        }
};
:
priority-queue<int, vector<int>, prioritize> pq; // calling

                OR

just call :- priority-queue<int, vector<int>, greater<int>> pq;
                                              ↓
                                   inbuilt to sort
                                   in reverse order
```

Note :- Don't subtract from size() unless typecasting to (int)

Q Set Queries - Module 0 - Applied Learning of STL

IMP Task → Add elements to a set and find the smallest integer that is not present in the set which is greater than or equal to ~~to~~ ~~an~~ a number input 'x'

⟹ MEX → minimum excluded number

Logic :- use 2 sets → set 1 for the actual storing
→ set 2 for storing the excluded terms.

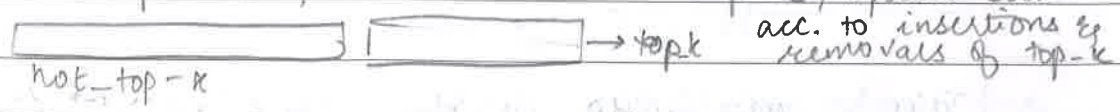
while inserting 'y' in set 1, remove it from set 2. If 'y+1' is not present in set 1, then insert ~~in~~ it in set 2.

for finding MEX with respect to x, first check in set 1, → if it is present, then find the lower bound of x in set 2
→ else MEX wrt x is x

Top-K Sum Continued

Q7) b) Accomodate remove() function as well in Q7. Note that remove is only called if the number is already present

⟹ Use 2 multisets, 1 to store all numbers which are not in top-K & other to store top-k, update sum acc. to insertions & removals of top-k



insert(x) → if x >= (first element of top-k), insert in top-k and remove the first element of top-k & insert it in not-top-k
→ else insert in not-top-k

remove(x) → ~~if x is present in top-k, insert last element of not-top-k in top-k after removing x from top-k~~
→ ~~if~~ else remove from not-top-k

STL Application 5 : Count Covering Interval

Q8) a) We have 'N', the number of intervals and $[L_i, R_i]_{i=1}^{N}$ being the intervals on a number line (x axis) We have 'Q' queries where we need to find $x = y_i$ cuts how many intervals for $i \in 1$ to Q. (end pts included)

Constraints :- $N, Q \leq 10^3$, $L_i, R_i, y_i \leq 10^9$, $L_i < R_i$

$\Longrightarrow$ Brute-force method works for these constrains

Time complexity $\rightarrow O(NQ)$

Q8) b) Same question as above but different constraints

Constraints :- $N, Q \leq 10^5$, $L_i, R_i, y_i \leq 10^9$, $L_i < R_i$

$\Longrightarrow$ Take 2 vector of pairs, sort 1 wrt L_i & 1 wrt R_i. Using lower & upper-bound find

No. of $R_i < y_i$ & No of $L_i > y_i$

$\downarrow$ N_R $\qquad$ $\downarrow$ N_L

N_R + N_L gives the number of intervals that Do not intersect $x = y_i$. we can directly add them as they are disjoint sets as $L_i < R_i$

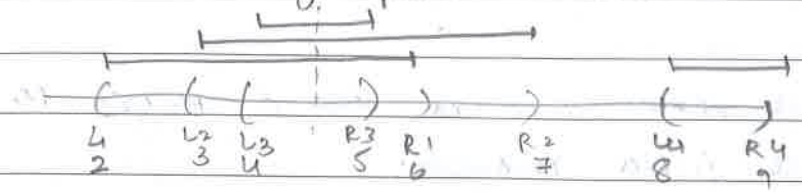
$N - (N_R + N_L)$ gives the no of intervals intersected by $x = y_i$

Time complexity $\rightarrow O(Q \log N)$

Another approach

Take a multimap with key = L_i or R_i & value = 1 if L_i, -1 if R_i

Take a map with key = Li or Ri value is the cumulative sum of previous value in the first multimap



multimap:- 2 3 4 5 6 7 8 9
1 1 1 -1 -1 -1 1 -1

map :- 2 3 4 5 6 7 8 9
1 2 3 2 1 0 1 0

Suppose $y_i = 4.5$
we find the key $\leq y_i$ i.e 4. The value at 4 is 3
hence, $y_i = x$ cuts 3 intervals

Time complexity → $O(Q \log N)$

STL Application 5: Set Range Maintenance

Q9) We are given Q queries of type

IMP

1) Fill Range [L, R] : fills the number line in the mentioned range. (if overlapping with any range then unifies them as one)
2) Clear Range [L, R] : Deletes the range from L to R
3) Check point x : Checks if x lies in the current filled range
4) Check Interval any [X, Y] : checks if [X, Y] contains any point of the filled range
5) Check interval all [x, y] : checks if [x, y] interval is completely inside the filled range

$Q \leq 10^5$
$L, R, X, Y \leq 10^9$

#end pts are included in all queries except clear range

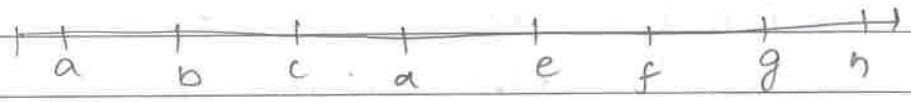
Time Complexity = $O(Q \log Q)$

⟹ ~~fo~~ After each ~~insert~~ Fill range or clear range, we are left with a set of disjoint unique ~~set~~ ranges.

Hence we can use a set of pair to store the filled ranges

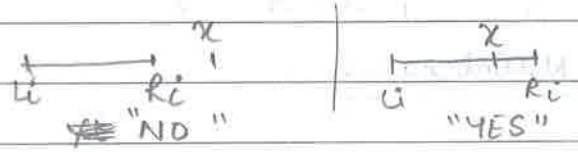
$L_1 \; R_1 \qquad L_2 \; R_2 \qquad L_3 \; R_3 \qquad L_4 \; R_4$

$\{(a, b), (c, d), (e, f), (g, h)\}$



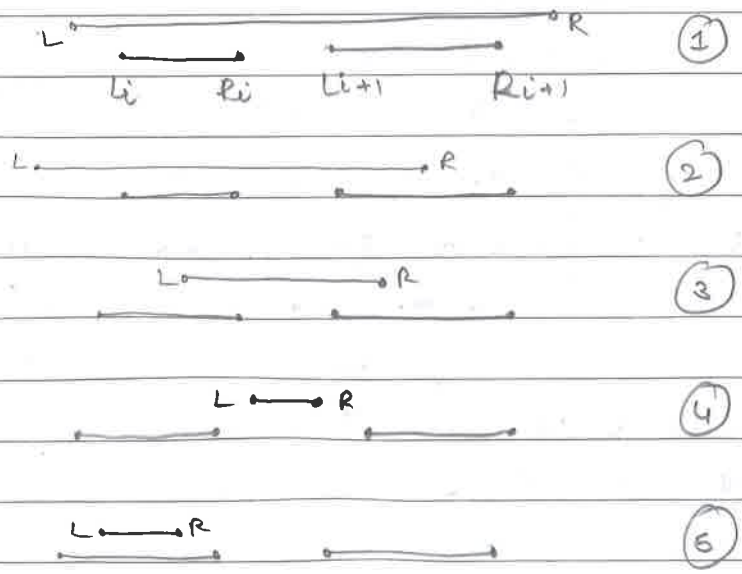
~~For check~~

For Query (3), check point x,
we find the largest $L_i <= x$.
For that L_i, if the corresponding R_i is $>= x$
then x is present in the fill range



Use similar logic for Query (4) & (5)

For Query (1) Fill Range L R
(2) Clear Range L R
we need a code to handle the following case



→ For Query (4) Check any $x\ y$,
We find the smallest $L_i > x$ and ~~ch~~ check if $R_i \geq y$
if yes, return 1. else we move to the preveous Range and check if $x \in [L_{i-1}, R_{i-1}]$
if yes Return 1 else we return 0

→ for Query (5), check all $x\ y$,
~~wf~~ we find largest $L_i <= x$ and check if $R_i >= y$
if yes, return 1 else return 0.

→ For Query (1), find the interval which contains L. Let it be (L_a, R_a). Similarly find the interval that contains R. Let it be (L_b, R_b).

If L is not present in any interval, (L_a, R_a) will be such that smallest $L_a \geq L$ & .
If R is not present in any interval, (L_b, R_b) is such that, largest $R_b \leq R$.

~~Dete~~ Delete all intervals b/w (L_a, R_a) & (L_b, R_b) if $L_a < L_b$, ((L_a, R_a), (L_b, R_b) included)
Insert an interval, $(\min(L, L_a), \max(R, R_b))$.

→ For Query (2)
Find (L_a, R_a), (L_b, R_b) st. $L \in [L_a, R_a]$ & $R \in [L_b, R_b]$.
If L does not belong to a filled range, maintain a check-$L = 1$. in that case (L_a, R_a) is st, smallest $L_a \geq L$.
Similarly maintain check-$R = 1$ if ~~ch~~ R does not belong to filled ~~Regiein~~ Region, in that case (L_b, R_b) is st, largest $R_b \leq R$

If $L_a \leq L_b$, delete all intervals b/w (L_a, R_a) & (L_b, R_b) with both end intervals included

If check_L is 0, insert (La, L)
If check_R is 0, insert (R, Rb)

Alternate check condition
- insert (La, L) if La<L
- insert (R, Rb) if Rb>R

Does not require check_L & check_R

Note :- For set of pairs, to find the interval just after x, we use

st.upper_limit({x, ~~R~~ ie g})

Similarly, we can use the lower_limit function

II Foundational Maths → (Revise)

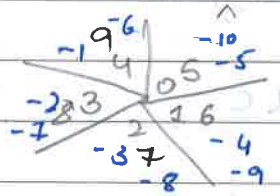
Maths Overviews

Modular Domain Concept

$A \equiv B \pmod{C}$ $\xrightarrow{Eg}$ $7 \equiv 2 \pmod{5}$

"congruent to"

mod 5 divides the integer number line in 5 parts or spaces



(will fill the space in clockwise direction)

Any 2 numbers in the same space are congruent each other wrt the ~~two~~ total number of spaces. This is what $A \equiv B \pmod{C}$ means. It says if total integer number line is divided into C parts, B & A will belong to the same part

Also $A\%C \longrightarrow$ remainder of A/C (integer division)

$\therefore$ $A \equiv B \pmod{C}$ means $A\%C = B\%C$

Note :- • if $A \equiv B \pmod{C}$ then

$\boxed{C \mid (A-B)}$ → C divides $A - B$

as $A = Cq_1 + r_1$, $B = Cq_2 + r_1$, $A - B = (q_1 - q_2)C$

• $\boxed{A = B + kC}$ for some k if $A \equiv B \pmod{C}$

Properties of Modulo function

- $(A+B) \bmod C$
  $= (A \bmod C + B \bmod C) \bmod C$

$A = q_1 C + r_1$, $A+B = (q_1+q_2)C + r_1 + r_2$
$B = q_2 C + r_2$ $\quad A\%C = r_1$, $B\%C = r_2$

- $(A-B) \bmod C = (A \bmod C - B \bmod C) \bmod C$
- $(A * B) \bmod C = ((A \bmod C) * (B \bmod C)) \bmod C$
- $A^B \bmod C = ((A \bmod C)^B) \bmod C$

$A = qC + r$, $A^B = (qC+r)^B = r^B + qC(\,^BC_1 \cdot r^{B-1} + \,^BC_2 qC r^{B-2} + \ldots)$
$(A^B) \bmod C = (r^B) \bmod C = (A \bmod C)^B \bmod C$

Note:-
★ $A \bmod B = (A+B) \bmod B$ → used when $A < 0$

Modulo Arithmetic → (10^9+7) prime

$\left(\frac{A}{B}\right) \bmod C = (A B^{-1}) \bmod C$
inverse with respect to mod.

now, B^{-1} is such that $B \cdot B^{-1} \equiv 1 \bmod C$
or $BX \equiv 1 \bmod C$ if $X = B^{-1} \bmod C$

Fermats little theorem :- let p be prime int.
Then for any $a \in \mathbb{Z}$, $\boxed{p \mid (a^p - a)}$

In $\left(\frac{A}{B}\right)\%C$, we know that C is always prime

now, $C \mid (B^C - B)$ acc. to fermat's little theorem
$= C \mid (B^{C-1} - 1)B$

we know that B is not divisible by C as the question would not make sense then.
Hence $(B^{C-1} - 1)$ is divisible by C

$$B^{C-1} - 1 \equiv 0 \mod C$$
$$B^{C-1} \equiv 1 \mod C$$
$$B B^{C-2} \equiv 1 \mod C$$

hence, $\boxed{B^{-1} = B^{C-2}}$ in the modular domain

Note:- The above only holds when <u>C is prime</u> & <u>$C \nmid B$</u>.

To get the smallest value, we use
$$\boxed{B^{-1} = B^{C-2} \mod C}$$

- $\left(\frac{A}{B}\right)\% C = (A \bmod C * B^{C-2} \bmod C) \bmod C$ | C is prime & $C \nmid B$

Iterative Fast Exponentiation (Binary Exponentiation)

$A^n \longrightarrow$ suppose n is large.
break n into binary form
let $n = 117$, so $n_{(2)} = \overset{6}{1}\overset{5}{1}\overset{4}{1}\overset{3}{0}\overset{2}{1}\overset{1}{0}\overset{0}{1}$
$A^{117} = A^{2^0 + 2^2 + 2^4 + 2^5 + 2^6} = A^{2^0} \cdot A^{2^2} \cdot A^{2^4} \cdot A^{2^5} \cdot A^{2^6}$
We get $A^{2^0} = A \xrightarrow{\text{square}} A^{2^1} = A^2 \xrightarrow{\text{square}} A^{2^2} = A^4 \xrightarrow{\text{square}} A^{2^3} = A^8 \ldots$
Iteratively we calculat $A, A^2, A^4, A^8, A^{16} \ldots$ by recursive squaring. We use the required values to find A^n, $n = 117$
$\boxed{\text{Time complexity} = O(\log n)}$

IMP

Code:-

```
// to calculate (a^b) % m
int binary_exponentiation (int a, int b, int m ) {
    int ans = 1;
    while (b>0) {
        if (b % 2 == 1)
        { ans = ans * a % m; } // while *, use LLI
        a = a * a % m;
        b = b/2;
    }
    return ans;
}
```

- b % 2 & b = b/2 give 1 & 0 to convert b to binary

Eg b=117

117 % 2 = 1 , 117 / 2 = 58
58 % 2 = 0 , 58 / 2 = 29
29 % 2 = 1 , 29 / 2 = 14
14 % 2 = 0 , 14 / 2 = 7
7 % 2 = 1 , 7 / 2 = 3
3 % 2 = 1 , 3 / 2 = 1
1 % 2 = 1 , 1 / 2 = 0

$b_{(2)} = 1110101$

- a = a * a squares after each step to give a, a^2, a^4, a^8 iteratively.

Note:- 1e9 reads as '1' multiplied by 10 power '9'

- Binary exponentiation works even if the base is a matrix or a float number

nCr Calculation

$^nC_r = \frac{n!}{(n-r)!\, r!}$, let $n! = A$, $(n-r)! = B$, $r! = C$

$(^nC_r) \bmod p = \left(\frac{A}{BC}\right) \bmod P = (A\, B^{-1}\, C^{-1}) \bmod P$

calculating A, B, C would take $O(n)$ each
" " B^{-1}, C^{-1} " " $O(\log p)$ "

Total order of the code would be $O(n)$

Suppose we have $T = 10^5$ Test cases with $n, r \leq 10^5$
The above code would not pass as the total time complexity would be $O(n^2)$

→ Idea of cacheing → Calculate $1!, 2! \ldots 10^5!$ and store it in an array and reuse it for every test case. Then we have a code of $O(T \log p)$ which would pass in 1 sec.

if $A \% B < 0$, add B to ans to get an equivalent +ve value. Eg $-2 \% 3 = -2 \rightarrow -2+3 = 1 \rightarrow -2 \% 3 = 1$

10^9+7 → Prime → Inverse always exist for all $a \in \mathbb{Z}$
→ $2 \times (10^9+7) < INT_MAX$, hence $(A+B) \% (10^9+7)$ can be saved as int

$(a \% b) \% c \neq (a \% c) \% b$

General problems faced using modulo
1) Multiplication overflows → $(a * b) \% (10^9+7)$, $a = b = 10^9$, int is not sufficient
2) Negative values → $-2 \% 3 = -2$
Can be handled using $((a \% m) + m) \% m = a \% m$

General Rules while solving modulo problems

1) keep all variables in long long
2) keep simple binary expressions i.e take mod after each operation

$$a = (b + c - d * f) \% e$$
$$= ((b + c) \% e - (d * f) \% e) \% e$$

3) while printing, always use $((a \% b) + b) \% m$ to avoid -ve numbers.

Note For Common solving

→ rbegin(), rend() only works for random iterators

→ we get RE (runtime error) if we try to erase an element from a set that is not present, so find first then erase

→ passing large vectors/arrays into function recursively can cause TLE. In that case, use globally declared ~~to va~~ vectors/arrays to avoid time wasted in sending the array and copying

Module 1 - Week 2

I Debugging Techniques

1. Creating edge cases

↳ Think about constraints

- code might fail in extreme values of constraints (like case 0, 1 or the last cases)
  check overflow & mod property

To run the last test cases, create an input.txt file using another code file. Input it to the main code and run it.

- ./a.exe > in.txt → saves output in in.txt
- ./a.exe < in.txt → takes input from in.txt

- another way to test large test cases is hard coding.

2. Flow checks

Creating test-cases to check all the conditions.
For:- eg we have a code with if → else if → else kind of condition. In that case, we need to create test cases to cater to if, else if, else conditions to check if they are working smoothly

Creating test cases to check every line of code at-least once is known as flow checks

3) Input type

Check if input is of correct type based on constraints

4) Using assert

assert (Statement); // this code gives RTE if it is triggered and the statement is false

assert can be used to check which code block is giving wrong answer.
If assert is hit and statement is false, it results in RTE. So, WA changes to RTE and we get to know the statement & block resulting in this.

5) Stress Testing → Consumes (10-20 mins)
→ Finds test cases that fail (most efficient)

We have → solve() function to solve the question
Write → solve_brute() " " " " "
using bruteforce method but inefficiently acc. to the time
Write → generate() function to generate random inputs

Inputs generated from generate() are sent to solve() & solve_brute() to obtain output 1 & output 2. If both are same, we again repeat the process until we find the case with dissimilar outputs and print it. Use this output and input combination to debug the code.

another way is by using a solve() function, generator() function & a checker() fun^n which checks if for given input the output generated by solve is correct.

→ W1 practice problems

For eg:- In rebuid array code, we can use generator() to build an array, and the using it, we can build the sub-set sum array. The original array is sent to checker() and the subset sum " is " " solve().
Output of solve() is sent to checker to compare with the original array(). If there are discrepencies, print the input and output. Else, repeat until finding error.

• use rand() % N to create numbers from 0 to N-1 randomly

II Prefix & Partial Sums

• to create different seq. every run use srand(time(0)) before rand() [only once in a code]

Prefix Sum Technique

Q) Find answers to Q query on static array A of size N :-
Q LR → Find the sum of elements in the range L to R

⟹ Let $A = A_1, A_2, A_3 \ldots A_N$

Define a prefix sum array P such that

$P_0 = 0$

$P_1 = A_1 = \sum_{i=1}^{1} A_i$

$P_2 = A_1 + A_2 = P_1 + A_2 = \sum_{i=1}^{2} A_i$

$P_3 = A_1 + A_2 + A_3 = P_2 + A_3 = \sum_{i=1}^{3} A_i$

$\vdots$

$P_N = A_1 + A_2 + \ldots + A_N = P_{N-1} + A_N = \sum_{i=1}^{N} A_i$

To find the sum of elements in the range L to R, we return $P_R - P_{L-1}$

Algorithm → $O(N+Q)$

Partial Sum Concept

Q. Given an array A of size N, initially all 0's.
There are Q queries of the form
+ L R X → add X to all the index in the range L to R
After all queries you need to find the final array

→ Step 1:- Build array P = A
Step 2:- For each query + L R X
do P(L) += X , P(R+1) -= X
if (R+1) < N
Step 3:- Print the Prefix sum array of P
↳ Total → O(Q + N)

The above works as P = $A_1, A_2 \dots A_N$
For Query 1, + L_1 R_1 X_1
$P = A_1, A_2, \dots, A_{L_1} + X_1, A_{L_1+1}, \dots, A_{R_1}, A_{R_1+1} - X_1, \dots A_N$
If we build the prefix sum of above, we get
$A_1, A_1 + A_2, \dots, \left(\sum_{i=1}^{L_1} A_i\right) + X_1, \left(\sum_{i=1}^{L_1+1} A_i\right) + X_1, \dots \left(\sum_{i=1}^{R_1} A_i\right) + X_1, \left(\sum_{i=1}^{R_1+1} A_i\right), \dots, \left(\sum_{i=1}^{N} A_i\right)$

So the effect of X_1 stays only from L_1 to R_1
Similarly this ~~reflects~~ works for Q queries.

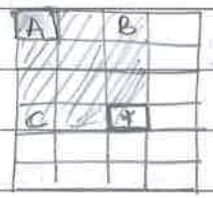
This method only works when we need to print the array in the final step

2D Prefix Sum Concept

Q. Given a 2D matrix Arr[][], with values in each cell.
There are Q queries of the form
? L R U D : Find the sum of values in rectangle formed by
$U \le i \le D$ & $L \le j \le R$ where i is the row index &
j " " column "

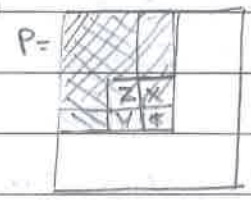
⇒ Using Brute Force → $O(QNM)$
Using 1D Prefix sum for each row & looping over the end pts of rows of the required region → $O(QN+NM)$

Idea of 2D Prefix sums :-



We need to build the 2D Prefix sum such that * has the sum of all values b/w A, B, C & * where A is always starting cell.

Building the 2D Prefix Sum :-
We can build prefix sum as follows



→ Building takes $O(NM)$

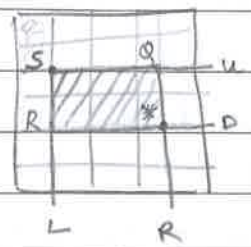
$* = X + Y - Z$

as X stores sum of all elements in ////
Y " " " " " " \\\\
and Z " " " " " " ////

when we add X & Y, Z area is added twice, hence we subtract it once.

Doing the above loop over rows, and in rows loop over cells.

Solving Query ? L R U D from P



//// is LRUD region

Ans = $* - Q - R + S$
as S region is subtracted twice

Total time complexity $= O(Q + NM)$

2D Partial Sum Concept

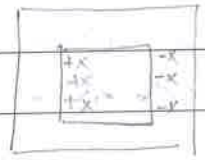
Q. Given a 2D matrix Arr[][], with 0 as initial value in each cell. There are Q queries of the form
+ L R U D X : Add X to the cell values in rectangle formed by $U \le i \le D$ & $L \le j \le R$ where i is the Row index & j is the column index

→ • Brute Force :- O(QNM)

• Using 1D Prefix Sum & 1D Partial Sum :- left cells of the required region, we add X. Right+1 " "
" " " , we subtract X.

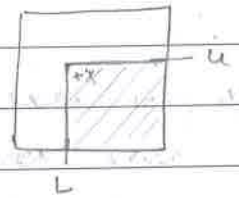
We do this for all queries. At the end, we build row-wise prefix sum.

Time complexity :- O(QN + NM)

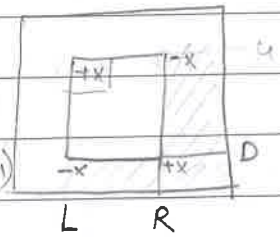


Idea of 2D Partial Sum :-

If we add X to (L,U) cell, it will have an effect on all the cells below & to right when we take the prefix sum



To remove this effect from cells outside LRUD, we subtract X from (U, R+1) and (D+1, L) cells.



Now the bottom corner region has a net value of +X-X-X--X
To remove that, we add X to (D+1, R+1) cell

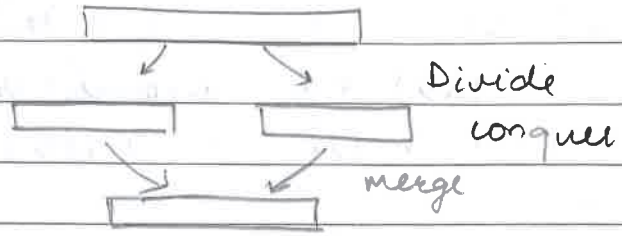
We do the same for all Queries & take 2D prefix sum at the end.

Time Complexity :- O(Q + NM)

III Divide & Conquer

Divide & Conquer Fundamentals

Merge sort →



Time complexity for divided array

$T(n) = 2T(\frac{n}{2}) + M(n)$ → Merge complexity.

Time complexity of Total array

Now $M(n) = O(n)$ as we need to traverse the whole array

using mastertheorem, as $O(n^{\log_2 2}) = O(n) = M(n)$

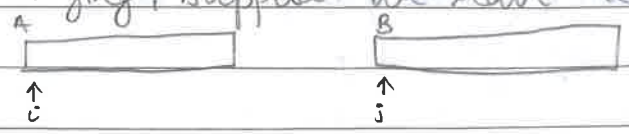
$T(n) = O(n \log n)$

Conquer can be easily coded as it works trivily for the smallest array part, i.e ~~2 elements or~~ 1 element.

~~If only 1 element return the element~~

~~else if $\boxed{a_1, a_2}$ i.e 2 elements, & $a_2 > a_1$ return $\boxed{a_2, a_1}$ else $\boxed{a_1, a_2}$~~

For merging, suppose we have 2 sorted arrays



we need to build a merged C arrays. Algorithm for merging is as follows:-

```
while ( i < size(A) & j < size(B)) { // i, j, k initialized to 0 outside loop
    if( A(i) <= B(j)) { // we use <= to have a stable sort
        C(k) = A(i)
        i++;
    }
```

