

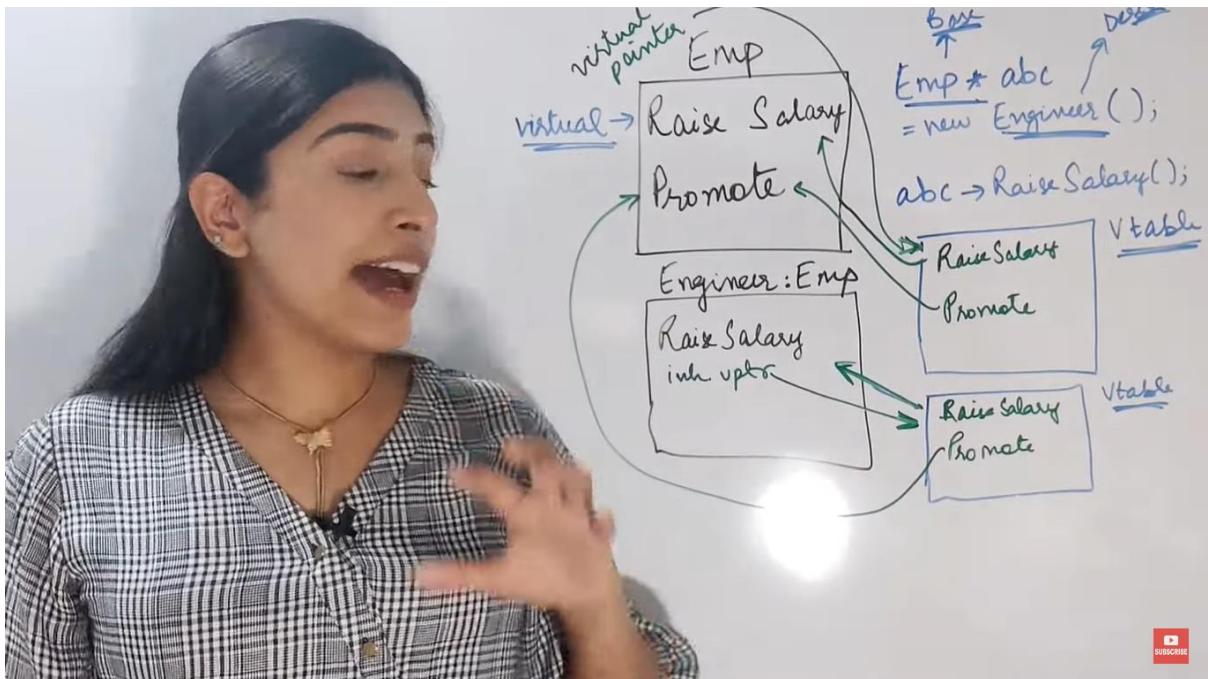
SQL vs NoSQL

1. SQL databases are relational, NoSQL are non-relational.
2. SQL databases use structured query language and have a predefined schema. NoSQL databases have dynamic schemas for unstructured data.
3. SQL databases are vertically scalable, NoSQL databases are horizontally scalable.
4. SQL databases are table based, while NoSQL databases are document, key-value, graph or wide-column stores.
5. SQL databases are better for multi-row transactions, NoSQL are better for unstructured data like documents or JSON.

AMAZON STORIES

- 51: "कैसे तोड़ते हैं बैकेन्ड पर"
- 52: Interns' Accident
- 53: Intern late deliverer का लिए जो अलग से अलग J.D. से आवाज दी जाती है वह काम करने की ओर की ओर बदल गया।
- 54: Two teams working on same project, And lot of duplicate work was being done.
- 55: Meeting में कौछल लगा भूल गया (जिसके कारण से उसके Problem हुए)
- Problem: कौछल लगा DSA के Ques लिए ही नहीं School level के लिए ही भी नहीं लगा तो उसकी वजह से Problem हो गई (अच्छी Difficulty + Ques Set होने)
- 56: Weakness: I got nervous talking to new people.
- 57: Not Satisfied with performance : Initially when I joined I used to depend more on other people.
- 58: Tension with Manager : Cfg में ER feature implement करना था उसमें जो videos paid कर रहे थे उन्हें Unpaid रखकर उसके लिए Revenue.

