17/NOV/21

STRIVER SHEET :- 180 Q.

DAY :- 1 :- Set Matrix Zero

Statement :- Given a mxn integer matrix, if an element is 0, set its entire row and column to 0's and return the matrix.

Ex :-

I/P :-	1	1	1
	1	0	1
	1	1	1

O/P :-	1	0	1
	0	0	0
	1	0	1

Code :-

```
class Solution {
    public:
    void setZeroes(vector<vector<int>>& matrix) {
        int col0 = 1, rows = matrix.size(), cols = matrix[0].size();

        for(int i = 0; i < rows; i++) {
            if (matrix[i][0] == 0) col0 = 0;
            for(int j = 1; j < cols; j++)
                if (matrix[i][j] == 0)
                    matrix[i][0] = matrix[0][j] = 0;
        }

        for(int i = rows - 1; i >= 0; i--) {
            for(int j = cols - 1; j >= 1; j--)
                if (matrix[i][0] == 0 || matrix[0][j] == 0)
                    matrix[i][j] = 0;
            if (col0 == 0) matrix[i][0] = 0;
        }
    }
};
```

Solution: 3 (optimised Approach) :-

Step 1 :- linearly traverse array from backward such that i^{th} index value of the array is less than $(i+1)^{th}$ index value. Store that index in a variable.

Step 2 :- If the index value received from step1 is less than 0. This means the given input array is the largest lexicographical permutation. Hence, we will reverse the input array to get the minimum as starting permutation. linearly traverse array from backward. Find an index that has a value greater than previously found index. Store index another variable.

Step 3 :- Reverse array from index+1 where the index is found at step 1 till the end of the array.

Code :-

```
class Solution {
public:
    void nextPermutations(vector<int>& nums) {
        int n=nums.size(), k, l;
        for (k=n-2 ; k>=0; k--) {  //traverse from second last
            if (nums[k]<nums[k+1]){ index.
            break;
        }
    }
    if(k<0) {
        reverse (nums.begin() , nums.end());
    }
    else {
        for (l=n-1 ; l>k; l--) {
            if (nums[l]>nums[k]){
            break;
        }
    }
    swap (nums[k] , nums[l]);
    reverse (nums.begin()+k+1 , nums.end());
}
};
```

Question :- 3 NEXT PERMUTATION

Statement :- Given an array Arr[] of integers, rearrange the NO. of the given array into the lexicographically next greater permutation of numbers.

If such an arrangement is not possible, it must rearrange it as the lowest possible order. (i.e, Sorted in ascending order).

Ex :- I/P :- Arr[] = {1,3,2}
O/P :- Arr[] = {2,1,3}

Explanation :- All permutations of {1,2,3} are {1,2,3}, {1,3,2}, {2,1,3}, {2,3,1}, {3,1,2}, {3,2,1}. So the next generation just after {1,3,2} is {2,1,3}.

Ex :- I/P :- Arr[] = {3,2,1}
O/P :- Arr[] = {1,2,3}

Explanation :- As we see all permutations of {1,2,3}. we find {3,2,1} at the last position. So we have to return the top most permutation.

= Solution 1 (Brute Force :) i) Find all possible permutations of elements present and store them.

(ii) Search Input from all possible permutations.
(iii) Print the next permutation present right after it.

T.C :- O(N! XN)
S.C :- O(1)

= Solution 2 (Better Approach) :- using c++ in built function.

```
int main() {
    int arr[] = {1,3,2};
    next_permutation(arr, arr+3);
    cout << arr[0] << " " << arr[1] << " " << arr[2];
    return 0;
}
```

This problem is a variation of the popular Dutch National flag Algorithm.

Approach :-

(i) Initialize the 3 pointers such that low, mid will point to 0th index and high pointer will point to last index.

```
int low = arr[0]
int mid = arr[0]
int high = arr[n-1]
```

(ii) Now there will be 3 different operations based on the value of arr[mid] and will be repeated until mid <= high.

(a) arr[mid] == 0;
Swap (arr[low], arr[mid])
low++ ; mid++

(b) arr[mid] == 1;
mid++ ;

(c) arr[mid] == 2;
Swap (arr[mid], arr[high])
high-- ;

Code :-

```
Class Solution {
public:
    void SortColors (vector<int>& nums) {
     int low = 0;
     int high = nums.size()-1 ;
     int mid = 0;

     while (mid <= high) {
        switch (nums[mid]) {
     // if the element is 0
     case 0:
```

Question 5) Sort an array of 0's 1's 2's :-

Statement :- Given an array consisting of only 0's, 1's and 2's. write a program to in-place sort the array without using inbuilt sort function.
(Expected :- Single pass - O(N) and constant space).

Ex :- i) **Input :-** nums = [2,0,2,1,1,0]
output = [0,0,1,1,2,2]

ii) **Input :** nums = [2,0,1]
output = [0,1,2]

iii) **Input :** num = [0]
output = [0]

= **Solution 1 (Brute force) :-** Sorting T.C O(NlogN)
S.C O(1)

= **Solution 2 :- (Keeping count of values) :-** Approach :-

(i) Take 3 variables to maintain the count of 0,1,2
(ii) Travel the array once and increment the corresponding counting variables.
[count-0 = a, count-1 = b, count-2 = c]
(iii) In 2nd traversal of array, we will now overwrite the first 'a' indices/ positions in array with '0', the next 'b' with '1' and the remaining 'c' with '2'.

T.C :- O(N)+O(N)
S.C :- O(1)

= **Solution 3 :- 3- pointer Approach** (most optimis

T.C :- O(N)

S.C :- O(1)

M.Imp :

Question :- 4 KADANE'S ALGORITHM :-

Statements :- Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

A subarray is a contiguous part of an array.

Ex :- 1) Input :- nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

output :- 6

Explanation :- [4, -1, 2, 1] has the largest sum = 6.

2) Input :- nums = [1]

output :- 1

3) Input :- nums = [5, 4, -1, 7, 8]

Output :- 23.

Code :-

```
class Solution {
    public:
        int maxSubArray(vector<int>& nums) {
        int sum = 0;
        int maxi = INT_MIN;
        for(auto it : nums) {
            sum += it;
            maxi = max(sum, maxi);
            if (sum<0) sum = 0;
        }
        return maxi;
    }
};
```

```
        swap(nums[low++], nums[mid++]);
        break;

    // If the element is 1
    case 1:
        mid++;
        break;
    // If the element is 2
    case 2:
        swap(nums[mid], nums[high--]);
        break;
    }
  }
}
```

Question :- 6 **Stock Buy and Sell**

Statement:- You are given array prices where prices[i] is the price of a given stock on the i^{th} day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Eg:- Input:- prices = [7,1,5,3,6,4]

Question No-8 Merge Overlapping Sub-intervals

Statement :- Given an array of intervals where intervals[i] = [start_i, end_i], merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

ex:-1) I/P :- intervals :- [[1,3],[2,6],[8,10],[15,18]]
O/P :- [[1,6],[8,10],[15,18]]

Explanation :- Since intervals [1,3] and [2,6] overlaps, merge them into [1,6].

(2) I/P :- intervals = [[1,4],[4,5]]
O/P :- [1,5]

Explanation :- [1,4] , [4,5] overlaps.

Code :-
```
class Solution {
    public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
    vector<vector<int>> mergedIntervals;

    if(intervals.size()==0){
        return mergedIntervals;
    }
    sort(intervals.begin(), intervals.end());
    vector<int> tempInterval = intervals[0];

    for(auto it : intervals) {
        if( it[0]<=tempInterval[1] {
            tempIntervals[1] = max(it[1], tempInterval[1]);
    } else {

        mergedIntervals.push_back(tempInterval);
            tempInterval = it;
    }
}
```

```
        mergedIntervals.push_back(tempInterval);
        tempInterval = it;
    }
}
mergedIntervals.push_back(tempInterval);
return mergedIntervals;
}
};
```

Question No. 9 Merge two Sorted Arrays without extra Space

Statement :- You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the No. of element in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in non-decreasing order.

I/P :- nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3

O/P :- [1,2,2,3,5,6]

Explanation :- The array we are merging are [1,2,3] and [2,5,6]. The result of the merge is [1,2,2,3,5,6].

Code :- (optimised) :-

```
class Solution {
public:
void merge(vector<int>& nums1, int m, vector<int>& nums2,
           int n)
{   // Resize the Size of vector.
    nums1.resize(m);
    nums1.insert(nums1.end(), nums2.begin(), nums2.end());
    Sort(nums1.begin(), nums1.end());
}
};
```

Brute force :-

```
int i = m-1, j=n-1, index = m+n-1;
while ( i>=0 && j>=0) {
    if ( nums1[i] >= nums2[j])
{
    nums1[index--] = nums1[i--] // not using another
}                                  line for
 else {
        nums1 ['index--] = nums2[ j--]
    }
}
while (j>=0)
{
    nums1[index--] = nums2[j--]
}
};
```

Question NO* 10 Find the duplicate in the array of N+1 integers

Statement :- Given an array of integers nums containing n+1 integers where each integer is in the range [1,n] inclusive. There is only one repeated NO. in nums return this repeated NO.

I/P :- nums = [1,3,4,2,2]
O/P :- 2

I/P :- nums = [3,1,3,4,2]
O/P :- 3

Code :- optimised (By slow and fast pointer approach)

```
class Solution {
    public:
      int findDuplicate(vector<int>& nums) {
          int slow = nums[0];
          int fast = nums[0];
    do {
          slow = nums[slow];
          fast = nums[nums[fast]];
    }
     while (slow != fast);

    fast = nums[0];
        while (slow != fast) {
              slow = nums[slow];
              fast = nums[fast];
        }

        return slow;
    }
};
```

Question NO # 11 REPEAT AND MISSING NUMBER

Question NO→13 Search in a 2D matrix

Statement :- Given an m*n 2D matrix and an integer, write a program to find if the given program exits in the matrix.

I/P :- Input matrix :- $\begin{bmatrix} 1 & 3 & 5 & 7 \\ 10 & 11 & 16 & 20 \\ 23 & 30 & 34 & 60 \end{bmatrix}$

target = 3

O/P = true

= **Solution 1 (Naive approach)** we can traverse through every element that is present in the matrix and return true if we found any element in the matrix is equal to the target integer. if the traversal is finished we can directly return false.

T.C :- O(m*n)
S.C :- O(1)

= **Solution 2 (Binary Search) (optimised)**

Approach :-

(i) Initially have a low index as the first index of the considered 1D matrix (i.e. 0) and high value as the last index of the considered 1D matrix [i.e. (m×n)-1]

```
int low=0;
int high = (m*n)-1;
```

(ii) Now apply binary Search. Run a while loop with the condition [low<=high]. Get the middle index as (low+high)/2. we can get the element at middle index using matrix[middle/m][middle%m]

```
while (low<=high)
    int middle = (low+high)/2;
```

(iii) If the element present at the middle index is greater than the target, than it is obvious that the target element will not exist beyond the middle index.

```
if (matrix[middle/m][middle%m] > target)
    high = middle-1;
```

(iv) if the middle index element is lesser than the target,

```
if (matrix[middle/m][middle%m] > target)
    low = middle+1;
```

(v) if the middle element is equal to the target integer, return true.

```
if (matrix[middle/m][middle%m] == target)
    return true;
```

(vi) Once the loop terminates we can directly return false as we did not find the target Element.

Code:- Passed 200/246 Test Cases in gfg.

```
class Solution {
public:
    bool SearchMatrix(vector<vector<int>>& matrix, int target) {
        int low=0;
        if (!matrix.size()) return false;
        int hi = (matrix.size() * matrix[0].size()) -1;

        while (low<=hi) {
            int mid= (low+(hi-lo)/2);
            if (matrix[mid/matrix[0].size()][mid%.matrix[0].size()]
                == target)
            return true;
        if (matrix[mid/matrix[0].size()][mid%.matrix[0].size()] < target)
            low= mid+1;
        }
        else {
            hi= mid-1;
        }
    }
    return false;
}
```

Question NO+14 Power(x,n)

Implement pow(x,n) which calculates x raised to the power n (i.e. x^n)

I/P :- input x = 2.0000 , n=10
output 1024.0000

I/P :- x = 2.10000 , n=3
output = 9.26100.

code :- Brute force :- loop from 1-n
[ans = ans×n]

if n is negative
$(x)^{-n} = \frac{1}{(x)^n}$ u have to take long type data type.

T.C :- O(N) looping from 1 to N
S.C :- O(1) // Not using any extra space.