```
else {
    C(k) = B(j)
    j++;
  }
  k++;
}

while (i < size(A)) // if elements are left in A
{
    C(k) = A(i);
    i++;
    k++;
}

while (j < size(B)) // if elements are left in B
{
    C(k) = B(j);
    k++;
    j++;
}
```

This idea can be extended for more than two arrays as well. In that case, we can merge in pairs to get $\frac{n}{2}$ sorted arrays and recursively merge to get 1 single sorted array.

Difference b/w Recursion & Divide & Conquor (DnC)

→ Recursion :- Has overlapping sub problems i.e. we cannot divide the problem into Disjoint Sub-problems Eg Fibonacci

→ DnC :- Can be divided in to non-overlapping or disjoint subproblems that can be merged easily DnC is a special type of recursive problems.

Karatsuba Multiplication

We need to find $X * Y = Z$. Let X, Y, Z be numbers of base B.

$$X = x_0 + x_1 B + x_2 B^2 + \dots + x_n B^n$$
$$* \; Y = y_0 + y_1 B + y_2 B^2 + \dots + y_n B^n$$
$$Z = z_0 + z_1 B + z_2 B^2 + \dots + z_{2n} B^{2n}$$

- If we use traditional method $\rightarrow O(n^2)$
- If we find, $z_i = \sum_{j=0}^{i} x_j y_{i-j}$, and add the carry over of $z_0, z_1 \dots z_{2n}$ in $O(n)$ time, we have a total time complexity of $O(n^2)$

- We know that addition & subtraction take $O(n)$ time for numbers like X & Y. Carry over can be handeled by traversing once from $(x_0 \pm y_0)$ to $(x_n \pm y_n)$
- even division with traditional method takes $O(n^2)$ time.

Multiplication using divide and conquor..

$X = X_L + B^{(\frac{n}{2})} X_R$, $Y = Y_L + B^{(n/2)} Y_R$

Let $X = x_0 + x_1 B + x_2 B^2 + x_3 B^3 + x_4 B^4$, $n = 4$

similarly Y. Now $n/2 = 2$

$\rightarrow X_L$ = terms of X with B power $< (n/2 = 2)$

$X_L = x_0 + x_1 B$, $X_R = x_2 + x_3 B + x_4 B^2$ (as $B^{n/2}$ is already multiplied

$Y_L = y_0 + y_1 B$, $Y_R = y_2 + y_3 B + y_4 B^2$

$\rightarrow X_R$ = term of X with B power $\geq (n/2 = 2)$ after dividing by $B^{n/2}$

Now,

$$Z = (X_L + B^{(n/2)} X_R) \cdot (Y_L + B^{(n/2)} Y_R) = Z_L + Z_m B^{n/2} + Z_R B^n$$

$Z_L = X_L Y_L$, $Z_R = X_R Y_R$

$Z_m = X_L Y_R + X_R Y_L$

In this way Z can be calculated by dividing recursively

In the above calculation, there are 4 subproblems of $\frac{n}{2}$ multiplication, and adding them would take $O(n)$ time.

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n) = O(n^{\log_2 4}) = O(n^2)$$

But this is the same as before. To reduce time complexity, consider

$Z_L = X_L Y_L$, $Z_R = X_R Y_R$

$$Z_m = X_L Y_R + Y_L X_R = \underbrace{(X_L + X_R)}_{O(n/2)}\underbrace{(Y_L + Y_R)}_{O(n/2)} - \underbrace{Z_L - Z_R}_{O(n/2)}$$

as addition & subtraction take less time than multiplication, their contribution is ignored.

The above has 3 subproblems of $n/2$ multiplication, and combining them would take $O(n)$ time

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n) = O(n^{\log_2 3}) = O(n^{1.585})$$

this is faster than $O(n^2)$

Inversion of Array Concept

Q. We need to find the total number of inversions in an array. An inversion is a case where $i < j$ but $A_i > A_j$.

→ Brute force method → $O(n^2)$

Using modified Merge Sort → $O(n \log n)$

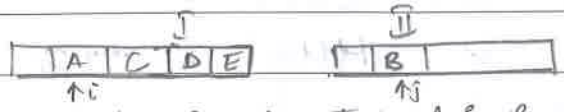
Modified Merge Sort :-

let $A > B$ i.e inverted.

At some point of merge sort, we would have an divided array such that

as we divide the array till only one element is left in an sub array

we ~~then~~ have sorted I & II ~~and~~ which we merge. While merging, we can keep a count of number of inversions



when we encounter A & B cells, B < A but C, D, E > A thus B < A < C < D < E. Hence while merging we add (length(I) − i) to the count of inversions. We do this for all merges and get a total count.

Algorithm:- C is merged array of A & B)

```
i, j, k = 0
while (i < size(A) & j < size(B)){
  if (A(i) <= B(j)){
      C(k++) = A(i++);
  }
  else {
      C(k++) = B(j++)
      inversions += size(A) - i ; // = is handled in if *
  }
}
while (i < size(A)){
    C(k++) = A(i++);
}
while (j < size(B)){
    C(k++) = B(j++);
}
```

// inversions is a global variable, so changes in it are reflected for all the recursions.

~~Parity~~

Note:- No. of inversions gives us the number of swaps required in Bubble sort.

To count number of swaps using Bubble sort $\rightarrow O(n^2)$
" " " " inversion " Merge " $\rightarrow O(n \log n)$

Parity of swaps is a check whether the number of ~~so~~ swaps required is even or odd.

If we find this using Bubble Sort, ~~we~~ Time complexity would be more. Hence we find inversions using Merge sort and check if they are even or odd.

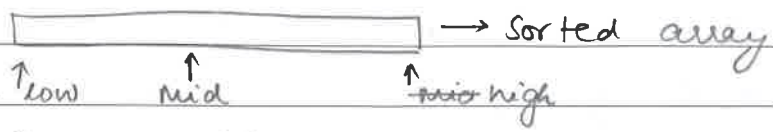
IV Binary Search

Binary Search Fundamentals

Binary search for Array → Seaches if element is present

→ Array needs to be ~~of~~ sorted

Idea :-



If x = mid → stop

x > mid → low = mid+1 → proceed to itr 2

x < mid → high = mid-1 → " " itr 2

Recursively / Iteratively divides array into 2 parts.

Time Complexity : $\boxed{O(\log N)}$

Q1:- We are given an array of 0's & 1's such that all zeros occur before all ones. We need to find the index of first 1

⟹ Use binary search in the following way

Conditions	low_val	mid_val	high_val	next step
(1)	0	0	$\boxed{0}$	stop, no 1 present
(2)	0	0	1	low = mid+1, proceed
(3)	$\boxed{1}$	1	1	stop, return ans = low_pos
(4)	0	1	1	high = mid, proceed

loop while low-pos <= high-pos

If low_pos == high_pos & low_val == 1 return ans = low_pos, stop

" " == " & low_val == 0, stop, 1 not present

Another strategy :- maintain return_pos = -1 (Similar to previous strategy)

loop (while low_pos ≤ high)

{if mid == 0 → low = mid + 1 → proceed

else → return_pos = mid, high = mid - 1 → proceed }

return (return_pos)

If Q1 is changed to find last 0, at the end, we update ans-- 1, and return
If there is no zero, it returns -1 → acc to logic 1

Acc to logic 2, if ans==-1, we return 0,
else we return ans--1

Code for logic 2 → finding the first 1

```
lo = 0, hi = n-1, ans = -1,
while (lo ≤ hi) {
    mid = (lo+hi)/2;
    if (a[mid] == 1)
    {ans = mid
     hi = mid-1; }
    else
    { lo = mid+1; }
}
return ans
```

for finding last 0

```
if (a[mid] == 0)
{ ans = mid
  lo = mid+1; }
else
{ hi = mid-1; }
```

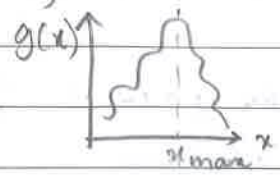
Monotone Functions & Spaces

Monotone Space :- Follows the following condition
→ has ≤ 2 elements
→ has ≤ 1 partions

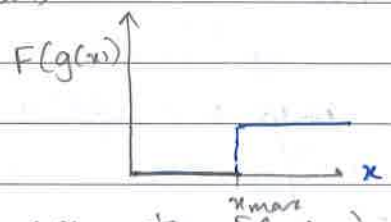
Let F(x) be monotone function
↳ Its ordered range is a monotone space

Using Binary Search on normal Problems

Suppose we have $g(x)$ & we need to find x where $g(x)$ is max



We need to find a monotone function which changes its value at $g(x_{max})$



Applying Binary Search to $F(g(x))$ would return the actual value i.e x_{max}

Eg:- Given a sorted array Arr, we need to find the first element ≥ 12

Arr = 1, 3, 9, 11, 12, 12, 13, 13, 14, 15

→ This problem is similar to 0, 1 problem

monotone_Arr = 0, 0, 0, 0, 1, 1, 1, 1, 1, 1

↳ monotone Array corresponding to Arr

But creating monotone Array would take ~~to~~ $O(N)$ time which is > than $O(\log N)$

Thus we can create a check function, which gives the monotone value for the respective element as we require monotone value for only mid elements. This takes ~~O(N)~~ ~~tin~~ $O(1)$ time per query.

Thus the whole algo remains $O(\log N)$

Q2) Given an rotated array which was originally sorted. We need to output the number of times it was rotated. 1 rotation is moving first element to the end

Eg:- Original array :- 1 2 3 4 6 7

Given array :- 4 6 7 1 2 3

Expected output :- 3 as

rotation 1 : 2 3 4 6 7 1

2 : 3 4 7 6 1 2

3 : 4 6 7 1 2 3

⇒ We observe that there is a partition

4 6 7 } 1 2 3

and final ans is (size of array − pos of (1))

↳ i.e 3

The monotone function for this ~~array~~ question would be return 1 if arr[mid] < arr[0]

else return 0

we get

0, 0, 0, 1, 1, 1

If ~~we~~ there is no 1 in the monotone array we get ans of binary search as −1

in that case,

we need to print (0) as that is the case of no rotations

Bitonic Array

An array that first strictly increases then strictly decreases.

Eg:-

- 1 2 3 4 5 4 3 2 1 → yes
- 1 2 3 → yes
- 9 3 1 → yes
- 5 9 3 2 1 → yes
- 1 2 3 2 1 2 → No

Q3) Find the max element of a bitonic array

→ Monotone Function

return 1 if $a[x]$ is the last element or $a[x] > a[x+1]$
else return 0

we get 0, 0, 0, 0, 1, 1, 1, 1, 1
for Arr = 1, 2, 3, 4, 5, 4, 3, 2, 1

The monotone function works as last element is always 1 either in decreasing only case, or in increasing only case or in ↗↘ case

We apply binary search to find the first 1 in the monotone array and print its position

Time complexity :- $O(\log(\underbrace{hi-low}_{O(N)}) \times \underbrace{(monotone_function)}_{O(1)})$ (per call)

in the above case $= O(\log(N) \times 1) = O(\log(N))$

$O(\log(hi-low))$ mid values are present for each of which we call monotone-function

Thus Total complexity $= O(\log(hi-low) \times monotone_function)$
↳ excluding input and multiple test cases

Binary Search Applications 1

Q4) Painter's ~~Partion~~ Partition Problem (STD Problem)

Given N walls that take $t_1, t_2, \ldots t_n$ time to be painted respectively. We have k painters. A painter can only paint continuous walls. We need to partition the walls among painters such that the total time taken is minimum, if all the painters start painting simultaneously.

Eg:- $\underbrace{2 \quad 7 \quad 1}_{10} \quad \underbrace{8 \quad 3}_{11} \quad \underbrace{4 \quad 5}_{9}$ → 7 walls, 3 painters

total_time = max (10, 11, 9) = 11

required output is partitions such that total_time is 10, as 11 is the minimum possible total_time

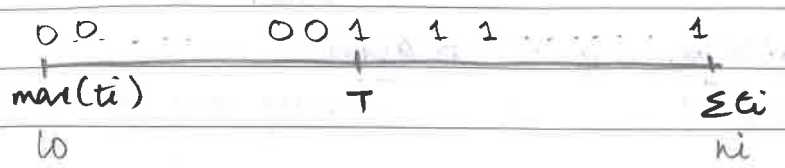
→ We need to minimize the maximum time taken

Minimize the maximum and maximize the minimum hint towards binary search problem

We know that the minimum time would be taken when there are N painters & maximum time would be taken when there is 1 painter.

The required ans T would lie in the range min = max (t_i) & max = Σt_i

We need to create a monotone space as below



The required monotone function is
return 1 if max time per painter ≤ mid for ≤ k painters
else return 0

Basically, in the monotone-function, we have "mid" amt of time per painter. We traverse through the walls and check how many painters are required. If the no. ≤ k, we return yes else no. This is because, if the walls can be painted in max of "mid" time with ≤ k painters, then they lie on the right side of optimum value

eg:- 2 7 1 8 3 4 5 ⟶ 7 walls
k = 3

Let mid = 10

painter 1 :- 2 + 7 + 1 = 10
2 :- 8 = 8 (as 8 + 3 > 10, not allowed)
3 :- 3 + 4 = 9 (as 9 + 5 > 10, "
4 :- 5 = 5 (walls exhausted)

As No of painters (4) > 3, monotone function returns 0

```
int monotone-fun (int mid){
int time_taken = 0, num_of_painters = 1; // time_taken used to store time of each painter so far
for (i from 0 to <N){
    if (time_taken + a[i] > mid){ time_taken = a[i]; num_of_painters++; }
    else { time_taken += a[i]}
}
if num_of painters > k , return 0
else return 1 }
```

```
int main() {
// input the wall array a and calculate
// lo = max(a[i]) , hi = Σ a[i]
int mid, ans; // no need to initiate ans as k ∈ [1, N] so ans always exists
while (lo <= hi) {
    mid = (lo+hi)/2;
    if (monotone-fun(mid) == 1) {
    ans = mid;
    hi = mid-1 }
    else {
        lo = mid+1;
    }
}
cout << ans << "\n";
}
```

Q5) We are given N points on number line $a_1, a_2, a_3 \ldots a_n$. We have to place k points on this number line such that max of neighbor distances is minimized. (Neighbor distance in dist. b/w any 2 consecutive pts) (All numbers in int domain)

→ <u>Method 1</u> :- Using set of pairs

Take out max, divide in ½ and in set

Do this k times → $O(k \log(N+k))$

But this might lead to wrong <u>ans</u>

Eg :- A = 0, 100 , k = 2

itr 1 :- A = 0, 50, 100

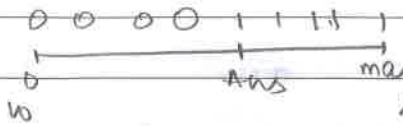
itr 2 :- A = 0, 25, 50, 100 → max neighbor dist = 50

required Ans => A = 0, 33, 66, 100 → " = 34 "

Method 2 :- Using Binary Search

We know, MND (Maximum Neighbor Dist) after inserting k pts would lie in range

$lo = 0$, $hi = \max(a_i - a_{i-1})\Big|_{i=1}^{n-1}$



$\longrightarrow$ required monotone space

monotone function F(mid) : return 1 if all neighbor distances $\leq$ mid with $\leq k$ points placed

as, if MND is $\leq$ mid with $< k$ pts then the mid lies in the 1's region.

```
int monotone-fun (int mid) {
  int pts=0; if (mid==0) return 0;
  for(i=1; i<n ; i++){
   int neigh-dist = a[i] - a[i-1];
   pts += (neigh-dist / mid)        // basically pts += ceil(neigh-dist/mid) -
   if (neigh-dist % mid ==0)
      pts -= 1;
  }
  if (pts > k) return 0;
  else return 1;
}
```

Apply BS on the lo to hi range calling the monotone-fun on each mid.

Time complexity $\rightarrow O(n * \log(\max(a_i - a_{i-1})))$

$\text{ceil}(A/B) = \text{floor}\left(\frac{A+B-1}{B}\right)$

Binary Search Application Idea 3

Q6) We are given 2 arrays A & B of N, M elements respective. Let C be an array of size $N \times M$ whose elements are pairwise sum of elements of A & B. We have to find the kth smallest element of C.

IMP

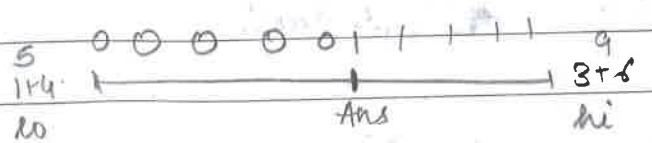
$1 \le N \le 10^6$, $1 \le M \le 10^6$, $1 \le k \le N \times M$, $1 \le a_i, b_i \le 10^4$

⟹ We cannot create the array C and then find the kth element as that would take $O(N \times M)$ time which exceeds the time bound.

Hence we use binary search.
We take A & B input and sort them respectively. We take A st it is the shorter array

Eg:- A = 1 2 3 Let k = 6
B = 4 5 6

We know that the kth smallest element of C would lie in the range



→ required monotone space

monotone function:- F(mid) :- return 1 if the number of C array elements ≤ mid ≥ k
else return 0

The above logic would work if there are multiple instances of ans

Eg:- C = 1 3 3 5 , k = 2

F(1) = 0 as $1 \leq 3$

F(3) = 1 as $1, 3, 3 \leq 3$

F(5) = 1

The above logic would also work for numbers not in C

Eg:- C = 1 2 5 , k = 2

$F(1+5/2) = F(3) = 1$ as $1, 2 \leq 3$

Thus, we need a monotone-function that returns 1 if $\geq k$ elements of C are $\leq$ mid.

Monotone-function:- A = 1 2 3 , k = 3

B = 4 5 6 , mid → input for f^n

```
int num=0
for( i=0; i<size(A); i++){// A is smaller array
    num += distance(B.begin(), upper_limit(mid - A[i]));
}// O(N log M) , N≤M
If num >= k , return 1
else return 0
```

Basically, we traverse over A in O(N) time and using upper_limit, we find the number of elements of B whose sum with A(i) ≤ mid

Overall function → $O(N \log(M) \cdot \log(2 \times 10^4))$

↓

as a_i, b_i have max value of 10^4 and lower limit is 0

can ignore $\log(2 \times 10^4)$ as it is $< \log M$

Time complexity → $O(N \log(M))$

Recorded Application Idea - 4

All the previous questions were Binary Search on Ans. (Till Q (7))
The following BS on each start (Q8)

Q7 Given an array of ~~si~~ size N of 0's & 1's.
we are given k flips: 1 flip is converting 0 to 1 or 1 to 0. We need to find the max length of sub-array made of only 1 after $\leq k$ flips

→ Suppose we have
A = 1 0 0 1 0 1 1 0 1 1

The ans lies in range
0 0 0 1 1 1 1 → required monotone space
1 Ans N

Monotone fun:- F(mid):- Return 1 if 'mid' length sub-array of 1's requires > k flips.
(O(N)) Else return 0

↓

```
monotone-fun(int mid) { // let P be prefix sum of count of zeros
                        // P=0 0 1 2 2 3 3 3 4 4 4
  int flips_req = 0;
  for(int i = 1 ; i <= (n - mid+1); i++) { // sliding window of mid size
    flips_req = P[i+mid-1] - P[i-1];
    if(flips_req <= k)
      return 0;
  } if (flips_req <= k)
      return 0;
  else return 1;
}
```

Binary search on lo = 1, hi = N
Time complexity = O(N log N)

Q8. Find the number of subarrays that we can make all 1's with $\leq k$ flips given an initial array A of size N.

Suppose given array is

1 0 0 1 0 1 0 1 0 0 0 1

and $k = 2$.

We can check each sub-array $\rightarrow O(N^2) \rightarrow$ TLE

Consider any index, let it be "i"

1 0 0 1 0 1 0 1 0 0 0 1

i_1 j_1 i_2 j_2

For every i, there exists a j until which the k flips get exhausted

All the sub arrays b/w i & j are valid & add up to the final ans. But to avoid double counting, caused when i is moved to i+1, we add only the sub-arrays starting from ith index so,

$ans += j - i + 1$.

We observe that for every i, we get a monotone space

1 0 0 1 0 1 0 1 0 0 0 1

i j

0 0 0 0 1 1 1 1

The idea is to find the j using BS on monotone-space for each i and adding up the ans.

~~Monotone fun(int x, int i)~~

We first build the prefix sum of count of 0's

```
          i   j
A =  1 0 0 1 0 1 0 0 0 1
P = 0 0 1 2 2 3 3 4 4 5 6 7 7
            0 0 0 0 0 1 1 1 1 → monotone space
```

Monotone-fun:- Return 0 if zeros b/w x & i (ends included) <=k else return 1

BS → search for the last 0=j → lo = i hi = n-1

```
monotone fun(i, x)
{if P[x] - P[i-1] > k  return 1
 else return 0} //O(1)
```

BS → O(log N) for every index i

Total → O(N log N)

For k=0, → to handle this
we take j = i-1 as default value

Note:-
Q(7) can be solved by taking max of (j - i + 1) for every i

Extra Q) Given an array A of size N containing elements $1 \le a_i \le 10^5$. Find the number of subarray which has ~~at mat~~ max k distinct elements

BS on Real Domain

// epsilon = (req-acc)/1000

```
long double lo=0 , hi=n;
int i=0
while ( abs (hi - lo) >= epsilon && i < k){
    long double mid= (lo+hi)/2;
    if(check(mid))
        hi = mid;
    else
        low= mid;
    i++;
}
ans = (low+ hi)/2;
```

// i is kept to check that the code does not go into TLE

k is decided based on the input n.
if n is of the range 10^9, k would be 40
as in every iteration, we divide n into 2 parts
So, in 40 iterations we would have a part of size
$n/2^{40} \simeq n/(10^3)^4 = n/10^{12}$ as $2^{10} \simeq 10^3$ i.e in 10 iterations, we reduce search by 10^3
So, we get a precision of 10^{-3}. We can increase precision acc. to requirement.

To avoid TLE, we consider n is around the range 10^5, so we can run the code for k = 20 times.

as $O(n \log n)$ requires $n \cong 10^5$ to avoid TLE

Module 1
Week 3 - Mathematics Finale

I Number Theoritic Maths

Factorization Techniques

Q1) Finding if a number is prime or not:
→ we know that if $A * B = N$,
$A = B = \sqrt{N}$ or $A < \sqrt{N}$ & $B > \sqrt{N}$ (as $A * B > N$ if $A, B > \sqrt{N}$)

Hence we know, if N is composite, we have a factor $x \leq \sqrt{N}$, hence we loop from 2 to $\sqrt{N}$ and find if any number is a factor of N
Time complexity → $O(\sqrt{N})$

Q2) Finding all factors of a number
→ Using the same logic as of Q(1), we loop from 1 to $\sqrt{N}$ and find factors $A < \sqrt{N}$ st $A | N$.
Corresponding to a we have $B > \sqrt{N}$ st $B | N$ i.e
$A * B = N$ which can be found as $\left(B = \frac{N}{A}\right)$.
If N is a perfect square $\sqrt{N} = A = B$ is a factor too.
Hence we loop to find all A's & corresponding B's for N
Time Complexity :- $O(\sqrt{N})$

Q3) Finding all the prime factors of N
→ Suppose $N = p_1 * p_2 * p_3 * \ldots * p_n$, according to the logic given ~~by~~ in Q(1), ≤ 1 prime factors are $> \sqrt{N}$, all other prime factors would be $\leq \sqrt{N}$.

Also, ~~all~~ prime factors may have a frequency ≥ 1
Eg :- $12 = 2 \times 2 \times 3$, $2 \rightarrow 2$ (Num, freq), $3 \rightarrow 1$

To address that, we store N in x, and loop from 2 to $\sqrt{N}$

```
x = N;
for( i = 2 to √N ){
if( x % i == 0 && x != 0 ){      // continuously dividing x can cause x=0
                                    resulting in erroneous factors & ∞ loop
    while (x % i == 0){
                                 // stack containing all prime factors
        x /= i;                  // we continuously divide x by i
    } S.push(i);                 ~~to~~ two get the frequency of i
                                 in the prime factorization of N
}

if ( x > 1)
    S.push(x)                    // accomodating pn > √N if it exists
                                 // after continuous division by factors,
                                    x = pn remains
```

Sieve of Eratosthenes

Q4) Finding all primes ~~to~~ till 10^7

Using the method proposed in Q(1), The time complexity would be $O(N\sqrt{N})$ which will result in TLE

We create an array from 2 to 10^7 with all 1's

2	3	4	5	6	7	8	9	....	10^7
1	1	1	1	1	1	1	1	....	1
1	1	0	1	0	1	0	0	....	0

1 is the sign that a number is prime. Initially we consider all numbers as prime. We iterate from 2 to 10^7. As 2 is prime we convert all multiples of 2 to '0's. Next prime is 3, hence all multiples of ~~that~~ 3 are converted to 0's. 4 has '0' so it is not prime (composite). 5 has 1 so it is prime, we repeat the same steps as for 2 & 3.

At the end, the numbers that have 1 left are prime.

```
for (i = 2 to 10^7)
 if (arr[i] == 1){
   for(j = 2×i; j <= N; j += i){
       arr[j] = 0;
   }
}
```

→ for (j = i×i ; j <= N ; j += i)
can be a more optimal loop as the multiples of i which are < i×i are of form p×i < i×i
Now, p < i, hence it would be striked of by a prime < i if p is composite ~~of~~ or by p itself if it is prime

Time complexity :- $O(N \log \log N)$ | for both forms

Note :- • $\left(\frac{N}{2}\right) + \left(\frac{N}{3}\right) + \left(\frac{N}{5}\right) + \left(\frac{N}{7}\right) + \cdots = \sum_{\substack{p \le N \\ p \text{ is prime}}} \left(\frac{N}{p}\right) = O(N \log \log N)$

all floor fractions

• $\left(\frac{N}{1}\right) + \left(\frac{N}{2}\right) + \left(\frac{N}{3}\right) + \cdots + \left(\frac{N}{N}\right) = O(N \log N)$

Segmented Sieve

Q5) Find all the primes in the range [a, b] where $0 < a <= b <= 10^{12}$ & $b - a <= 10^6$

→ Idea 1 :- ~~or~~ Checking every number b/w a & b to be prime individually

Complexity $= O((b-a)\sqrt{b}) = O(10^{12})$ ✗

Idea 2 :- Normal Sieve till 'b'

Complexity $= O(b \log \log b) > O(10^{12})$ ✗

Idea 3:- Segmented sieve

To check if a number 'x' is prime, we only need to " if it is divisible by primes $\leq \sqrt{x}$

As b is the maximum number to ~~be~~ be checked for prime, we need a list of primes $\leq \sqrt{b}$

So we find all the primes from 2 to $\sqrt{b}$ using normal sieve. (PHASE 1)
Complexity $\rightarrow O(\sqrt{b} \log \log \sqrt{b}) \cong O(10^6)$

Once we have the primes from 2 to $\sqrt{b}$, we create a vector from a to b (PHASE 2)

We then convert all the multiples of the primes (except the primes themselves) to 0's.

This gives a vector from a to b with '1's at places of primes.

? Complexity:- $O((b-a) \log \log b)$ as the max prime can be 'b'
↳ as length of array is b-a

Total complexity:- $O((b-a) \log \log b)$ → dominating term

Note:- There are $\left(\frac{N}{\log N}\right)$ primes from 2 to N approximately

Complexity of phase 2:- $\frac{(b-a+1)}{2} + \frac{(b-a+1)}{3} + \frac{(b-a+1)}{5} + \dots = \sum_{\substack{p \leq b \\ p \text{ is prime}}} \frac{(b-a+1)}{p}$

$= O((b-a) \log \log b)$

Fast factorization ($O(N \log N)$)

Q6) Find the prime factorization of all numbers from 1 to 10^6

→ If we use the method in Q(2) → $O(N\sqrt{N}) \cong O(10^9)$ → TLE

For any number N, it can have log(N) number of prime factors at max, as the number of prime factors is maximum when $N = 2 \times 2 \times \dots \times 2$

***Any number N has at max (logN) primefactors**

To harness the property, we build a sieve in a modified way

Take an array from 2 to 10^6 like following

	2	3	4	5	6	7	8	9	10	...	10^6
arr =	2	3	4	5	6	7	8	9	10	...	10^6

Suppose
arr[i] = i → then i is prime

iterate from 2 to 10^6 storing the smallest prime factor of any number, i.e

if x is a multiple i, (i is prime) and a(x) = x (i.e, it has not been visited yet), arr[x] = i

As we are moving in ascending order, the above stores the smallest prime factor of each number. Once a number is visited by its smallest prime number, its index ≠ value, so it doesn't enter the if.

After updation the array would look like so

2	3	4	5	6	7	8	9	10	...
2	3	2	5	2	7	2	3	2	...

To find the prime factor of a number x, we recurse as follows:

~~the~~ prime factor of x = arr[x] + prime factor $\left(\frac{x}{arr[x]}\right)$
(stop when $x = arr[x]$ i.e $\frac{x}{arr[x]} = 1$)

Eg:-

$pf(12) = 2 \times pf\left(\frac{\cancel{12}^{6}}{2}\right) = 2 \times 2 \times pf\left(\frac{\cancel{6}^{3}}{2}\right) = 2 \times 2 \times 3 \times pf\left(\frac{\cancel{3}^{1}}{3}\right)$ stop

$pf(12) = 2 \times 2 \times 3$

Pseudo Code:-