RUN-TIME POLYMORPHISM



[link1](#) [link2](#)

Copy Constructor

Singleton Class

Can a constructor be private in C++?

Lock v/s Mutex v/s Semaphore

Data Structure

- A data structure is a particular way of organizing data in a computer so that it can be used effectively.

7Chocolate distribution(Sorting) V/S Minimum Pages Allocation (BSearch)

Chocolates ese baatni h ki har bcche ko exactly ek pkt mile aur jis bcche ko max chocolate mili aur jisko minimum chocolate mili vh diff minimum hona chahiye.

Pages vale m ese book deni h ki ek bnde ko jo maximum pages mil skte h vh minimum ho jese 113 ans tha th mtlb 113 se kam pages agar m du ek bnde ko th bnde km pd jaenge

When storing the number store by using $(\text{nums}[i]\% \text{maxs}) * \text{maxs}$ and when retrieved the original will be retrieved by $(\text{nums}[i]) \% \text{maxs}$) and the other no. at the same index by $(\text{nums}[i] / \text{maxs})$.

ARRAYS

Merge two sorted arrays without extra space

```
int maxs=100005;
void merge(long long ar1[], long long ar2[], int n, int m)
{
    int i=0,j=0,k=n-1,idx=0;
    int mins=min(ar1[0],ar2[0]);
    if(mins>0)
        mins=0;
    for(int x=0;x<n;x++)
    {
        ar1[i]=ar1[i]-mins;
    }
    for(int x=0;x<m;x++)
    {
        ar2[i]=ar2[i]-mins;
    }
    while(i<n&&j<m)
    {
        if((ar1[i]%maxs)<(ar2[j]%maxs))
        {
            if(idx<n)
                ar1[idx++]+=(ar1[i]%maxs)*maxs;
            else if(idx>=n)
            {
                ar2[idx-n]+=(ar1[i]%maxs)*maxs;
                idx++;
            }
            i++;
        }
        else
        {
            if(idx<n)
                ar1[idx++]+=(ar2[j]%maxs)*maxs;
            else if(idx>=n)
            {
                ar2[idx-n]+=(ar2[j]%maxs)*maxs;
                idx++;
            }
        }
    }
}
```

```

        j++;
    }
}
while(i<n)
{
    if(idx<n)
        ar1[idx++]+=(ar1[i]%maxs)*maxs;
    else if(idx>=n)
    {
        ar2[idx-n]+=(ar1[i]%maxs)*maxs;
        idx++;
    }
    i++;
}
while(j<m)
{
    if(idx<n)
        ar1[idx++]+=(ar2[j]%maxs)*maxs;
    else if(idx>=n)
    {
        ar2[idx-n]+=(ar2[j]%maxs)*maxs;
        idx++;
    }
    j++;
}
for(int i=0;i<n;i++)
{
    ar1[i]=ar1[i]/maxs;
    ar1[i]-=mins;
}
for(int i=0;i<m;i++)
{
    ar2[i]=ar2[i]/maxs;
    ar2[i]-=mins;
}
}

```

Last k 2 for loops m ar1[i]+=mins and ar2[i]+=mins aaega

Rearrange positive and negative with order maintained

```
void rearrangePosNeg(int arr[], int n){  
    //find min and max in the array  
    int min = 0, max = 0;  
    for(int i=0; i<n; i++)  
        if(arr[i]<min) min = arr[i];  
        else if(arr[i]>max) max = arr[i];  
    //Converting the array of positive and negatives into an array of positives only  
    for(int i=0; i<n; i++)  
        arr[i] -= min;  
    //Any element above or equal to -min is positive else negative  
    //Pushing negatives in the front.  
    int divisor = max - min + 1;  
    int k = 0;  
    for(int i=0; i<n; i++)  
        if(arr[i]%divisor < -min)  
            arr[k++] += (arr[i]%divisor) * divisor;  
    //Pushing positives afterwards.  
    for(int i=0; i<n; i++)  
        if(arr[i]%divisor >= -min)  
            arr[k++] += (arr[i]%divisor) * divisor;  
    //Converting numbers back to their original form  
    for(int i=0; i<n; i++) {  
        arr[i] /= divisor;  
        arr[i] += min;  
    }  
}  
int main() {  
    int arr[] = { -12, 11, -13, -5, 6, -7, 5, -3, -6 };  
    int n = sizeof(arr) / sizeof(arr[0]);  
    printArray(arr, n);  
    rearrangePosNeg(arr, n);  
    printArray(arr, n);  
    return 0;  
}
```