$(2)^{10} = (2\times2)^5 = (4)^5$ // Here n=10(even) so (2x2) and n/2
$(4)^5 = 4\times4^4 = 4\times4^4$ // Here n=5(odd) so ans = ans×x and n = n-1
$(4)^4 = (4\times4)^2 = (16)^2$
$(16)^2 = (16\times16)^1 = 256^1$
$(256)^1 = 256\times(256)^0 = 256$ when n=0 we stop our code.

observation :-

[n%.2==0] → (x×x)
n/2
[n%.2==1] → ans = ans×x
n = n-1

T.C :- $\log_2 n$

Code :- class Solution {
public :
double myPow(double x, int n){
double ans = 1.0 ;
long long nn = n; // stores copy of n.

```
if (n<0)  nn = -1 * nn ;   // for making n positive no.
while (nn) {
    if (nn%2) {
        ans = ans * x ;
        nn = nn-1 ;
    }
    else {
        x = x*x ;
        nn = nn/2;
    }
}
if (n<0)  ans = (double)(1.0)/(double)(ans);
return ans;
};
```

(II) Method :-

```
class Solution {
public:
    double myPow(double x, int n) {
        if (n==0)
            return 1;
        double temp = myPow(x, n/2);
        if (n%2==0)
            return temp * temp ;
        else {
            if (n>0)
            return temp * temp * x;
            else
            return (temp*temp)/x;
        }
    }
};
```

Question No: 15 Majority Element that occurs more than N/2 times

Statement:- given an array of N integers, write a program to return an element that occurs more than N/2 times in the given array.

I/P :- N = 3 nums[] = {3,2,3}

Result = 3

Explanation :- N = 3

N/2 => 3/2 => 1.5

Hence no. > 1

3 occur 2 times.

Solutions :-

(I) **(Brute Force) :-**

- Check the count of occurence of all elements of the array one by one. Start from the first element of the array and count the no. of times it occurs in the array. If the count is greater than N/2 then return that element as answer.

T.C :- $O(N^2)$

S.C :- $O(1)$

(II) **(Better Approach) :-** ✓

(i) use a hashmap and stores as (key, value) pairs.

(ii) Here the key will be the element of the array and value will be the no. of times it occurs.

(iii) Traverse the array and update the value of key. Simultaneously check if the value is greater than floor of N/2.

→ if yes, return the key

→ no, carry forward

T.C :- $O(N \log N)$

S.C : $O(N)$.

III Optimal Approach :- Moore's Voting Algorithm

Approach :-

(i) Initialize 2 variables -

1. Count - for tracking the count of element.
2. Element - for which element we are counting.

(ii) Traverse through nums array.

(i) If count is 0 then initialize the current traversing integer of array as Element.

(ii) If the traversing integer of array and Element are same increases count by 1.

(iii) If they are different decreases count by 1.

(iii) The integer present in Element is the result we are expecting.

T.C :- O(N)
S.C :- O(1)

Code (optimised) :-

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int count = 0;
        int candidate = 0;

        for (int num : nums) {
            if (count == 0) {
                candidate = num;
            }
            if (num == candidate)
                count += 1;
            else
                count -= 1;
        }
        return candidate;
    }
}
```

Question 16 Majority Elements-Y (N/3 times)

Statement :- Given an array of N integers. Find the elements that appears more than N/3 times in the array. If no such elements exits, return an empty vector.

1)

Input: N = 5, array[] = {1, 2, 2, 3, 2}

Output :- 2

Explanation :- Here count(1) = 1 count(2) = 3 , count(3) = 1

N/3 = 5/3 => 1.6

Hence count of (2) is greater than N/3.

2) Input : N = 6 arr[] = { 11, 33, 11, 33, 33, 11 }

output : 11 33

Explanation :- Here we can see that count(11) = 3, count(33) = 3

So we return both.

Solutions 1) BRUTE FORCE :-

Approach :- Simply count the no. of appearance for each element using nested loop and whenever you find the count of an element greater than N/3 times, that element will be your answer.

Code :-

```
#include <bits/stdc++.h>
using namespace std;
vector<int> majorityElement(int arr[], int n) {
    vector<int> ans;
for(int i=0; i<n; i++) {
    int cnt = 1;
for(int j = i+1; j<n; j++) {
    if (arr[j] == arr[i])
    cnt ++;
}
    if (cnt > (n/3))
    ans.push_back(arr[i]);
}
    return ans;
}
```

```
int main() {
    int arr[] = {1, 2, 2, 3, 2};
    vector<int> majority;
    majority = majorityElement(arr, 5);
    cout << "The majority element is" << endl;
    set<int> s(majority.begin(), majority.end());
    for(auto it:s) {
        cout << it << " ";
    }
}
```

O/P :- The majority element is 2.

T.C :- $O(n^2)$
S.C :- $O(1)$

Soution 2 : Better Soution :-

Approach :- Traverse the whole array and store the count of every element in a map. After that traverse through the map and whenever you find the count of an element greater than N/3 times, store that element in your ans.

Code :-

```
#include<bits/stdc++.h>
using namespace std;
vector<int> majorityElement(int arr[], int n) {
    unordered_map<int, int> mp;
    vector<int> ans;

    for(int i=0; i<n; i++) {
        mp[arr[i]]++;
    }
    for (auto x : mp) {
        if(x.second > (n/3))
        ans.push_back(x.first);
    }
    return ans;
}
int main() {
    int arr[] = {1,2,2,3,2};
    vector<int> majority;
```

```
majority = majorityElement(a88,5);
cout << "The majority element is" << ;

for(auto it: majority) {
        cout << it << " " ;

    }
}
```

o/p :- The majority element is 2.

T.C :- O(n)
S.C :- O(n)

Solution 3 :- optimal solution (Extended Boyer Moore's voting algorithm)

Approach :- In our code, we start with declaring a few variable.

- num1 and num2 will store our currently most frequent and second most frequent element.
- c1 and c2 will store their frequency relatively to other numbers.
- we are sure that there will be a max. of 2 element which occurs > N/3 times because there cannot be if you do a simple math addition.

- if ele == num1, so we increment c1
- if ele == num2, " " " c2
- if c1 is 0, we assign num1 = ele.
- if c2 is 0, we assign num2 = ele
- In all other cases we decrease both c1 and c2.

Code :-

```
#include <bits/stdc++.h>
using namespace std;
vector<int> majorityElement(int nums[], int n){
    int sz = n;
    int num1 = -1, num2 = -1, count1 = 0, count2 = 0, i;
    for(i = 0; i < sz; i++) {
        if (nums[i] == num1)
            count1++;
```

```
else if (nums[i] == num2)
    count2++;
else if (count1 == 0) {
    num1 = nums[i];
    count1 = 1;
}
    else if (count2 == 0) {
        num2 = nums[i];
        count2 = 1;
    } else {
        count1--;
        count2--;
    }
}
vector<int> ans;
count1 = count2 = 0;
for(i=0; i<sz; i++) {
    if (nums[i] == num1)
        count1++;
    else if (nums[i] == num2)
        count2++;
}
if (count1 > sz/3);
    ans.push_back(num1);
if (count2 > sz/3)
    ans.push_back(num2);
return ans;
}
int main() {
    int arr[] = {1,2,2,3,2};
    vector<int> majority;
    majority = majorityElement(arr, 5);
    cout << "The majority element is" ;

    for (auto it : majority) {
        cout << it << " ";
    }
}
```

T.C :- O(N)
S.C :- O(1)

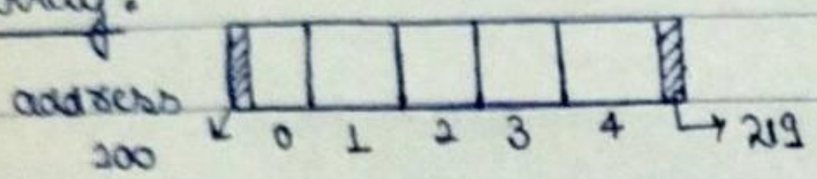
LINKED LIST :-

LEC :-01 use of linked list :- (i) Stack and queues.
(ii) Image gallery, Music player [Doubly linked list]
(iii) Browser (Doubly linked list)
(iv) Hash Map/ Hash Set.

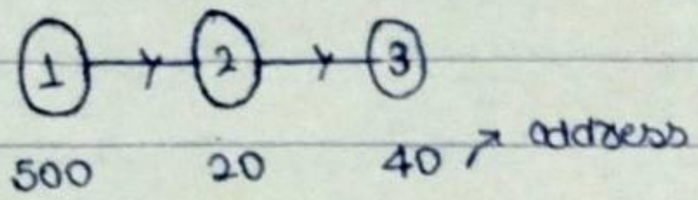
Representation of linked list :-

array :-

1 int = 4 bytes



linked list :-



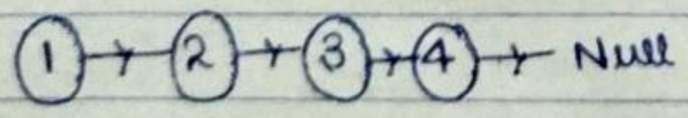
The only link b/w these two is we have to provide address of next node.

Code :-

```
Struct Node
{
    int val;      → //value of starting node
    Node *next;   → //Store the address of next node
};
```

'New' keyword allocates the memory in the heap section.
"1 Node is known as head node."

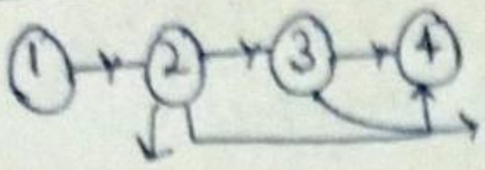
Deletion from linked list :-



traversal of a linked list :-

```
class Solution {
public :
    void deleteNode (listNode * node)
    {
        listNode *temp = node;
        while (temp != NULL)
        cout << temp->val << " ";
        temp = temp->next;
    }
};
```

☑ deletion :- we have to make connection b/w 2 and 4 and remove 3.

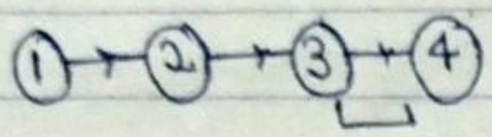


temp (stores address of 2)

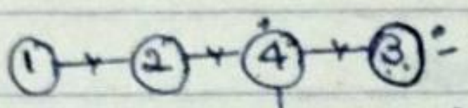
temp 2 = temp→next ; // temp→next stores address of 3.

temp→next = temp2→next ; // temp2→next stores address of 4

delete temp2 ; // delete temp2 (means 3) from memory.



Swap these two by trick



Now Simply delete 3 from the List.

final list :- ①

4	Next temp2 Node-Next =

3	temp2- Next.

temp2→next means address of the node which we have to delete.

Code by Swap :-

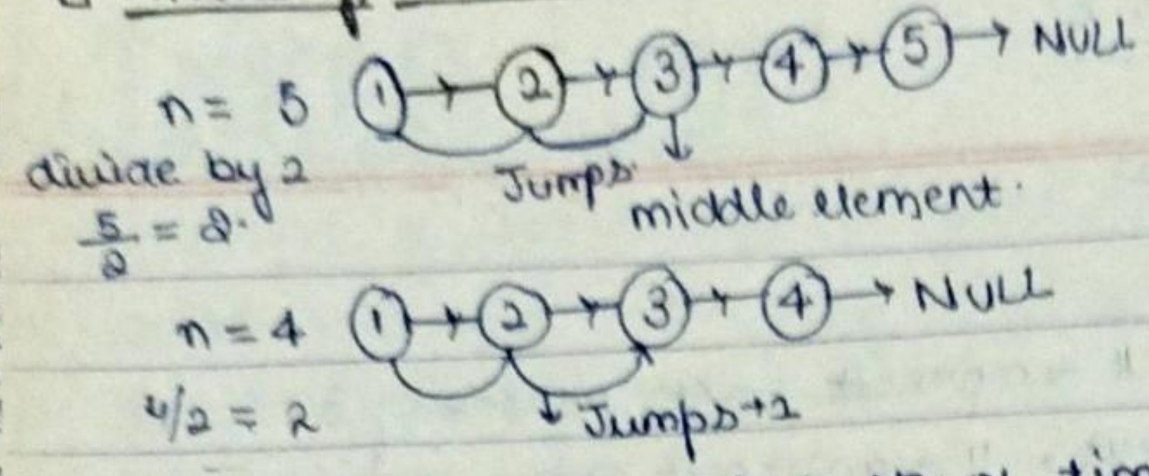
```
class Solution {
public:
    void deleteNode (dist Node* node)
{       // Base condition
    if (node->next == NULL) delete node

    Swap (node->val, node->next->val);
     dist Node * temp2 = node->next;
      node->next = node->next->next or temp2->next;
      delete temp2;
    }
};
```

☑ ARRAYS Vs LINKED LIST :-

(i) In memory allocation arrays linked list are most efficient compared to arrays.
(ii) Linked list is dynamic type data structure.
(iii) Access time in linked list [O(N)] Random access not possible.
(iv) deletion in linked list [O(1)]
(v) deletion in Array O(N)

Questions:-

Middle of the linked list:-



we have to traverse half no. of times.

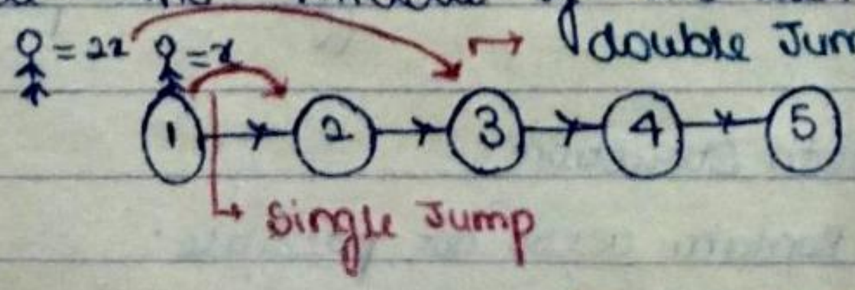
Code of this approach:- Brute force.