```
for (i from 1 to 10^6)
    arr[i] = i ;

for (i from 2 to 10^6){
  if (arr[i] == i){ // i is prime
      for (j=2×i ; j <=10^6 ; j+=i ){
            if (arr[j] == j)
                arr[j] = i ;     // i is the smallest pf of j
}}
}   // O(N log log N)
// To find Pf of x, call prime_factor function
cout << "1 = 1" ;
for (i from 2 to 10^6)
    cout << i = ;
    prime-factor(i);
    cout << endl;   // O(N log N)

prime-factor (x) {
  if (x == 1)
      return;

cout << arr[x] ;
prime-factor (x/arr[x]);
}
```

Sieve Applications → after fermat's little theorem

Factorization Based Counting

Formula to find number of divisors of x

Suppose $x = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_n^{\alpha_n}$ → prime factorization
where p_i's are prime factors & α_i's are their corresp-
-ondingg powers in the prime factorization of x

Then,

num of divisors of $x = {}^{\alpha_1+1}C_1 \; {}^{\alpha_2+1}C_1 \dots {}^{\alpha_n+1}C_1$

$= (\alpha_1+1)(\alpha_2+1) \dots (\alpha_n+1)$

$$\boxed{S_0 = \sum_{d|x} d^0 = (\alpha_1+1)(\alpha_2+1)\dots(\alpha_n+1) = \text{number of divisors of } x}$$

logic :- for every divisor d of x, we need to choose a power of each prime factor from 0 to α_i respectively

Hence we ~~have~~ have (α_i+1) choices wrt each prime factor.

Formula to find sum of all divisors of x

let $x = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_n^{\alpha_n}$

Now,

sum of divisors $= S_1 = \sum_{d|x} d^1$ can be written as

$$S_1 = (1+p_1+p_1^2+\dots+p_1^{\alpha_1})(1+p_2+p_2^2+\dots+p_2^{\alpha_2})\dots(1+p_n+p_n^2+\dots+p_n^{\alpha_n})$$

as different combinations of prime factors gives a unique divisor of x

we notice that

$1 + p_i + p_i^2 + \dots + p_i^{\alpha_i}$ is a GP and using the formula of sum of ~~g~~ GP $a\frac{(r^n - 1)}{r-1}$ we get

$$1 + p_i + p_i^2 + \dots + p_i^{\alpha_i} = 1\frac{(p_i^{\alpha_i+1} - 1)}{p_i - 1}$$

$$\boxed{\therefore \text{ Sum of all divisors of } x = S_1 = \left(\frac{p_1^{\alpha_1+1} - 1}{p_1 - 1}\right)\left(\frac{p_2^{\alpha_2+1} - 1}{p_2 - 1}\right)\cdots\left(\frac{p_n^{\alpha_n+1} - 1}{p_n - 1}\right)}$$

<u>Formula to find product of all Divisors of x</u>

Deriving the formula using an example

let $x = p_1^2 p_2^3 p_3^1$

We know that some divisors have p_1^0, some have p_1^1 while others have p_1^2

We need to determine the number of times p_1 appears in the product

Let d_{10} be the number of divisors containing p_1^0. Therefore p_1^0 appears in the product d_{10} times.

Similarly p_1^1 appears d_{11} & p_1^2 appears d_{12} times

Now $d_{10} = (3+1)(1+1)$ as $\alpha_2 = 3$ & $\alpha_3 = 1$, so there are $(3+1)(1+1)$ factors with p_1^0

$d_{11} = d_{12} = (3+1)(1+1)$

Contribution of p_1 in product $= (p_1^0)^{(3+1)(1+1)} \times (p_1^1)^{(3+1)(1+1)} \times (p_1^2)^{\cdots}$

$= p_1^{(3+1)(1+1)(0+1+2)} = p_1^{\left(2 \times \frac{(2+1)}{2}\right)(3+1)(1+1)}$

Similarly for p_i

Contribution of $p_i = p_i^{(\alpha_i)\frac{(\alpha_i+1)}{2}\prod_{j\neq i}(\alpha_j+1)} = p_i^{(\alpha_i)\frac{(\alpha_i+1)}{2}\frac{\prod(\alpha_j+1)}{(\alpha_i+1)}}$

$= p_i^{(\alpha_i)\frac{}{2}\,S_0}$

In this case total product is

$\prod$ divisors $= p_1^{2\frac{(2+1)}{2}(3+1)(1+1)} \times p_2^{(3)\frac{(3+1)}{2}(2+1)(1+1)} \times p_3^{(1)\frac{(1+1)}{2}(3+1)(2+1)}$

General formular

$\prod$ divisors $= \prod_{i=1}^{n} p_i^{(\alpha_i)\frac{(S_0)}{2}}$ where S_0 is the sum of divisors

$= \left(\prod p_i^{\alpha_i}\right)^{\frac{S_0}{2}} = x^{\frac{S_0}{2}}$

If S_0 is even $\rightarrow$ x is not a perfect square but S_0 is divisible by 2

so $\boxed{\text{product of divisors} = x^{\left(\frac{S_0}{2}\right)}}$

If S_0 is odd $\rightarrow$ x is a perfect square but S_0 is not divisible by 2

$\boxed{\text{product of divisors} = (\sqrt{x})^{S_0}}$

analogous to sum of AP, finding the middle term and adding it n times

Fermats little Theorem

If p is a prime & a & p are coprime (i.e $p \nmid a$)

then $\boxed{a^{p-1} = 1 \mod p}$ or $\boxed{a^p = a \mod p}$

<u>Application</u> :- find $a^{b^c} \% p$

Let $b^c = m(p-1) + n \Rightarrow b^c \% (p-1) = n$

So

$$a^{b^c} = a^{m(p-1)+n} = a^{m(p-1)} a^n$$

Now

$$a^{m(p-1)} \% P = 1$$

Thus,

$$\boxed{a^{b^c} \% p = a^n \% p = a^{(b^c \% (p-1))} \% p}$$ iff $p \nmid a$ & p is prime

<u>Euler's Theorem</u> - If a & m are coprime, then

$$\boxed{a^b \% m = a^{b \% \phi(m)} \% m}$$

When m is prime, all non-multiples of m are coprime with m & $\phi(m) = (m-1)$

Thus

imp

$a^b \% m = a^{b \% (m-1)} \% m$ when m is prime & $m \nmid a$

↳ Fermat's little theorem

Sieve applications

→ Used to find properties of numbers in range [1 to N] provided the property has some relation with factorization / divisors

Eg:- S_0 & S_1 (No of divisors & Sum of divisors) can be found using previous formulae & fast factorization

Q) Let $F(x)$ = count of numbers a in $[1, x]$ st $\gcd(x, a) = 1$, i.e the count of coprime numbers.

We need to find $F(x)$ $\forall x$ in 1 to 10^6

$F(x)$ → known as totient of a number → $\phi(x)$

⟹

let $x = p_1^2 p_2^3$

All the multiples of p_1 from ~~from~~ 1 to x are NOT coprime to x. The same goes for p_2. All other numbers are coprime to x

$$F(x) = x - \left(\frac{x}{p_1}\right) - \left(\frac{x}{p_2}\right) + \left(\frac{x}{p_1 p_2}\right)$$

→ to handle double counting

↳ Number of values b/w 1 & x divisible by p_1

$$F(x) = x\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)$$

Note that powers of p_1, p_2 make no difference to $F(x)$

General formula

$$\boxed{\phi(x) = x\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_n}\right)}$$

Pseudo-code

```
for i = 1 to N
    a[i] = i ;        // initiate a[i] = i for all i ∈ [1, N)

for i = 2 to N {
    if ( a[i] == i ) {
        for (j = i ; j <= N ; j += i) // j starts from i as φ(i) = i-1
            a[j] -= a[j]/i ;  // recursively updates for all
                                 multiples of i
    }
} // O(N log log N)
```

Totient properties

- $\phi(p) = p-1 \longrightarrow$ p is prime
- $\phi(p^k) = p^k - p^{k-1} \rightarrow p^k\left(1-\frac{1}{p}\right) = p^k - p^{k-1}$
- $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$ if a, b are coprime

Intuitive proof :-

Let a & b be prime

$$\phi(a \cdot b) = a \cdot b\left(1-\frac{1}{a}\right)\left(1-\frac{1}{b}\right) = (a-1)(b-1) = \phi(a)\,\phi(b)$$

Proof :-

Let $x = a \cdot b$, a & b are coprime. Let $x = \overbrace{p_1^{\alpha_1} \times p_2^{\alpha_2} \cdots p_n^{\alpha_n}}^{a} \times \overbrace{q_1^{\beta_1} \times q_2^{\beta_2} \cdots q_m^{\beta_m}}^{b}$

$$\phi(x) = \underset{a \cdot b}{x}\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_n}\right)\left(1-\frac{1}{q_1}\right)\left(1-\frac{1}{q_2}\right)\cdots\left(1-\frac{1}{q_m}\right)$$

$$\phi(a) = a\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_n}\right)$$

$$\phi(b) = b\,a\left(1-\frac{1}{q_1}\right)\left(1-\frac{1}{q_2}\right)\cdots\left(1-\frac{1}{q_m}\right)$$

$$\phi(a)\,\phi(b) = \phi(a \cdot b)$$

$p_i \neq q_j$ for any i, j as a, b are coprime, so they dont have any common prime factors

III Combinatorial Mathematics

Binomial Coefficients

IMP

$$(a+b)^n = {}^nC_0 a^0 b^n + {}^nC_1 a^1 b^{n-1} + {}^nC_2 a^2 b^{n-2} + \dots + {}^nC_k a^k b^{n-k} + \dots + {}^nC_n a^n b^0$$

Binomial Theorem

$$\boxed{{}^nC_k = \frac{n!}{k!\,(n-k)!} = {}^nC_{n-k}}$$

Methods to calculate ${}^nC_k$

1. Factorial Method

→ Used when $n \leq 10^6$ Eg when we want ans % p (p is prime)
store $0!$, $1!$ ... $n!$ in an array

$$\cancel{}\ {}^nC_r = \frac{n!}{r!\,(n-r)!} = \left(\left(n! \times (k!)^{-1}\right) \% p \times (n-r)!^{-1}\right) \% p$$

Basically inverse needs to exist
When p is prime $x^{-1} = x^{p-1}$ w.r.t. %

$O(N + \log p)$ → complexity (total)
$O(\log p)$ → per query
We can also maintain an array of
$0!^{-1}$, $1!^{-1}$ ... $n!^{-1}$, then every query takes $O(1)$ time

2. Pascals Triangle

1											
1	1										
1	2	1									
1	3	3	1								
1	4	6	4	1							

Skewed version

$n=0 \rightarrow$											
$n=1 \rightarrow$											
$n=2 \rightarrow$											

for $\boxed{\sum_{i=0} i^2 \, {}^nC_i}$ divide x again

3. hockey stick formula → ~~to~~ LHS & RHS answers form a hockey stick like structure in pascals triangle

IMP

$$\sum_{m=k}^{n} {}^mC_k = {}^{n+1}C_{k+1}$$

Intuition

Suppose we need to choose $(k+1)$ people from $(n+1)$ people

$\overbrace{\circ \quad \circ \quad \cdots \quad \circ \quad \circ \quad \circ}^{n+1}$

↓ last person included

then we have to choose k people from n people → ${}^nC_k$

+

OR

$\circ \quad \circ \quad \cdots \quad \circ \quad \circ \quad \times$ → last person excluded

↓ second last person included → ${}^{n-1}C_k$

+

OR

$\circ \quad \circ \quad \cdots \quad \circ \quad \times \quad \times$ → last 2 people excluded

↓ 3rd last person included → ${}^{n-2}C_k$

+

⋮

+ ${}^kC_k$

$\overline{{}^{n+1}C_{k+1}}$

4. Vandermonde's Identity

IMP

$$\sum_{i=0}^{k} \binom{n}{i}\binom{m}{k-i} = \binom{n+m}{k}$$

Intuition:- let n = no. of men, m = no. of women

we need to choose total of k people

${}^nC_0 \, {}^mC_k + {}^nC_1 \, {}^mC_{k-1} + {}^nC_2 \, {}^mC_{k+2} + \dots + \dots + {}^nC_k \, {}^mC_0 = {}^{n+m}C_k$

0 men & k women (OR) 1 man & k-1 women (OR) 2 men & k-2 women (OR) ... k men & 0 women

k people out of n+m people

Special Case of Vandemonde's Identity

IMP

$$\sum_{i=0}^{n} \binom{n}{i}^2 = \binom{2n}{n} \longrightarrow \sum_{i=0}^{k} \binom{n}{i}\binom{m}{k-i} = \binom{n+m}{k}$$

put $m = n$, $k = n$

$$\sum_{i=1}^{k} \binom{n}{i}\binom{n}{n-i} = \sum_{i=1}^{k} \binom{n}{i}\binom{n}{i} = \binom{2n}{n}$$

→ No. of Solutions of $\boxed{x_1 + x_2 + \dots + x_n = R}$, $x_i \in [0, R]$, $x_i \in \mathbb{N}$

↳ $\boxed{{}^{n+R-1}C_{n-1}}$

IMP

$\underbrace{\circ \quad \circ \quad \circ \quad \circ \quad \circ}_{R \text{ identical balls}}$

$\underbrace{- \quad - \quad - \quad - \quad - \quad - \quad - \quad -}_{R+n-1 \text{ empty positions}}$

we need to choose $n-1$ partitions from $R+n-1$ positions so that we get a combination of the form

$\underbrace{\circ}_{x_1} \; \underbrace{\circ}_{x_2} \; | \; \underbrace{\circ}_{x_3} \; | \; | \; \underbrace{\circ}_{x_n}$

i.e. The remaining R positions are filled with balls. These partitions divide R among $x_1, x_2 \dots x_n$

Fibonacci from ${}^nC_r$

IMP

$$\sum_{k=0}^{(n/2)} \binom{n-k}{k} = Fib(n+1)$$

1 = Fib(1)
1 = Fib(2)
2 = Fib(3)
3 = Fib(4)

1
1 1
1 2 1
1 3 3 1

IMP

Lucas Theorem

→ recursively use the same formula until n, m are small enough

$$\binom{n}{m} \% p = \binom{\frac{n}{p}}{\frac{m}{p}} \cdot \binom{n \% p}{m \% p} \% p$$

↳ floor function

Used when N is large & p is small prime

Catalan Numbers

Series of the form → 1, 1, 2, 5, 14, 42

C_n → Catalan number n

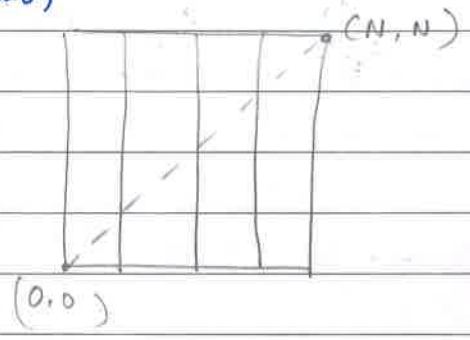
$$\boxed{C_n = \left(\frac{1}{n+1}\right)\binom{2n}{n}}$$ → Formula

$$\boxed{C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}}$$ for $n \geq 0$, $C_0 = 1$

↳ Recurrance

In questions look for 1-1 correspondence with Q1 or look for recurrence relation

Use of Catalan Numbers



Q1 Find the number of paths from (0,0) to (N,N) that lie below the diagonal

IMP

⟹ $\frac{1}{(n+1)}\binom{2n}{n}$ → Proof Not Understood !!

Q We are given '(' & ')' brackets. We need to build a balanced sequence of paranthesis of length 2n. How many such sequences exist?

⟹ This problem can be seen as a type of previous problem where (→ right move
) → left move

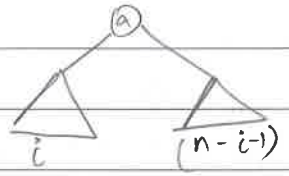
A sequence is unbalanced when number of) increases above the number of (, This corresponds to going above the diagonal.

Thus we want, the total number of paths below the diagonal $\Longrightarrow$ $C_n = \frac{1}{n+1}\binom{2n}{n}$

Q. Find the number of Binary trees with n nodes

$\rightarrow$ let $F(n)$ be the number of binary trees with n-nodes

For a parent node ⓐ, we can have 2 child trees with i, $n-i-1$ nodes each st $0 \le i \le n-1$



$$F(n) = \sum_{i=0}^{n-1} F(i)\,F(n-i-1)$$

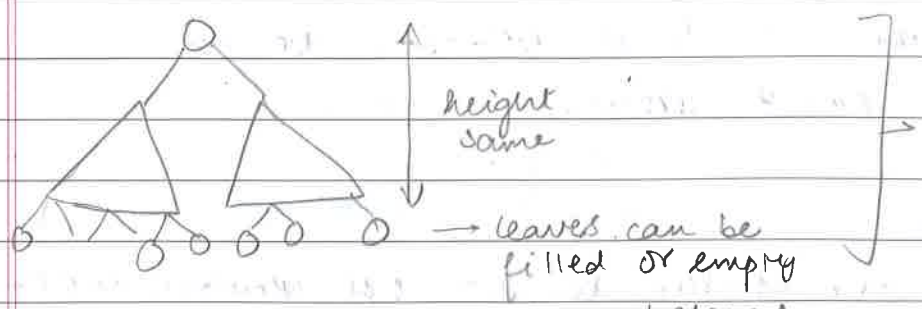
replacing n with n+1

$$F(n+1) = \sum_{i=0}^{n} F(i)\,F(n-i) \rightarrow$$ Catalan Number recurrence relation

Hence $F_n = C_n$

Q. Find the number of Full Binary trees with n+1 leaves (Full Binary Trees means equal height of each child tree for every parent tree) $\rightarrow$ Sol^n C_n

$\Longrightarrow$

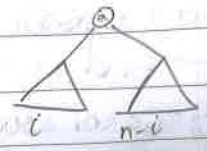


let f_n be the number of balanced binary trees with (n+1) leaf nodes

For a parent node ⓐ, we can have 2 children with 'i' leaf nodes and 'n-i+1' leaf nodes respecively

This 'i' can vary from 0 to n

So we can choose left sub-tree in f_i ways and right subtree in f_{n-i} ways

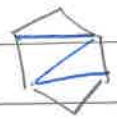


$f_n = \sum_{i=0}^{n} f_i\, f_{n-i+1} \Longrightarrow$ gives C_n like in previous question

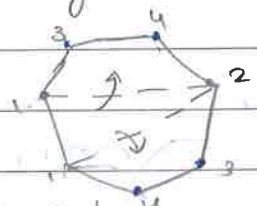
Q. Find the number of triangulation ways for a convex polygon of $(n+2)$ sides

(Triangulation means Drawing non-intersecting diagonals to divide the polygon into triangles)

Eg



When we are given a $n+2$ convex polygon



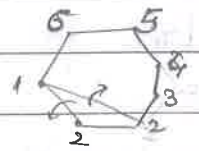
$n+2 = 7 \Rightarrow n = 5$

We can choose a base, and triangulate it at any point. This point can be chosen in $(n+2-2)$ ways

The sides are then considered as bases and we are left with two sub problems

Let $f(n+2)$ be number triangulation ways

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



The minimum points on any side, are 2 and the other side has $(n+2-i+1)$ pts when total pts are $(n+2)$

The formula is similar to catalan relation

Hence $f(2) = C_0, f(3) = C_1, \ldots f(n+2) = C_n$ (as catalan starts from 0)

$$f(n+2) = f_2 f_{n+1} + f_3 f_n + f_4 f_{n-1} + \ldots + f_{n+1} f_2$$
$$= C_0 C_{n-1} + C_1 C_{n-2} + C_2 C_{n-3} + \ldots + C_{n-1} C_0$$
$$= \sum_{i=0}^{n-1} C_0 C_{(n-1)-i} = C_n$$

Q → Find Number of ways to connect $2n$ pts on a circle so that no 2 intersect → C_n

(Solved similar to above problem)

Inclusion Exclusion Techniques

Note:- For combinatorial sequences like Fibonacci refer to oeis.com

We know that

$\underbrace{x+y+z}_{k} = \underbrace{6}_{n}$ → no. of solutions where $x, y, z \in [0,6]$ is ${}^{6+3-1}C_{3-1} \Rightarrow {}^{n+r-1}C_{r-1}$

Q. now, if we have to find the no. of solutions for $x+y+z \leq 6$ st. $x, y, z \in [0,6]$

Sol:- The ans would be $\boxed{{}^{n+r}C_r}$, as we can look at this as

$\underset{x_1}{__} \; \underset{0}{__} \; \underset{1}{__} \; \underset{x_2}{__} \; \underset{1}{__} \; \underset{x_3}{__} \; \underset{1}{__} \; __ \; __ \rightarrow n+r$ empty spaces

↳ need to choose 3 partitions, as last number can be any b/w $[0, 6-x_1-x_2]$

Or we can consider $x+y+z+a=6$, where a takes up the residual value to equate to 6.
hence, there are ${}^{n+k+1-1}C_{r+1-1}$ choices i.e

$\Rightarrow$ $\boxed{\underbrace{x+y+z}_{k} \leq \underbrace{6}_{n} \text{ can be solved with } {}^{n+k}C_r \text{ combinations}}$

Note:- $\boxed{{}^{n}C_r + {}^{n}C_{r-1} = {}^{n+1}C_r}$ → standard result

Q. $x+y+z=10$, $x, y, z \in [2,6]$, $x, y, z \in \mathbb{Z}$, find the number of solutions

→ let $x = a+2$, $y = b+2$, $z = c+2$ as 2 is min value for all

$\Rightarrow a, b, c \in [0, 4]$

$a+b+c = 10-6 = 4$

Total ways $= {}^{4+3-1}C_{3-1} = {}^{6}C_2$ ways

Q. $x+y+w+z \leq 20$, $x \in [3,20]$
$y \in [1,20]$
$w, z \in [0,16]$, $x, y, z, w \in \mathbb{Z}$

Find number of solutions

$\longrightarrow$ let us add the dummy variable :
$x+y+w+z+p=20$, p can adjust acc. to requirements

Let $x = a+3$, $y = b+1$

$\Rightarrow a+b+w+z+p = 20-4 = 16$ $\quad a \in [0,17]$ but acc. to eq^n
$a \in [0,16]$
similarly $b \in [0,16]$ &
$p \in [0,16]$

Total number of solutions $= {}^{16+5-1}C_{5-1} = {}^{20}C_4$

Q. $x+y+z = 8$, $x, y, z \in \mathbb{Z}$, $[0,7]$ find the no. of ways
$\Longrightarrow$ First find the number of ways $x+y+z=8$, $x,y,z \in [0,8]$
$\Rightarrow {}^{8+3-1}C_{3-1} = {}^{10}C_2$

Now, we can choose x, y, z in ${}^3C_1$ ways and make them violate the bound in ${}^1C_1$ ways
Total ans $= {}^{10}C_2 - {}^3C_1 \times {}^1C_1 = {}^{10}C_2 - 3$

Q. $x+y+z=6$, $x, y, z \in \mathbb{Z}$, $[0,4]$ Find the no. of ways
$\Longrightarrow$ when $x, y, z \in [0,6]$, we have ${}^{6+3-1}C_{3-1}$ ways i.e ${}^8C_2$ ways
Consider the case where 1 of ~~the~~ x, y, z is > 4, and other two are bound less
We can choose that 1 in ${}^3C_1$ ways.
Suppose it is x, as it violates the bounds, $x \in [5,6]$
$x+y+z=6 \Rightarrow a+5+y+z=6 \Rightarrow a+y+z=1 \Rightarrow {}^{1+3-1}C_{3-1}$ ways
$= {}^3C_2$ ways

we notice that at max, only 1 element can violate the [0,4] bound at a time. Hence we subtract ${}^3C_1 . {}^3C_2$ from ${}^8C_2$

Total ways :- $\underbrace{{}^8C_2}_{\text{ways in which } \geq 0 \text{ values break bounds}} - \underbrace{{}^3C_1 . {}^3C_2}_{\text{ways in which } \geq 1 \text{ values break bounds}}$

or violations possible (5,0,0) in 3 ways and (5,1,0) in 3×2 ways
$\Rightarrow$ Total 9 ways of violation $= {}^3C_1 . {}^3C_2$

Q.2 $x+y+z+w = 100$, $x, y, z, w \in \mathbb{Z}$, $[0, 40]$

$\Rightarrow$ no. of ways where $x, y, z, w \in [0, 100]$ $\Rightarrow {}^{100+4-1}C_{4-1}$
$= {}^{103}C_3$

Case 1 :-

1 violates the bounds, all others are free
We choose the 1 in ${}^4C_1$ ways.
let it be x. $\Rightarrow x \in [41, 100]$, y, z, w are free
$x+y+z+w = 100 \Rightarrow a+y+z+w = 100-41 = 59 \Rightarrow {}^{59+4-1}C_{4-1} = {}^{62}C_3$
$\hookrightarrow$ Contains double counting of cases where 2 of the ~~bounds~~ variables violate the bounds

Case 2 :-

2 variables violate the bounds, all others are free.
Choose the 2 in ${}^4C_2$ ways
let them be x, y, $\Rightarrow x, y \in [41, 100]$, z, w are free
$x+y+z+w = 100 \Rightarrow a+b+z+w = 100-41-41 = 18 \Rightarrow {}^{18+4-1}C_{4-1} = {}^{21}C_3$

3 variables cannot violate the bounds at a time, as $41 \times 3 > 100$, which is the minimum value

Thus :- Total count $= \underbrace{{}^{103}C_3}_{\text{ways where } \geq 0 \text{ violate the bounds}} - \underbrace{{}^4C_1 \, {}^{62}C_3}_{\text{ways where } \geq 1 \text{ break bounds (has double counting)}} + \underbrace{{}^4C_2 \, {}^{21}C_3}_{\text{where } \geq 2 \text{ break bounds (to handle double counting)}}$

Can also be written as

Total Count $= {}^{100+4-1}C_{4-1} + {}^{4}C_1 \, {}^{100-41+4-1}C_{4-1} + {}^{4}C_2 \, {}^{100-2\times 41+4-1}C_{4-1}$

Generalized formula :- $x_1 + x_2 + \dots + x_r = n$, $x_1, x_2 \dots x_r \in [0, b]$

IMP →

$$\boxed{\sum_{i=0}^{\left\lfloor \frac{n}{b+1} \right\rfloor} {}^{r}C_i \; {}^{n+r-1-i(b+1)}C_{r-1} (-1)^i}$$ → Inclusion Exclusion formula

Q. Find the number of numbers from 1 to 1000 divisible by either 4 or 6 or 9

⟹ A → no. of numbers divisible by 4 $= \left\lfloor \frac{1000}{4} \right\rfloor$

B → " " " " " 6 $= \left\lfloor \frac{1000}{6} \right\rfloor$

C → " " " " " 9 $= \left\lfloor \frac{1000}{9} \right\rfloor$

D → " " " " " 4 & 6 $= \left\lfloor \frac{1000}{\text{lcm}(4,6)} \right\rfloor = \left\lfloor \frac{1000}{12} \right\rfloor$

E → " " " " " 6 & 9 $= \left\lfloor \frac{1000}{\text{lcm}(6,9)} \right\rfloor$

F → " " " " " 9 & 4 $= \left\lfloor \frac{1000}{\text{lcm}(9,4)} \right\rfloor$

G → " " " " " 9 & 4 & 6 $= \left\lfloor \frac{1000}{\text{lcm}(9,4,6)} \right\rfloor$

Ans $= \underbrace{(A+B+C)}_{\text{divisible by } \geq 1} - \underbrace{(D+E+F)}_{\text{divisible by } \geq 2} + (G)$ → divisible by ≥ 3

↳ to handle double counting

$\boxed{\text{lcm}(a,b) = \frac{a \times b}{\gcd(a,b)}}$

Application of $n(A \cup B \cup C) = n(A) + n(B) + n(C) - n(A \cup B) - n(B \cup C) - n(C \cup A) + n(A \cup B \cup C)$

Q. Find the kth number divisible by either 4 or 6 or 9.

⟹ Using Binary search

lo ——— x ——— hi

4 ——— $k \times 4$ (as $k \times 9$ & $k \times 6 > k \times 4$) & from 4 to $k \times 4$ there are k numbers for sure

monotone function:- Return 1 if no. of values divisible by 4 or 6 or 9 that are $\leq x$ is $\geq k$, else return 0

(Use previous Q formula) ⟹ $O(1)$

BS → find the first 1

Time Complexity → $O(\log k)$

Another Method - Find LCM of 4, 6, 9 → 36

Count the number of values divisible by 4 or 6 or 9 which are ≤ 36. let it be p.

1 ——— p ——— 36 ——— p ——— 2×36

The pattern repeats for every 36 values. Thus there are $n \times p$ values divisible by 4 or 6 or 9 which are $\leq n \times 36$

Let $k \geq n * p$ & $\leq n+1(p)$ → window to find the kth divisible number

Not efficient method as LCM's can be huge & time complexity can shoot to $O(k)$

IMP

Q

Ugly Number

Find the kth value which is divisibey by only 2 or 3 or 5

→ Maintain a set S and an output vector v.

Insert in S $2^0 3^0 5^0 = 1$

~~Ans~~ while (length(v) <= k) {

insert s.begin() in v;

remove s.begin() from S;

insert 2×s.begin(), 3×s.begin(), 5×s.begin() in S;

}

cout < v[k] <

The above algorithm first inserts $2^0 3^0 5^0 = 1$ in v

Then it insert 2, 3, 5 in S, it selects the smallest i.e 2 & inserts in v

V = 1, 2, S = 3, 5

now, it inserts 2×2, 3×2, 5×2 in S

S = 3, 4, 5, 6, 10

The algorithm repeats the steps until ~~reach~~ reaching k.

O(k log k) → time complexity

Fibonacci Numbers

→ $F_0 = 0$ $\quad F_1 = 1$ $\quad \boxed{F_n = F_{n-1} + F_{n-2}}$, $n \geq 2$

0, 1, 1, 2, 3, 5, 8 - - - ↳ used to find F_n in $O(n)$ time

→ Approx formula :- $F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$

can lead to decimal rounding error

→ Matrix formula :- $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$

used along with Fast exponentiation to find F_n in $(\log n)$ time

Identities

1) Running Sum Identity → $\sum_{i=0}^{n} F_i = F_{n+2} - 1$

$$\begin{pmatrix} F_0 \\ + \\ F_1 \end{pmatrix} + \begin{pmatrix} F_1 \\ + \\ F_2 \end{pmatrix} + \begin{pmatrix} F_2 \\ + \\ F_3 \end{pmatrix} + \cdots + \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} \rightarrow F_{n+2} - F_1$$

2) Convolution Property → $F_n = F_m \cdot F_{n-m+1} + F_{m-1} F_{n-m}$

3) Sum of squares → $\sum_{i=0}^{n} F_i^2 = F_n F_{n+1}$

4) $F_n \mid F_m \Rightarrow n \mid m$ $\quad , n \geq 2$

5) $\gcd(F_n, F_m) = F_{\gcd(n,m)}$

IV Miscellaneous Techniques

Application Idea 1

IMP

Q1) Find the number of solutions for $\frac{1}{a} + \frac{1}{b} = \frac{1}{n}$
$n \in I - \{0\}, a < n < 10^6, \quad a, b \in I - \{1\}$

$\Longrightarrow \frac{1}{a} + \frac{1}{b} = \frac{1}{n}$

$\Rightarrow \frac{b+a}{ab} = \frac{1}{n}$

$\Rightarrow -na - nb + ab = 0$ adding n^2 on both sides

$\Rightarrow -na - nb + ab + n^2 = n^2$

$\Rightarrow -na + b(a-n) + n^2 = n^2$

$\Rightarrow -n(a-n) + b(a-n) = n^2$

$\Rightarrow \underset{x}{(b-n)}\underset{y}{(a-n)} = \underset{z}{n^2}$

we need to find the factors of z using the iteration till $\sqrt{z}$ method

Suppose di is a factor of z

$di = x = b - n \Rightarrow \boxed{b = di + n}$

$\frac{n^2}{di} = y = a - n \Rightarrow \boxed{a = \frac{n^2}{di} + n}$ → corresponding -ve factors

$b = -(di + n)$

$a = -\left(\frac{n^2}{di} + n\right)$

Application Idea 2

Q) Find $\left\lfloor \frac{N}{1} \right\rfloor^3 + \left\lfloor \frac{N}{2} \right\rfloor^3 + \dots + \left\lfloor \frac{N}{N} \right\rfloor^3 \quad N \leq 10^{12}$

→ Calculating directly → TLE

let $N = 12$ $\quad \frac{12}{1} = 12, \quad \frac{12}{2} = 6, \quad \frac{12}{3} = 4, \quad \frac{12}{4} = 3$

$\frac{12}{5} = 2, \quad \frac{12}{6} = 2$

$\frac{12}{7} = \frac{12}{8} = \frac{12}{9} = \frac{12}{10} = \frac{12}{11} = \frac{12}{12} = 1$

We notice that for any N, $\left\lfloor \frac{N}{x} \right\rfloor$ can take at max $O(\sqrt{N})$ values.

We see that for a range of $x = i$ to j

$\left\lfloor \frac{N}{i} \right\rfloor = \left\lfloor \frac{N}{i+1} \right\rfloor = \dots = \left\lfloor \frac{N}{j} \right\rfloor$, we need to find this j

IMP || We also notice that j is always a factor of N such that $j = N / \left\lfloor \frac{N}{i} \right\rfloor$

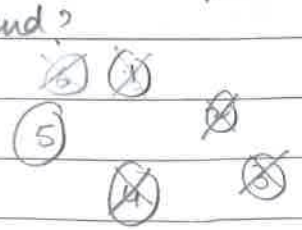
Hence we loop from

~~for (i = 1 ; i <= N ; i += j) {~~
~~j = N / (N/i);~~
~~ans += (N/i)^3 × (j - i + 1),~~
~~} // O(√N)~~

Find factors → 1, 2, 3, 4, 6, 12 // $O(\sqrt{n})$
for (i ∈ factors) {
ans += $(N/i)^3$ × (i - prevfactor)
} // $O(\sqrt{n})$
→ pseudocode

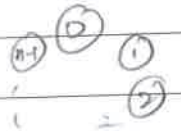
Josephus Problem

We have N people arranged in a circle. Initially the knife is with the first person. He will kill the kth person & pass the knife to $k+1^{th}$ person & this continues. Who stays alive at the end?



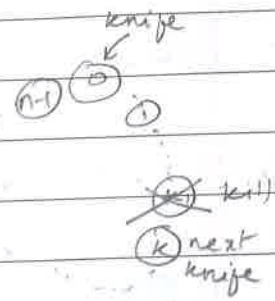
$N = 3$
$k = 2$

→ We need a recurrance. For ~~case~~ case of recurrence we relabel the people from (0) to (n-1)



→ In 0 indexing, (k-1) is killed when 0 has knife

This is done as we need to take (% n) in recurrence as things go round & round



$$\boxed{J(n,k) = (k + J(n-1,k)) \% n}$$

n people and knife is held by '0' — kill's k^{th} person from

gives the last man standing

to handle if the loop is reentered

If k person is seen to be '0' person he kills next k^{th} person among n-1 people

as J(n-1, k) gives ans for knife being held by '0' person, so we add k to bring it to correct indexing

$\boxed{J(1,k) = 0}$ → trivial case

Pseudo Code

