Competitive Programming

⇒ IDE Setup for CP online Judges for Java/C++.

⇒ Fast Input/Output in Java & C++

⇒ Template for C++ & Java coders

⇒ Pre-requisities & Syllabus to be covered!

⇒ Books, notes and coding practice/contests platform

Fast Input/output

```java
public class Main {
    static class FastReader {...}

    static class FastWriter {...}

    static FastReader in = new FastReader();
    static FastWriter out = new FastWriter();

    static {
        if (System.getProperty("ONLINE_JUDGE") == null) {
            try {
                System.setOut(new PrintStream(
                        new FileOutputStream("output.txt")));
                System.setIn(new FileInputStream("input.txt"));
            } catch (Exception e) {
            }
        }
    }

    public static void main(String[] args) {
        try {
            int testCases = in.nextInt();
            while (testCases-- > 0) {
                solve();
            }
            out.close();
        } catch (Exception e) {
        }

    }

    public static void solve() {

    }
}
```

Time Complexity & Space Complexity Analysis.

Big Integers in Java

Standard Template Library in C++

or Java collection framework in Java

Floor Division

$10\%3 \neq 0$ {

$\lfloor 10/3 \rfloor = 3$

$\lceil 10/3 \rceil = 4 = 3+1$

$10\%2 == 0$ {

$\lfloor 10/2 \rfloor = 5$

$\lceil 10/2 \rceil = 5 = 5+0$

Syllabus

(1) Introductn, Template, Fast Input/output, Big Integers

(2) Time & Space Complexity, Collectn Frameworks / Standard Template library

(3) Bit manipulatn (4) Number Theory & Maths (5) Modular Arithmetic

(6) Range Query → Segment Tree, Fenwick Tree, SQRT decomposition

(7) String matching algo → kmp algo, Z algo, Rabin Karp (Rolling Hash)