```
class Solution {
public:
    ListNode* middleNode(ListNode *head)
    {
        int n=0;                    // Size of the list.
        ListNode* temp = head;
        while (temp != NULL)
        {
            n++;
            temp = temp->next;
        }
        int half = n/2;
        temp=head;
        while (half--)
        {
            temp = temp->next;
        }
        return temp;
```

T.C:- O(N)

2 Approach without finding the length:- (Two pointers Approach)

Here two pointers slow and fast. fast is double the slow so when fast reaches at Null then slow will be at the middle of the list.



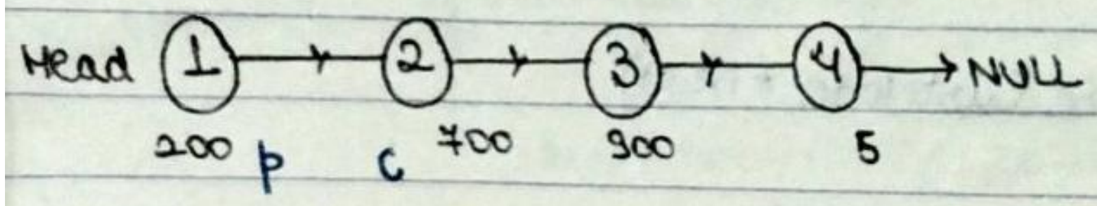

```
slow = slow->next;
fast = fast->next->next;
```

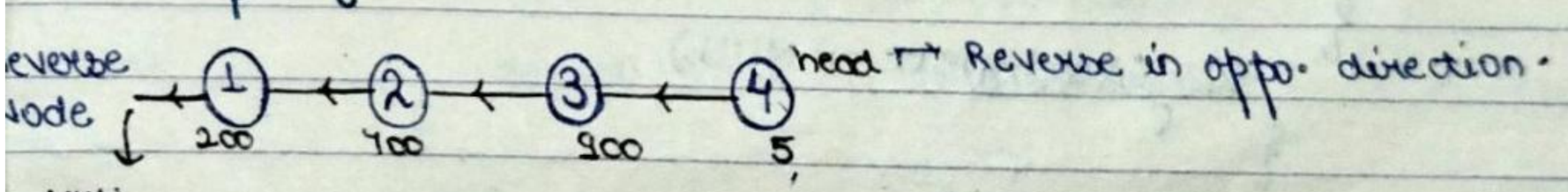
Code :-
```
class Solution {
    public:
    ListNode *middleNode(ListNode* head)
    {
        int n=0;
        ListNode *slow=head, *fast=head;
        while (fast!=NULL && fast->next!=NULL)
        {
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }
};
```

IMP:-

Reverse a linked list :-

we have to reverse the value of Address of the Node.






```
c->next=p;
p=c;        // previous node = current node
c=n;        // current node = next node
n=n->next;  // initialise next node with address of the next node.
```

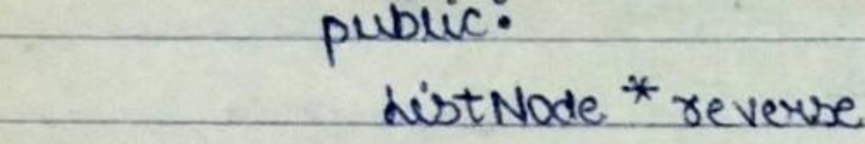
Code :- Iterative Approach :-

```
class Solution {
    public:
    ListNode* reverseList(ListNode *head)
    {   // Base condition
        if (head==NULL) return NULL;
        ListNode *p=NULL, *c=head, *n=head->next;
                          // previous, current, next
        while (c!=NULL)
        {   c->next=p;
            p=c;
            c=n;
            if(n!=null)n=n->next;
```

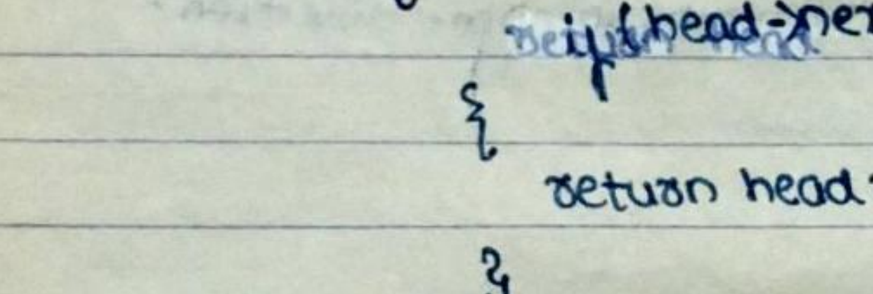
```
        return p;          // p becomes head after reversal of the linked list.
    }
};
```

Questions :-

Reverse of a linked list [Recursive Approach]



↳ with the help of recursion we reverse from 2 to last of the list, then we have to reverse only from 2 to 1 and make [1→next = NULL].

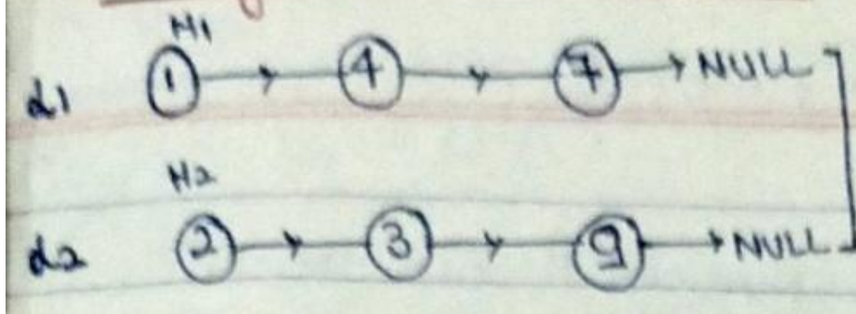


Code :-

```
class Solution {
public:
    ListNode * reverse (ListNode * head)
    {
        if(head->next == NULL)
        {
            return head;
        }
// This is for reversing 2 to 1.
// 1 is head.
        ListNode * reverseHead = reverse (head->next);
        head->next->next = head;
        head->next = NULL;  //Now 1 is head so making [1->next == NULL].
        return reverseHead;
    }

    ListNode * reverseList (ListNode * head)
    {
        if (head == NULL) return NULL;
        return reverse (head);
    }
};
```

Questions:-

Merge two Sorted list:-



Compare both values of H1 and H2 because both list are given in sorted order so the smallest value b/w these two is the head of the merged list.

[L1->Value < L2->Value]

Hold this value and apply recursion from its next to the last node (of the second list).

Recursive approach:-

```
class Solution {
    public:
    ListNode *merge(ListNode *L1, ListNode *L2)
    {
        if (L1==NULL) return L2;
        if (L2==NULL) return L1;
        if (L1->val < L2->val)
        {
            L1->next = merge(L1->next, L2);
            return L1;
        }
        else
        {   L2->next = merge(L1, L2->next);
            return L2;
        }
    }
    ListNode *mergeTwoLists(ListNode *L1, ListNode *L2)
    {
        return merge(L1, L2);
    };
```

// Base condition

It

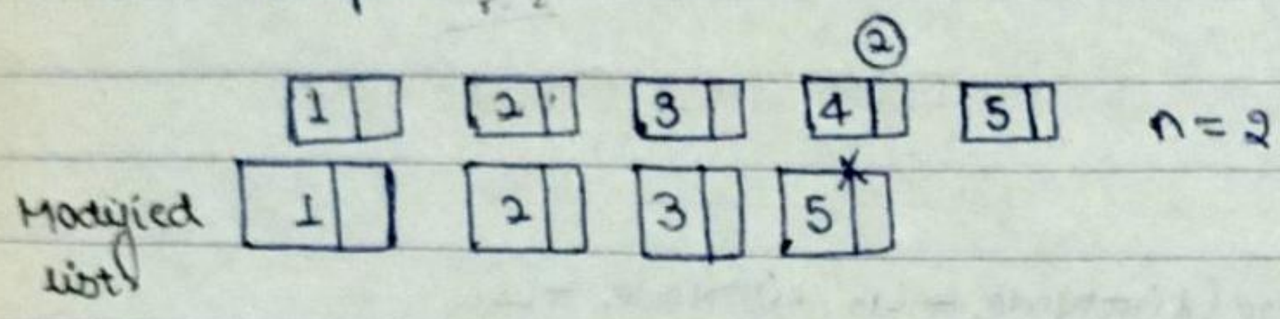
Questions :-

Remove N-th Node from the End of a linked list :-

Statement :- Given a linked list and a Number N. Find the Nth node from the end of this linkedlist and delete it. Return the head of the New modified linkedlist.

Ex 1) :- Input : head = [1,2,3,4,5] , n = 2

output : [1,2,3,5].



Solution :- Naive (Traverse 2 times)

We can traverse through the linkedlist while maintaing a count of nodes let's say in variable count and then traversing for the 2nd time for (n-count) nodes to get to the nth node of the list.

T.C ≒ O(2n)

S.C ≒ O(1)

Solution 2 Two Pointers Approach :-

Approach :- (i) Take two dummy nodes who's next will be pointing to the head.

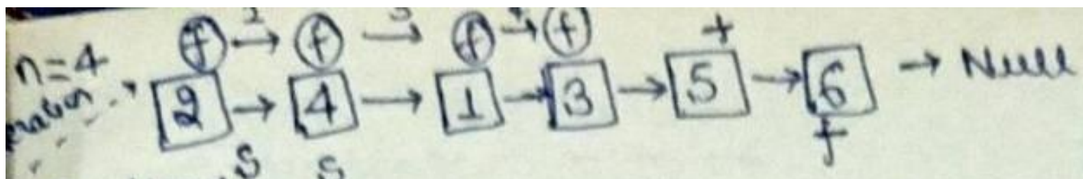
(ii) Take another node to store the head ; initially it is a dummy node (start) and the next of the node will be pointing to the head. The reason why we are using this extra dummy node is because there is an edge case. If the node is equal to the length of the linked list then this slow's will point to slow's next → next . and we can say that our dummy start node will be broken and will be connected to the slow's next → next.

(iii) Start traversing until the fast pointer reaches the Nth Node.

(iv) Now start traversing by one step both of the pointers until the fast pointer reach the end.

(v) when the traversal is done, just do the deletion part.

(vi) last, return the next of start.



fast = d
slow = d

as fast point reaches the last node we simply delete slow->next.

Edge case if n=6 Hence fast pointer moves by 6.
slow->next will point to slow->next->next.

[T.C :- O(N)
S.C : O(1)]

Code :-