```
int ans=0; // trivial case
for (i= 2 to =n)
    ans = (ans+ k) % i;
ans += 1; // to convert to 1 indexing
```

O(N)

Note: There exists a $\boxed{O(K \log N)}$ solution to the above problem which involves recursively deleting all the kth ~~nat~~ values

#Note When $k=2$, represent N in binary.

IMP. Take MSB & put it ~~to~~ after LSB.

you get the ans.

Eg:- $N = 101$, $k = 2$

$101_{(10)} = 1100101_{(2)}$

Ans $= 1001011_{(2)} = 75_{(10)}$

Power of X in Fact [N]

Q. Find largest a st. $X^a \mid N!$ & $X^{a+1} \mid N!$

$\Rightarrow$ Similar to finding the trailing 0's in factorials.

When X is Prime :- maximum power of X in $N!$

$$= \left\lfloor \frac{N}{X} \right\rfloor + \left\lfloor \frac{N}{X^2} \right\rfloor + \left\lfloor \frac{N}{X^3} \right\rfloor + \dots \left\lfloor \frac{N}{X^k} \right\rfloor$$

until $X^k \leq N$

For composite Numbers, $X = P_1^{\alpha_1} P_2^{\alpha_2} \dots P_n^{\alpha_n}$

We find maximum power of $P_1^{\alpha_1}$ in $N!$ $= a_1$, $P_2^{\alpha_2}$ in $N!$ $= a_2$ ... $P_n^{\alpha_n}$ in $N!$ $= a_n$

$$\boxed{a_i = \left\lfloor \left(\left\lfloor \frac{N}{P_i} \right\rfloor + \left\lfloor \frac{N}{P_i^2} \right\rfloor + \dots \left\lfloor \frac{N}{P_i^k} \right\rfloor \right) / \alpha_i \right\rfloor \text{(floor)}}$$

The minimum of these would act as a limiting factor

$$\boxed{\text{ans} = \min(a_1, a_2 \dots a_n)} \rightarrow O(\log X)$$ as X has at max $\log X$ primes

Eg:- $X = 10 = 2 \times 5$, let $N = 36!$

$\boxed{10^8 \mid 36!}$

$a_1 = \frac{36}{5} + \frac{36}{5^2} = 7 + 1 = 8$

$a_2 = \frac{36}{2} + \frac{36}{2^2} + \frac{36}{2^3} + \frac{36}{2^4} + \frac{36}{2^5} = 18 + 9 + 4 + 2 + 1 = 34$

In factorial, as 5' appears less frequently than 2',
5 becomes the limiting factor as there is ≥ 1 2's for every 5

Eg :- $x = 12 = 3 \times 2^2$, $N = 36!$

12^{17}	$36!$

$a_1 = \frac{36}{3} + \frac{36}{3^2} + \frac{36}{27} = 12 + 4 + 1 = 17$

$a_2 = \left(\frac{36}{2} + \frac{36}{2^2} + \dots \frac{36}{2^5}\right) / 2 = 17$

as 2 has power '2'

we see that $3 < 2^2$ but still $a_1 = a_2$ hence, we need to check for all primes and then decide the limiting factor

Note :- For $x = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_n^{\alpha_n}$

$a_1 = \left(\frac{N}{p_1} + \frac{N}{p_1^2} \dots \frac{N}{p_1^{\alpha}}\right) / \alpha_1$ (floor) $\rightarrow O(\log_{p_1} N)$

$\vdots$

$a_n = \left(\frac{N}{p_n} + \frac{N}{p_n^2} \dots \frac{N}{p_n^{\alpha}}\right) / \alpha_n$ (floor) $\rightarrow O(\log_{p_n} N)$

we take $a_2 = \frac{\left(\frac{36}{2} + \dots \frac{36}{2^5}\right)}{2}$ if 2^2

as 2^3 also has 2^2, and this wont be countd if we take $a_2 = \frac{36}{2^2} + \frac{36}{2^4} \dots$

time complexity to find all a_i's :- $O(\log x \log N)$

for every prime, $\log N$ time to calculate a_i

There can be at max $\log x$ number of primes

Time complexity with factorization $= O(\sqrt{x} + \log x \log N + \log x)$

$\sqrt{x}$: for factorization

$\log x$: to find min of a_i

$\boxed{\text{Total time complexity} = O(\sqrt{x} + \log x \log N)}$

as $\log x < \log x \log N$ hence ignored

Two pointers

Week 4

Module 1

I Two pointers

Two pointers fundamentals

Binary Search → BS on Ans → 90%
→ BS on every sort → 10%

out of 10% → 90% can be solved with two pointers
→ 10% BS only

(continuous)

Q) Find the number of subarrays with $\leq k$ 0's
The array contains only 1's & 0's
→ Eg:- 1 00 1 0 1 0 1 1 0
Sol^n using Binary Search → $O(N \log N)$
" " Two Pointers → $O(N)$

Rules to use Two pointers:-
- Every L has an unique R
- For $L' > L$, $R' \geq R$

These conditions are satisfied in above problem.
So, we initiate L at the 0'th index, and move R accordingly, and in the meanwhile, we count the number of subarrays

Pseudo code:-

```
l = 0, r = -1;   // while counting l & r are inclusive, hence
zeroes = 0; ans=0; // to avoid over counting, we initiate r = l-1
while ( l < n) { // using '0' indexing
    while (r+1 < n && (a[r+1]==1 || (a[r+1] == 0 && zeros<k)))
O(R)  {
        r++;
        if (a[r] == 0) zeros ++;   // keeping a count of zeros
      }
O(A) ← ans = r-l+1;   // as from l to r there are r-l+1 subarrays
                         starting from l
```

O(L)

```
if ( l <= r) {
    if (a[l] == 0) zeros--; // as we are moving forward
    l++;
} else { // case 8      r = l-1;
    l++;
    r = l-1;
}
```

Debug:- print l & r for each iteration & check

Two pointers Application 1

```
l=0; r=l-1;
while (l <n) {
    • Move r as far as possible      // O(R) per move of r++
    • process the current window     // O(A) per analysis
    • l++, update accordingly        // O(L) per move of l++
}
```

Total time complexity = $O(N*(R+A+L)) = O(N*(A+L+R))$
$= O(N(L+A+R))$

Two pointers

- Counting problems
- max/min length

For an array question solution is either
- → using Binary Search
- → " Two pointers
- → " Mathematics (like Sieve etc.)

Q) Given an array A of N integers ($A \in [0, 10^9]$, $N \leq 10^5$)
Find the number of sub-arrays with $\leq k$ distinct elements.

```
map<int,int> m; // to maintain the distinct elements & their frequency in the active window
→ l=0; r=-1; ans=0; // size of m gives the no. of distinct elements in active window
while(l<n){
    while(r+1<n && ([a[r+1] is not present in m & size of m <k]
                    or a[r+1] is present in m))
    {
        r++;
        if(a[r] is not present in m)
            insert(a[r],1)
        else a[r]++;
    }
    ans += r-l+1;
    if(l<=r){
        a[l]--;
        if(a[l]==0) remove a[l] from m;
        l++;
    }
    else { // case of r=l-1
        l++;
        r=l-1;
    }
}
//O(N logN)
```

We choose frequency map as the data structure as we need to,
- insert,
- delete,
- find frequency
- check number of distinct elements in the window

(for two pointers condition checking)

In the previous question, "zeros" served as the data structure

We can create 5×10^7 size global array.

Alternate Sol^n :- Maintain a global variable "distinct" which ~~count~~ has the global count of distinct elements in ~~array~~ active window. (only if $1\le A_i\le 10^6$)

Maintain a global "freq" array, which has frequency of elements in active window. "freq" is of size 10^6 initiated to {0}

While adding an element to ~~away~~ active window

```
freq[a[r]]++;
if(freq[a[r]] == 1) distinct++;
```

While removing an element

```
freq[a[l]]--;
if (freq[a[l]] == 0) distinct--;
```

At the end of each test case,
set $freq[a[i]] = 0 \quad \forall\, i \in [0, n-1]$
as running freq of 0 to 10^6 = 0 will result in TLE

$\boxed{\text{The above sol}^{\underline{n}} \text{ is } O(N)}$

This is automatically covered in the loop, as we go till $l<n$, so, all elements are removed from the window. So, we can ignore this part of the code.

Two Pointer Application Idea 2

Q) 3 sum

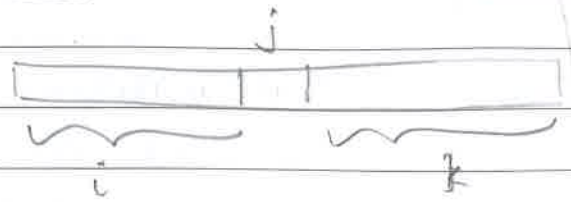
Given an array A of size N, we need to find i, j, k st $0 \le i < j < k \le n-1$ & $A[i] + A[j] + A[k] = T$

→ Brute force method :- $O(N^3)$

3 variants of the above question
(a) Find one solution
(b) " if a " exists
(c) " the number of such solutions
(d) " " " # of unique "

for (d) → use map / set to find uniqueness.
[Can't do without "$n^2 \log n$" TC.] and at ☆ push ans and do i++

$O(N^2)$ Algorithm



We fix 'j' and find 'i' from left & 'k' from right
st. $a[i] + a[j] + a[k] = T$.
We traverse through j from 0 to n-1

So the problem changes to 2 sum as we need to find
$\underbrace{a[i] + a[k]}_{\text{needs to be done in } O(N) \text{ time}} = T - a[j]$

- We first "SORT" the array. 1 — as we are adding i, j, k can be interchanged
- On sorted array we start from $i = 0$ & $k = n-1$
- if $a[i] + a[j] + a[k] = T \longrightarrow$ stop *
  $< T \longrightarrow i{+}{+}$
  $> T \longrightarrow k{-}{-}$

we do till $i = j-1$ & $k = j+1$, then $j{+}{+}$ and repeat.

For (a) & (b) this solution works.
For (c), there is one extra step at *, we dont stop, but
if $a[i] + a[j] + a[k] = T$,
do $i{+}{+}$ until ($a[i] \neq a[i\ original]$ or $i == j-1$)
do $k{-}{-}$ until ($a[k] \neq a[k\ original]$ or $k == j+1$)
$ans += (i\ original - i) \times (k\ original - k)$

to handle multiple cases

Eg:- A = 15 5 5 5 5 1 (indices 0 1 2 3 4 5 6) T = 15

$j = 0 \longrightarrow ans = 0$	$j = 3 \longrightarrow ans = 3 + 2\times2 = 7$
$j = 1 \longrightarrow ans = 0$	$j = 4 \longrightarrow ans = 7 + 3 = 10$
$j = 2 \longrightarrow ans = 1\times3 = 3$	$j = 5 \longrightarrow ans = 10$
	$j = 6 \longrightarrow ans = 10$

Imp Note Q: Remove k integers,

→ x appears in subarrays $(n \times m)$ times

K Odd Number

Q Given an array A of N integers.
We need to find ~~the~~ an subarray with $\leq k$ odd no.
We need to find maximum ~~of~~ sum of such subarrays.
Note that this sum $\leq D$

$A_i \in 10^{-9}$ to 10^9, $D \in 10^{-18}$ to 10^9

$N \in 1$ to 10^5

$1 \leq k < N$

→ As A_i can be negative, sum is not always increasing so we cannot apply 2 pointers ~~to~~ wrt D
But no. of odd numbers is always increasing with increase in window size so we apply 2 pointers wrt k.

2 10 -5 D = 9 — valid
— invalid

So two pointers wrt D is not possible as there is no unique r wrt an l

```
l=0; r=-1; odd=0; // to keep a count of odd numbers in window
ans= INT_MIN;
while (l<n){
    while((r+1)<n && ((a[r+1]%2==1 && odd<k) || a[r+1]%2==0)))
    {
        r++;
        if(a[r]%2==1) odd++;
    } // O(1)
    ans= max(ans, analyze(prefix_sum, l, r)); // prefix sum of array a O(N)
    if(l<=r){
    if(a[l]%2==1) odd--;
    l++;}
    else { l++; r=l-1; } // O(1)
}
```

we check sum of each sub array starting with l

```
int analyze (prefix_sum, l, r){
  int temp=0; int ans= INT_MIN
  if(l) temp= prefix_sum[l-1];
  for(i=l; i<=r; i++){
      if (a[i]-temp ≤ D){
          ans = max(ans, a[i]-temp);
      }
  }
  return ans;
} //O(N)
```

Total time complexity=(O(N²))

Alternate Solution O(N logn)

```
l=0; r=-1; odd=0;
ans= INT_MIN; multiset<int> m; // to maintain the prefix sums
while (l<n){
    while (same as before){
        r++;
        if (a[r] % 2 == 1) odd++;
        m.insert (prefix_sum[r]);
    } // O(log N)
    int temp=0;
    if (l) temp= prefix_sum[l-1];
    auto it =m.upper_bound (D+temp); // we need to find r-temp ≤ D
    if (it != m.begin()) it--;
    ans= max (ans, it* - temp);
    } // O(logn)
  if (l<=r){
    if(a[l] % 2 == 1) odd--;
    m.erase (m.find (a[l]));
    l--; }
  else { same as before };
  //O(logn)
} if ans==INT_MIN cout impossible, else cout ans;
```

II Bit Manipulation

Bit Manipulation Basics

Binary / Bitwise Operators

1) $\wedge \rightarrow$ XOR $\left(\begin{matrix} 10 \rightarrow 1 \\ 01 \rightarrow 1 \end{matrix} \middle| \begin{matrix} 00 \rightarrow 0 \\ 11 \rightarrow 0 \end{matrix} \right)$ $\quad x \wedge x = 1 = \bar{x}$, $x \wedge 0 = x$

Eg:- $5 \wedge 6 \Rightarrow 101 \wedge 110 = 011 = 3$

$12 \wedge 10 \Rightarrow 1100 \wedge 1010 = 0110 = 6$

2) $| \rightarrow$ OR

Eg:- $(12) | (10) \Rightarrow (1100) | (1010) = 1110 = 14$

3) $\& \rightarrow$ AND

Eg:- $12 \& 10 \rightarrow 1100 \& 1010 = 1000 = 8$

4) $<< \rightarrow$ left shift

Eg:- $8 << 2 \Rightarrow 1000 << 2 = 100000 = 32$

5) $>> \rightarrow$ right shift

Eg:- $13 >> 2 \Rightarrow 1101 >> 2 \Rightarrow 0011 = 3$

Note:- $x << y$ is equivalent to $(x * 2^y)$

$x >> y$ " " " $\left\lfloor \frac{x}{2^y} \right\rfloor$

6) $\sim \rightarrow$ negation (1's compliment)

Eg:- $\sim 5 = \sim \underbrace{(00000101)}_{\text{8 bit representation}} = 11111010 = 250$

Note that ~ 5 negates in 32 bits if 5 is an int variable.

If 5 is an long long variable, $\sim$ negates in 64 bits

~~Bitwise operators take (O(1)) cost per bit. It depends on the number of bits in the variable.~~
~~As the number of " " " " is fixed,~~
~~the actual time taken $T = K \times O(1)$. which boils down to O(1).~~ (here $K \leq {}_2\log n$ for any number n)
~~Therefore $T = O(\log n)$. But~~
~~AND Also reason is — parallel computing~~
~~Even though $O(\log n)$ computations take place~~

Bitwise operators take $\boxed{O(1) \text{ time per operation.}}$
Actually they take $O(\log_2 n)$ time" " , as the number of bits in any number $n \leq \log_2 n$, but due to fast parallel computing, the cost of bit operators falls down to $O(1)$

Note:- !, &&, || are logical operators & not bit operators.

Note:- '~' is not actually a bit operator but we can use it as 2's compliment, hence it is somewhat a bit operator only

Eg:- $-6 \Rightarrow \underbrace{0110}_{\text{assume 6 is a 4 bit number}} = +6$

$\sim(0110) + 0001 = 1001 + 0001 = \underbrace{1010}_{\text{in 4 bit representation}} = -6$

↳ one's compliment of +ve number ↳ adding '1' to LSB

$\boxed{\text{\# 1 in MSB indicates a negative number}}$

Using '~' we can subtract in terms of addition

Eg:- $16 - 12 = 16 + (-12) \Rightarrow$

$$\left.\begin{array}{r} 00010000 = 16 \\ +\ 11110100 = (-12) \\ \hline \boxed{1}\ 00000100 \quad 4 \end{array}\right\} \text{in 8 bits}$$

ignore

Eg:- $5 - 12 = 5 + (-12) =$

$$\begin{array}{rl} & 2^7\ 2^6 \ldots 2^1\ 2^0 \\ 00000101 & = 5 \\ 11110100 & = -12 \\ \hline 11111001 & -7 \end{array}$$

MSB = 1 → -ve number

2^s compliment of 11111001 = 00000111 = 7

Hence 11111001 = -7 → 8 bit numbers can range from $(-2^7$ to $2^7-1)$ only

Note:- As the MSB is indicative of -ve numbers. int numbers range from $(-2^{31}$ to $2^{31}-1)$ even though int occupies 32-bits

In unsigned int, MSB is not indicative of -ve numbers and hence they range from $(0$ to $2^{32}-1)$

- $(-x) = (\sim x) + 1$ → depends on the bit representation of x
- Arithmetic operators have higher precedence than bitwise operators
- '-' is actualy an arithmetic operator

$1 << 3 - 1$ → gives ans = 4 as $3-1$ is evaluated first

$(1 << 3) - 1$ → gives ans = 7

For coding tests

* Always put brackets as precedence is confusing
* If there are any MCQ regarding what an output would be, run in the coding terminal & select the respective answer.

Bit operations

$(109)_{(10)} = (0\,1\,1\,0\,1\,1\,0\,1)_{(2)}$

$7\ 6\ 5\ 4\ 3\ 2\ 1\ 0 \longrightarrow$ positioning of Bits

1) set $(x, i) \longrightarrow$ sets i^{th} position bit to '1' in number $x_{(10)}$

$\boxed{x \mid (1 << i)}$

2) unset $(x, i) \rightarrow$ sets i^{th} bit in $x_{(10)}$ to '0'

$\boxed{x \,\&\, (\sim(1 << i))}$

3) Check $(x, i) \rightarrow$ checks if i^{th} bit of $x_{(10)}$ ~~the~~ is '0'

$\boxed{(x >> i) \,\&\, 1} \rightarrow$ method 1 $\rightarrow$ YES when output is 1; No " " is 0

$\boxed{x \,\&\, (1 << i)} \rightarrow$ method 2 $\rightarrow$ YES " " " ≥ 1; NO " " " 0

4) flip $(x, i) \longrightarrow$ flips the ith bit of $x_{(10)}$

$\boxed{x \wedge (1 << i)}$

eg:-

$$\begin{array}{r} 0\,1\,1\,0\,1\,1\,0\,1 \\ \wedge\ 0\,0\,0\,1\,0\,0\,0\,0 \\ \hline 0\,1\,1\,1\,1\,1\,0\,1 \end{array} \quad , \; i = 4$$

$$\begin{array}{r} \wedge\ 0\,1\,1\,0\,1\,1\,0\,1 \\ 0\,0\,0\,0\,1\,0\,0\,0 \\ \hline 0\,1\,1\,0\,0\,1\,0\,1 \end{array} \quad i = 3$$

5) count $(x) \longrightarrow$ counts the number of '1's in $x_{(10)}$

--builtin_popcount (x) $\rightarrow$ for int
--builtin_popcountll (x) $\rightarrow$ for ll

$\Big] \rightarrow$ counts number of 1's in $x_{(10)}$

works when $\boxed{x \geq 0}$ as >> is only defined for ≥0 numbers. and $x/2$ does not work acc. to 2's compliment

Method 1 :- count(x) = count($\frac{x}{2}$) + (x & 1) (recursive)

O(number of bits)

base case (count(0) = 0);

($\frac{x}{2}$ can also be written as x >> 1)