(8) Dynamic Programming → Digit DP, DP with Bitmasking, DP on tree

(9) Game Theory (10) Geometry (11) PBDS (12) Interactive Problems

(13) Advanced Trees (14) Ternary search (15) Advanced Graphs

Time & Space Complexity

Online Judge ⇒ Time limit ⇒ 1 sec = 10^8 operations

Time Complexity = Rate of Runtime wrt input

↳ usually worst case

10^8 op ⇒ 1s

10^9 op ⇒ 10s

10^{10} op ⇒ 100s

10^{11} op ⇒ 1000s

10^{12} op ⇒ 10000s

Time & Space Complexity

online Judge ⇒ Time limit ⇒ 1 sec = 10^8 operations

Time Complexity = Rate of Runtime wrt input

↳ usually worst case

10^8 op ⇒ 1s

10^9 op ⇒ 10s

10^{10} op ⇒ 100s

10^{11} op ⇒ 1000s

10^{12} op ⇒ 10000s

Time Complexity Trend

$O(1) < O(\log_2 N) < O(\sqrt{N}) < O(N) < O(N * \log N)$

constant — logarithmic — root = $n^{1/2}$ — linear

$N_{max} = 10^{16}$ — $N_{max} = 10^8$ — $N_{max} = 10^5$ to 10^6

$< O(n^2) < O(n^3) <<< O(2^N) < O(3^N) < O(N^N)$ or $O(N!)$

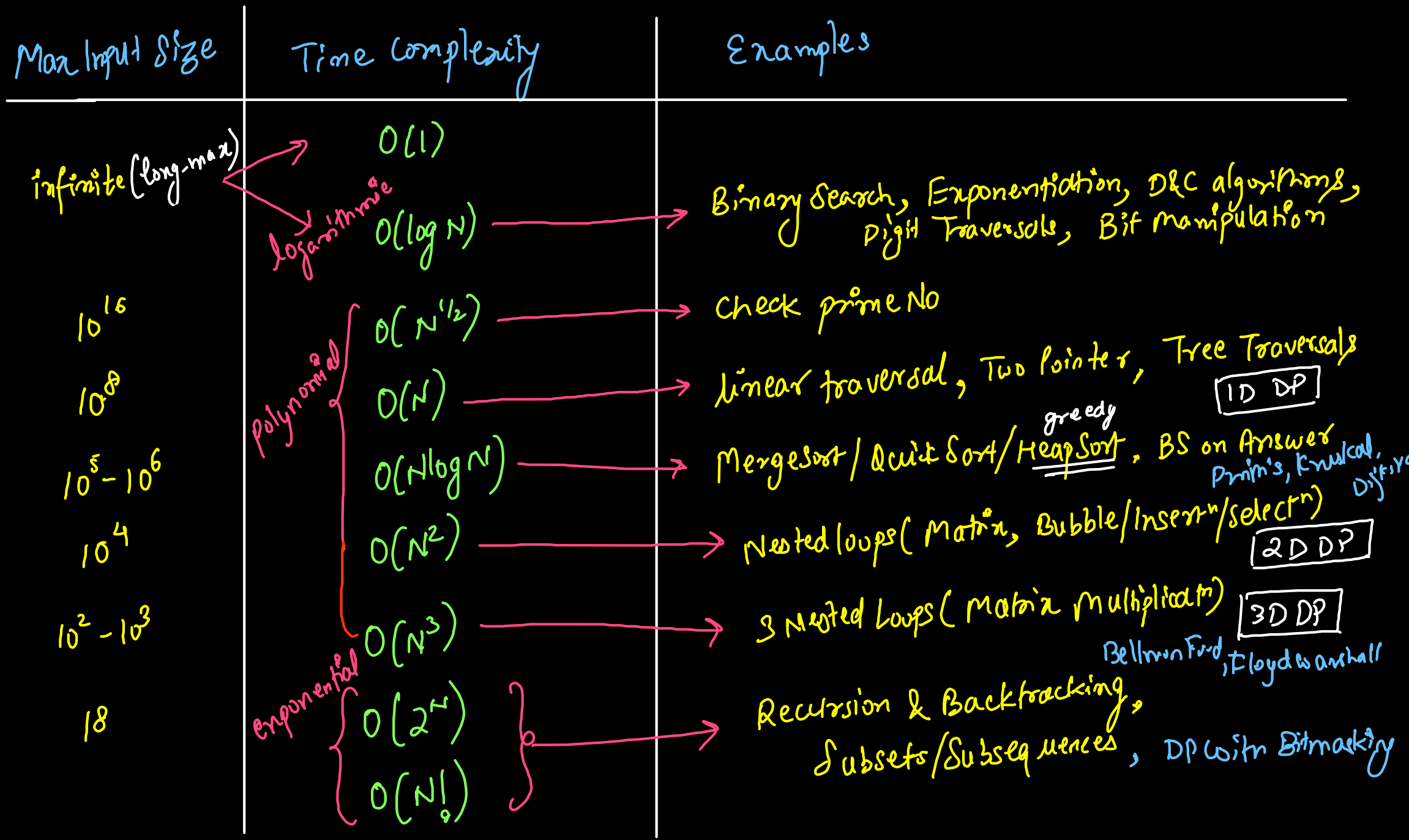
quadratic — cubic

$N_{max} = 10^4$ — $N_{max} = 500 - 600$

$N = 32$ → $N = 18$

polynomial

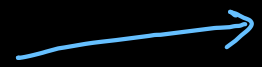
exponential


Max Input Size
Time Complexity
Examples
infinite (long-max)
O(1)
Logarithmic
O(log N)
Binary Search, Exponentiation, D&C algorithms, Digit Traversals, Bit manipulation
10^{16}
10^{8}
$10^{5} - 10^{6}$
10^{4}
$10^{2} - 10^{3}$
18
Polynomial
$O(N^{1/2})$
Check prime No
O(N)
linear traversal, Two Pointer, Tree Traversals
1D DP
$O(N \log N)$
Mergesort / Quick Sort / HeapSort
greedy
, BS on Answer
Prim's, Kruskal, Dijkstra
$O(N^2)$
Nested loops (Matrix, Bubble/Insertn/selectn)
2D DP
$O(N^3)$
3 Nested Loops (Matrix multiplicatn)
3D DP
Bellman Ford, Floyd Warshall
Exponential
$O(2^N)$
$O(N!)$
Recursion & Backtracking, Subsets/Subsequences, DP with Bitmasking

Is A relationship
(Inheritance)