```
class Solution {
    public :
        ListNode *removeNthFromEnd (ListNode *head, int n) {
            ListNode *start = new ListNode(0);
            start->next = head;
            ListNode *fast = start;
            ListNode *slow= start;

            for(int i=1; i<=n; ++i)
                fast = fast->next;
            while (fast->next != NULL)
            {
                fast = fast->next;
                slow = slow->next;
            }
            slow->next = slow->next->next;
            return start->next;
        }
};
```

Add two Numbers as LinkedLists :-

Statement :- Given the heads of two Non-empty linkedlists representing two Non-negative integers. The digits are stored in reverse order. and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

Example:- Input format:

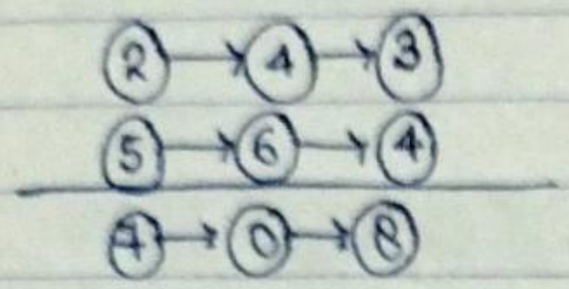
num1 = 342 , num2 = 465

l1 = [2, 4, 3]

l2 = [5, 6, 4]

Result: Sum = 807 ; L = [7, 0, 8]

Explanation:- Since the digits are stored in reverse order, reverse the numbers first to get the original no. and then add them as 342 + 465 = 807.



Pseudocode :- (i) Create a dummy node which is the head of new linked list.

(ii) Create a node temp, initialise it with dummy.

(iii) Initialise carry to 0.

(iv) Loop through list l1 and l2 until you reach both ends, and until carry is present.

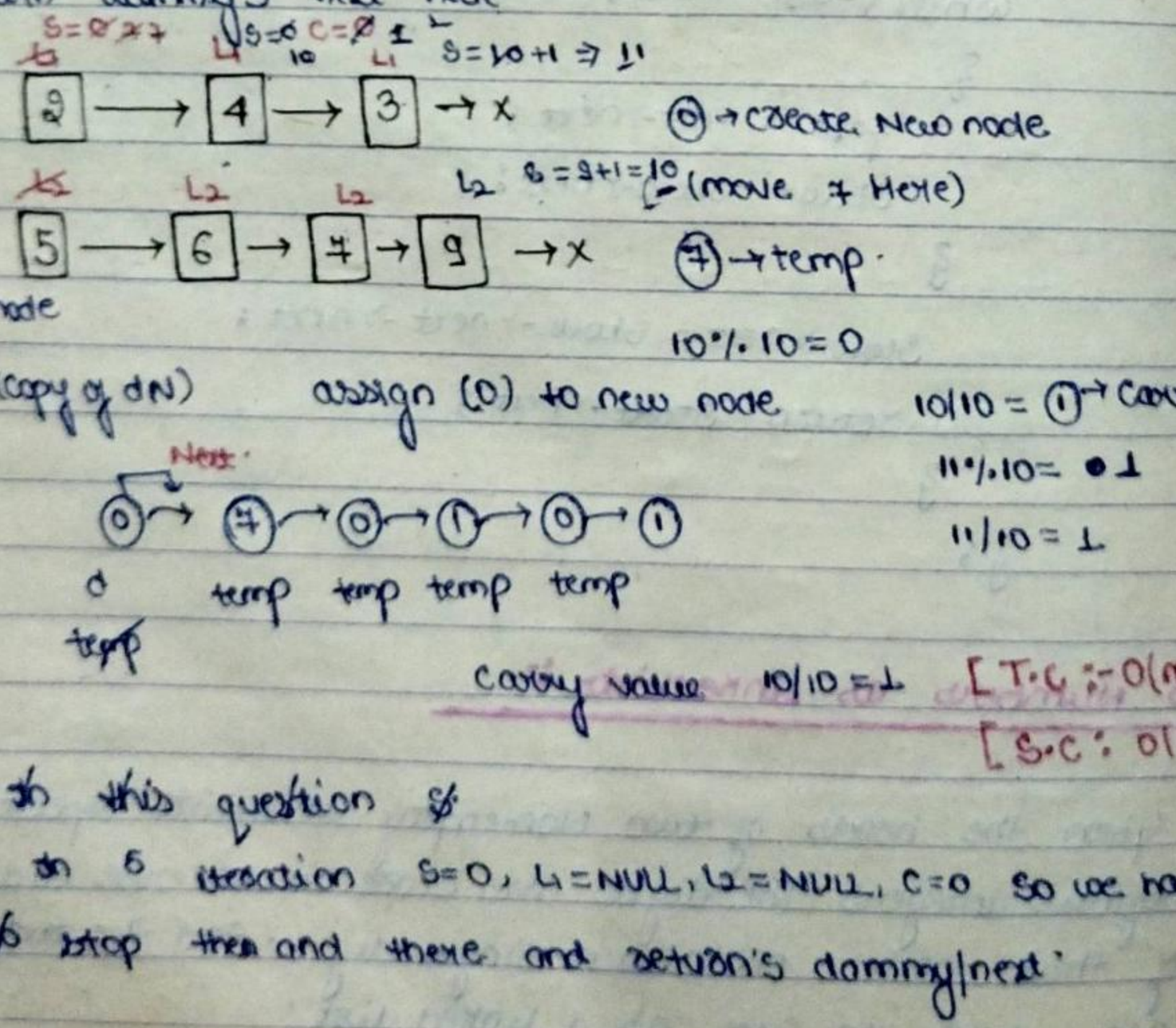
(v) Set Sum = l1.val + l2.val + carry.

(vi) update carry = Sum/10

(vii) Create a New node with the digit value of (Sum%10) and set it to temp node's next, then advance temp node to next.

(viii) Advance both l1 and l2.

(ix) Return dummy's next node.



In this question if in 5 iteration S=0, l1=NULL, l2=NULL, C=0 so we have to just stop there and there and return's dummy/next.

Code:-

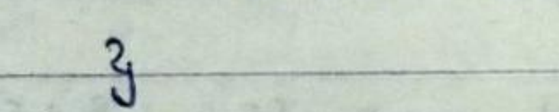
```
class Solution {
    public:
    ListNode * addTwoNumbers(ListNode *l1, ListNode *l2) {
    ListNode *dummy = new ListNode();
    ListNode *temp = dummy;
    int carry = 0;

    while ((l1 != NULL || l2 != NULL) || carry) {
        int sum = 0;
    if (l1 != NULL) {
        sum += l1->val;
        l1 = l1->next;
    }
    if (l2 != NULL) {
        sum += l2->val;
        l2 = l2->next;
    }
    sum += carry;
    carry = sum/10;
    ListNode *node = new ListNode(sum%10);
    temp->next = node;
    temp = temp->next;
    }
    return dummy->next;
    }
};
```

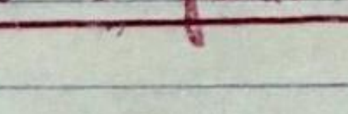
Question :-

Delete a given Node when a node is given O(1) Solution :-

Problem statement :- Write a function to delete a node in a singly-linked list. You will not be given acess to the head of the list, instead you will you will be given access to the node to be deleted directly.

(or 1)





Input :- head = [4,5,1,9], node = 5

output = [4,1,9]

Explanation :- You are given the second node with value 5.

Exp [1] → [4] → [2] → [3] → NULL

Node

copy 3 to 2 and make 2->next → NULL.

T.C : O(1)
S.C :- O(1)
] constant in nature.

Exp

add Code :- class Solution {

public :

void deleteNode (ListNode *node) {

node->value = node->next->value ;

node->next = node->next->next ;

}

};

Question :- 31 Intersection of Two linked lists :-

Statement :- Given the head of two slightly linked-lists headA and head B, return the node at which the two list intersect. If the two linked list have no intersection at all, return NULL.

Ex 1) A: 4 → 1 → 8 → 4 → 5

B: 5 → 6 → 1 → 8 → 4 → 5

Input :- intersect val = 8, list A = [4,1,8,4,5], list B = [5,6,1,8,4,5], skip A = 2, skip B = 3

Output :- Intersected at '8'

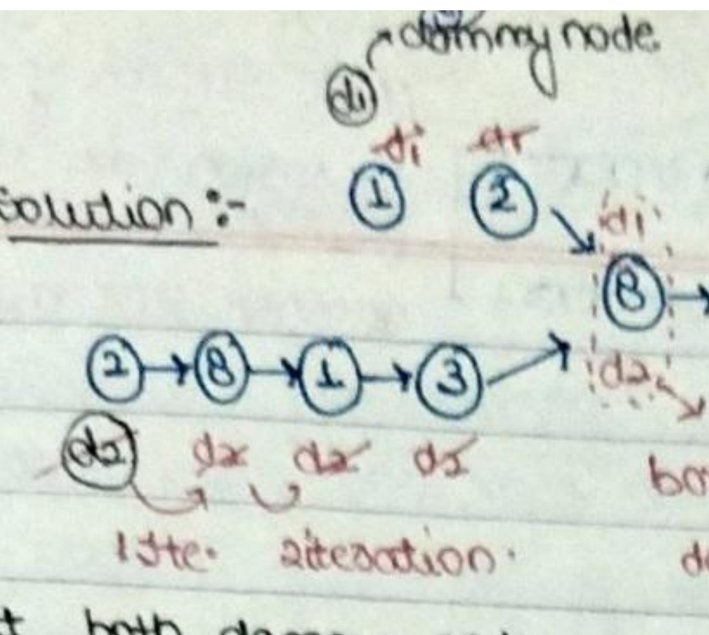
Brute force :-

A: 4 → 1 → 8 → 4 → 5

B: 5 → 8 → 1 → 8 → 4 → 5

Start moving from B and check whether node 5 is equal to the node 4 and so-on (basically we are checking the address of node 5 is same 4 or not). and same we are doing with list B also and we get that 8 is the same node we are standing so the intersection node of both the list is 8.

Optimised solution :-



(i) traverse both dummy node simultaneously till they reach at end and in doing so you keep the count of list 1 and 2.

(ii) again put both dummy nodes at the head of both linked list and find the diff b/w length of both linked list.

$|L_1 - L_2| \Rightarrow |5-7| = 2$.

(iii) Move longer linked list by the difference. by this you covered the diff

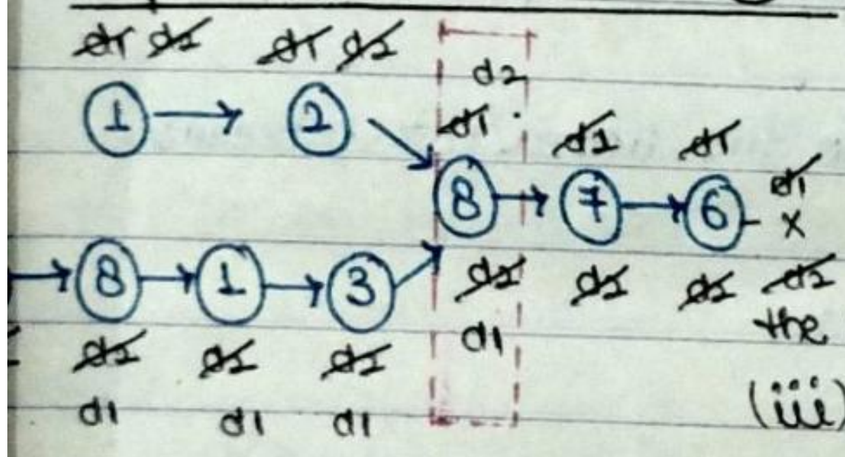
(iv) Move simultaneously dummy 1 and dummy 2, the moment d1 and d2 collides, we can say that this is the node which is actually the intersection pt. of both linked list.

T.C :- $O(M) + O(M-N) + O(N)$ ← again traversing through shorter list. ← diff. ← length

$\Rightarrow O(2M)$

S.C :- $O(1)$

Optimised solution :- II



(i) assume two dummy nodes d1 and d2 and traverse simultaneously.

(ii) The moment any of the node reaches Null then assign this dummy nodes as the head of the second list.

(iii) Again start traversing both the nodes simultaneously.

(iv) The moment where the both the nodes collides is that intersection node of both linked list.

Code :-

```
class Solution {
public: ListNode *getIntersectionNode(ListNode *headA, ListNode
                                      *headB) {
    if (headA == NULL || headB == NULL)
        return NULL;

    ListNode *a = headA;
    ListNode *b = headB;        // Initialize two dummy nodes with head of both linked list.