```
while (x)  {                    (iterative)
    ans += (x & 1)
    x >>= 1 }
```

↳ short hand notation

Method 2 :- count(x) = count(x & (x-1)) + 1 (recursive)

O(number of 1's present) base case (count(0) = 0)

→ Method 2 is faster than Method 1 &

does not depend on the sign of the number.

It works for all integers from $-\infty$ to ∞

as it is dependent on bit representation only.

Eg :- (x) = 10100 10000

− 1

(x-1) = 10100 01111

→ (x-1) flips the right most 1 and adds its contribution to the 0 bits after it. This is due to carry over.

(x) & (x-1) = 10100 00000 → removes the contribution of right most 1

→ recurses 3 times for this problem as there are 3 ones

→ we add 1 at every recursion as every recursion remove one '1' from the number

→ Method 2 gives same ans. as builtin_popcount() for all x but Method 1 " " " only for x ≥ 0

Note $x\%2$ can be -1 for $-ve$ numbers, so always take $abs(x\%2)$

while using $(1<<i)$ kind of functions for ~~log~~ long long, type cast 1 to long long, else it would give error

Basics and Masking

Masking :- representing set with a integer

Application on set

Eg:- set $\{1, 5, 4\}$ can be represented by $(1\,1\,0\,0\,1\,0)_{(2)}$
$\quad\quad\quad

Q) set = {2, 5, 7} print all subsets

→ for i = 0 to n-1
    cin >> a[i];

```
for (mask=0; mask < (1<<n); mask++) { // loop from 0 to 2^n-1
    for (i = 0 to n-1) { // checking for on positions from 0 to n-1
        if ((mask >> i) & 1) { // checking if i^th position is on
            cout << a[i] << " ";
    } }
} cout << endl;
```

In bit Manipulation:- finding contribution per each bit is key idea for many problems #

Contribution Technique

Finding ans. using contribution of each element in the ans

Eg:- set = {2, 5, 7}, sum of all sub-set sums?

⇒ $(2+7+5) \times 2^{3-1}$

for any set of size n, ans = (Σ elements) × 2^{n-1}

as element x appears in 2^{n-1} subsets.

x is present for sure, for other n-1 elements there are 2 choises to be present or not to be present, hence 2^{n-1} subsets.

Thus contribution of x ⇒ $x \times 2^{n-1}$

int a[x] gives empty output with no error when x is -ve

avoid the use of pow & sqrt functions. Also, do not use pow function to find fractional powers, it gives error

$\boxed{x \wedge x = 0}$ Eg $\begin{array}{r} 6 = \ 1\ 1\ 0 \\ 6 = \wedge\ 1\ 1\ 0 \\ \hline 0\ 0\ 0 \end{array}$, hence xor gives

1, if there are odd number of 1's at a position, else it gives zero

$\wedge$, |, & are commutative & associative

Also, $\boxed{x \wedge 0 = x}$

NOTES DISCUSSION

Bit Magic

→ $x \& (x-1)$ ⇒ Removes the right most 1

→ $x \& (\sim(x-1))$ ⇒ Gives the right most 1

→ $\wedge$(Array) ⇒ gives 1 at positions where there are odd 1's

→ $x \wedge x = 0$

→ $x \wedge 0 = x$

Note:- Finding xor of 1 to N (MAGIC Pattern)

$1 \ldots \wedge \ldots 8 \longrightarrow 8$	
$1 \ldots \wedge \ldots 7 \longrightarrow 0$	
$1 \ldots \wedge \ldots 6 \longrightarrow 7$	
$1 \ldots \wedge \ldots 5 \longrightarrow 1$	
$1 \wedge 2 \wedge 3 \wedge 4 \longrightarrow 4$	
$1 \wedge 2 \wedge 3 \longrightarrow$	$1 \wedge 2 \wedge 3 \longrightarrow 0$
$1 \wedge 2 \longrightarrow$	$1 \wedge 2 \longrightarrow 3$
$0 \wedge 1 \longrightarrow$	$1 \longrightarrow 1$
$0 \longrightarrow$	$0 \longrightarrow 0$

pattern

	Ans
$n \% 4 == 1 \longrightarrow$	1
$n \% 4 == 2 \longrightarrow$	$n+1$
$n \% 4 == 3 \longrightarrow$	0
$n \% 4 == 0 \longrightarrow$	n

Finding xor from L to R

$1 \wedge 2 \ldots \wedge (L-1) \wedge L \wedge (L+1) \ldots \wedge R = F(R)$

$1 \wedge 2 \ldots \wedge (L-1) \qquad = \wedge F(L-1)$

cancel out (In xor both addition & subtraction like prefix sum is done using xor) $\qquad F(L \text{ to } R)$

Remember :- Try solving BIT-WISE.

classmate
Date
Page

Note || XOR once → to count contribution
" twice → " remove "

XOR can be seen as "addition without carry"

→ Given an array, minimum XOR of pairs would be a_i (XOR) a_{i+1} for some i when a is sorted
Always true only for XOR [NOT for AND]

IMP Question: Total number of set bits from 0 to n
—x—

Application Idea 2 :- Important Question

Q:- Given

Application Idea 2 || Contribution of ith element in subarrays
↳ (# elements till i] × [i to n # of elements)

Important Question

Q:- Given array $A = A_1 \ldots A_N$
Perform the following operation as many number of times as desired

Operation :- $f(A_i, A_j) \Rightarrow A_i := A_i$ OR A_j
$A_j := A_i$ and A_j

Aim :- Maximize $\sum_{i=1}^{N} (A_i)^2$

→ Some imp pts :-

- $a+b = \underbrace{(a\&b)}_{\text{small}} + \underbrace{(a|b)}_{\text{large}}$ → only bits shift in respective positions hence $\sum A_i$ is always constant

- $x+y = c$, c is constant
To maximize x^2+y^2, we minimize (xy) (as $(x+y)^2 = c^2$)
to minimize (xy) → take x, y at extremes ($x=0, y=c$)
" maximize (xy) → " x, y near the middle value ($x=\frac{c}{2}, y=\frac{c}{2}$)

classmate
Date
Page

Module 2 - week 5

BackTrack When No Way out!

I. Brute force & Backtracking

(Video 1) [Brute force & Backtracking formulation]

→ Prune while using Brute force

N-Queens Problem → $(O(N * N!))$ or $(O(N \times N!))$

Terminology :- (for general BackTracking)

- Level :- level (i) → levels (or row) i to N-1 are empty, rest filled
  ↳ current configuration ↳ only in this case
- Choices :- The values ~~assi~~ available to be assigned at a particular level (column numbers in ~~our~~ this case)
- Check :- Out of choices, checking with previous configuration to satisfy the condition of question (here checking for the valid choices of column number such that no previously placed queens attack)
- Move :- 1) Set :- ~~place the~~ make a valid choice for current level
  2) recurse :- move to next level
  3) revert :- after exploring all solutions of the branch, return to the level and restract with another possible choice

Pseudo Code:-

```
recursion(level){
    //Base Case
    for x ∈ choice(level){
        if(check(x))
            move → set(x)
                   → recursion(level+1)
                   → unset(x)
    }
}
```

(# choices for L1 × # choices for L2 × ... × # choices for LN)

$O(\underbrace{\#\text{ of Answers possible}}_{N!} \times \underbrace{\#\text{ of levels}}_{N} \times \text{Avg check complexity})$

negligible

Brute Force Exploration (K-knights problem)

Rem! No. of permutations of (A, A, B, C, A) = $\frac{5!}{3!}$ → as A_1, A_2, A_3 are same

Memory complexity of Brute force codes:- O(Data structure used to maitain the current state + (levels × (parameters size))) recursion

For K-knights problem ⇒ $O(n^2 + (n^2 \times (4+4))) = O(n^2)$

(levels == cell) — board — levels — number of knights parameter saved as int; level number saved as int

Debugging Brute force code:-
1) Print level
2) Print set move
3) Print unset move

Tic-Tac-Toe

level: i^{th} value of 'x' or 'o' to be inserted in the board after given configuration

choices:- all positions of board

check:- if position is free

Configuration 1 → W, W, W } for opp player → losing state for current player

2 → L, L, L } " " " → winning state for curren player

3 → W, L, L } " " " → Winning state

4 → D, L, L } " " " → Draw state

5 → W, P, L } " " " → Winning state for current player

Note :- when $n \leq 13, 19$ or 7 etc
→ Brute Force Possible

Note :- • Given x, to find py such that py is the greatest multiple of y which is $\leq x$.
use:- $\boxed{py = (x/y) * y}$

• Masking → can be used to optimize the check function

Meet in the Middle

→ mainly used when constrants are 2 or more times the number of parable constrains (imp hint)

→ IDEA :- • Divide problem into 2 sub-problems (1 with first half of constraints and second with the other)

Divide
↓
Solve • Solve both using brute force or any other paradigm
↓
Merge • Find merged ans without actually merging (using iteration on one & BS on other for eg)

Tough Questions :-
→ Brute Force :- • Prime Cycles • Prime Palindromes
• K^{th} Permutation - Hard

$n = 35$ → Subset → Meet in Middle

Brute Force ⟶ Meet in middle / DP
Binary search / Two Pointers
Graphs / Segment Tree
Mathematics / Greedy

M2. W6 → Sometimes It is Better to be Greedy!

I. Miscellaneous Techniques

Pigeon-Hole Principle

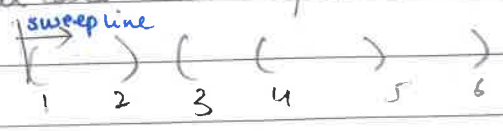
If we insert (N+1) balls in N boxes, atleast 1 box would have 2 balls.

INP Q). Find if there exist a subset/subarray whose sum % N == 0. The total array is of size N. $N \le 10^5$, $A_i \ldots A_N \le 10^9$

Rem:- Sub-set questions are brute-force questions. If the constraints are large, try to find a simpler sub-problem to solve as large constraints DONT pass through Brute-force

Sweep-line Technique → (Internal problems)

idea:- we move through the real line and hit opening & closing brackets of intervals. Doing so, we calculate the required things



Offline Method of Solving:-
- Obtain all queries
- Process them in any order required
- Return acc. to original order

Online Method of Solving:-
- Process ~~out~~ queries in the order of they have been obtained.
- Basically process queries one at a time.

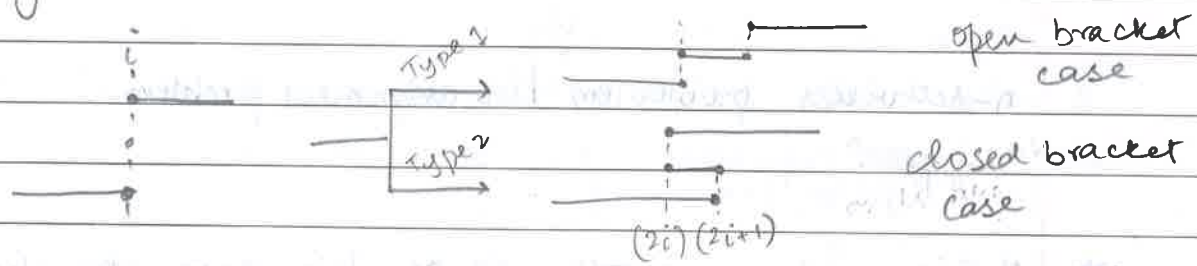
Imp Q :- Pair Product, Solve Intervals 3

Sweep Line hard Problems → (✓✓ imp Questions) ★★

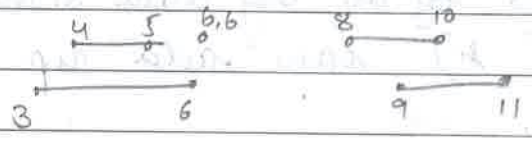
Prefer Brute force over greedy if Brute code passes in the time-constraints.

→ Handling a point interval "(a,a)"

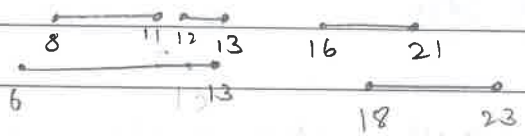
Generally use $(a-\varepsilon, a-\varepsilon)$, in sweepline problems to actually make it an interval



Eg:- Type 2



⇓ converted to



Starting pts → (* 2)
ending pts → (* 2+1)

Ideas based on fractions (Interesting Question)

```
int gcd (int a, int b) {
    if (a==0 or b==0) return a+b;
    return gcd (b, a%b);
}
```

→ Code to find gcd

→ To store fractions, store N^r & D^r in reduced form ($\gcd(N^r, D^r) = 1$)

IMP **Permutation Matching** → Interesting Question

↳ fixed window Approach

↳ keeping the count of values mismatched with permutation

II Greedy Algorithm and Techniques

Techniques generally used in Greedy
- → Sorting
- → Reversal of Approach (Thinking Backward)
- → Splitting Dimension "(2D problems having 1D greedy observation)
- → Multiset / priority queue maintainence
- Classical → Range Based Ideas (starting / ending as fast / late as possible)

n-Activities probleblem / Job assignment problem
→ find jobs that end the earliest

Note:- passing vectors ~~of~~ or big data structure without reference (&) can add up copying them to the total time.
→ Arrays are by default passed by reference

Must knows Reading 1&2 → classical problems

→ Stock Prices Question :- Draw Graph & Visualize

For Greedy Questions → Find all possible Greedy solutions
- Then -
→ Find counter Examples to disprove the ~~too~~ logics to boil ~~to~~ down to 1 logic

Q5 & Q4 → Must knows Reading 2 → IMP

IMP
Q. Joining Ropes → Must knows 1

Q. Find min product of subset of #s ($A_i \in (-1e9, +1e9)$)
↳ Must know! Q4

Q. Given a number, remove k digits without disturbing existing orders to make max number of the remaining
→ Use stack, if a large number is followed by small digit, remove

Greedy Classicals

1) Stock Problem → Solve using graph
2) Job Assignment Problem → Solve using "greedily find the one which ends first" method.
3) Fractional knapsack → Sort acc. to lowest price per unit
Iteratively fill the query

Note:- $\left(\frac{a}{b} < \frac{c}{d}\right)$ function can be written as $(1LL^{*}a^{*}d < 1LL^{*}c^{*}b)$ to avoid floating pt

4) Huffman Encoding → • Iteratively find values with 2 smallest frequency, say v_1, v_2
• make $f_{v_1 v_2}$ → 1 for v_1, → 0 for v_2
• push $f_{v_1 v_2}$ back with frequency as sum of frequencies of v_1 & v_2.
• Keep doing this until only 1 function is left

[Used in problem where complexity = Σ freq × e (where e is encoding length)]

Greedy Ideas & Structuring

Q. Maximize dearrangement in array.

→ Idea:- Rotate array x times, where x is the max. frequency of any element of the array.

→ Sort the array acc. to frequencies, f_1 | f_2 | f_3 ...

→ rotate acc. to f_1 times as f_2, f_3 ... wont overlap in that case

→ $\boxed{\text{If } 2 \times f_1 > n, \ 2f_1 - n \text{ would overlap}}$, else there is no overlap at all.

Ans $= n - \min(0, 2f_1 - n)$

→ for finding the rotated array maintain pair (element, position), Sort acc to frequencies (v_1), rotate f_1 (v_2) times.

$\boxed{\text{New array}[v_1[i].\text{second}] = v_2[i].\text{first}}$ → v_1, v_2

Greedy Live Session 1

Problems which can be solved with other techniques → avoid GREEDY

Swapping Technique :- & Used with Greedy Sorting Questions (or rearrangement)

Eg. N-cards game :- (Aim : To maximize the difference b/w scores)

- Take 2 cards $\boxed{\begin{matrix}A_1\\B_1\end{matrix}}$ $\boxed{\begin{matrix}A_2\\B_2\end{matrix}}$

- Suppose picking A_1 is better picking A_2 for P_1

i.e $A_1 - B_2 > A_2 - B_1 \Rightarrow \boxed{A_1 + B_1 > A_2 + B_2}$

→ Take 2 elements

→ Assume ① > ② & derive the condition acc. to problem (aim of problem)

2D problems → reduce to 1D & think

#Note: $\sum |x_i - x|$ is minimum at Median

$\sum (x_i - x)^2$ is " " Mean

For questions involving ±(min) → try to ITERATE On mins to obtain ans.

Greedy Live Session 2

Finding Top-k in N log k → use priority queue

Ugly Numbers problem (Explained in video) IMP

Important Questions

- Jobs → d_1, d_2 belongs to d_2
- Selling cookies → Rest, Highest Nbr
- Make it smooth
- Palindrome Swaps → Greedily swap exterior elements
- Fibbonacci Breakdown → Highest No. first
- Fast Solving
- Light em up
- Kill Mosters

Module 2 - W7 & W8

DP Demystified

I. Iterative Decomposition of problems

Live Session DP - I

- DP → Brute force + ~~hashing~~ memoization (saving)
  ↳ Generally used in recursive codes
- Iterative Codes :-
  - Time Efficient
  - Space can be optimize
  - Tough to visualize
- Recursive Codes :-
  - Easy to visualize
  - Not space efficient
  - Not time efficient → Can be made using pruning & ~~hashing~~ memoization
- BS problems are generally NOT counting problems/combinatorial probs

II Recursive Decomposition of Problems

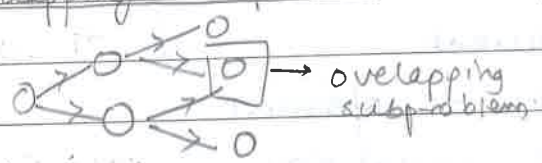
DP Problem Exploration 1

required for DP

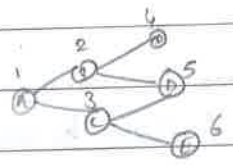
DP ⇒ Brute force + Memoization

Terms :-

1) Overlapping Subproblem → Only ~~at~~ in this DP is used



2) Optimal substructure → The states on which the current state depends on



2 & 3 are optimal substructure of 1

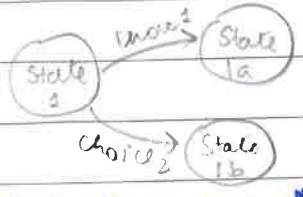
Solving DP :- → Reducing to Brute Force
→ Knowing the Sol^n
→ Modelling into a Form

Note:- Subset creation problems are generally Brute Force/DP. They are generally NOT Greedy

Brute Force Method

→ Step 1:- Write Brute Force
→ Step 2 :- Remove Bulky structure & write as parameters
→ Step 3 :- Make "rec" fn return an essential parameter.
We can no more save the ans as a global parameter
(Return what parameter is asked in the problem)

only Input is global, rest all is local



- State 1a returns ~~what~~ required/best value when choice 1 is taken by State 1
- Similarly 1b

State 1 returns max(choice 1 + State)

State 1 returns :- max/sum/any function (choice 1 cost (+ or x ...) State 1a , choice 2 cost (+ or x ...) State 1b)

→ Step 4 :- • Maintain a DP array (sometimes map) of n dimensions if n is the number of params in recursion.
• The limit for each dimension can be set acc. to the corresponding parameter constraints.

DP is only possible if the recursion parameters are bounded

→ Use memset for setting an array to ~~a~~ a value
memset(arr-name, -1, sizeof(arr_name));

Timecomplexity of DP codes

$$\boxed{O(\#\text{ of states} \times (1 + \underbrace{\text{Avg transition cost per state}}_{\text{Time complexity of Transition part of the code}}) \times \text{Cost of (caching)})}$$

Structure of code

```
return_type DP[param_1][param_2]....;
return_type rec(pram 1, param2 ....){
  // Pruning

  // Base Case

  // Cache Check

  // Choices / Transitions

  // Saving & Returning

}
```

Note :- • params of rec are level & constraint parameters generally
• Returning value is the ans asked in the question generally

DP General Framework

Steps

Recognize → Try to recognize Form (OR) Reduce from Brute Force & Guess the states

State Formulation → DP(state) returns (Meaning / Value) IMP

IMP

↳ Imp & tough to guess

(For DP array, need to consider space constraints)

Transition Design → Current state $\xrightarrow{\text{find}}$ all feasible states

DP(state) = optimization function (DP(next state))

IMP

(max, min, sum etc)

DP(state) needs to be unique → DP(state) needs to have the same value despite of the path taken to reach it

Constraints:-

→ value constraint :- used to calculate Time complexity

Eg $N \leq 10^6$

→ Restrictive Constraint :- Used as "rec" fun^n parameter i.e state

Eg:- $\leq P \leq k$ etc

Time complexity & code → Check if sol^n passes in given time constraint & code else Optimize

DP Exploration 2

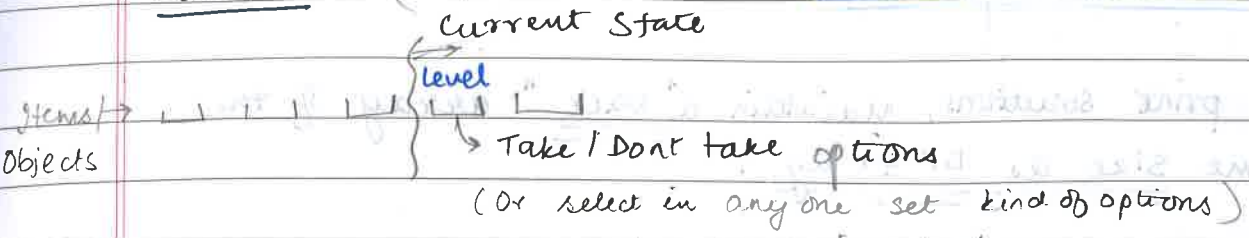
→ Prefix sum & two pointers → mainly for subarray

Bruteforce, DP, Greedy → " " Sub-sets

→ DP Question with Q queries, → Each query adds O(1) time only to the whole code

Total time complexity → O(DP part) + O(Q)

FORM I → (knapsack) [used for sub-set creation type questions]



In such questions with sums involved we can take sum-needed & sum-taken-already, either of these as a state variable

Note :- Take a state variable independent of the query value (Generally prefer sum_needed, as query can be sent in recursion call)

General State :- DP(level, sum_needed) → returns required ans
(or) other restrictions

Note :- Variables in rec fun^n "same" variables in DP array

DP Solution Generation

To generate the sub-sets that satisfy the condition of the question → just Back-Track the DP array

(Check DP sol^n Generation for code)

Pseudo Code

```
vector< _ > Sol
void Generate (state variables) {
    // Base Case

    // Transitions possible
    // push_back into sol vector
    // recursive call
}
```

DP Problem Exploration 3 → Stone Paper Scissor – FORM 1

→ To print solutions, maintain a "back" array of the same size as DP Array.

→ While moving from state a to state b, store the optimal move for state a in the "back".

→ Print Ans should be the last step of code after calculating the req val/result.
↳ Using Back array & generate function (brute-force recursive), trace the optimal states & store the ans.

→ To generate ALL solutions, store all optimal moves in Back vector

Converting Recursive to Iterative Code

→ Write Recursive code → Top-down Approach

→ Find the dependency of current state 'A' to the next state 'B' → B should be present before A in iterative code → Bottom-Up Approach

→ Write loops acc. to " " " & copy-paste recursive code.

→ Save in DP array instead of return

→ Remove cache & prune in transitions

→ Call DP array not rec fn!

DP Exploration Problem 4 – IMPORTANT – FORM 4 (LR DP)

↳ Tough Question

imp

Q. Given convex polygon with N point in clock-wise, find the minimum triangulation cost. Triangulation cost $= \sum |\text{Diagonal added}|^2$

→ Iterative code is different → check

Classical Questions 1) Kadane's Algorithm → Longest Sum contigious Sub-array
2) LCS → Longest Common Sub-sequence

classmate
Date
Page

Forms To Solve Problems

Form 2 & Form 3

FORM 2 → Ending form (generally used in Sub-array sum kind of problems)

DP[level] → asked optimum parameter when we take end at level

Transition

rec[level] = optimum-function(arr[level], rec[level-1] + arr[level], 0)

Note:- • In the compute / transition stage, don't call "DP" array, always call "rec" funⁿ
• Call DP array only in the cache check & saving part

In form 2:- • we generally get 'ans' at any end pt.
• So we have to go through all levels in solve part.

FORM 3 → Multisequence DP / Matching form
(Sub-sequence matching problems)

(2 array set up)

DP[level 1, level 2] → Best match/cost from (level 1 to n) of arr 1 & (level 2 to m) of arr 2

↙ for arr 1 ↓ for arr 2

arr 1 — [level 1 — — ...]
arr 2 — — [level 2 — ...]
→ valid areas

Transition

rec(level 1, level 2) — only if level 1 matches level 2 → cost + rec(l1+1, l2+2)
Dropping l1 → rec(l1+1, l2)
Dropping l2 → rec(l1, l2+1)
↓ optimization funⁿ

Note :- If we have 'k' strings to match, & k may vary, we can use map<vector, int> to cache the recursion & vector<int> as recursion parameter

Form 4 & Form 5

FORM 4 ⟶ LR DP

→ Questions involving maintaining the L & R ptrs of the current active subarray

DP(L, R) ⟶ returns the required optimum value for the subarray from L to R

Note:- Suppose we have a game (symmetric) & players p_1 & p_2

P_1 turn ⟶ takes move x ⟶ recursion goes in DP(x)

$\boxed{P_1 \text{ score} = x + -DP(x)}$ ⟵ returns optimal val for P_2

Debugging DP codes :- ~~Return~~ Print State & its return value

FORM 5 ⟶ Game form

DP(moves left) ⟶ Win / Lose

IMP Note :-

DP(S_1) $\xrightarrow{\text{moves}}$ DP(S_2), DP(S_3), ⋮, DP(S_n)

↓ for P1 ; DP(S_n) ↓ for P_2

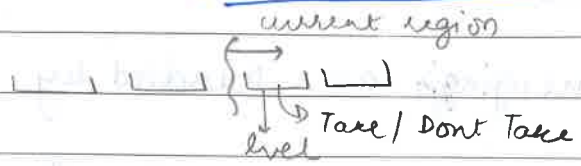
DP(S_2) = win if any of DP(S_2) ... DP(S_n) are losing

DP(S_2) = lose if all of DP(S_2) ... DP(S_n) are ~~losing~~ winning

Note:- In Game Theory we can observe ~~codes~~ patterns & use them to create O(1) code

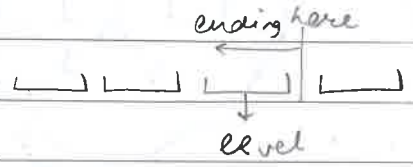
BASIC FORMS

sub-set **Form 1** (Sub-set)



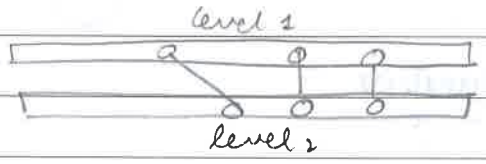
DP(level, restrictions) → ans required

sub-array, sub-seq **Form 2** (Ending) jumping



DP(level) → calculation ending at level

multiple arrays **Form 3** (Matching Multi)

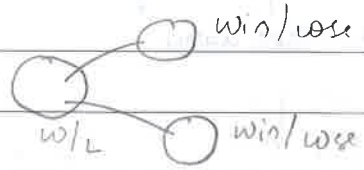


DP(level 1, level 2..) → required count/ans

sub-array **Form 4** (LR-DP) (MCM)

DP(L, R, ...) → optimum ans. in the active region

games **Form 5** (Game)



DP(state/moves) → W/L

IV. Learning More Application Ideas

Mixed Ideas Session I (Tough) (Must know BOTH)

Q. Min cuts to make all substrings palindromes (FORM 2) (classical)

We have to decompose into sub-arrays → Form 2 or Form 4

DP 1 $\xrightarrow{\text{calls}}$ DP 2 → Calling takes O(1) time due to memoization

Hence :- Total time = $O(DP_1 + DP_2)$

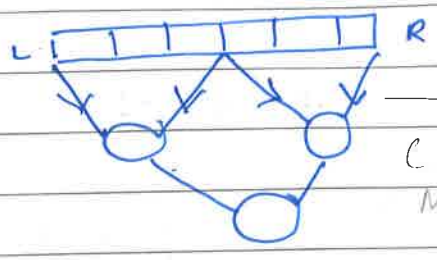
If the transition part takes more time, try to write another DP to cache it

Q2:- Classical (Tough)

Note:- Question of $\boxed{N \le 50 \text{ to } N \le 500}$ (Generally 100, 200, 800) Are of ~~for~~ FORM 4. as FORM 4 generally is of $\boxed{O(N^2) \text{ or } O(N^3)}$

→ Sub-array mergings are handled by FORM 4

For LR Merging Problems:- (See from the scope of last merge)



→ Classical MCM (Matrix Chain Multiplication)

Recognizing a DP Question

→ For a question check for all possible paradigm
→ Try to formulate for the other paradigms along with DP
→ Do TL check for all paradigms and conclude
↳ Constraints are a give away

→ B.S, Greedy, DP, Graph, Brute Force (Mainly) — obvious
→ Maths, 2 ptr (Rarely) — obvious

Mixed Ideas Session-2

Q. Banned Substring (Counting) (FORM 1) → If transforming then Graph
Hint → Use Finite State Machine Diagrams to check for moves banned/available

Q. Given a string, find no. of distinct sub-sequences (FORM 2) IMP

→

	a	b	a	b
ε	a	b	ax	bx
1		ab	aa	abx
	1	2	aba	bb
			ba	abb
			3	5
			= 1+2	= 2+3

no. of distinct sub-seq ending at $i = \sum_{j=k}^{i-1}$ distinct sub-seq " " j

where k = (last occ of ith char) − 1

(iterative)

Q. Finding distinct sub-strings → $O(N^3 \log N)$ solution
(For weak test cases, repeated " len ≤ 100)
↳ Maintaining Map

Mixed Ideas Session - 3 → At coder problems

Counting Problems

Q. Given N, find number of ways to decompose N (FORM I)

→ Eg 4:- 4 = 1+1+1+1, 2+2 etc

{1, 2, 3, 4} DP(level, sum-req) → ways of decomposition

0 1 2 3

Position beyond which vals can be taken

DP(0,4) = DP(0,3) + DP(1,2) + DP(2,1) + DP(3,0) → O(N³)

sum req

DP(level+1, sum_req) + DP(level, sum_req - level) → O(N²)

⟹ B, C, I At coder Questions

Q. Find LIS (Longest Increasing Subsequence) → IMP Q

Pb FORM 2 → (N²) LIS

⟹ O(N log N) idea

Printing sol? ??

vector [ ] → ending value (e1, L1) (e2, L2)

index → length

when e1 == e2, greater L is preferred

" L1 == L2, smaller e is "

Hence vector always sorted.

Find e < current e, update the next length with e

[≤ for L Non-Decreasing S]

Twist Q. Find LCS with Arrays having all distinct element

⟹ O(N log N) soln

IMP

A → 1 3 2 4 6 X → pos → postions of B which have same elements as the pos of A

B → 1 2 3 4 5 6 → pos

As B is sorted, from A, find the LIS → O(N log N)

Grid Based Application 1 & 2

Q. No of paths from (1,1) to (N, M) → with walls / with points (find path with max pts)

→ Generally FORM 2

Grid Based Application 3 (IMPORTANT) Classical

Q. Given grid of 1's & 0's, Find max Area of Sq containing only 1's.

→ FORM 2

C	B
D	A

DP(A) = 1 + min(DP(B), DP(C), DP(D))

Cherry Pickup Problem

Grid Based Application 4 (IMPORTANT - Classical)

Q. Given a grid with weights in each cell, we need to maximize the path weight of a round trip from (1,1) to (N,N). Note:- revisiting the same cell wont add to the path weight.

→ Wrong Approaches:- Finding Optimum path from (1,1) to (N,N), Setting its weights to 0 & repeating

Reason:- There might be multiple optimum paths.

FORM 1

Solution:- Consider 2 persons simultaneously moving from (1,1) to (N,N) & if they meet only one weight is considered

$DP(\underbrace{x_1, y_1}_{P_1}, \underbrace{x_2, y_2}_{P_2})$ → moves / Transitions
- → P_1 down P_2 right
- → P_1 down P_2 down
- → P_1 right P_2 right
- → P_1 right P_2 down

meaning → max of Transitions $+ A[x_1][y_1] + A[x_2][y_2]$ (counted only once if $x_1 = x_2$, $y_1 = y_2$)

Observation:- To reach the same spot, both need same number of moves, hence they would reach ~~the~~ at same cell at same level if they ever meet

DP(State) → max path weights removing double counting while both ~~ra~~ reach (N,M)

Note:- Refrain from using INT_MAX / INT_MIN / LONG_MAX / LONG_MIN in DP as ~~it~~ when something is added / sub, it might become a V. small / large No.

State Space Reduction (Alert → Not for all problems)

If there is a relation b/w state variables which is always true, we can "DROP" variables till we are left with an "INDEPEDENT" state ($x_1 + y_1 = x_2 + y_2$ → redundant)

Note :- If there are k people :- • Use Mask to ~~simulate~~ go through all possibilities

Trouble Shooting DP Issues

→ **WA Error** :- • Print state & ans (even in base case) → Narrow Down search
• If for two runs, the same rec gives different values → states is not unique
• Check the states that current state depends on are called or not.

→ **Base Case** :- Use ∞, −∞, 0 or 1 generally (take correct meaning)

→ **TLE** :- Optimize

Making rec independent of test case :- Take all restrictions as state variables

→ **Recogizing a Question** :-
• Bit manipulation :- ^, & | in Q
• Two Pointers :- Similar to Q on platform
• Graph, linked list, BST :- One type only

Q → Implementation → STL only
Q → Counting → Maths / DP
Counting → small constraint → Brute force
Q → Optimize → Greedy
Optimize → DP/Brute
Optimize → BS (generally if N log N solⁿ req)

check using constraints:
Greedy
Brute force
DP
Binary Search
Maths

→ **Overcounting** :- DP(state) → State a, State b
if Disjoint → no overcounting
else → overcounting

IMP

Maximum K-Subarrays Sum — IMPORTANT (FORM 2)

LIS Code, Solution Generation and Variants

Code:- imp

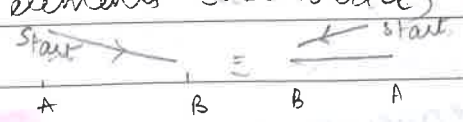
important code