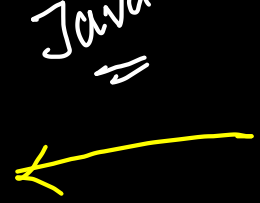
class Car{
}

class BMW extends Car{
}



List{
}

class Stack implements List{
}



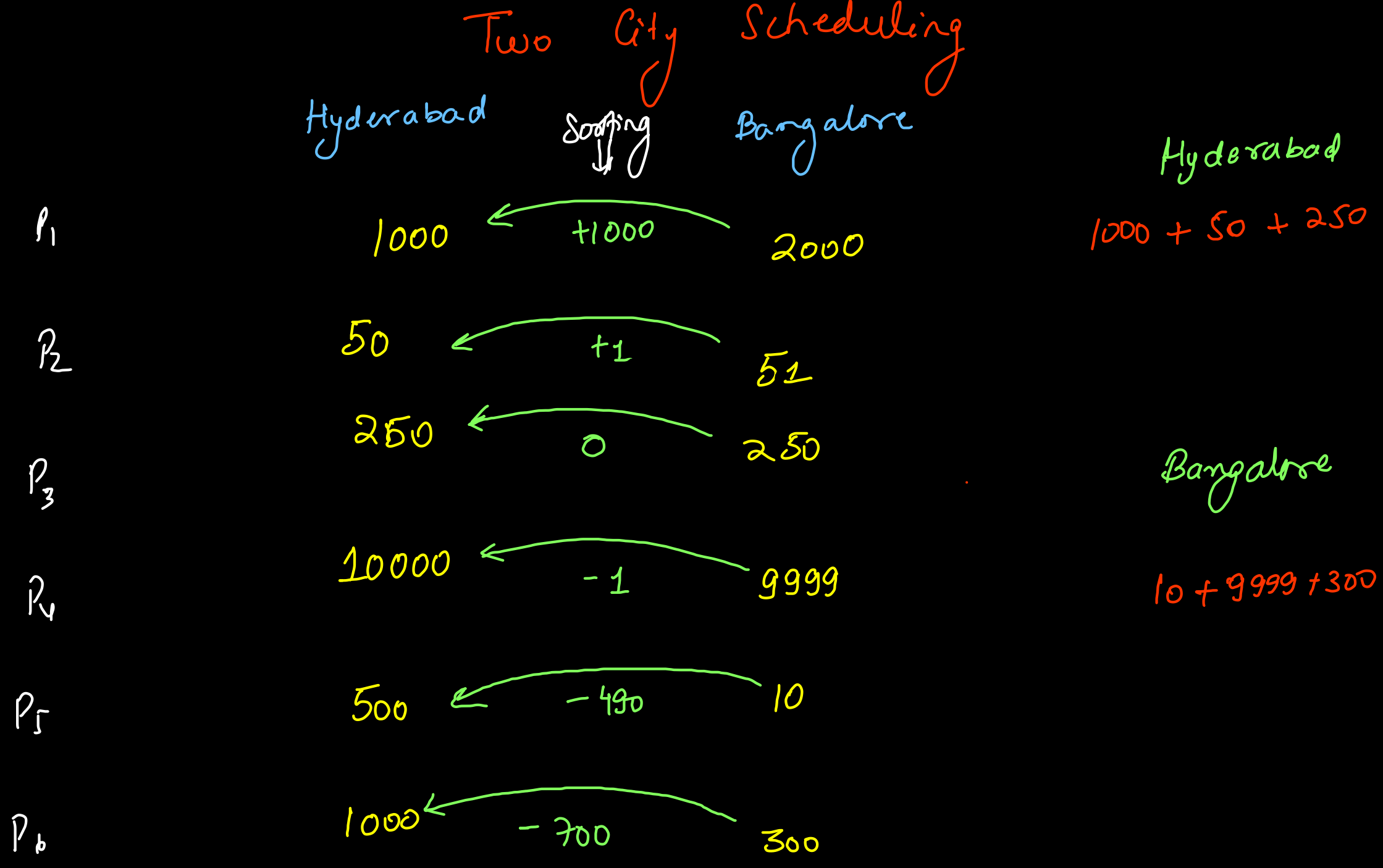
Has A relationship
(Composition)