// if a & b have different len, then we will stop the loop after second iteration.
```

```
while (a != b) {
    a = a == NULL ? headB : a->next;
    b = b == NULL ? headA : b->next;
}
return a;
}
};
```

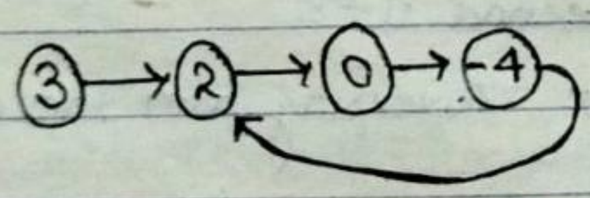
// for the end of the first iteration, we just reset the pointer to the head of another linked list.

Question: No: 32 **Detect a cycle in a linked list :-**

Statement :- Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as parameter.

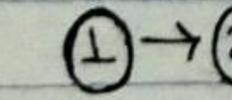
Return 'true' if there is a cycle in the linked list. Otherwise return 'false'.

Ex.1) 

Input :- head = [3,2,0,-4] , pos = 1

Output :- true

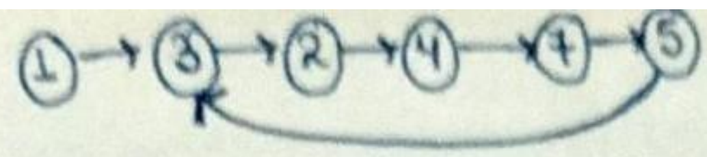
Explanation :- There is a cycle in a linked list, where the tail connects to the 1st node (0-indexed).

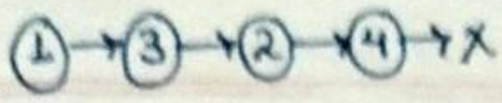
Ex-2 

Input :- head = [1,2] , pos = 0

output :- true

Explanation :- There is a cycle in the linked list where the tail connects 0th node.

Solution :-  Here's a cycle in it.

 Here's No cycle present.

(1) Naive approach :- **By Hashing**

(i) Create Hash table and take dummy Node and check does that ① complete node (1) is present if it is not Hash complete node 1 into the hash table and same with all the other element.

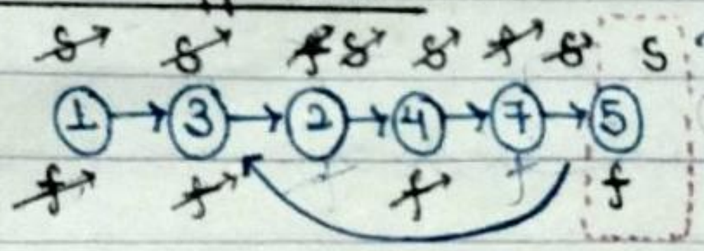
(ii) The moment it comes back to (3) and it is present on the hash table that is the reason you find a cycle in hash table.

(iii) If you reach Null and any of the traversed node did not exit on Hash table you can say there is no cycle present.

T.C :- O(N)
S.C :- O(N)

for Hash table.

(2) Optimised Approach :-



when we repeat in cycle s → 1 step then we have f → 2 step. to give 1 gap for f.

(i) Take two dummy node slow pointer and fast pointer. Move slow pointer by 1 step and move fast pointer by 2 steps.

Code :-

```
Class Solution {
    public:
        bool hasCycle (ListNode *head) {
            if (head == NULL || head->next == NULL)
                return false;

            ListNode *slow = head;
            ListNode *fast = head;

        while(fast-> next && fast-> next-> next) {
            fast = fast-> next-> next;
            slow = slow-> next;
            if (fast == slow)
                return true;
        }
        return false;
    }
```

M-dmp HARD PROBLEM

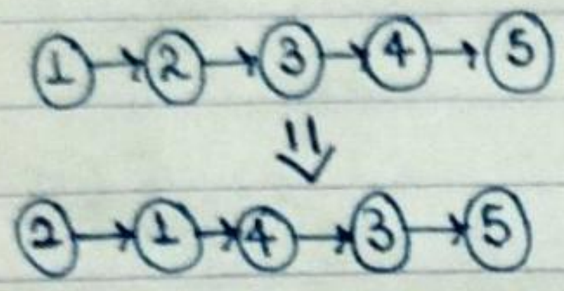
Question No-33 :- Reverse a LinkedList in a group of Size K.

Statement :- Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.

k is a positive integer and is less than or equal to the length of the linked lists. If the number of nodes is not a multiple of k then left-out nodes, in the end, should remain as it is.

You may not alter the values in the list's nodes, only nodes themselves may be changed.

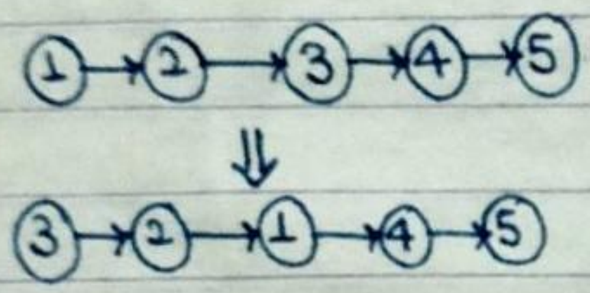
Ex 1)



Input :- head = [1,2,3,4,5] , k=2

output :- [2,1,4,3,5]

Ex 2)



Input :- head = [1,2,3,4,5] , k=3

Output :- [3,2,1,4,5]

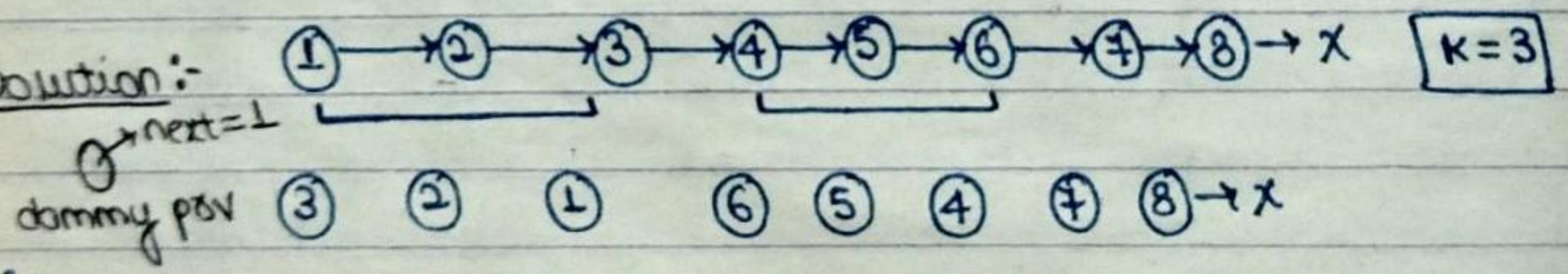
Ex 3) Input :- head = [1,2,3,4,5] , k=1

output :- [1,2,3,4,5]

Ex 4) Input :- head = [1], k=1

output :- [1]

Solution :-



(i) count the size of linked list by taking a dummy node. cnt= at last the value of cnt =8.

(ii) And then we divide it by k to get groups of size k.

8/3 = 2 groups are present in the list of size 3.

(iii) Now perform operation on the I group. Make a dummy node curr pointing to curr 1.

curr->next = next->next

next->next = pre->next

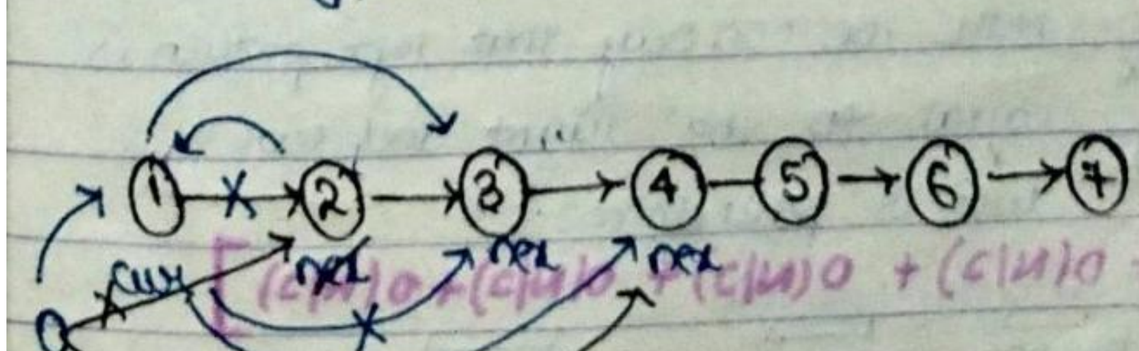
pre->next = next;

next = curr->next;

Here prev-next pointing to 1
↓
dummyNode.

Code :-

```
Class Solution {
    public:
        ListNode * reverseKGroup (ListNode *head, int K) {
// Base condition if (head == NULL || K == 1) return head;
// Initialising dummy node.   ListNode *dummy = new ListNode(0);
                              dummy-> next = head;

// Initialising cur, prev, next.  ListNode *cur = dummy, *nex = dummy, *pre = dummy;
                                  int count = 0;
                                  while (cur->next != NULL) {   // loop for count of list
                                      cur = cur->next;
                                      count++;
                                  }
                                  while (count >= K) {
                                      cur = pre->next;
                                      nex = cur->next;
                                      for (int i = 1; i < K; i++) {
                                          cur->next = nex->next;
                                          nex->next = pre->next;
                                          pre->next = nex;
                                          nex = cur->next;
                                      }
                                      pre = cur;
                                      count -= K;
                                  }
                                  return dummy->next;
    }
};
```



Code :-

```
class Solution {
public:
    bool isPalindrome(ListNode* head) {
        if (head == NULL || head->next == NULL)
return true;
ListNode *slow = head;
ListNode *fast = head;

while (fast->next != NULL && fast->next->next != NULL){
        slow = slow->next;
        fast = fast->next->next;
}
slow->next = reverseList(slow->next);
slow = slow->next;

while (slow != NULL){
    if (head->val != slow->val)
        return false;
    head = head->next;
    slow = slow->next;
}
return true;
}
};

//ListNode *reverseList (ListNode *head) {
    ListNode *pre = NULL;
    ListNode *next = NULL;
    while (head != NULL){
    next = head->next;
    head->next = pre;
    pre = head;
    head = next;
}
return pre;
};
```

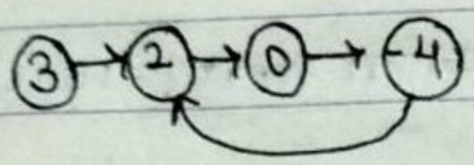
Here head is a dummy Node.

code for reverse list

[M-Imp]

Question NO-35 Find the Starting point of the loop of the linked list.

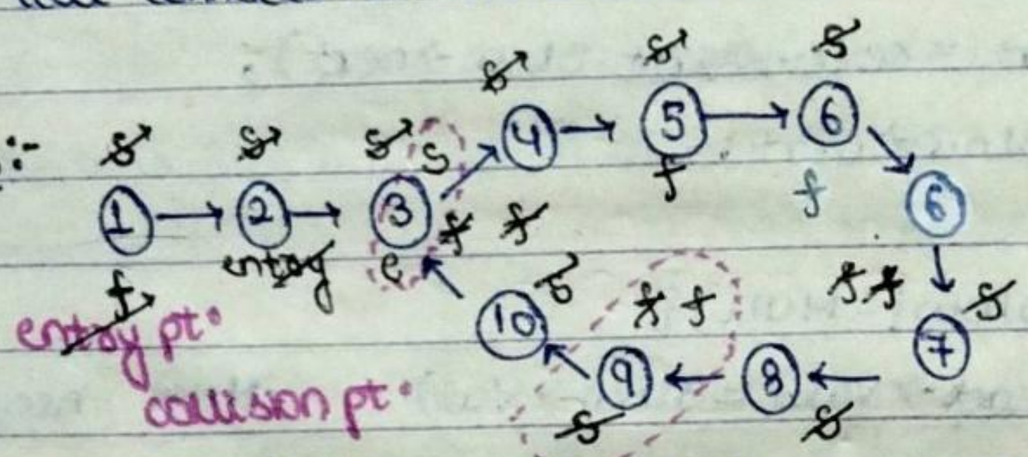
Statement :- Given the head of a linked list, return the node where the cycle begins. If there is no cycle, return NULL.
There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer.

Ex. 

Input :- head = [3,2,0,-4], POS = 1

output :- tail connects to node index 1

Explanation :- There is a cycle in the linkedlist, where tail connects to the second node.

Solution :- 

entry pt. and slow pointer collides.

(i) **Naive approach** :- By Hash table and keep on hashing each entire node.

T.C : O(N),

S.C :- O(N) for Hashing.

(ii) **optimised solution** :- Slow and fast pointer method

(i) find out collisson point of slow and fast pointer. S→1, f→2

(ii) take one more pt. and put this at head of linkedlisty.
[Entry pt →1, Slow pointer→1.] until they are colliding.

(iii) The node where they both are colliding is our starting pt. of the loop of linked list.

T.C :- O(N)

[S.C :- O(1)]

Code :-

```
class Solution {
    public:
    ListNode *detectCycle(ListNode *head) {
      if (head == NULL || head->next == NULL)
      return NULL;

    ListNode *slow = head;
    ListNode *fast = head;
    ListNode *entry = head;

while (fast->next && fast->next->next) {
        slow = slow->next;
        fast = fast->next->next;
      if (slow == fast) {
      while (slow != entry) {
        slow = slow->next;
        entry = entry->next;
      }
      return entry;
    }
  }
  return NULL;
  }
};
```

JMP.

Question NO→ 36 Flattening of a linked list

Given a linked list of size N, where every node represents a sub-linked-list and contains two pointers.

(i) a next pointer to the next node.

(ii) a bottom pointer to a linked list where this node is head.

Each of the sub-linked-list is in sorted order.

Flatten the link list such that all the nodes appears in a single level while maintaing the sorted order.

NOTE:- The flattened list will be printed using the bottom pointer instead of next pointer.

6x+1) Input: 5→10→19→28
1 1 1 1 → represent bottom pointer.
7 20 22 35
1 1 1
8 50 1
1 40
30 1
45

Output:- 5→7→8→10→19→20→22→28→30→35→40→45→50

Explanation:- The resultant linked list has every node in a single level.

Approach:- Start flattened two list from last, merge them and then again merge it with 3 list and then the last one.

Recursive Call

16 flatten(L1) → merge(14,16) (13,14)→15
↓
flatten(L2) → merge(12,15) (15,12)→16
15 ↓
flatten(L3) → merge(13,14) (11,16).
↓
flatten(L4)

- T.C :- O(Summation)
- S.C :- O(1)

Code:- Node *mergeTwoLists(Node *a, Node *b) {

```
Node *temp = new Node(0);
Node *res = temp;
while(a!=NULL && b!=NULL) {
    if(a->data < b->data){
        temp->bottom = a
        temp = temp->bottom
        a=a->bottom;
    }
    else {
        temp->bottom= b;
        temp= temp->bottom;
        b= b->bottom;
    }
}
if(a) temp->bottom =a;
 else temp->bottom =b;
```

Here whole code represent merge of two lists from last.