Find the Duplicate Number (hare and

```
int findDuplicate(vector<int>& nums) {
    int n=nums.size();
    int hare=nums[0];
    int tortoise=nums[0];
    while(1)
    {
        tortoise=nums[tortoise];
        hare=nums[nums[hare]];
        if(hare==tortoise)
            break;
    }
    tortoise=nums[0];
    while(hare!=tortoise)
    {
        hare=nums[hare];
        tortoise=nums[tortoise];
    }
    return hare;
}
```

tortoise)

*At meet point

$$\text{hare} = d + nl + x$$

$$\text{tortoise} = d + x$$

$$\Rightarrow d + n * l + x = 2 * (d + x)$$

$$\Rightarrow d + x = n * l$$

*After meeting comparing position

$$\Rightarrow \text{tortoise} = d$$

$$\Rightarrow \text{hare} = (d + x) + d = n * l + d = d$$

Pascal's Triangle II

sbse phle th ith index p 1 krdo fr usse phle ka sum lelo $a[i] + a[i-1]$ ka ese hi 2 se row tk k liye chlado

```
vector<int> getRow(int row) {
    if(row==0)
        return {1};
    else if(row==1)
        return {1,1};
    else
    {
        vector<int> v(row+1,0);
        v[0]=1;
        v[1]=1;
        for(int i=2;i<=row;i++)
        {
            v[i]=1;
            int temp=v[0];
            for(int j=1;j<i;j++)
            {
                int temp2=v[j];
                v[j]+=temp;
                temp=temp2;
            }
        }
        return v;
    }
}
```

Diagonal Traverse

Key Idea=>In a 2D matrix, elements in the same diagonal have the same sum of their indices.

```

vector<int> findDiagonalOrder(vector<vector<int>>& nums) {

    //Key Idea => In a 2D matrix, elements in the same diagonal have
    //same sum of their indices.

    unordered_map<int,vector<int>>m;
    int maxs=0;
    for(int i=0;i<nums.size();i++)
    {
        for(int j=0;j<nums[i].size();j++)
        {
            m[i+j].push_back(nums[i][j]);
            maxs=max(maxs,i+j);
        }
    }
    vector<int>ans;
    for(int sum=0;sum<=maxs;sum++)
    {
        for(auto it=m[sum].end()-1;it>=m[sum].begin();it--)
        {
            ans.push_back(*it);
        }
    }
    return ans;
}

```

First Missing Positive (TC:O(N))

Hard 6767 1104 Add to List

Given an unsorted integer array `nums`,
return the smallest missing positive integer.

You must implement an algorithm that runs
in $O(n)$ time and uses constant extra
space.

Example 1:

Input: `nums = [1,2,0]`
Output: 3

Example 2:

```

4 *      /*
5 * Put each number in its right place.
6 * For example:
7 * When we find 5, then swap it with A[4].
8 * At last, the first place where its number is not right, return the place.
9 */
10 * int firstMissingPositive(vector<int>& nums) {
11 *     int n=nums.size();
12 *     for(int i=0;i<n;i++)
13 *     {
14 *         while(nums[i]>0&&nums[i]<=n&&nums[i]!=nums[nums[i]-1])
15 *         {
16 *             swap(nums[i],nums[nums[i]-1]);
17 *         }
18 *     }
19 *     for(int i=0;i<n;i++)
20 *     {
21 *         if(nums[i]!=i+1)
22 *             return i+1;
23 *     }
24 *     return n+1;
25 *