class Car{
Engine engine;
}

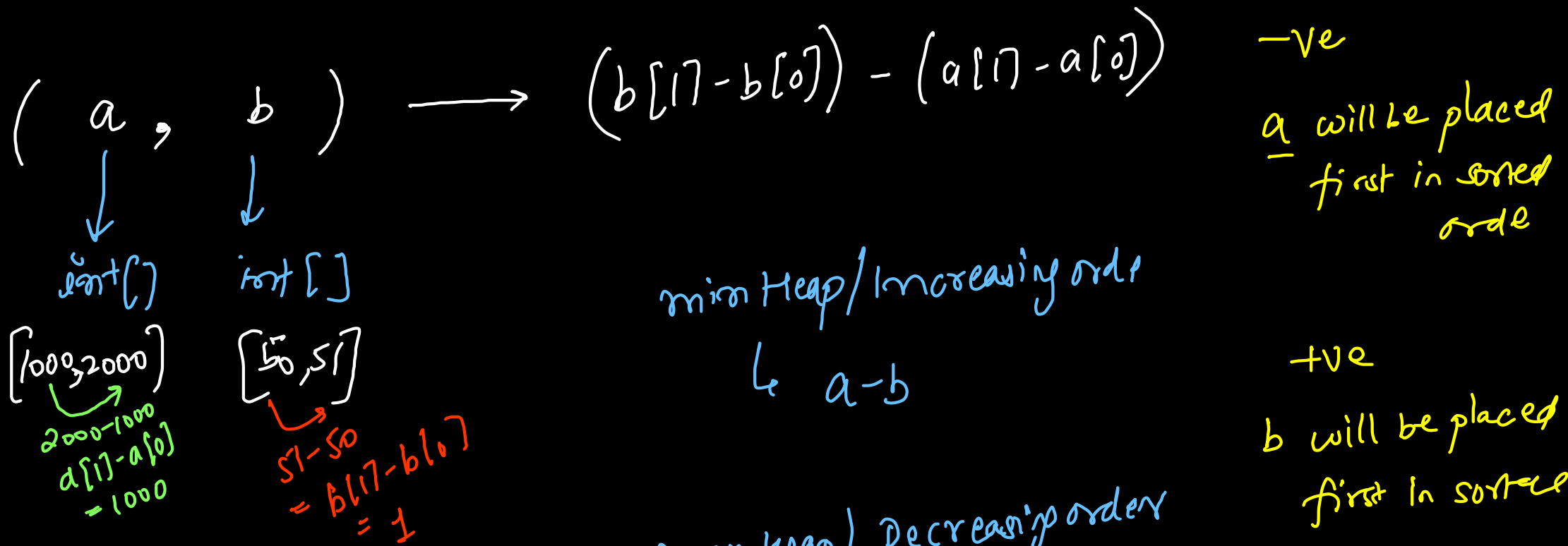
class Engine{
}

class Stack{
ArrayList<Integer>
}

Two City Scheduling



Comparator



min Heap / Increasing order

↳ a-b

max Heap / Decreasing order

↳ b-a

-ve

a will be placed first in sorted order

+ve

b will be placed first in sorted

0

a = b

```java
class Solution {
    public int twoCitySchedCost(int[][] costs) {
        Arrays.sort(costs, (a, b) -> (b[1] - b[0]) - (a[1] - a[0]));

        int total = 0;
        for(int idx = 0; idx < costs.length/2; idx++)
            total += costs[idx][0];
        for(int idx = costs.length/2; idx < costs.length; idx++)
            total += costs[idx][1];

        return total;
    }
}
```

Sort based on profits / Hyd vs Banj

Take people with max profit in city A (Hyderabad)

Take remaining people in city B (Bangalore)

Largest Number

Given a list of non-negative integers `nums`, arrange them such that they form the largest number and return it.

Since the result may be very large, so you need to return a string instead of an integer.

Example 1:

```
Input: nums = [10,2]
Output: "210"
```

Example 2:

```
Input: nums = [3,30,34,5,9]
Output: "9534330"
```

210

199

3 ✓

99 ✓

21

192

19

91 ✓

60 ✓

99916032121019919219

21 + 210 = 21210

210 + 21 = 21021

199 + 192 + 19 = 19919219 ✓

199 + 19 + 192 = 19919192

19 + 199 + 192 = 19199192

```java
public String largestNumber(int[] nums) {
    String[] arr = new String[nums.length];
    for(int idx = 0; idx < nums.length; idx++){
        arr[idx] = Integer.toString(nums[idx]);
    }

    Arrays.sort(arr, (a, b) -> (b + a).compareTo(a + b));

    StringBuilder res = new StringBuilder();
    for(String str: arr) res.append(str);

    if(res.charAt(0) == '0') return "0";
    return res.toString();
}
```

arr:

6 210

7 a = "199"

4 3

1 b = "99"

5 21

8 192

9 19

2 91

3 60

b+a a+b

99199 > 19999

true

210+21 21+210

21021 21210

"000"

[0,0,0]

a = "21"

b = "210"

b+a

"210" + "21"

"21021"

a+b

"21" + "210"

"21210"