```
        return res->bottom;
}

Node * flatten (Node * root)
{
    if(root == NULL || root->next == NULL)
        return root;
        // recursion for list on right
        root->next = flatten (root->next);
         // Now merge
        root = mergeTwoLists(root , root->next);
        // return the root.
        return root;
```

Question No→37 Rotate a linked list

Statement :- Given the head of a linked list, rotate the list to the right by k places. [k can be greater than size of linked list]

Ex 1:- ① → ② → ③ → ④ → ⑤ Input : head = [1,2,3,4,5] , k=2

rotate 1: ⑤ → ① → ② → ③ → ④ output : [4,5,1,2,3]

rotate 2: ④ → ⑤ → ① → ② → ③

Approach :- Naive Approach :- Pick up the last not and put it in front at k no. of times.

[T.C: O(KXN)]

[S.C: O(1)]

Optimal Solution :- Case 1) K<length Case(ii) K>= length.

NOTE :- Any multiple of length is gone quit you original linkedlist

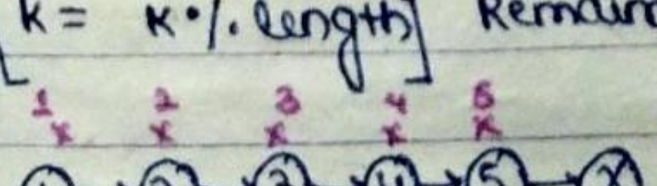
So for K>= length

[k = k%.length] Remainder is the actual location

K=12 => 12%.10 = 2

Len = 5 So first 10 rotation remains same.

Hence [K=2]

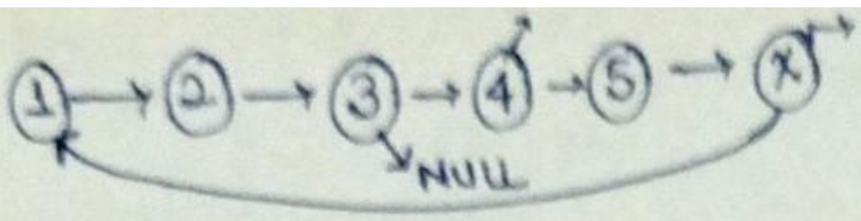


(i) compute length of the list.

(ii) Point last node to the head of the list. and head is pointing to 4

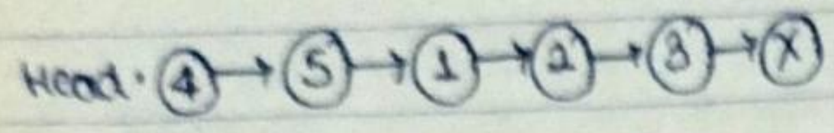
K = len-K

= 5-2 = [3]. pointing 3rd Next to Null



k = len - k
= 5 - 2
k = 3

point last Node to the head so that it become circular list.



(i) count the length of linked list
(ii) points last → next → head.
(iii) get (length - kth Node) and point it to NULL.

Code :-

```
class Solution {
    public:
    ListNode* rotateRight (ListNode *head, int k) {
     // edge cases
    if (!head || !head->next || k == 0)
      return head;
     // compute length
    ListNode *cur = head;
     int len = 1;
     while (cur->next && ++len)
          cur = cur->next;
     // go till that node.
     cur->next = head;
      k = k % len;
      k = len - k;
     while (k--)  cur = cur->next;
// make that node head and break connection.
      head = cur->next;
      cur->next = NULL;

      return head;
    }
};
```

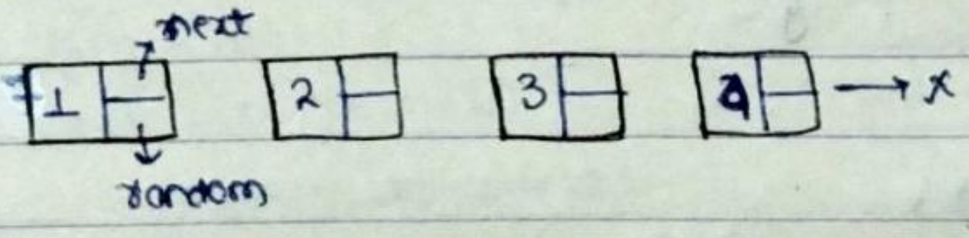
Question No-38 Clone a linked list with random and next pointer.

Statement :- A linked list of length n is given such that each node contains an additional random pointers, which could point to any node in the list, or null.

Construct a deep copy of the list. The deep copy should consist of exactly n brand new nodes, where each new node has its value set to the value of its corresponding original node. Both the next and random pointer of the new node should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state. None of the pointers in the new list should point to nodes in the original list.

Eg Input :- head = [[7,NULL], [13,0], [11,4], [10,2], [1,0]]
output :- [[7,NULL], [13,0], [11,4], [10,2], [1,0]]

Solution :-

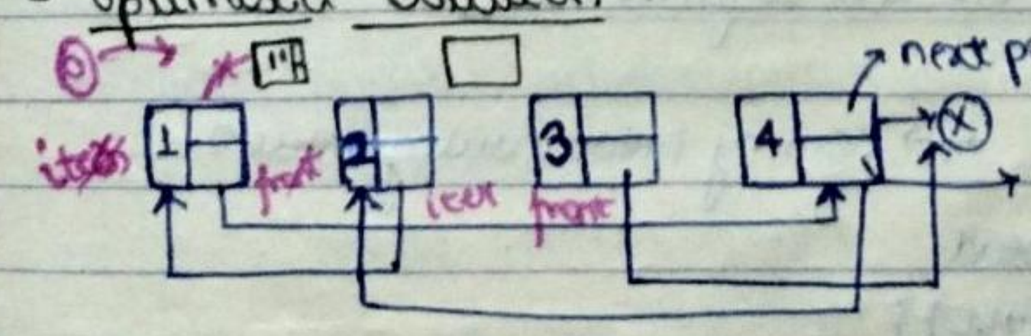


① Brute force approach :- By Hash Map traverse each node and store it in hash table [create deep copy of every node].

T.C :- O(N) + O(N)

S.C :- O(N) for Hash Map.

Optimised Solution :-



(i) create a copy nodes of Node 1, 2, 3, 4 and insert them right after the original Node.
(ii) and 1 -> next pointing to 1' and 1'->next pointing to 2 and similarly with 3 and 4. and the next of 4' pointing to NULL.
(iii) Point the Random pointer :-

iter pointing head.

By this random copy node. pointing of deep

2 step:-

[iter->next->random = iter->random->next]

deep copy->random = current random of iter

3 step:- iter move by 2 steps iter->next->next and pointing to 2.

4 step:- take dummy node and iter.

[T.C:- O(N) + O(N) + O(N) ⇒ O(3N) ⇒ O(N)]

[S.C:- O(1)]

Pseudocode Step 1):- Initialise iter = head

front = head.

```
while (iter != NULL) {
    front = iter->next;
    copy = New Node (iter->val)
    iter->next = copy;
    copy->next = front;
    iter = front;
}
```

Step 2) Random pointer:-

```
iter = head;
while (iter != NULL) {
    if (iter.random != NULL)
        iter->next->random = iter->random->next ;
    iter = iter->next->next;
}
```

(3) Step 3) Differenciate Deep copy list and original list :-

```
iter = head;
pseudohead = 10) ⟶ dummy Node having value 0.
copy = pseudohead.
while (iter != NULL) {
    front = iter->next->next;
    copy->next = iter->next;   // Extraction of deep copy
    iter->next = front;  // Link for original Link.
    copy = copy->next;
    iter = iter->next;
};
```

Code:-

```
class Node {
    public:
        int val;
        Node *next;
        Node *random;
        Node (int _val) {
            val = _val;
            next = NULL;
            random = NULL;
        }
};

class Solution {
    public:
        Node* copyRandomList (Node *head) {
            Node *iter = head;
            Node *front = head;

// First Round: make copy of each node, and link them together side by side in a single list.
            while (iter != NULL) {
                front = iter->next;

                Node *copy = new Node (iter->val);
                iter->next = copy;
                copy->next = front;

                iter = front;
            }

// Second Round: assign random pointers for the copy nodes.
            iter = head;
            while (iter != NULL) {
                if (iter->random != NULL) {
                    iter->next->random = iter->random->next;
                }
                iter = iter->next->next;
            }
// Third Round: restore the original list
```

```
        iter = head;
        Node *pseudoHead = new Node(0);

        Node *copy = pseudoHead;

    while (iter != NULL) {
        front = iter->next->next;

    // extract the copy
        copy->next = iter->next;
    // restore the original list
        iter->next = front;

        copy = copy->next;
        iter = front;
    }
        return pseudoHead->next;
    }
};
```

Question No-39 3 Sum :- Find triplets that add up to a zero

Problem Statement :- Given an array of N integers, your task is to find unique triplets that add up to give sum zero. In short you need to return an array of all the unique (a,b,c) triplets [arr[a], arr[b], arr[c]] such that i!=j, j!=k, k!=i, and their sum is equal to zero.

Ex :- Input : nums = [-1,0,1,2,-1,-4]

output :- [[-1,-1,2], [-1,0,1]].

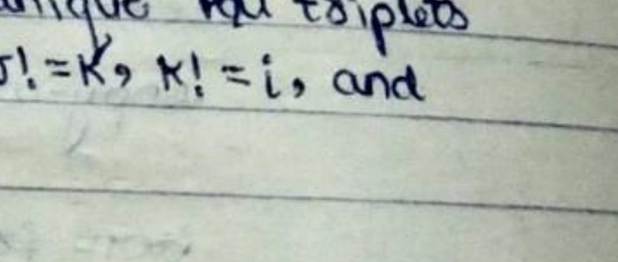
Explanation :- Out of all possible unique triplets possible, [-1,-1,2] and [-1,0,1] satisfy the condition of summing up to zero with i!=j!=k

Solution :- Brute force :- [-1,0,1,2,-1,-4]

a+b+c=0

```
for (i=0→n-1)
    for (j=i+1→n-1)
        for (k=j+1→n-1)
```

triplets: (a[i]+a[j]+a[k] == 0)



T.C :- O[n³Logm]

↳ time taking to insert triplets in set data st.

S.C :- O(M)

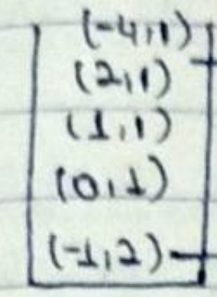
↳ No. of unique triplets.

① Optimised solution :- [-1, 0, 1, 2, -1, -4] By using Hashing.

$$a+b+c=0$$

(i) Iterate a loop on a and b and for c use Hashing.

(ii) we create a Hash map and store elements with their occurence.



(iii) Iterate value of a and b :- for(i=0; i<n-1, i++)

:- for(J=i+1; J<n-1, J++)

(iv) $a+b+c=0$

$[c = -(a+b)]$ If this exit in hash map then we can say it is the triplet we are looking for.