```
lis ( vector <int> arr) {
    vector<int> temp;
    for (auto v: arr) {
        if (temp.empty() || temp.back() < v) {
            temp.push_back(v);
        }
        else {
            auto it = lower_bound(temp.begin, temp.end, v);
            *it = v;
        }
    }
    return temp.size();
}
```

Printing lis:- maintain Positions of array values inserted in temp array. In reverse order find the first position having lis-1, lis-2 .... & so on.

For LNDS:- we use upper-bound (as we can have >1 same elements back to back)
& temp.back ≤ v

For LDS:- = rev(LIS(rev(Arr)))



For LNIS:- = rev (LNDS(rev (Array)))

Longest Bitonic Sub-sequence → FORM 2 → $O(N^2)$

$O(N \log N)$

$\max_{i=0 \text{ to } N} (\text{LIS till } (i) + \text{LDS from } (i) - 1)$

↳ due to double counting

Save LIS & LDS for every index & apply the above formula

Alternate :-

$O(N^2)$ → Form 2 :-

$$\text{ans} = \max(\text{all}) \begin{cases} \underset{\text{inc}}{DP(i,0)} \longrightarrow \max_{\substack{j<i \\ a[j]<a[i]}} (DP(j,0)+1) \\ \underset{\text{dec}}{DP(i,1)} \longrightarrow \max_{\substack{j<i \\ a[j]>a[i]}} (DP(j,0)+1,\ DP(j,1)) \end{cases}$$

Imp

Number of LIS → FORM 2

$DP(i, \text{size}) \longrightarrow$ No. of LIS ending at i with length "size"

↓ length of LIS ending at i

$O(N^3)$

$$DP(i, \text{size}) = \sum_{\substack{j<i \\ a[j]<a[i]}} DP(j, \text{size}-1)$$

Loop over size = 1 to n for all i to get No of LIS

— x —

$O(N^2)$ code :- Idea for any i, only LIS with max-size matter

Hence

$DP(i) \xrightarrow{\text{meaning}} \{\text{max-size}, \text{No of LIS with max-size}\}$ (a, b)

$$DP(i) \overset{\text{Transition}}{=} \sum_{\substack{j<i \\ a[j]<a[i]}} b_j \text{ if } a_j == \max \text{ of all } a \text{ for } j<i$$

↳ loop from for $j<i$, $a[j]<a[i]$ to get maximum 'a' = amax.

loop again to set

$$DP(i) = \{(a_{max}+1), \sum_{\substack{j<i \\ a[j]<a[i]}} b_j \text{ if } a_j == a_{max}\}$$

Tough Questions

- → Atcoder Question I → Coins
  Idea :- For Σ Products, use $\sum_{i=1}^{\#\text{transitions}}$ (curr state × possible transition(i))
- → Atcoder Question E → Knapsack 2 → state rotation

→ Scambled String :- Update rec param acc to input

→ Sort the strings :- $\Sigma|s_i| \leq 10^6 \longrightarrow N \leq 10^6$

worst case :- $s_i = 1 \ \forall \ s$, $N = 10^6$ → $TC = O(N \times 2) = O(N)$

($\frac{s_i}{N}$ length strings)

or

$s_i = 10^5$, others $s_i = 1$

→ Coloring Array & Rock, Paper Scissor :- • Do not start from -1

- $\boxed{\text{If TLE} \longrightarrow \text{Iterative}}$:-

→ For Back → Set a default value along with ans

" "

- Maximizing the cost → Imp
- → Count Binary strings → global DP

Relook :-

- Prime Subsequence
- Max Sum in Grid
- Bricks Colouring
- Balanced Bracket Sequence

Module 3 - Week 9-10

Graph is your 'Shortest Path' to Success

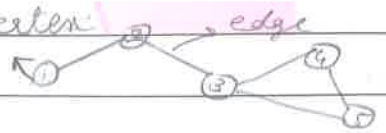
I Graph Foundations

Graph Introduction

- **Graph** → Function $\boxed{G(V,E)}$

  Set of vertices — Set of edges

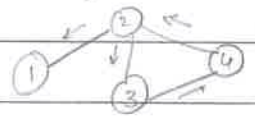
  Node/vertex — edge



  $\{1,2,3,4,5\}$ $\{(1,2)\ (2,3)\ (3,4)\ (4,5)\ (3,5)\}$

- **Weighted** :- Edges with weights → OPP → Unweighted
- **Directed** :- " have direction → OPP → Undirected
- **Sparse** Graph :- $M = |E| < O(N^{3/2})$, $N = |V|$ (Gen b/w 0 to N)

  Dense Graph :- $M = |E| > O(N^{3/2})$, $N = |V|$ (Gen towards N^2)
- **Degree** :- No. of Neighbors of Undirected graph Node



deg(2) = 3 (if undirected)

In deg(2) = 1

out deg(2) = 2

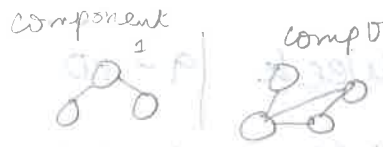
↳ INDEGREE / OUTdegree For directed graph

- **Path** (walk) :- Seq. of vertices $[v_1, v_2 \dots v_k]$ s.t. $(v_i, v_{i+1}) \in E$
- **Cycle** :- Path where starting Node is the Ending Node (without repeating edges)
- **Isolated Node** :- The node that cannot be reached by any other node of the graph. (degree / Indeg / outdeg = 0)
- **Reachable** :- Node x is reachable from Y if there exists a path from Y to x
- **Connected** :- $\forall x \forall y$, $\exists$ path (y to x) → Undirected

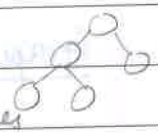
  Weakly " :- $\forall (x,y)$ if $x \to y$ or $y \to x$ } Directed

  Strongly " :- $\forall (x,y)$ if $x \to y$ & $y \to x$ }
- **Self-loop** :- $(x,x) \in E$ edge
- **Multiple Edge** :- repeated edge in E
- **Simple Graph** :- NO self loop & multiple edges
- **Multi Graph** :- has " " or " "
- **Sub Graph** :- Vertex induced → select vertices → Take any required edges

  Similarly, Edge induced



- **Component**
- **DAG** :- (Directed ACyclic Graph)
- **Tree** :- Connected Graph of (N-1) Edges, N = |V|
  - ↳ 1 simple path b/w any 2 nodes



  Forest :- Group of trees
- Easy **Eulerian** :- Every edge traversed once exactly → linear
- Tough **Hamiltonian** :- " node " " " → NP Hard

  ↳ Graph if cycle is present in graph

Rule to check if Eulerian path exists :- (Hierholzer's Algo)

→ Count No. of nodes with **ODD degree**

↳ If "zero" → Eulerian Cycle exists

If "two" → " path possible (mostly)

If > two → No " "

} Graph has to be connected

↳ Every Edge increases total deg of Graph by '2', So there cannot be odd no. of odd degree nodes as their sum has to be even

- **Bipartite** :- Graph with no two neighbors as same color when we color nodes with 2 colors
- **Complete** :- Contains all possible edges.

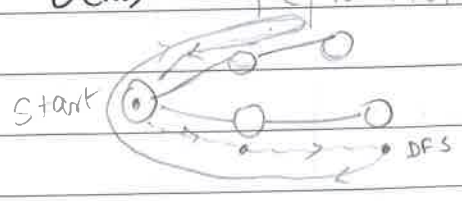
iff · no. odd len cycle

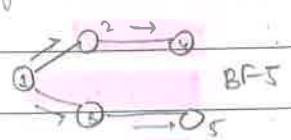
Representations :-

			Space complexity
1.	Adjecency Matrix	—	$O(N^2)$ →
* 2.	Adjecency List	→	$O(N+M)$ → 1 → { } 2 → { }
3.	Edge List	→	$O(M)$ → { (from, to, weight) }

Stack **DFS :-** $O(V+E)$

Queue **BFS :-** $O(V+E)$





Graph DFS Coding Intro :-

Q. Find (x, y) st. x < y & x, y belong to different components

→ Find the number of nodes in the independent components and ans = sum of all pair-wise product

$(\Sigma a_i)^2 = \Sigma a_i^2 + 2(\Sigma \text{pair wise products})$

- DFS → Short code. easy tweaking using recursion
- BFS → used to find shortest path

Explicit Graph → node & edge form
Implicit " → graph automatically formed using rules. Eg maze

Pseudo Code → O(N+M)

```
vector<vector<int>> g;      // adjcency list
vector<int> vis;            // visited = 1 else 0

void dfs(int node){
    vis[node] = 1;
    for (all nodes x reachable from node){
        if (vis[x] != 1){
            dfs(x);
    }}

Solve (){
    cin >>n>>m;
    g.clear();
    g.resize(n+1);
    vis.assign(n+1,0);
    // Input
    // For all nodes, if vis[node]==0 call dfs(node)
    // calculation & output
}
```

DFS Cycle Finding

I. Printing cycles in directed Graphs

edge → ⓐ if x not visited even once → FORWARD EDGE → DFS Tree
" x visited & we are currently exploring its neighbors → BACK EDGE
" " " " we have completed " " " → CROSS EDGE

BACK EDGE GIVES US CYCLES

Pseudo code:-

```
vector<int> status, par ; // status → 0 unvisited  1 - currently exploring  (back edge)
                                  2 → completely done.
                                  // parent of each node => par

dfs(node, parent){
  par[node] = parent;
  status[node] = 1;
  for( neighbors x of node){
    if (status[x] == 0)  dfs(x, node);
    else if (status[x] == 1){ // loop over parents till we reach node & maintain output vector for the first cycle.
    }}
  status[node] = 2;
}
```

→ Finding all nodes that are a part of a cycle

Sol$^{\underline{n}}$:- • On hitting "Back" edge at x, do partial sum +1 at x and partial sum -1 at parent[curr_node].

• Maintain an order of nodes that are getting completed first & loop in the same order adding prefix values to parents.

IMP

→ For Undirected Graphs, there are no ~~ex~~ cross edges.

→ As UG are two way graphs, there would appear to be a ~~path~~ cycle b/w all parents - child.

Sol$^{\underline{n}}$:- Don't explore x if it is parent

→ Handle self loop / multiple edges in input.

BFS Coding (Minimum path)

FORM 2

→ DP on Graphs is only applied on DAG's → Eg. no of paths, Maximum weighted path (allowed)

Q. Find the shortest path from a starting node (Given Grid with walls)

moves Available → L R U D → Not DAG

→ Pseudo Code:-

```
void bfs (starting node){
  // set dist for all cells as ∞
```

```
// In bfs queue, push starting cell & set its dist = 0
while (queue not empty){
    // pop ~~top~~ front cell & obtain front cell distance = F
    // Explore all neighbors using a loop
    for (all neighbors feasible N) {
        if ( F+1 < dist(N)) {
            dist(N) = F+1;
            queue.push(N);
        }
    }
}
}
```

→ To print the path taken, maintain parent array

→ To loop over implit neighbours, " dx & dy "

Finding Number of shortest path:- Along with dist, maintain a count of that " in pair

MultiSource BFS → (MSSP → Multi-Source Shortest Path works even with 01 BFS & Dijkstra) #

Case 1:- 1 source S, multiple Exits E_i's → Start - S Normal BFS.

Case 2:- multiple Sources S_i, one exit E → Start - E Normal BFS.

Case 3:- For Each source S_i, find nearest exit E_j → MS-BFS

Idea → For every node, we find nearest Start point. (Start points)

→ Push all Exits in BFS queue with distance zero, Run BFS normally

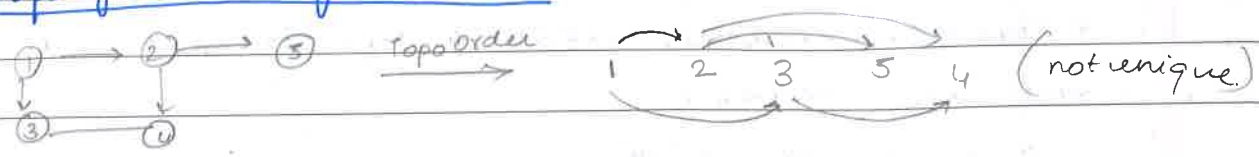
→ Every source S_i will be first reached by the nearest exit E_j

IMP Q. Person vs Monster in Grid ...

II Topological Ordering → (Only for DAG) → DP → FORM 2

Path or Max Path Questions

Topological Ordering with DFS



Use:- Finding longest path in DAG (Also DP Sol^n)

Pseudo Code:-

```
DFS () {
// exploring all neighbors
// topo.push_back(node); → topo saves in reverse order
}
```

Topological Ordering with BFS - Kahn's Algo

Idea:- Go on deleting the nodes with indeg = 0
Order of deletion = Topological Ordering

Pseudo Code:-

```
BFS () {
// Push all nodes with indeg = 0 in queue
while ( queue is not empty) {
// pop front & push in topo
// explore all neighbors of N front & reduce their indegree by 1
// If the indegree of the neighbor becomes zero, push in queue
}
}
```

→ If size of topo < total nodes, then Graph has a cycle
→ To find lexographically smallest topo, use Priority Queue

III Graph Shortest Path

~~Formulation Video~~ 0-1 BFS

→ Idea :- BFS Queue maintains distinction b/w nodes at current level & next level → Exploit by adding 0 wt nodes in front & 1 wt nodes at back using deque

Pseudo Code (for Single Source Shortest Path - SSSP) | O(V+E)

```
BFS10 (start){
    //set all dist as ∞
    // push start node in deque & set its dist to 0
    while (deque not empty){
        // obtain front & pop
        for (all neighbors N of front){ // adjecency list contains pair of node & edge wt
            if (dist(N) > dist(front) + edge_wt){
                dist(N) =
                if (edge_wt) == 0 → push front
                else → push back
}}}}
```

Note:-
For MSSP (Multi SSP) we can use same technique as normal BFS in 10-BFS

0-1 BFS Code

Problem 2:- 3 cities, length of shortest bridge connecting them
↳ IMP

Dijkstra Theory - For weighted edges without -ve cycle

Idea:- From start nodes process nodes acc. to the dist they are from", ie from shortest dist. to longest dist. nodes

O[(V+E) log E] - Time complexity → uses set / priority queue maintain (dist, node) pair

Pseudo Code:- (for SSSP)

```
dijkstra (start){
    //set dist as ∞ & visited as 0 // VISITED mandatory (else TLE)
    // push (-0, st) in pq (priority queue) & set dist[start] = 0
    while (pq not empty){
        //obtain the front element & pop it
        // if (visited(front) == 1) continue, else set visited(front) = 1.
        for (neighbors n of front){
            if → (dist(n) > dist(front) + edge_wt){
                dist(n) = dist(front) + edge_wt,
                push (-dist(n), n) in pq
}}}}
```

Dijkstra Eg Problem 1 → IMP (Tough)

Q. Given graph of cities & wt on edges. (A) —— L —— (B), L petrol cost to go from A to B. Car has capacity C of petrol which it can refill at any city i paying price P_i. Find the least cost of petrol spent on moving from start city to end city.

Dijkstra Eg Problem 2 → INTERMEDIATE

Q. Burning ropes & nodes. Time to burn the whole structure?

BELLMAN ford - SSSP for graphs with -ve cycle - O(VE)

Idea :- • looping over all edges (V-1) times and relaxing them gives us the shortest dist. to nodes reachable by start.

• looping ~~aa~~ again for (V-1) times helps us detect the nodes affected by -ve cycle which ~~whit~~ will have $-\infty$ dist from start

Pseudo code :-

```
run (v-1 times) {
  for ( edges e in the graph){
    if (dist(to_node) > dist(from_node)+e)
        dist(to_node) = dist(from_node+e
  } }
```

↳ Repeat process & set all nodes that reduce dist. to $-\infty$

Note :- all nodes with $-\infty$ dist from start are NOT a part of -ve cycle

Floyd Warshal (All Pair shortest Path) - $O(V^3)$

Idea :- Relax dist b/w any two nodes wrt a third node.

Pseudo code :-

```
//Take input as adjecency matrix
// dist[i][i] = 0 , dist[i][j] = ∞ if i≠j → Set default
// take input and set dist[i][j] = min(input, dist[i][j])
```

```
for (k : 1 to N) {
    for (i : 1 to N) {
        for (j : 1 to N) {
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
}   }   }
```

Note :- In place of floyd warshal, we can use BFS on every node ($O(VE+V^2)$)

Floyd Warshal Problems

Q → Finding the path taken in APSP

Sol$^{\underline{n}}$:- • Maintain a parent array & by default par[i][i] = i;

• if dist[i][j] < dist[i][k] + dist[k][j]

update dist[i][j] normally & set par[i][j] = par[k][j];

• Print in reverse the parents from j until we reach i

Q → Finding nodes reachable from A

Sol$^{\underline{n}}$:- • In input if path exist from A to some x, set its dist 1 (Default 0)

• If any node is reachable from A through another node, set dist = 1

Q → Finding shortest cycle of graph / -ve cycle in graph

Sol$^{\underline{n}}$ → Set dist[i][i] = ∞ & run floyd marshal. Find min(dist[i][i])

For DIAMETER of graph → max(all dist[i][j], i≠j)

↳ longest(shortest path b/w any 2 nodes)

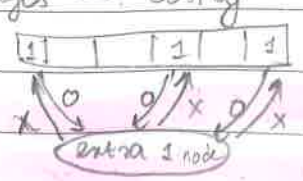
Graph Modelling Ideas 1

Q → Given an array you can move from a cell to any other cell with same value giving a cost 'x'. You can move to adjacent ~~nodes~~ cells giving a cost 'y'. Find min cost from st. to end.

⟹ Connecting all pairs of cell with same value using edge of cost 'x'

↳ $O(n^2)$ edges → costly

Idea :-



→ $O(n)$ edges → cheap

↓

add an extra node ~~to channelize in & out~~

↓ cost = 0 ↓ cost = h

Q. Given a binary string st of size = 20, convert it to ~~st~~ end string of size = 20 by flipping one bit at a time. Find least no. of moves. (Note, we should never reach the forbidden string)

IMP

→ • Storing binary string as int number → saves time

• From every non-forbidden number, building edges to numbers formed by flipping one bit if they are not forbidden, then BFS

Graph Modelling Ideas 2

Q. Given a game, we need to move from start to end in a M×M board. If we move in one direction, we ~~chan~~ can change direction only on hitting wall or boundary. Find min moves. ↔ allowed

→ BFS Approach :-

For every cell find where we stop if we move in four directions. Build an edge from current cell to all stop cells and run BFS

DI-BFS Approach :-

State :- (x coordinate, y coordinate, current_direction)

if → (next cell is not wall / boundary) move in the same direction with same no increase in cost

else → change direction with +1 increase in cost

Note :- If state has 'x' variables, dist array should also have the corresponding 'x' " as they represent a node uniquely in graph

Q. Given suitcase lock, we need to reach unlock configuration without reaching the forbidden values.

→ Previous video idea $\xrightarrow{\text{Small trick}}$ State(i, j, k) → save positions rather than numbers for easy coding

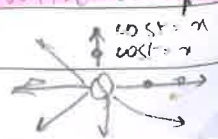
Q. Given Grid with walls, we need to move queen from start to end.

→ 0-1 BFS Approach :- $O(8^2 N^2)$:- dir $\xrightarrow{\text{same}}$ cost = 0, $\xrightarrow{\text{diff}}$ cost = 1

IMP

BFS Approach :- $O(N^2)$:- From a cell, move in all 8 dir with cost = 1 edges, if cell not visited

(from Queen) same cost ←

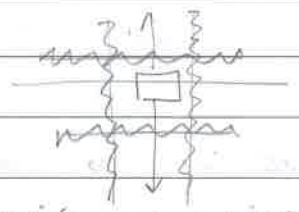


Harder Modelling Problem Ideas

Q. There are dots of k colors in grid. Find shortest line length ~~that~~ that has ~~all~~ dots of all colors. ↕ allowed

IMP → Idea:- • Maintaining if the specific color dot is present or not using bit mask. Bitmask + cell location → State

Q. Light bulb-path problem:- Man has to move from start to end ~~with~~ only on lit cells. He can make current row/col lit for price = 1. From such row/col he can only move to originally lit cell. The the row/col light switches off. Min cost to reach end. ↕ allowed

→ 

□ curr cell
— can move to any with cost 1
∿ can " " " lit cell with cost 1

Directly creating edges → $E = O(N^2)$

Creating edges by connecting to an extra node for each col/node → $E = O(N)$

Shortest Path/ Min cost etc → Graph

IV Union Find & MST

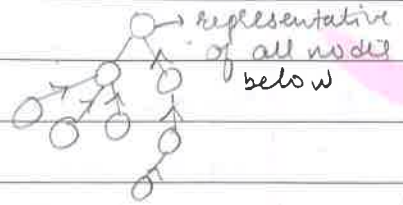
Union Find

Q. Given a graph & Q queries, we need to tell if the nodes in the query are joined then the number of components decrease or not.

→ Union Find

Find(x) → finds representative of x

merge(x,y) → merges " " x with " " y



Rank Compression :- Joining component with smaller size / height to a larger one

Pseudo Code (Initially all ranks = 1, parent(i) = i ∀ i)