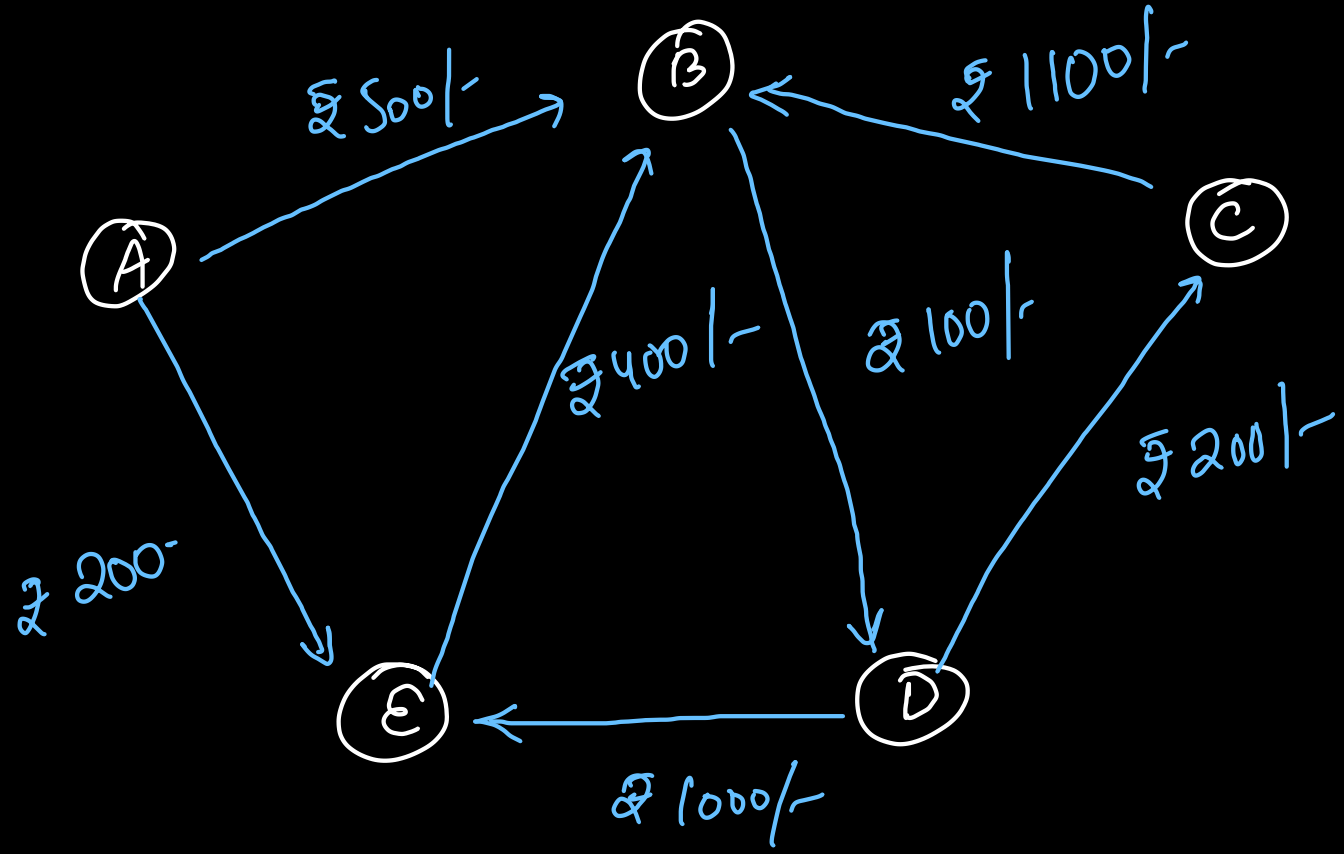
a = "199"

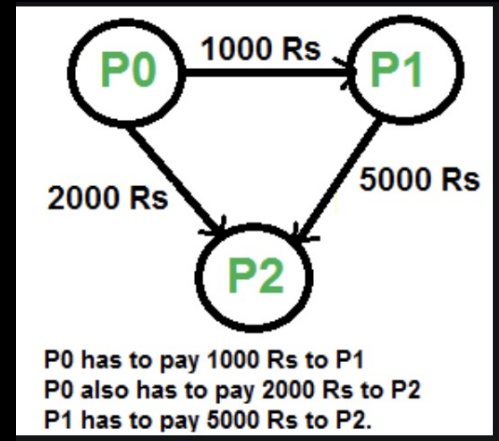
b = "19"