```
for (i=0→n-1)
    hash[a[i]]--
for(J=i+1→n-1)
{
    hash[a[J]
    .....
}
hash[a[J]]++
hash[a[i]]++ ;
```

[T.C :- $O(N^2 \times \log M)$]

[S.C :- $O(M) + O(N)$]

② Best Solution :- **By using two pointer approach**

(i) For using 2-pointer approach we need to sort the array first.

a → low low low low low low Hi Hi

[-2, -2, -1, -1, -1, 0, 0, 0, 2, 2, 2]

$a+b+c=0 \Rightarrow \boxed{b+c=-a}$

↓ const.

(i) a = -2

[b+c=2]

low + Hi ⇒ -2+2 = 0 ↑↑ Have to increase, so we move towards right.

-1+2 = 1 ↑↑

[0+2 = 2] → this is the Eq?

b = 0

c = 2

a = -2.

our triplets = [-2, 0, 2]

when u get equivalent (2) we have to move low as well as Hi pointer by 1 step. Move low until they do not reach the no. which is not equal to the previous one.

The moment low crosses high we have to stop

Now take, a=-1.

b+c=1.

lo=-1, Hi=2] -1+2=1

a=-1, b=-1, c=2

[-1,-1,2]

Now lo=0 →
Hi=0
[0+0+1] ↑↑

Now take a=0 and apply the same concept and for a=2 also.

T.C :- O(N×N) ≈ O(N²)
S.C :- O(M) → for storage of ans.

Code :-

```
class Solution {
    public :
    vector<vector<int>> threeSum(vector<int>& num) {

        sort(num.begin(), num.end());
        vector<vector<int>> res;
        // moves for a (int)                    → for 3rd last element.
        for(int i=0; i<num.size()-2; i++) {
            if(i==0 || (i>0 && num[i]!=num[i-1])) {
            int lo=i+1, hi=num.size()-1, Sum=0-num[i];
                                (int).
            while(lo<hi) {
                if(num[lo]+num[hi]==Sum){
                vector<int> temp;
                temp.push_back(num[i]);
                temp.push_back(num[lo]);
                temp.push_back(num[hi]);
                res.push_back(temp);
                while(lo<hi && num[lo]==num[lo+1])lo++;
                while(lo<hi && num[hi]==num[hi-1]) hi--;

                lo++; hi--;
                }
                else if(num[lo]+num[hi]<Sum) lo++;
                else hi--;
            }
            }
        }
```

```
        return res;
    }
};
```

Question No-40 **TRAPPING RAIN WATER PROBLEM :-**

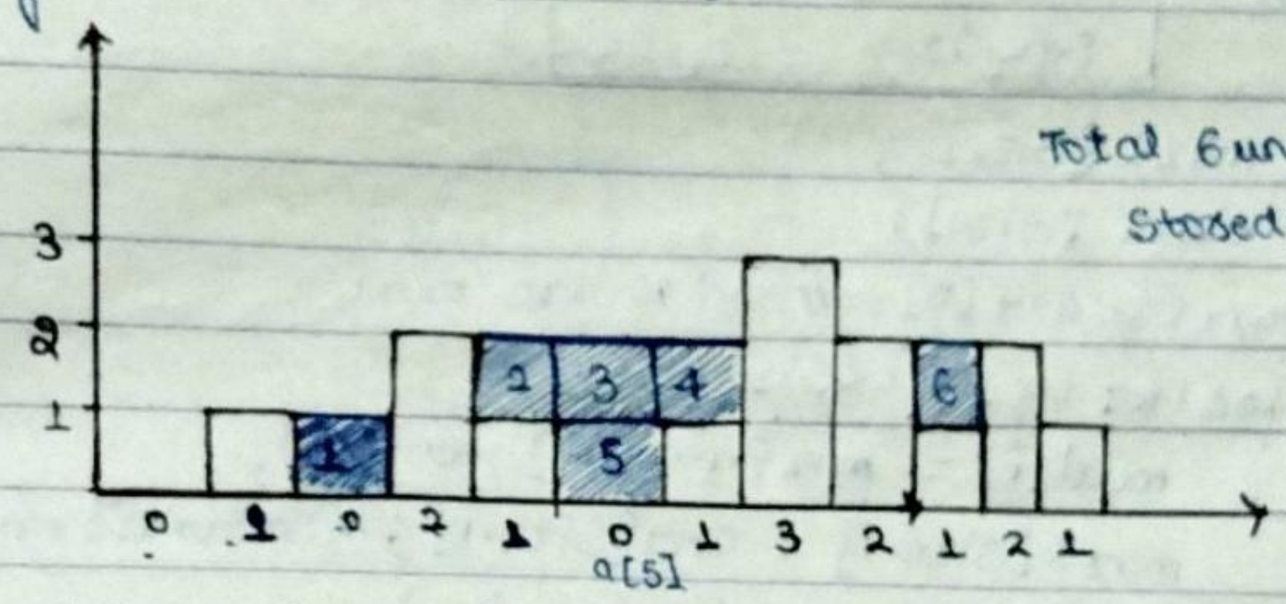
Problem Statement :- Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Ex 1.

Input :- height = [0,1,0,2,1,0,1,3,2,1,2,1]

output :- 6

Explanation :- The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]



Soln :- (i) Brute force Approach :- For every index we have to find the unit of water stored. The amount of water stored is equivalent to min of (max left height and max right height).

water [i] = min[pleft(i), mright(i)] - a[i]

min [2, 3] - a[5]

2 - 0 ⇒ 2

for left max :- (i to 0)

for right max : (i to n-1)

T.C : $O(N^2)$ ↳ for left max and right max.

S.C :- O(1)

(ii) Better Solution :- (Prefix Sum)

How to calculate max L and max R as prefix sum and suffix sum.

array :- 3 0 0 2 0 4

max.L :- ③ 3 3 3 3 4

compare 0 and 3 which one is max. write it.

array :- 3 0 0 2 0 4

max.R :- 4 4 4 4 4 ④

array :- 0 1 0 2 1 0 1 3 2 1 2 1

max L :- [0 1 1 2 2 2 2 3 3 3 3 3]

array :- 0 1 0 2 1 0 1 3 2 1 2 1

max R :- [3 3 3 3 3 3 3 3 2 2 2 1]

Ex :- 3 0 0 2 0 4

max l 3 3 3 3 3 4

max R 4 4 4 4 4 4

min 3 3 3 3 3 4

- 3 0 0 2 0 4 → Height of Building

= 0 3 3 1 3 0

= 3 3 1 3 0 ⇒ 3+3+1+3+0 ⇒ 10 units of water stored.

Code :- Given :- arr[] : 3 0 0 2 0 4 Size :- o/p :- 10.

```
{
  {  int maxl[size] ;
     int maxr [size];
       maxl = arr[0]  // Initialize maxl
     for(int i=1 ; i<Size ; i++)
         maxl[i] = max[maxl[i-1] , arr[i]);
       maxR[Size-1] = arr[Size-1] ; // Initialize maxR.
     for( int i= Size-2 ; i>= 0 ; i--)
         maxR[i] = max( maxR[i+1] , arr[i]);
     int water[Size];
     for(int i= 0 , i< Size ; i++)
   water[i] = min (maxl[i] , maxR[i]) - arr[i];
   int sum = 0;
    for (int i=0 ; i<Size ; i++)
       sum = sum + water[i] ;
     return Sum;
  }
};
```

Final Code :- Most optimised Solution [2 pointer's approach]

(i) Initialise left pointer [l = 0] and right pointer to [r = n-1] and [res = 0] ; leftmax = 0 , [rightmax = 0]

(ii) check if (a[l] <= a[r])
    if (a[i] > = leftmax) leftmax = a[i]
    else res += [leftmax - a[i])
    l++;

Code :-

```
class Solution {
    public:
    int trap (vector <int> & height) {
        int n = height.size();
        int left=0;  int right = n-1;
        int res =0;
        int maxleft =0, maxright =0;

    while (left <=right) {      // for right pointer moves till
        if (height [left] <= height [right]) {  it doesn't crosses
                                                left.
            if (height [left] >= maxleft)
                maxleft = height [left];
            else
                res+= maxleft - height [left];
            left ++;
        }
        else {
            if (height [right] >= maxright)
                maxright = height [right];
            else
                res+ = maxright - height [right];
            right --
        }
    }
        return res;
    }
};
```

T.C :- O(N) for 2 pointer's approach.
S.C :- O(1)

Question No-41 Remove Duplicate from Sorted Array.

Problem Statement :- Given an integer array sorted in non-decreasing order, removes the duplicates in place such that each unique element appears only once. The relative order of the elements should be kept the same.

If there are k elements after removing the duplicates, then the first k elements of the array should hold the final result. It does not matter what you leave beyond the first k element.

Ex 1) :- **Input :-** arr [1, 1, 2, 2, 2, 3, 3]
output :- arr [1, 2, 3, ----]

Explanation :- Total No. of unique element are 3 i.e. [1, 2, 3] and therefore return 3 after assigning [1, 2, 3] in the beginning.

→ **Bruteforce :-** By using Hash Set.
Iterate for every element and insert it and set doesn't insert duplicate.

T.C : O(NLogN)+O(N)
S.C : O(N) for Hash Set.

→ **optimised Approach :-** 2 pointer's approach

i j j j j j j ⑤ when J pointer crosses we have to stop.

a[] = [1, 1, 2, 2, 2, 3, 3]
i ①

(i+1) index.

check a[i] != a[J]

T.C :- O(N)
S.C :- O(1)

Code :-

```
class Solution {
public:
 int removeDuplicates(vector<int>& nums) {
    if(nums.size()==0)
        return 0;
    int i=0;
    for(int J=1 ; J<nums.Size(); J++){
        if(nums[i]!=nums[J])
            i++;
        nums[i] = nums[J];
    }
    return i+1;
};
```

Question NO: 42 **Max. Consecutive ones :-**

Statement :- Given an array that contains only 1 and 0 return the count of maximum consecutive ones in the array.

Example :- (i) **Input :-** prices = {1, 1, 0, 1, 1, 1}

output :- 3

Explanation :- There are two consecutive 1's and three consecutive 1's. Hence maximum is 3.

(ii) **Input :-** prices = {1, 0, 1, 1, 0, 1}

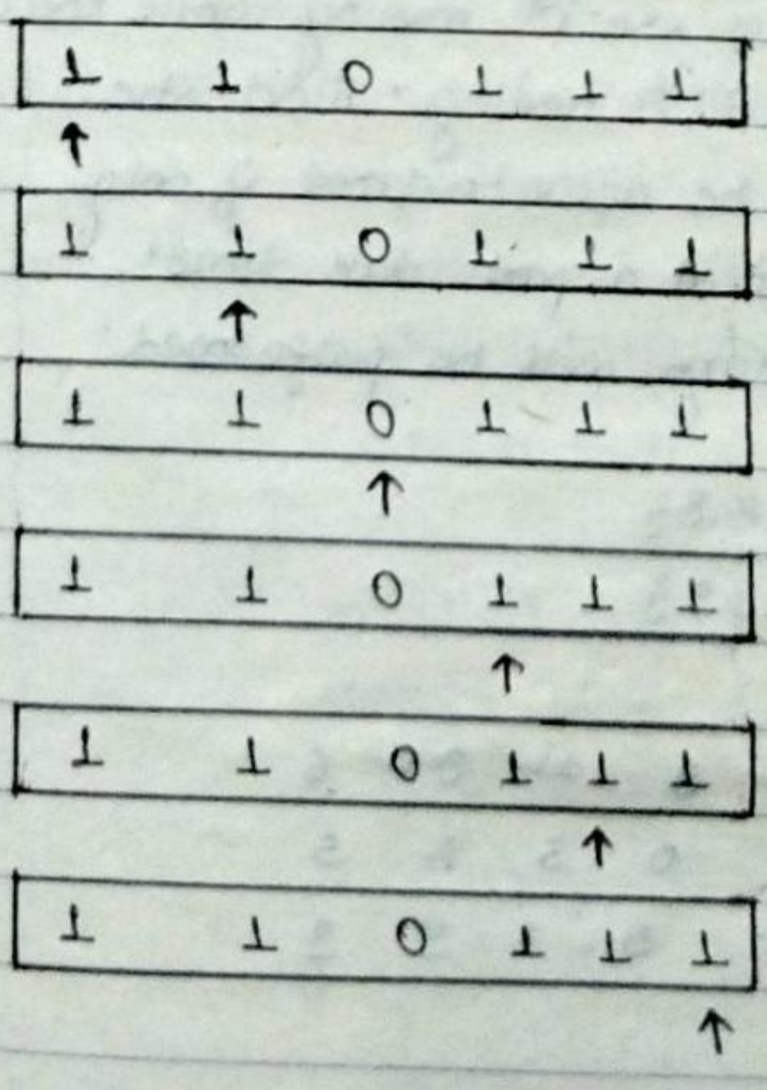
output :- 0 2

Explanation :- There are two consecutive 1's in the array.

Solution :- We start traversing from the beginning of the array. Since we can encounter either a 1 or 0 there can be two situation.

(i) If the value at the current index is equal to 1 we increase the value of count by one. After updating the count variable if it becomes more than max-count update the max-count.

(ii) If the value at the current index is equal to zero we make the variable count as 0 since there are no more consecutive ones.



Code:-
```
class Solution {
    public:
      int findmaxConsecutiveOnes(vector<int>& nums) {
       int cnt = 0;
       int maxi = 0;
      for(int i=0; i<nums.size(); i++) {
          if (nums[i] == 1) {
              cnt++;
          }
          else {
              cnt = 0;
          }
          maxi = max(maxi, cnt);
      }
       return maxi;
    }
```

[T.C :- O(N)]
[S.C :- O(1)]

M. Imp:

Question No. 43 — N Meeting in one Room (GREEDY Algorithm)

Statement :- There is one meeting room in a firm. You are given two arrays, start and end each of size N. For an index 'i', start[i] denotes the starting time of the i^{th} meeting while end[i] will denote the ending time of the i^{th} meeting. Find the maximum No. of meeting that can be accommodated if only one meeting can happen in the room at a particular time. Print the order in which these meetings will be performed.

Ex :- Input :- N = 6,
start[] = {1, 3, 0, 5, 8, 5}
end[] = {2, 4, 5, 7, 9, 9}

output :- 1 2 4 5

Explanation :-

Meeting No.	1 ✓	2 ✓	3	4 ✓	5 ✓	6
Start time	1	3	0	5	8	5
End time	2	4	5	7	9	9

Soln :-

S[] :- 1 0 3 8 5 8 → resting in order of their

F[] 2 6 4 9 7 9 → Sort the finishing tym

1 2 3 4 5 6

take a vector and store their with positions.

↓ Sort acc. to their finishing tym.

✓	(1,2,①)
✓	(3,4,③)
x	(0 6,2)
✓	(5,7,5)
✓	(8,9,4)
x	(8,9,6)

①③⑤④ end limit = x * * 9 update cross tym

O/P : 1 3 5 4

traverse through sorted order of my finishing tym

T.C :- O(N) + O(N LogN) ≃ O(N LogN)

↳ for Sorted.

S.C :- O(N)

↳

Code :-