```
merge(x,y) { // O(log N)
  x = find(x); y = find(y); if(x==y) return;
  if( rank(x) < rank(y)) rank(y) += rank(x); parent(x) = y;
  else                   rank(x) += rank(y); parent(y) = x;
}
```

Find(x) { $O(\log N)$

if par(x) == x return x

else return (par(x) = find par(x)) // Path Compression

}

Total Find & Merge complexity = $O(\log \log N) \sim O(1) \Rightarrow \alpha(N)$ → operation fun^n

Q. Given a graph & Q queries, remove the edges in the query one by one & report the remaining components after each query :- Sol^n :- Do Union Find in reverse order of the queries

IMP

Q. Given a graph, we need to find the value of $\sum_{i} \sum_{j} \min(\text{edges b/w } i \,\&\, j)$ $(i<j)$

→ Start from a graph with no edges, & add the edges in decreasing order of weights. On each addition, add (wt × left component nodes × right component nodes)

no. of nodes on left — w — no. of nodes on right

MST - Minimum Spanning tree (Always exists in a connected graph)

Spanning tree :- Tree connecting all nodes in graph

MST :- Spanning tree with min Σ(of chosen edges)

Kruskal's Algorithm :- Sort all edges acc. to wts & go on connecting using union find (i.e) if the nodes are already a part of the same component, skip the edge, else connect.

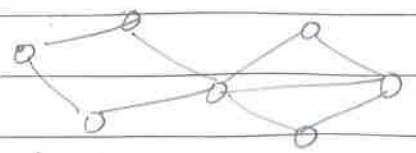
$\boxed{E \log E + (E+V)\alpha(V)}$

↓ sort ↓ union find

IMP

Q. Given N factories, connected by roads, we need to ensure each factory gets goods either from itself or from other factory through road. D_i → road cost

C_i → self cost



→ Idea :- Connect all nodes to a supreme node, cost to come/go to/from this node is C_i. Find MST

V. Tree Discussions

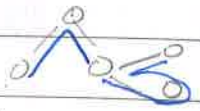
Tree Application Ideas 1

→ N-1 edges with 1 component
→ Only 1 simple path b/w any 2 nodes } Tree → Rooted / → Unrooted
→ No cycle

→ Trees → DFS (As only one path to reach each node from another)
↳ Can also determine dist b/w nodes in trees

Finding a diameter in a tree :- • Take random node 'x' as root
• Find node 'y' farthest from 'x'.
• Run dfs from 'y' to find 'z' as the farthest node from 'y'.
z to y → diameter

PYQ :- IMP Given a ~~ter~~ tree, we have to find the min dist required to go to all nodes using edges



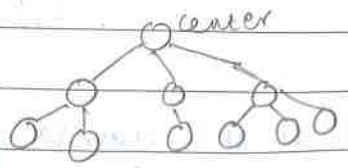
→ Contribution Technique :- [2(N-1) - diameter] → Ans
(As we go through all edges 2wice except diameter)

Tree Application Ideas 2

Q. Finding the Number of diameters in trees
→ Center of ~~di~~ tree :- Mid point/s (1 or 2) of any diameter of tree
All diameters pass through center of tree

Number of diameters :-

Case 1 :- 1 Center tree :- • Let D be the diameter length (odd)

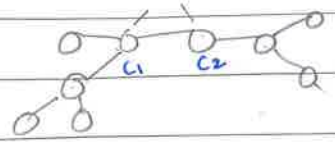


Ans = 2×1 + 2×3 + 3×1
= 11

• Root the tree at center
• Suppose center has n childs & a_i be the number of leaves in ith child tree with depth = $(D-1)/2$
• Number of diameters = $\sum a_i \times a_j$, $i<j$

Case 2:- 2 center tree :- • let D be the diameter length (D/2)

centers



• For each center as root, find the number of leaves at depth = D/2, let them be a_1 & a_2

• Ans = $a_1 \times a_2$

Ans = 2×2 = 4

D = 6

#:- To not explore C_2 side while exploring C_1 call DFS on C_1 with C_2 as parent

Finding Center/s :- While finding the diameter, maintain parents too. from root to the farthest node, go (D/2) steps to reach the parent (for a center node)

Tree Application Ideas 3

Centroid :- The node at which if we root a tree, all subtrees have $\leq n/2$ nodes.

Center ≠ Centroid always

Trees with even nodes might have 2 centroids (Both connected)

Algorithm :- • Root a tree at some random node & find a sub-tree with $> n/2$ nodes

• Move to that child & redo the process until we find centroid

Q. Given a tree, find $\sum\sum_{i<j} dist(i,j)$

→ contribution technique → for every edge multiply nodes on either end side

Ans = $\sum_{v=1}^{N}$ sub_tree size (v) × (N − subtree size (v))

Q. Given a tree with weight on nodes find $\forall x$ - max |(val[x] − val[y])| (y is ancestor of x)

→ Save (min & max) values as we go down the tree → save in recursion state

(If min was asked. maintain a global set of all ancestors so far)

If large datastructure needs to be maintained in DFS, maintain it globally. (for ancestral query)

VI Euler Tour, Articulation Points & Bridges → DFS Tree (Undirected)

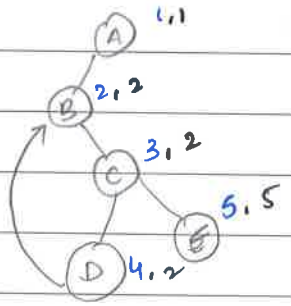
Articulation Pt :- Removing this NODE will result in 2 separate components

Bridge :- " " edge " " " " " "

IMP

DFS number / In time :- number assigned ~~acc.~~ in the order to which the nodes are explored from root

lo number / lo time :- The lowest in time node reachable from a node when taken back edges or untraversed edges.



• - lo time
• - in time

Parent $y \longrightarrow x$ child

if low_time(x) ≥ in_time(y) → y is AP

" " > " (y) → Bridge (x,y)

iff y is not root

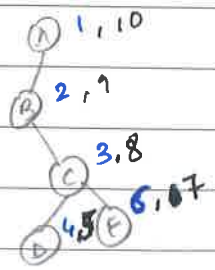
Q. Find the number of components a graph splits into on deleting a Node (given).

Euler Tour

Q. Given a rooted tree & Q queries, we need to find if u is ancestor of v for each ".

⟹ Maintain **Outtime** (:- The order in which exploration of nodes was completed)

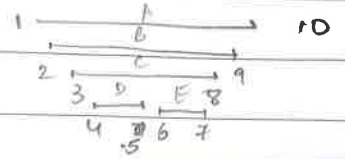
IMP



• Outtime
• Intime

u is ancestor of v iff intime(u) ≤ intime(v) & outtime(u) ≥ outtime(v)

Flattening technique :-



Tough Questions

Union Find & MST

→ Weighted Union Find Tree :- Maintaining dist to the root

→ Min Edge wt :- Contribution Technique

→ Topological Ordering (labelling) :- Topological ordering using outdeg

Tree

→ Tree Centroid :- When ~~dia~~ nodes are even, 2 connected centroids can exist

→ Tree Diameter 2 :- Tree with 1 node has 1 dia of length 0

I. Segment Tree Concepts

Segment Tree Concept 1 & 2 | Segment Tree Implementation

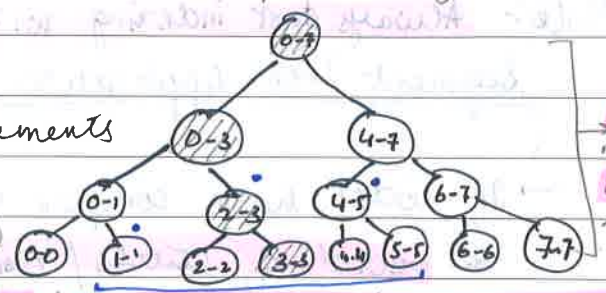
Q. Given an N sized array and Q queries of form
1 l r → find sum of sub-array b/w l & r
2 x v → change x^{th} index value to v

⟹ Using prefix sum :- $O(NQ)$
Usig segment tree :- $O(Q \log N)$

Idea :- creating tree over array → node responsible for a range

Eg :-

2	3	2	4	5	1	0	4
0	1	2	3	4	5	6	7

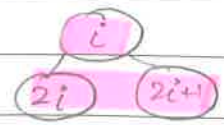


Each nodes saves sum of elements it is responsible for
(not necessary to have all leaves)

logN ← Changing 3 will only affect (3) nodes (1 node per level at max)

Sum 1 to 5 is given by : nodes → logN (At max 2 nodes per level)

Building segment tree :- Save nodes in array following the scheme for 'i'



Pseudo Code :-

```
int a[n], t[4×n]; // a is original array, t saves seg tree
void build (int i, int l, int r){ // i for seg tree, l & r corresponding
                                  // indices from a for which
                                  // node i saves value
    if (l==r) t[i] = a[l], return;
                  // leaves case
    mid = (l+r)/r;
    build (i×2, l, mid);
    build (i×2+1, mid, r);
    t[i] = t[i×2] + t[i×2+1]; // as t stores sum of range here
                              // merge
}
```

// Point update

```
void update{ int i, int l, int r, int pos, int val){
    if (pos < l || pos > r) return; // pos not b/w l & r
    if (l == r){ t[i] = a[l] = val; return; }
    // find mid & merge into t[i] after updating children
}
```

// range query

```
int query (int i, int l, int r, int lq, int rq){
    if ( rq < lq || r < lq) return 0; // (lq, rq) not b/w (l, r)
    if ( lq ≥ l & rq ≤ r) return t[i]; // " completely b/w "
    // find mid
    // find 2 children query & return merged value
}
```

Note:- Always start indexing with '1' for '2x' & '2x+1'

Segment Tree Application 1 of code.

→ If nodes have complex value / multiple values to save use vector of structs / required data structures

→ Change leaf condition acc. to requirement

→ Build a merge function.

Eg:-

```
struct node{
    int mini;
    int cnt;
}
```

```
node t[n×4];
t[i] = node(val_1, val_2);
t[i].mini; // gives val_1;
t[i].cnt; // gives val_2;
```

imp

Segment Tree Application 2

Range Update — Point Query
- 1 l r v → add v to all elements b/w l & r
- 2 x → find value of "

Idea 1:- • Convert Range update into 2 point updates.
of partial sum (+v at l, -v at r+1)

• For point query, find sum from (0 to x)
(Convert " " to range query)

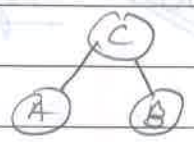
Idea 2 :- • For range update, add '+v' to all node responsible for the range
• For pt query, add all the node values in the path to x (add all ancestors)

Segment Tree Application 3

Q. Query 1 :- update arr[x] with value v
Query 2 :- find max sub array sum b/w l & R

→ All no.s are +ve
→ >0 length subarray

⟹ For every node, save the following as state
- max sum - subarray → ①
- " " " starting at l → ②
- " " " ending " r → ③
- total sum of range → ④



$$\left.\begin{aligned} C.1 &= \max(A.1,\ B.1,\ A.3 + B.2) \\ C.2 &= \max(A.2,\ A.4 + B.2) \\ C.3 &= \max(B.3,\ B.4 + A.3) \end{aligned}\right\} \text{mergings}$$

$C.4 = A.4 + B.4$

and leaf conditions.

Note:- The mergings, will slightly change if -ve numbers are allowed or if ≥0 length sub arrays are allowed

k^{th} Segment Tree

Q. Query 1:- Insert x
" 2:- Delete x
" 3:- find k^{th} element in set

→ $Q \le 10^5$
$K \le 10^9$

IMP

⟹ Offline:- • Take all queries & check which nos are being inserted.
• Assign a representative number to them from 0 to 10^5.

Coordinate compression

Idea :- • Maintain segment tree from 0 to 10^5, all initially 0
• If i is inserted, update i^{th} leaf to 1, for delete use 0
• ~~Rat~~ Each node contains no. of on bits in l to r
• from A move to B if $K \le$ (val of B) & find k^{th} element in B
else, move to C & find (val of C) - k^{th} element in C

Segment Tree Application 4

IMP

Q. We are given N people with the information that for each person, how many people taller than him are before him. We are given N heights in scrambled order. Find height of each person

→ • Sort heights

• Start ~~for the~~ from the last person. Suppose k people are taller than him before him → • Find $(N-k)^{th}$ height

Repeat ← • Remove it from list

Segment Tree Application 5

IMP

Q. Find longest subseq such that $|A_i - A_{i+1}| \le D \ \forall \ i$ in subseq

$N \le 10^5$, $A_i \le 10^9$, $D \le 10^9$

→ Form 2 DP → $O(N^2)$ ✗

Form 2 DP with Segment tree → $O(N \log N)$ ✓

Idea:- for A_i, we need longest subseq ending anywhere b/w $(A_i \pm D)$

(COORDINATE COMPRESSION)

• We maintain an array of max subseq length ending at A_i

• Range query to get max b/w $(A_i \pm D)$

• Update A_i with max length after passing through each A_i

Segment Tree Lazy Propagation & Code

Range Update & Range Query

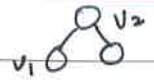
→ Range Query Normally

→ Range Update: Mark log N nodes lazy & use Push to correct the nodes when ever reached

Push:- 1) Corrects the current node → $O(1)$ or $O(\log N)$

2) Implements lazy merge (logic to merge lazy values b/w child & parent)

3) pushes lazy to child nodes

Note:- 

→ v_1 has been marked before v_2. if it was otherwise, v_2 would have be pushed while tying to update v_1

Pseudo Code :- $O(N \log N \times \text{push operation})$

```
update ( ){
// Push
// If valid update, do update & push
// Move to child nodes}

query ( ){
// Push
// Normal Query merge & return}

push( ){
// If node not marked, return
// set the lq parameters acc. to lazy prop if node is marked
// push to child nodes if exist and unmark the curr node
}
```

Debugging Segment Tree codes:- Print out l & r in update & query

II. Live Session & Practice Problemes

Segment Tree Abstraction

Q. Given Array of size N. We can jump from i to j if $a[i]+i \le j \le b[i]+i$. Cost of landing on $j = c[j]$. Find min cost from 0 to N-1

→ DP(i) → min cost to reach N-1 from i → Traverse from 0 to N-1 & build a min tree of DP(i)

→ update $DP(i) = cost(i) + DP(j)$ st $a[i]+i \le j \le b[i]+i$

More Application Ideas

IMP Q. Given an array, answer queries of form Q → L R K → find num of values ≤ K b/w L & R included.

→ Solution#1 → Offline technique

- Sort queries in order of K
- Build segment tree which store sum of indices & initiate them with zero
- For K = Ki, fill all indices which have val ≤ Ki with 1 & run segment tree query b/w L & R

TC → O(N log N)
MC → O(N)

Solution#2 → Online Technique

- In each node maintain the sorted order of the elements in a vector.
- For query, go to required nodes & find the number of values ≤ K in those nodes in log N time

NC → O(N log N)
TC → O(N log² N)

Q. Find the # distinct values b/w L & R → Q queries of this kind

IMP

→ Eg:-

4	2	3	4	4	5	6	7	3	4

→ Given Array → A

-1	-1	-1	0	3	-1	-1	1	2	4

→ built array → B

Build array of the index of last seen same element on A

distinct values b/w L & R in A = # of values b/w L & R ≤ L-1 (0 indexing) in B

Revise Questions → Prev Year
→ graph

Quant Prep

→ OOPS, DAA (OS)

n days

$A_i \rightarrow$ on i^{th} day (expiry day)

4 1 2 1 2

1

4

4 -4

+1 +2 +1 -1 +2

+4 -4

~~4 -3 2 1 +1~~

4 1 2 3 4 5

a

$n \rightarrow$ dri $\rightarrow$ (x $\rightarrow$ pos)

$k \rightarrow$ batteries $\rightarrow$ (y $\rightarrow$ key)

b

---

driver batteries

$n \rightarrow k$

12

Stones [i] $\leq 10^7$

$n \leq 10^5$

1000 $\rightarrow$ drivers

2000 $\rightarrow$ batteries

$P \leq 10^9$

12

$360^\circ \rightarrow 12 \times 60 \times 60$

3

10 40 60 80

40 10 10 30

k players → 1 to k → 10^6

string s → 10^6

00

3

2 1 1 2

3 % 4 = 3

7 % 4 =

1 2 3 4

8 % 4 = 0

1 2 3 4

4

1 2 3 4

0 + 1

1

2 % 4

8 % 4 = 0

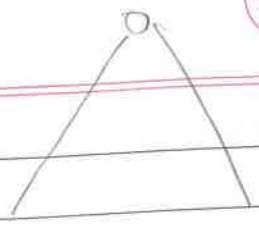
4

1 2 3 4

6 + 7

7 % 4

3



n nodes

$\lfloor i/2 \rfloor \leftarrow i \rightarrow 2i$

$\rightarrow 2i+1$

cost of travelling through the road

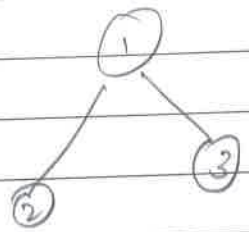
x to parent $x/2$ $\rightarrow$ cost $[x-2]$ (0 - idx)

root node 1 $\rightarrow$ leaf $\Rightarrow$ sum of cost of edges

inc cost on some edges

2^{10} $2^{10} = 10^3$

$10^4 - 1$



0 $\rightarrow$ 1 to 2

1 $\rightarrow$ 1 $\rightarrow$ 3

2 $\rightarrow$

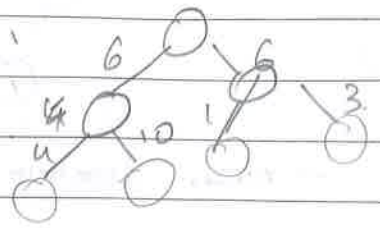
Given s of length n

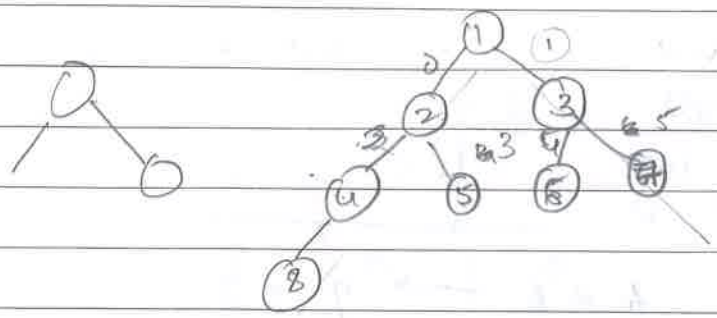
$26 \times 26 \rightarrow 670.$

10^5

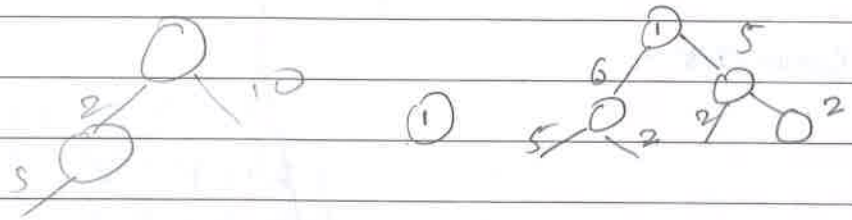
1 2 3

0 1 2 3 4
7 6 6 4 10



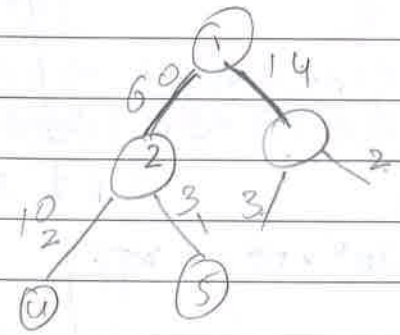


26×10^5



0 1

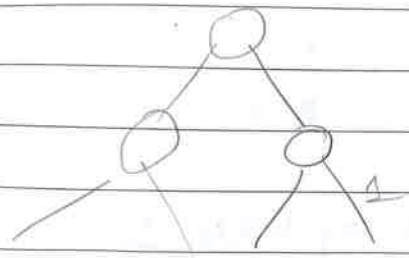
a a b c



0 1 2 3 4
abbac → 5

650×10^5

6×10^7



5 6

$(2N-1) \rightarrow$ bat

$E \rightarrow e_i$ $\quad K \rightarrow k_i$

$(2N-1) \rightarrow$ batches $\rightarrow$ each containing e_i Eng
k_i kashmiri

$T \rightarrow 5$	$T \leq 20$
$N \leq 100$	$N \leq 10^5$
$e_i \leq 00$	
$k_i \leq 100$	

A & B $\longrightarrow$ m & n

Karan dist =

1 2 1 2 1 2
1 2 3 1 2 3

1 2
1 2 3

$10^5 \times 50 \times 25$

$n = 2$
$m = 3$

$$\begin{array}{r} 2 \\ 1\overline{2} \\ 1\,2\,3 \\ \hline 3 \end{array}$$

0
1

1 2 1 2 1 2 1 2 1 2
1 2 3 4 5 1 2 3 4 5

A = 12 $\longrightarrow 2 = m$
B = 123 $\longrightarrow 3 = n$

A $\rightarrow$ n times
B $\rightarrow$ m times

1 2 1 2 1 2
1 2 3 1 2 3

1 2 1 2 1 2 1 2
1 2 3 4 1 2 3 4

1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 4 1 2 3 4 1 2 3 4

1 2 1 2 1 2 1 2 1 2 1 2 (2)
1 2 3 4 5 6 1 2 3 4 5 6

1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
1 2 3 4 5 6 | 1 2 3 4 5 6 | 1 2 3 4 5 6 | 1 2 3 4 5 6

(2)

~~12~~

2, 3 = 1
2, 4

0 1 2
1 2 3 4

4

0 1 2 3

1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 4 5 6 | 1 2 3 4 5 6

1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 m
1 2 3 4 5 | 1 2 3 4 5 | 1 2 3 4 5 | 1 2 3 4 5 n

$j \times 5 + i$

i

$(j \times n + i) \% n$

$x \to a = 10$

10 20 20 30

$P_1, P_2 \ldots P_{12}$

each

→ rolls a die → secret

I have a 4 → median ≥ 3.5

4

13 people have

$\frac{5}{11} \times \frac{1}{2}$

~~+9~~ ~~8~~ $\frac{1+2+\cancel{6}\,3+4+5+6}{6}$

$5 \times 4 \times 3 \times 2 \times 1$

$\frac{12+9}{6}$

$\frac{\cancel{+6}\,8}{11} \times \frac{1}{2}$

$2 + \frac{3}{2}$

3.5

→ Expected value of each of the other 11 players is 3.5

→

Probability

1 2 3

3 4 5 6

4 3.5 ~~8~~ 1 2 3 4 5 6

1 2 3 4 5 6

~~1 1~~ ~~2 2~~ ~~3~~ 3 4 ~~4~~ ~~5 5~~ ~~6 6~~

$\frac{2}{\cancel{3}6} \times 5$

$\frac{3}{6} \times 6$

1, 2, 4, 8, 16,

4 11 22 3 44 55 66

4 5 6

$\frac{3}{6}$ $\frac{1}{2}$

kyabola wo?

6 or > 6 get $\frac{1}{2}$

$\frac{1}{2} \times \frac{5}{11} + \frac{1}{2} \times \frac{6}{11}$

Ans Bhey → Ask if correct

$\frac{5}{11} \times \frac{1}{2} + \frac{1}{2} \times \frac{6}{11}$

$\frac{5}{11} \times \frac{1}{2} + \frac{64}{11} \times \frac{1}{2} + \frac{3}{11} \times \frac{1}{2} + 2 + 1 + 0$

5×

9 + 6 + 1

$\frac{8}{11} \times \frac{1}{2}$

3.5 × 11

55
33
38.5
+ 6
44.5

Photo Nahi hai iska

Q 1

NOT visible

1 2 3 4 5 6

2 2

$\frac{1}{3}$ 0.33?

odds > evens?

or # perfect sq > 5

$\frac{5}{11} \times 0 \cancel{6} 2$

Ernie → First

-10^4

-10^5

0 0 0 0 0

→ spl string

0 0

ernie -1

ϕ ϕ 0

best -1 best -1

ernie wins

e-1

b-2

1 → b

3 → e

5 → b

7

0 ϕ ϕ ϕ 0

ϕ

ϕ ϕ ϕ ϕ 0

b2 e

ϕ ϕ

e - 4

b - 2

0 ϕ ϕ ϕ ϕ ϕ ϕ ϕ

ϕ ϕ ϕ ϕ ϕ

ϕ ϕ ϕ

e → 1

4, 2, 3, 1

0 ϕ ϕ 0

3 → ernie

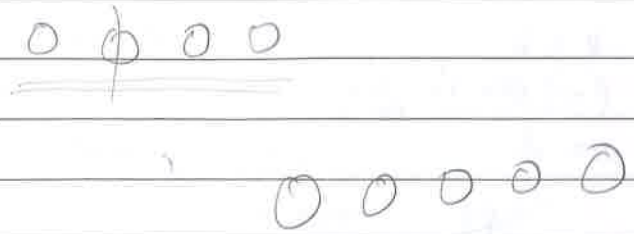
ϕ ϕ ϕ ϕ

0

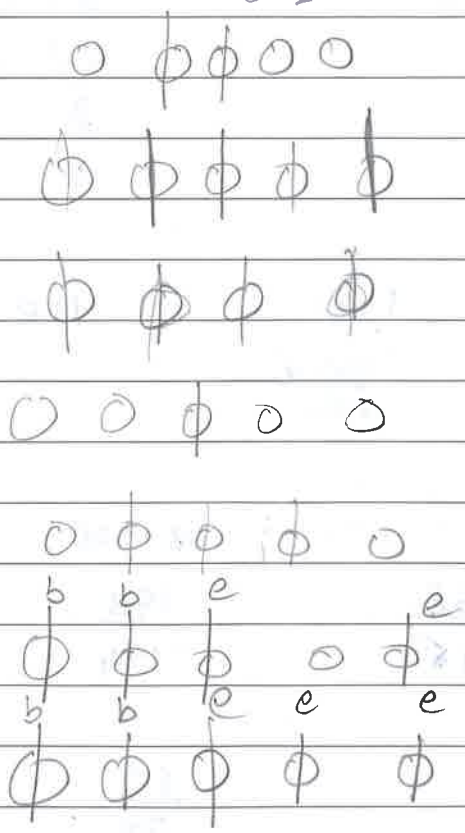
ϕ ϕ ϕ

spl st, level

DPC level, spl st



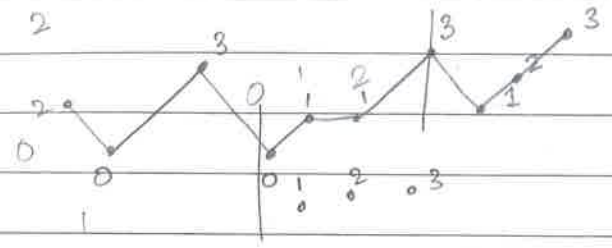
e - 1
b - 2



odd → even

~~z → 0~~
~~w → 2~~
~~g → 8~~
~~x → 6~~
~~v → 5~~
~~o → 1~~
~~s → 7~~
~~f → 4~~
~~h → 3~~
i → 9

K. days in past k , 0 10^5 10^7



$k = 2$
$f = 0$
$n = 3$

~~DPF~~
10^5

$k = 2$
$f = 3$
$n = 3$

2 2 5 ~~3~~
1 2 2

0 1 1
1 2 3

$2 = k$
$3 = f$
$n = 3$

2 2 5
1 2 2

3
5
7

100
110
110 + 27.5

120
$\frac{20x}{100}$

110

~~$\frac{68}{49}$~~

10+2
$(37.5)x$

$\frac{63}{484}$

68 + 125
$\frac{193}{174}$

1.556
0.75

112.5

$\frac{57}{76}$

$(37.5)x = (35)y$

$\frac{x}{y} = \frac{35}{375} = 0.9333$

111

$\frac{11}{100}$

$\frac{20x}{100} - ?$ $\frac{x}{100} = 10000$

~~while~~

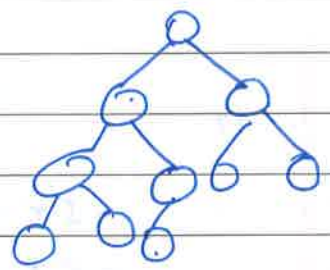
multiset

a) → pair int, int → (price, profit)

b) → int → price