a+b
= 19919

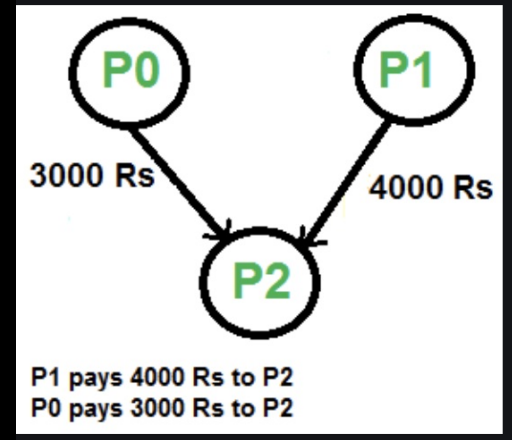
b+a
= 19199

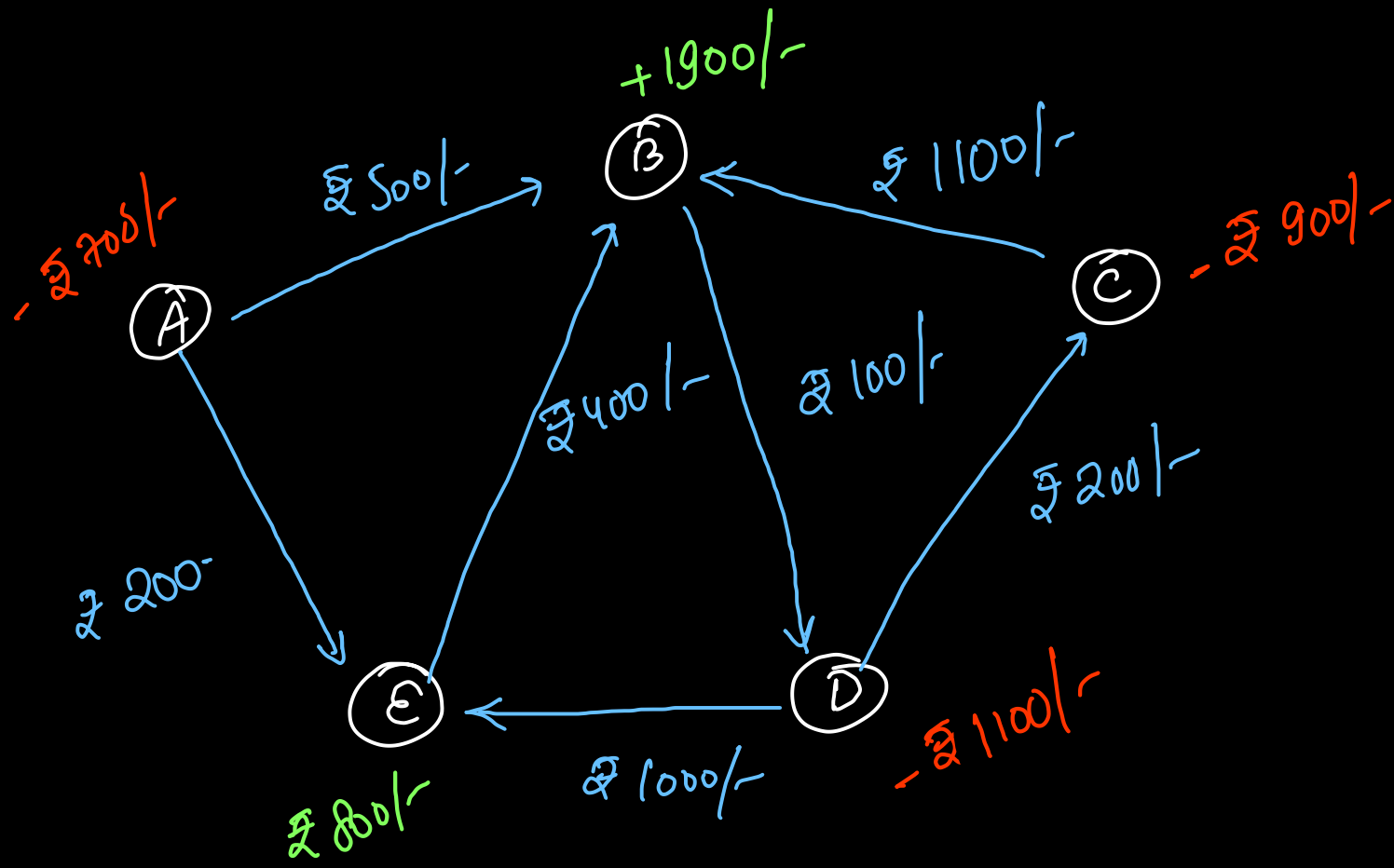
Minimizing Cash flow





Above debts can be settled in following optimized way