```
# include <bits/stdc++.h>
using namespace std;

struct meeting {
    int start;
    int end;
    int pos;
};
class solution {
public:
    bool static comparator(struct meeting m1, meeting m2) {
        if (m1.end < m2.end) return true;
        else if (m1.end > m2.end) return false;
        else if (m1.pos < m2.pos) return true;
        return false;
    }
void maxMeetings(int s[], int e[], int n){
    struct meeting meet[n];
    for(int i=0; i<n; i++) {
        meet[i].start = s[i], meet[i].end = e[i], meet[i].pos =
                                                    i+1;
}
```

```
sort(meet, meet+n, comparator);
    vector<int> answer;

    int limit = meet[0].end;
    answer.push_back(meet[0].pos);

for(int i=1; i<n; i++) {
        if(meet[i].start > limit) {
        limit = meet[i].end;
        answer.push_back(meet[i].pos);
    }
}

cout<<" The order in which the meetings will be performed is "<<endl;
for(int i=0; i<answer.size(); i++) {
        cout<<answer[i]<<" ";
    }
}
};

int main() {
    Solution obj;
    int n=6;
    int start[] = {1,3,0,5,8,5};
    int end[] = {2,4,5,7,9,9};
    obj.maxMeetings(start, end, n);
    return 0;
}
```

[for second method open geeg]

(MOST IMP.) [Myntra Round 2]

Question No. 44 **Minimum No. of platform required for a railway.**

Statement :- We are given two arrays that represent the arrival and departure times of trains that stop at the platform. We need to find the minimum No. of platform needed at the railway station so that no train has to wait.

Input :- N = 6

arr [] = {9:00, 9:45, 9:55, 11:00, 15:00, 18:00}

dep [] = {9:20, 12:00, 11:30, 11:50, 19:00, 20:00}

output : 3

Start = [120, 50, 550, 200, 700, 850]

End = [600, 550, 700, 500, 900, 1000]

(i) firstly ask whether they are in sorted or not. If not sort both start and end in ascending order.

Start [] : [50, 120, 200, 550, 700, 850]

End [] : [500, 550, 600, 700, 900, 1000]

Platform = 1

[T.C : O(2n log n) + O(2n)
S.C : O(1)]

Code :-

```
class Solution {
    public:
     int findPlatform (int arr [], int dep[] , int n)
    {
     Sort (arr, arr +n);
     Sort (dep, dep+n );

     int plat-needed = 1 , result = 1;
      int i=1 , J=0;
    while (i<n && J<n) {
         if (arr[i]<=dep [J]) {
          plat-needed ++;
```

```
        i++;
    }
    else if (arr[i] > dep[j]) {
        plat_needed--;
        j++;
    }
    if (plat_needed > result)
        result = plat_needed;
}

return result;
}
```

Question No:- **Job Sequencing Problem**

Statement:- You are given a set of N Jobs where each Job comes with a deadline and profit. The profit can only be earned upon completing the Job within its deadline. Find the number of Jobs done and the maximum profit that can be obtained. Each Job takes a single unit of time and only one Job can be performed at a time.

I/P:- N = 4 , Jobs = { (1,4,20) , (2,1,10) , (3,1,40) , (4,1,30) }

O/P:- 2 60

Explanation:- The 3rd Job with a deadline 1 is performed during the first unit of time. The 1st Job is performed during the second unit of time as its deadline is 4

profit = 40+20 = 60

Soln:-

id	deadline	profit
1	4	60
2	1	10
3	1	40
4	1	30

T.C :- O(NlogN) + O(M*N)

S.C :- O(M) ↳ for array.

Steps :-

(i) To find maximum profit, first we sort ~~id~~, deadline and profit in decending order.

ex 2)

id	deadline	profit
6	2	80
3	6	70
4	6	65
2	5	60
5	4	25
8	2	22
1	4	20
7	2	10

(ii) check for the maxm deadline so we have an extra sym to accomdate the other jobs.

(iii) Take an array of 6 (max deadline) and initialise with -1.

8	6 (id)	5	2	4	3
-1	-1	-1	-1	-1	-1
1	2 (deadline)	3	4	5	6

80+70+65 +60+25+22

profit = 80+70+65+60+25+22

No. of Jobs = 6

(iv) for Job Sequencing return the array of sequence (8, 6, 5, 2, 4, 3)

Intuition :- (i) For every Job check if it can be performed on its last day.

(ii) If possible mark that index with the Job id and add the profit to our answer.

(vi) If not possible, loop through the previous indexes until an empty slot is found.

Code :-

```
# include <bits/stdc++.h>
using namespace std;
struct Job {
    int id;
    int dead;
    int profit;
};
class Solution {
public:
    bool static comparison (Job a, Job b) {
        return (a.profit > b.profit);
    }
pair <int, int> JobScheduling (Job arr[], int n) {
```

```
sort(arr, arr+n, comparison);
int maxi = arr[0].dead;
for(int i=1 ; i<n; i++) {
        maxi = max(maxi, arr[i].dead);
}

int slot[maxi+1] ;
for(int i=0 ; i<=maxi; i++)
  slot[i] = -1 ;

int countJobs = 0 ; jobProfit = 0;
for(int i=0 ; i<n ; i++){
    for(int j = arr[i].dead ; j>0; j--) {
        if( slot[j] == -1) {
            slot [j] = i;
            countJobs ++;
            JobProfit += arr[i].profit ;
            break;
        }
    }
}
return make-pair(countJobs, JobProfit);
}
};
int main() {
    int n=4;
    Job arr[n] = {{1,4,20}, {2,1,10}, {3,2,40}, {4,2,30}};

    Solution ob;
// function call
    pair<int, int> ans = ob.JobScheduling(arr, n);
    cout << ans.first << " " << ans.second << endl;

    return 0;
}
```

Question :- **Fractional Knapsack Problem**

Statement :- The weight of N items and their corresponding values are given. We have to put these items in a knapsack of weight w such that total value obtained is maximized.

I/P :- N = 3, w = 50
values [] = {100, 60, 120}
weight [] = {20, 10, 30}

O/P :- 240.00

Explanation :- The first and second items are taken as a whole while only 20 units of the third item is taken.
Total value = 100 + 60 + 80 = 240.00

Sol^n :- values [] = {60, 100, 120}] find value per weight.
weight [] = {10, 20, 30}]

v = (6 5, 4) → value per weight

(i) Sort the item in value per weight (descending order)

20	×4 → 80
20	×5 → 100
10	×6 → 60

w = 50

⇒ total = 100 + 80 + 60 ⇒ 240.

T.C : NlogN + N
S.C : O(1)

Code :-

```
# include <bits/stdc++.h>
using namespace std;

struct Item {
    int value;
    int weight;
};
class Solution {
    public:
    bool static comp(Item a, Item b) {
        double r1 = (double) a.value / (double) a.weight;
    }
};
```

```
cout <<" The minimum number of coins is " << ans.size() << endl;
cout <<" The coins are " << endl;
for (int i=0 ; i<ans.size() ; i++) {
        cout << ans[i] << " " ;
}
return 0;
}
```

O/P:- The minimum Number of coin is 5

The coins are

20 20 5 2 2

T.C :- O(V)

S.C :- O(1)

→ B.S can be implemented to any Search space which is monotonic (increasing)

Question No:- **BINARY SEARCH [The N^th Root of an Integer]**

Statement:- Given two numbers N and M, find the N^th root of M.

→ The N^th root of a number M is defined as a number x when raised to the power n equals M.

Eg 1) I/P: N=3 M=27

O/P:- 3

Explanation:- The cube root of 27 is 3

Step 1:- Take low and high. low will be equal to 1 and high will be M. We will take the mid of low and high such that the Searching space is reduced using $\frac{low+high}{2}$.

Step 2:- Make a Sepate function to multiply mid N times.

Step 3:- Run the while loop till (high-low > eps). Take eps as $1e^{-6}$ (we have to return ans at 6 places of decimal).

Step 4:- If the mid is less than M, then we will reduce Search space to low and mid. Else, if it is greater than M then Search space will be reduced to mid and high.

Code :-

```
class Solution {
    public :
    double multiply (double number , int n) {
        double ans = 1.0 ;
        for (int i = 1 ; i <= n ; i++) {
            ans = ans * number ;
        }
        return ans ;
    }
void getNthRoot (int n , int m) {
    double low = 1 ;
    double high = m ;
    double eps = 1e-6 ;

    while ((high - low) > eps) {
        double mid = (low + high) / 2.0 ;
        if (multiply (mid , n) < m) {
            low = mid ;
        }
        else {
            high = mid ;
        }
    }
    return low ; or return high ;
```

T.C :- N log (M x 10^d)

S.C :- O(1)

Question No :- Matrix Median

Stat :- Given matrix of 'N' rows and 'M' columns filled up with integer where every row is sorted in Non-decreasing order. we have to find overall median of the matrix ie. if all the elements of the matrix are written in a single line.

Eg:- $\begin{bmatrix} 1 & 3 & 6 \\ 2 & 6 & 9 \\ 3 & 6 & 9 \end{bmatrix}$ N=3, M=3 [N×M → odd]

All Sorted (Row)

1, 2, 3, 3, [6], 6, 6, 9, 9

middle = median

"Naive Approach :- Iterate to each and every element and store it in some data structure and then sort (linearly). Middle of that linear array is N/a as median.

T.C :- N×M Log(N×M)
- Log(N×M) → for sorting
- N×M → for storing in DS

S.C :- O(N×M)
- N×M → for storing in data structure.

Efficient Approach :- [1 ... 10^9] median can lie b/w these range.

Eg :- [1 7 15], for understanding only.

mid $\frac{1+15}{2}$ = 8 7 find how many ele. (<=8).

In Row 1 ; 3 are <=8
2 ; 2 are <=8
3 ; 2 are <=8

3+2+2 = 7

mid = $\frac{7+1}{2}$ = 4

Algorithm ; For a Number to be median there should be exactly (N/2) numbers which are less than this NO.

- First we find the minimum and maximum elements in the matrix.

(i) Minimum element can be easily found by comparing the first element of each row.

(ii) Maximum element can be found by comparing the last element of each row.

(iii) Find the mid of the min and max.

(iv) And get count of numbers (in the matrix) less than our mid. and accordingly change the min or max.

```
if (count < (r*c+1)/2)
    min = mid + 1   // the median must be greater than the selected
else                NO.
    max = mid   // the median must be less than or equal to the
                selected NO.
```

Code :-

```
class solution {
    public :
        int binaryMedian (int m[][MAX] , int r, int c)
        {
            int min = INT_MAX, max = INT_MIN;
            for(int i=0; i<r; i++)
            {
                // Finding the minimum element
                if (m[i][0] < min)
                    min = m[i][0];
                // Finding the maximum element
                if (m[i][c-1] > max)
                    max = m[i][c-1];
            }
            int desired = (r*c+1)/2;   // Index of median if mat is put in array format
            while (min<max)
            {
                int mid = min + (max-min)/2;
                int place = 0;
                // Find count of elements smaller than mid
                for(int i=0; i<r; ++i)
                    place += upper_bound (m[i], m[i]+c, mid) - m[i];
                if (place < desired)
                    min = mid+1;          // No. of elements that is less than mid.
                else
                    max = mid;
            }
            return min;
        }
};
```

T.C :- $O[32 * r * \log(c)]$

S.C :- $O(1)$

- upperbound function :- `upper_bound (start_itr , end_itr, val)`

(i) returns iterator to first greater element.

(ii) for sorted array/vector.