```
while ( k > 0)
    it = a.begin()
    while (a! empty  and  it -> first <= f)
        b.insert (it -> second)
        a.erase (it);   it = a.begin

    if (b is empty) break)
    it = b.end(); it --;
    k--;
    f += value at it;
```



A +

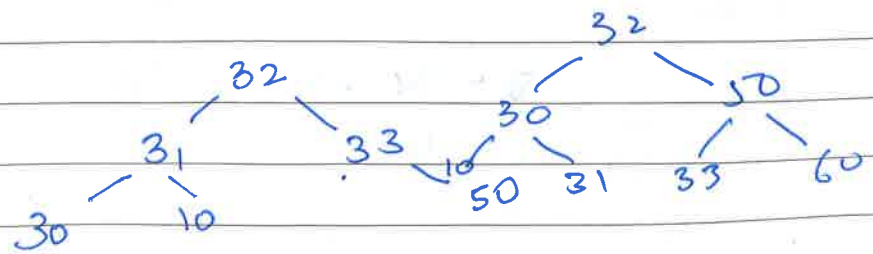
A+ ((B+C) + D)

[[A+] + [[B+C] + D]]

7 → Always

~~10~~ ~~20~~ ~~30~~ ~~31~~ 32 33

~~40~~ ~~50~~ ~~60~~ ~~70~~



$$\overset{S_1}{\frac{x}{6}} = \overset{S_2}{\frac{x}{4}}$$

$B = 22$

$P = 20$

$$t\left(\frac{x}{6}\right) + t\left(\frac{x}{4}\right) = x$$

2 8

3 2

$$t\left(\frac{1}{4} + \frac{1}{6}\right)$$

$$t\left(\frac{10}{24}\right)$$

$t = 2.4$

$= 2$ hrs, 0.4×60

, 24

#

train $\neq$ bike $\forall$ m

bikes == scooters $\forall$ bikes

$25 - 4(-3)$

$25 + 12$

$-5 \pm \sqrt{37}$

$-5 \pm 6.x$

$-11.x$, $1.x$ $\quad$ 11.09

$$\frac{11.x}{1.x} + \frac{1.x}{11.x}$$

$11 + \frac{1}{11}$ $\quad$ 10.909

$8 + 5 \times 6 + 3^2 = (6 + 6 / 6)^2 - (1 + 1 / 1)$

7

$8 + 30 + 9$ $\quad$ $49 - 1 = 48$

47

N-1 N

N-1 > N

6

2 × 3

O O O O O

A B C D E F G

12 H I J K L M N

8 O P Q R S T U

V W X Y Z

12 4 boxes

↳ 4 doen each

8 $40 each

12 Discar % 10%.

8 → 80

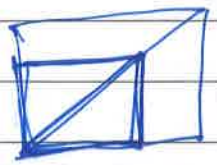
12 → 12

120 8

12

10 8

N



10^3

x $x + \frac{x}{2}$

m 10^5

n 30×10^3

3×10^4

$\frac{m \times n}{p}$ p 3×10^6

$\frac{25}{2}$

$\frac{25}{5}$

5 5 125

F _ _ C _ B

A C GGA GG CT G
GTCGGA GGCA

10^3

$10^6 \rightarrow$

$N \rightarrow 10$

10^7

Tree [n cities
n → bi-di road

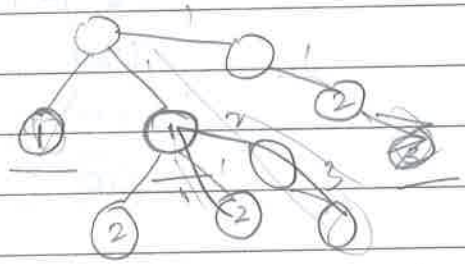
~~CTG~~ ~~3 CGT~~
CTCGTTCATT

a1, a2, ... an
l r x → give # of subsets b/w al & ar such that bitwise XOR is an

10^5

dist b/w

1 6
2 6



2 → (1, 1)
3 → (3, 3)
4 → (3, 4)

A → N ints

S → int

N → 10^6

A_i, S → (-1e9 to 1e9)

A & B → array → N ints

index k → fair if $\left(A[0] + \dots A[k-1]\right) + \left(A[k] + \dots A[N-1]\right)$

B B B B

are all equal

0

$\frac{\Sigma a}{n} = 0$ 0 0

$\Sigma a = 0$

0 | 0

5 1

1 $^6C_2$ 1 2 3

$\frac{6 \times 5}{2} = 15$

$1 + 2 + 3 + 4 + 5 = 15$

0	1	2	3

0	0

0	1	2	n-1 3

0	1	2	3

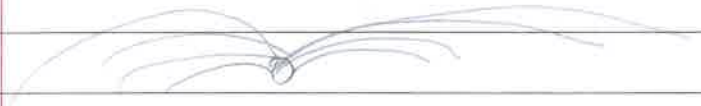
A of length N ($N \in (1, 2\times10^6)$)

$A_i \in [1, 10^6]$

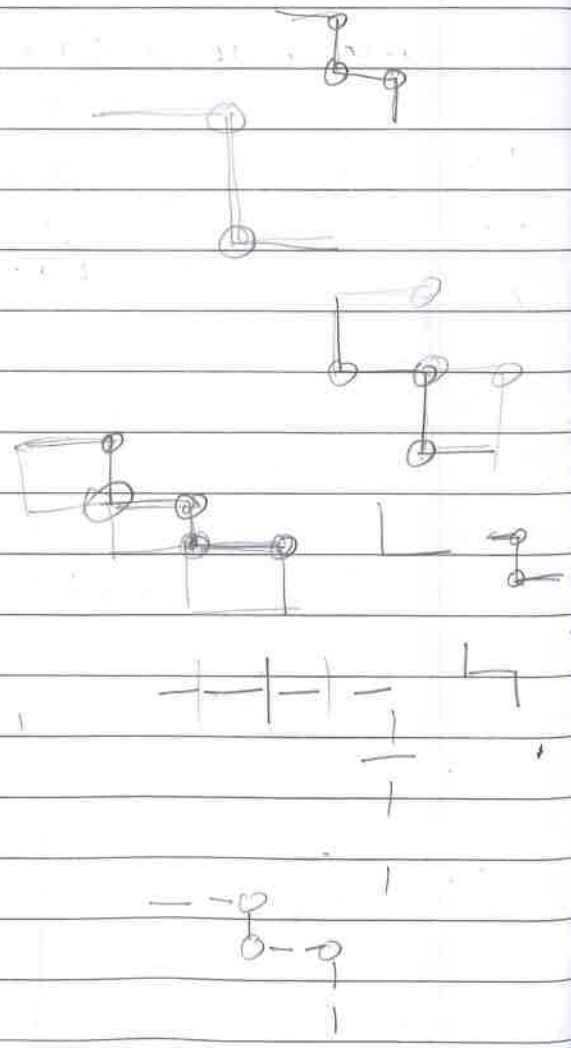
$(N \times N)$

S_x, S_y to E_x, E_y

$N = 1000$



0,0



Ans 3

Case 1 :- c is odd

down steps → b

right steps → a

when c is odd :- we divide a in (c-1) parts & b in (c-1) parts, and alternately place a & b parts in 2 ways

$$\# \text{ of ways} = \begin{cases} 2 \times {}^{a-1}C_{c-2} \times {}^{b-1}C_{c-2} & \text{if } c > 2 \text{ and } c \text{ is odd} \\ 2 & \text{if } c = 1 \end{cases}$$

when c is even : we divide a in (c) parts & b in (c-1) parts and alternatively place a & b parts or vice versa

$$\# \text{ of ways} = \left({}^{a-1}C_{c-1} \times {}^{b-1}C_{c-2}\right) + \left({}^{a-1}C_{c-2} \times {}^{b-1}C_{c-1}\right) \text{ if } c \text{ is even}$$

$$\# \text{ of ways} = \begin{cases} 2 & \text{if } c = 1 \\ 2 \times {}^{a-1}C_{c-2} \times {}^{b-1}C_{c-2} & \text{if } c > 2 \text{ and } c \text{ is odd} \\ \left({}^{a-1}C_{c-1} \times {}^{b-1}C_{c-2}\right) + \left({}^{a-1}C_{c-2} \times {}^{b-1}C_{c-1}\right) & \text{if } c \text{ is even} \end{cases}$$

Ans 4

$f(A,B) = \max\{A,B\} / \min\{A,B\} >= k$

$\max\{A,B\} >= \min\{A,B\} \times k$

a.i.) $\therefore$ $pdf(x) = \begin{cases} \frac{1}{b-a} & , x \in [a,b] \\ 0 & , \text{otherwise} \end{cases}$

we need $x, y \in [a,b]$ such that

$x \geq y$ & $\frac{x}{y} \geq k \Rightarrow x \geq yk$

for every y, x has to be btween yk & 1

probability of getting $x = 1 - yk$ for a particular y

y ranges from 0 to $\frac{1}{k}$

probability $= \int_0^{1/k} (1-yk)\, dy = \frac{1}{k} - k\left(\frac{y^2}{2}\right)_0^{1/k} = \frac{1}{k} - \frac{1}{2} \times \frac{1}{k} = \frac{1}{2}\frac{1}{k}$

Ans $= \frac{1}{2k}$ $\longrightarrow$ ranges from 0 to ∞, need to normalize it to 0 to 1

N Hitman

k buttets → $k < N$

Subarray sum with max K

N Question ($1 \le 105$)

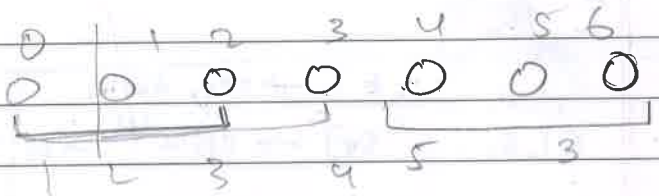
k given mandatory ques $\le 10^5$

each Ques → list of others to be solved

3 — ✓ — 1 ✓

1 - 4 ✓ 2 ✓

5 ✓



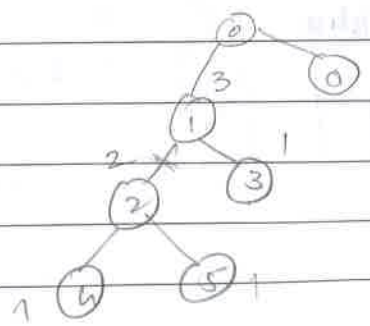
$7 - 3 = 4$

27

55

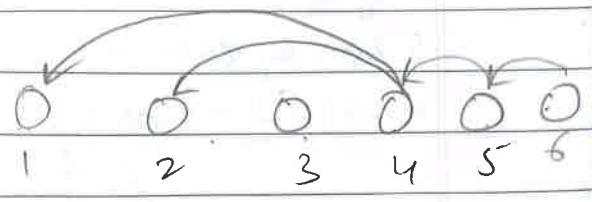
11

parity



N nodes

N-1 edges



N blocks → 1 ingri each from list
→ Req ingri → I
→ To prep dish → D

Group blocks of all I ingridients together → swapping
min swap

$N \in (1, 10^6)$
$A_1, A_2 \ldots A_N$
I

$X_i [x_1 \ldots x_k]$ → req skills $Y [y_1 \ldots y_L]$ → Alice skills
$S [S_1 \ldots S_n]$ → skills All → $N <= 15$

$B [B_1 \ldots B_m]$ ≤ 100 → books → can have ≥ 1 skill
$P_1 \ldots P_m$ → Price & $P_i \leq 100$

→ Total skills
N, M → Total books
k → # req skills
$X_1, \ldots X_k$
L → # skills owned by alice
$Y_1 \ldots Y_m$
~~B_m~~
$P_1 \ldots P_m$
# of skills in B_1
{ list of skills in B_1 } m lines

5, 5
4
1, 2, 3, 4 → req. skills
2
1, 5 → have

40, 30, 60
1
2 3 5
2
4 4
3
2 3 4

classmate

Date

Page

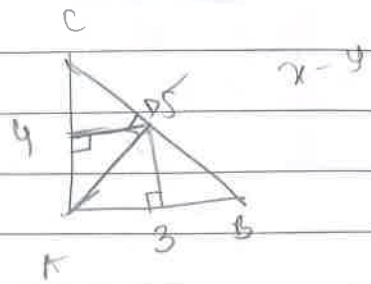
108662

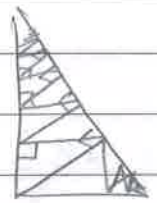
3 non empty ~~ex~~ vectors

→ min

→ max

→ min & max





15

B 100 ~~104~~ 15

2^{15}

set

$142 \times \frac{5}{100}$

15 log 15

$X \to \{2, 3, 4\}$

① → 0.99909990

② → 0.9991003148

③ 0.9991001799

1046

x

57.5

$\frac{5}{100}x + x \to y1$

$\frac{5}{100}\left(\frac{105}{100}x\right) + \frac{105}{100}x$

$\frac{(5 \times 105 + 10500)x}{10^4} \to y2$

$\frac{525 + 10500}{10^4}$

$\left(\frac{11025}{10^4}\right)x$

$\frac{5}{100}\left(\frac{11025}{10^4}x\right) + \frac{11025}{10^4}x$

$\frac{\log 9}{\log 27}$ $\log 9$

$\log 27 = \log 3^3 = 3 \times \log$

$2 \times \log 3 = ?$

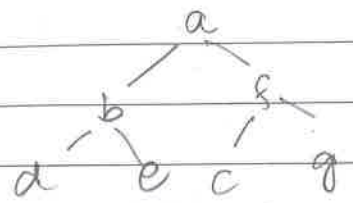
$3 \times \log 3 = \frac{1.431 \times 2}{3}$

$\left(\frac{5525 + 110250}{10^6}\right)x = \frac{1652.862}{3}$

0.954

$\frac{1157625}{10^6}$

~~dbaacfg~~ → in
abdefcg



8 $\log_2 8 = 3$
4
2

debcgfa

$n \times (\log n + \log \log n)$

$p = \log n$
$q = \log \log n$
$n = n$

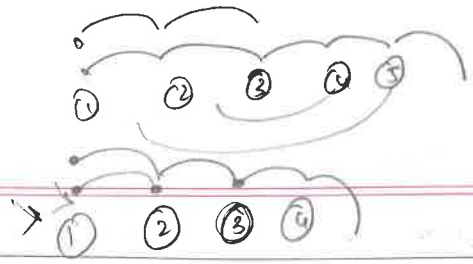
$n \mid \log \log n$

n nodes

$\log_2 37$

2

$b = 2$



1 2 3

6 5 4

16 25 34

1, 2 10

a, b, c d 5

A → 1, 2 — 10

J → 11, Q → 12, K → 13

9 → 2 pts others 0

1 → 1 pt

10 → 1 pt

11 → 3 pts

N → words of letter x, y, z

- rearranging the letters of old word
  - ↳ no char in same position as opp
  - ↳ lexiographically the smallest

xxxyz

xyzyx

yxzxz

N×M → 0 healthy

1 → covid

~~14, 10, 9, 8, 7, 4~~

~~55~~

15 50

35

50

−35

15

50

65

$$\frac{50 \times 13 + 65}{14}$$

$$\frac{650 + 65}{14} = 51.071$$

B → $P(B) = \frac{3}{8}$, $P(B') = \frac{5}{8}$

$P(C) = \frac{23}{88}$, $P(C') = \frac{65}{88}$

$\frac{3}{8} + \frac{23}{88} + P(A) = 1$

$\frac{33 + 23}{88} + P(A) = 1$

56 $P(A) = \frac{88 - 56}{88} = \frac{32}{88}$

~~14~~ 7

~~28~~

~~56~~

~~32~~

~~16~~ ~~8~~ 4

$$\begin{array}{r} 29 \\ -13 \\ \hline 11 \end{array}$$

$-6 \quad -7 + 24$

1 ✓

$\cancel{2} \quad -10 \quad -14 \quad +24$

$$\begin{array}{r} 24 \\ -20 \\ \hline 0 \end{array}$$

3

$3(-4$

$-12 \quad -21 + 24$

$n^2 - \cancel{7n} + 24 > \cancel{7n}$

$n^2 + 24 > 14n$

n^2

$^{10}C_4$

$\frac{1}{2} \times$

$$\begin{array}{r} 10! \\ \hline 4!\,6! \end{array}$$

$1x \quad 1x$

$\frac{1x}{2x} = \frac{1}{2}$

^{2x}C

$\frac{1x}{2x} \times$

$\frac{^{x}C_2}{^{2x}C_2}$

$\frac{x(x-1)}{2x(2x-1)} = \frac{7}{15}$

$15x - \cancel{15} = 28x - \cancel{14}$

$\frac{x(x-1)}{2x(2x-1)} = \frac{7}{15}$

$15x - 15 = 28x - 14$

~~S T u v w x~~

S	Q	~~T~~	P	~~R~~	U	W	V
4	2	1	3	1	6	8	1

S Q T P R U W V
4 2 1 3 1 6 8 1

N nodes, N-1 edges
root → 1

each node → char

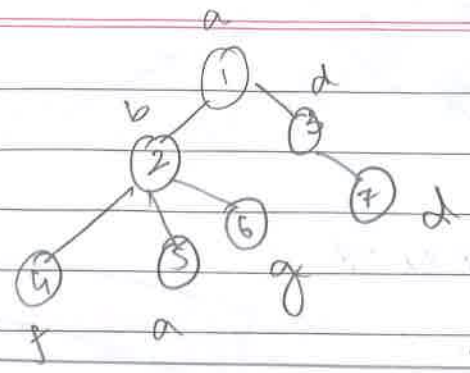
Q queries → M → node #
S → string

X → level order traversal of subtree M
obtain S

Find min # of chars to be added to X to get S

$N, Q \leq 10^5$

$\Sigma |s| \leq 10^6$



$\text{lcs}(u,v) = x$

$\text{lcs}(m,n) = y$

String S → 0 to 9

partition in parts → # formed ≤ k &
Σ nums → max

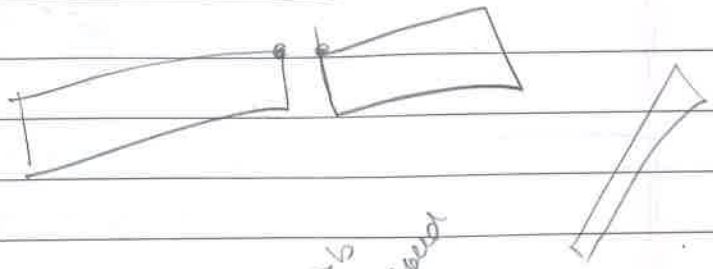
also
find # ways → max sum
% 1e9+7
if not (0,0) → output

$T \le 30$

$\Sigma |s| <= 3 \times 10^3$

$k \in [0, 10^{12}]$

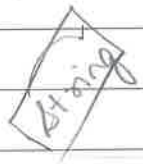
Array ints → 10 6
10a



a → b
given speed

private
color

getter - setter
function



Ans:
company specific

string → ( )

book → output → valid or not

$(\{(\ )^3\})$

$N \to 10^6$

Σ

( )

A → N elements

$A_i \in [1, N]$

0 1 2 3 4 5 6

1 2 2 3 4

A 1 to N →

M

$1 \wedge 2 \wedge 3$

$1 \wedge 2 \wedge 2$

$2 \wedge 5$

$\boxed{10^5}$

$2 \wedge 2^5$

Intro

- Why do you want to join :- Company & Profile Specific
  ↳ values ?

- Why should we take you :-
  / Your qualities / Your strengths / Your skills
- Any significant skills apart from CV

- Tell about your story

- What are your weakness :-

- Where do you see yourself in 5 years? Your post after 5 years.

- Greatest Achievement so far ?

- Challenge — Illu

STAR → Situation
Task
Action
Result

3 boxes
↳ green & blue balls ⊢ wrongly named

→ one only blue
" " green
" mix

B
G & B → B | G seen
Green → G & B | Blue
Blue → Green | G & B

1 2 3 4 5 6 7
~~P~~, ~~Q~~, R, S, T, U, & ~~V~~ → finish

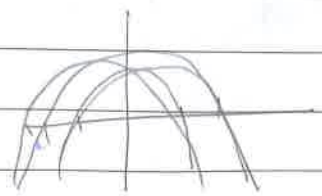
→ No tie

7	6	5
Q	P	V

ax^2+bx+c & $a<0$

$f(x)=0$ has 2 distinct roots $\{-2, -1, 0, 1, 2\}$

2×2



Tank, 3 pipes $\overset{1}{O} + \underset{2}{O} = O^3$

1 + 5 hrs = 2
2 + 4 hrs = 3

~~1 + 2 = 3~~
~~1 + 1 + 5 = 2 + 4~~
~~1 + 5~~

~~1 + 5 hrs =~~

2 + 5 hrs = 1

3 + 4 hrs = 2

40×39

~~780~~

~~1560~~

~~2~~

7400

700

40×39

1560

A → 6500 for 6 months

B → 8400 " 5 "

C → 10,000 " 3 "

$\frac{7400 \times 5}{100}$ 370

$6500 \times \frac{6x}{100} = 370$

$x = \frac{370}{65} \times \frac{1}{6}$

$\frac{8400 \times 5}{100} \times \frac{370}{65} \times \frac{1}{6}$

100 → 20% absent, 80 present

$\frac{30}{100} \times 80$ → busy

$\frac{80}{100} x \times \frac{70}{100} \times \frac{84}{100} = 112$

y, P, I D

A	B	C	D	E	F	G	H
I	J	K	L	M	N	O	P
Q	R	S	T	U	V	W	X
Y	Z						

x km

k km → y km/h

z km/h

$\frac{k}{y}$

+ve int → n → find smallest int > n with same

1 3 4 6 5 13 6 4 5

112

121

$n = (2, 10^3)$

$A_i \in [1, 10^3]$

5 3

3 - 0

5 - 0 > 3 - 0

DP(L, R)

N children

A_i → price seq

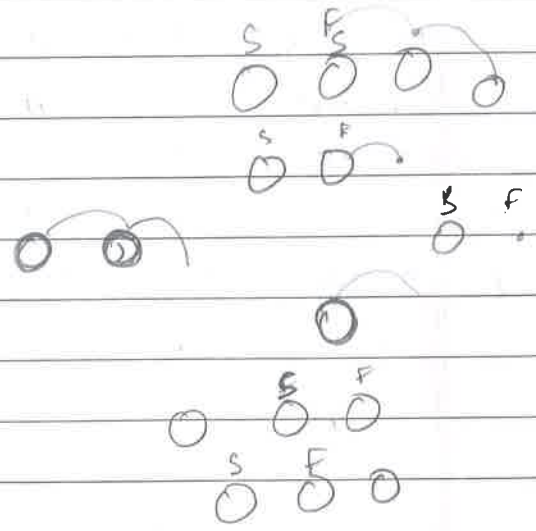
B_i → goodness "

~~each child >= A_i~~

~~>= B_i~~

M Gifts → C_i Price

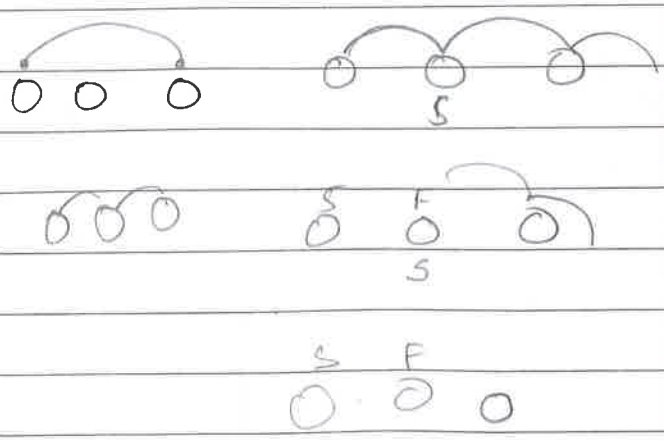
D_i Goodness

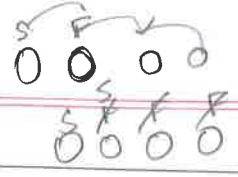


$N, M \in 1, 10^6$

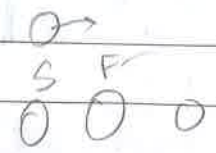
$A_i, B_i, C_i, D_i \in 1, 10^9$

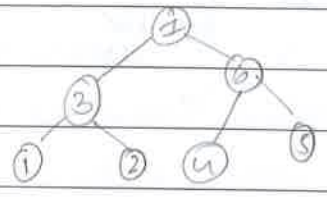
i value ≤ K → pure





$n = 2 \times 10^7$





2, 4, 6

15

$2 \times _ \leq 15$

$_ \leq \frac{15}{2}$ 7.5

~~4i~~ $A_i \in (1, 10^4)$

Array of int $\longrightarrow$ 10^4 Array

Greatest sum of divisible by 3.

i, 0 - 2

(2, 4, 6) $\rightarrow$ 3

$\leq 3 \rightarrow 1$

6 <

2 4 6

2 4 6

Module 0

24/25 → Applied learning of C++

1 2 3 4 (tricky) 5 ‖ 6 (tricky) 7 8 9 10 ‖ 11 12 13 (=18)

14 15 (tricky) ‖ 16 17 (imp) 18 19 20 ‖ 21 22 23 (tricky) 24 25 ×

#• sqrt(n) might give rounding off error → avoid using it as much as possible

• abs(n) gives absolute value of n

• "cout << fixed << setprecision(n) << variable" used to set precision of n

• 1e9+7 modulo generally used

→ Applied learning of STL

1 2 3 4 5 ‖ 1 2 (imp) 3 (imp) 4 5 ‖ 1 2 3 ×

24/25 4 5 ‖ 1 2 3 4 (good) 5 ‖ 1 2 3 (good) 4 5

#• multiset.erase(key) erases all occurances of key. To erase one occurance use multiset.erase(multiset.find(key))

• multiset count(key) gives the count of key.

→ Mixed Practice

14 1 2 3 (imp) 4 5 ‖ 1 2 3 (tricky) 4 (imp) ‖ 1 2 3 (imp)

4 ‖ 1 (imp) 2 3 4 ‖ 1 2 3 4 5

Module 1 w1

→ STL Application Ideas

1 2 3 4 5 6 7 8 9 10

11 12 13 14 15 16 17 18 19

→ Foundational Maths

1 2 3 4 5 6 7 8

#• $^nC_r + {}^nC_{r-1} = {}^{n+1}C_r$ • $a^b \% p = (a^{b\%(p-1)}) \% p$ if $a \nmid p$ & p is prime

• $(a^{-1}) \% p = (a^{p-2}) \% p$ if p is prime & $a \nmid p$

Module 1 W2

→ Partial and Prefix Sums — Advanced ×

5 — 1 2 3 (imp) || 1 2 || 1 2 3 4 5

• For prefix sum, always maintain a -1th position = 0

→ Divide and conquer — 1 2 3

• vector can be directly copied by = if size is same

→ Binary Search — unique — ask Esmita

10 — 1 2 3 4 5 6 7 8 9 10 11

• in problems like maximize the min neighbor distance, we have to find the last zero, not the first 1 (Q10)

• (Q7) Counter Example :- n=9, k=3, A(i) = 0, 2 2 2 3 3 4 5 5

• recognizing BS → where O(logN) is required → maximize / minimize

Module 1 W3

→ Mathematics Finale

1 2 3 4 5 6 7 8 || 1 2 3 4 5 6 || 1 2

Module 1 W4

→ Two pointers — tricky — similar

10 — 1 2 3 4 5 6 7 8 9 10

• Use "LLONG_MAX" for LL & "INT_MAX" for int

→ ~~Bit Manipulation~~ Bit Manipulation

7 — 1 2 3 || 1 2 3 4

• -ve numbers bit representation can be checked by num & (1<<i)

• For 2^n or n/2 → use fast expo % p as n can go beyond 63 and result in overflow (avoid use of 1<<i if i > 62)

Module 2 W6

→ Brute Force & Backtracking

8 — 1 2 3 4 5 6 7 8 9 10 11

• $|2!| < 1e9 < |3!|$ • $9! = 3 \times 10^5$ • $11! = 4 \times 10^7$

4 → Meet in the Middle ~~1~~ ~~2~~ ~~3~~ ** ~~4~~