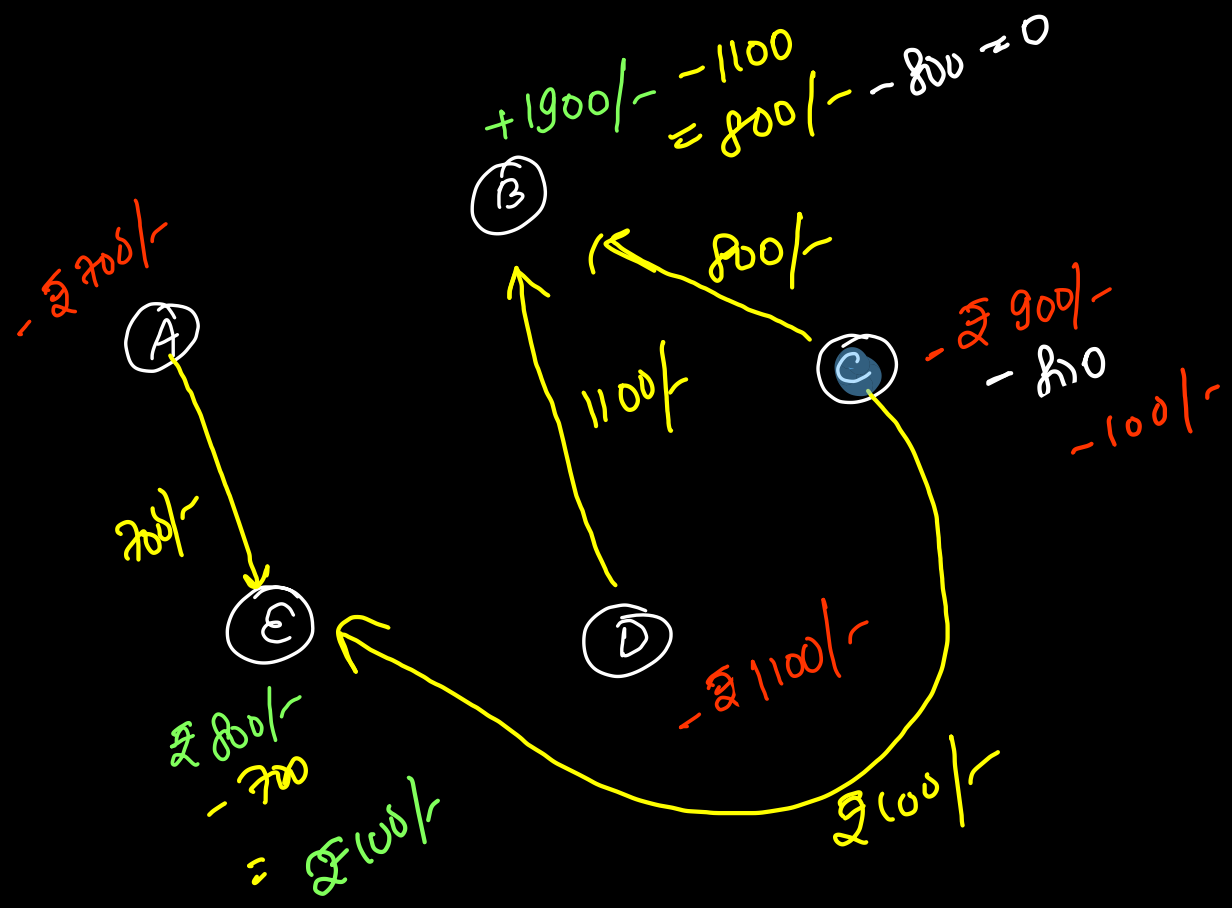
700 + 900 + 1100

=

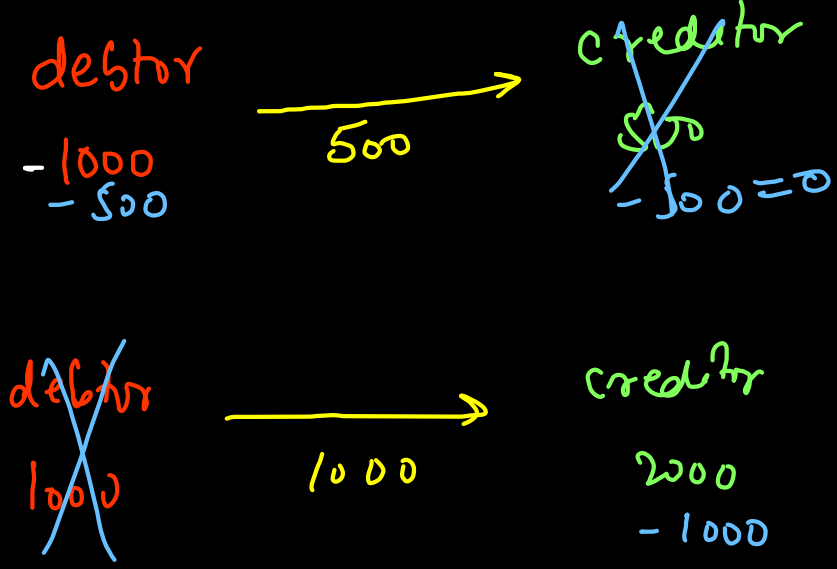
1900 + 800

Green → Max heap (creditor)

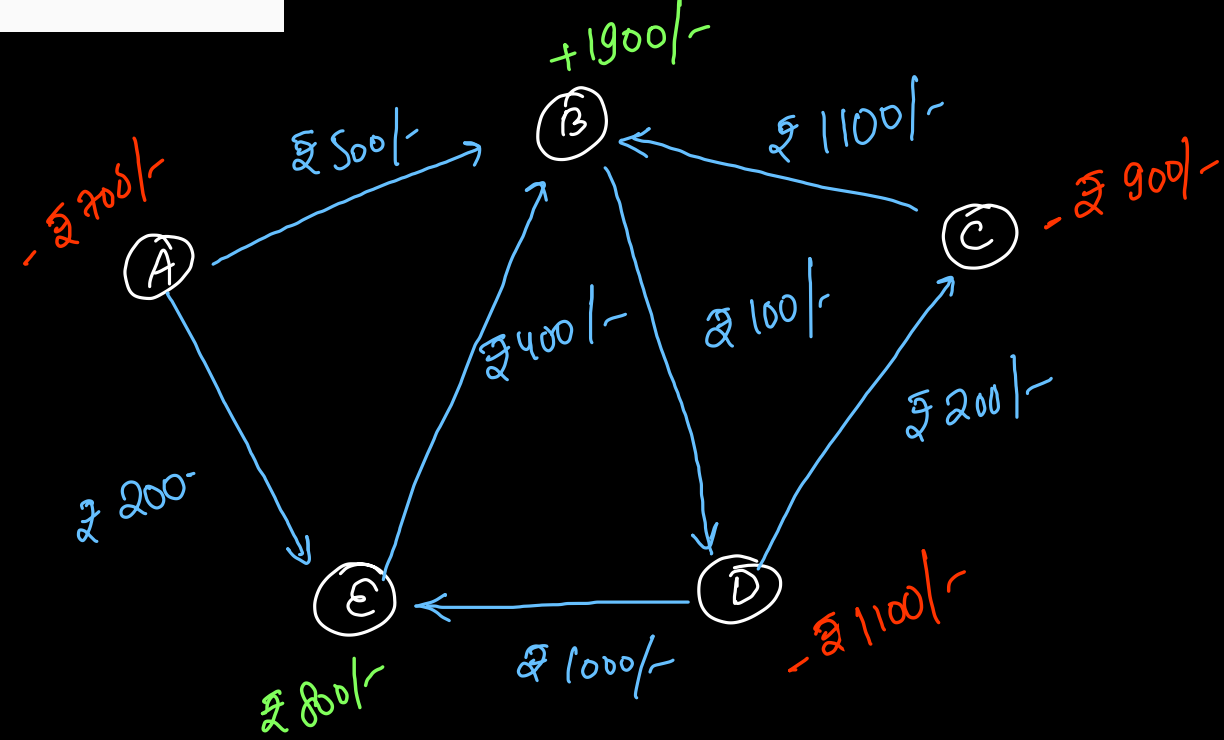
Red → Min heap (debtor)


+1900/- −1100
= 800/- −800 = 0
B
800/-
A
-₹700/-
C
-₹900/-
−800
-100/-
1100/-
700/-
E
D
-₹1100/-
₹800/-
−700
= ₹100/-
₹100/-

```
int[][] res = new int[n][n];
while(debtor.size() > 0 && creditor.size() > 0){
    int debt = debtor.remove();
    int cred = debtor.remove();

    res[debt][cred] = Math.min(-net[debt], net[cred]);

    if(-net[debt] > net[cred]){

    } else {

    }
}
```



```java
int[] net = new int[n];
for(int r = 0; r < n; r++){
    for(int c = 0; c < n; c++){
        net[r] -= mat[r][c];
        net[c] += mat[r][c];
    }
}

PriorityQueue<Integer> debtor = new PriorityQueue<>(); // min heap
PriorityQueue<Integer> creditor = new PriorityQueue<>(Collections.reverseOrder());

for(int idx = 0; idx < n; idx++){
    if(net[idx] > 0) creditor.add(idx);
    else if(net[idx] < 0) debtor.add(idx);
}
```



Iterate

① Index Based Loop

② foreach for(int val: —)

③ Lamba Expression for each Arrays.forEach(——);

Data Structures (Sorted)

① TreeSet/ TreeMap (BST)

② PriorityQueue (Heap)