```

Set Matrix Zeroes

```
void setZeroes(vector<vector<int>>& matrix) {
    int n=matrix.size();
    int m=matrix[0].size();
    bool is_first_row_zero=false,is_first_col_zero=false;
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            if(matrix[i][j]==0)
            {
                if(i==0)
                    is_first_row_zero=true;
                if(j==0)
                    is_first_col_zero=true;
                matrix[i][0]=0;
                matrix[0][j]=0;
            }
        }
    }
    for(int i=1;i<n;i++){
        for(int j=1;j<m;j++){
            if(matrix[i][0]==0||matrix[0][j]==0)
                matrix[i][j]=0;
        }
    }
    if(is_first_row_zero)
    {
        for(int j=0;j<m;j++)
            matrix[0][j]=0;
    }
    if(is_first_col_zero)
    {
        for(int i=0;i<n;i++)
            matrix[i][0]=0;
    }
}
```

Largest Number

```
bool comparison(string x, string y)
{
    string s1=x;
    string s2=y;
    string c1=s1+s2;
    string c2=s2+s1;

    return(c1>c2);
}

string Solution::largestNumber(const vector<int> &a) {
    vector<string>temp;
    for(int i=0;i<a.size();i++)
        temp.push_back(to_string(a[i]));
    sort(temp.begin(),temp.end(),comparison);
    string ans="";
    for(int i=0;i<temp.size();i++)
        ans+=temp[i];
    while(ans.size()>1&&ans[0]=='0')//for 0 0 0 0 if
        //this is not included it
        //will show output as 0000 but we
        //have to show output 0
    ans=ans.substr(1);
    return ans;
}
```

Merge Intervals

```
vector<vector<int>> merge(vector<vector<int>>&a) {
    sort(a.begin(),a.end());
    vector<vector<int>>ans;
    for(int i=0;i<a.size()-1;i++)
    {
        if(a[i][1]<=a[i+1][1]&&a[i][1]>=a[i+1][0])
        {
            a[i+1][0]=min(a[i][0],a[i+1][0]);
            a[i+1][1]=max(a[i][1],a[i+1][1]);
            a[i][0]=-1;
            a[i][1]=-1;
        }
        else if(a[i+1][0]>a[i][0]&&a[i][1]>a[i+1][0])
        {
            a[i+1][0]=min(a[i][0],a[i+1][0]);
            a[i+1][1]=max(a[i][1],a[i+1][1]);
            a[i][0]=-1;
            a[i][1]=-1;
        }
    };
    for(int i=0;i<a.size();i++)
    {
        if(a[i][0]!=-1&&a[i][1]!=-1)
            ans.push_back(a[i]);
    }
    return ans;
}
```

Insert Interval

Example 1:

```
Input: intervals = [[1,3],[6,9]],  
newInterval = [2,5]  
Output: [[1,5],[6,9]]
```

Example 2:

```
Input: intervals = [[1,2],[3,5],  
[6,7],[8,10],[12,16]],  
newInterval = [4,8]  
Output: [[1,2],[3,10],[12,16]]  
Explanation: Because the new  
interval [4,8] overlaps with  
[3,5],[6,7],[8,10].
```

Example 3:

```
public:  
#define pb push_back  
vector<vector<int>> insert(vector<vector<int>> &intervals, vector<int>& newInterval) {  
    vector<vector<int>>result;  
    for(int i=0;i<intervals.size();i++)  
    {  
        if(newInterval[0]>intervals[i][1])  
        {  
            result.pb(intervals[i]);  
        }  
        else if(newInterval[1]<intervals[i][0])  
        {  
            result.pb(newInterval);  
            newInterval=intervals[i];  
        }  
        else if(intervals[i][1]>newInterval[0]||newInterval[1]>=intervals[i][0])  
        {  
            newInterval[0]=min(intervals[i][0],newInterval[0]);  
            newInterval[1]=max(intervals[i][1],newInterval[1]);  
        }  
    }  
    result.pb(newInterval);  
    return result;  
}
```

Find Permutation

```
vector<int> Solution::findPerm(const string s, int n) {
    vector<int> v;
    for(int i=1;i<=n;i++)
        v.push_back(i);
    for(int i=0;i<s.size();i++)
    {
        int j;
        if(s[i]=='D')
        {
            j=i;
            while(s[j]=='D'&&j<s.size())
                j++;
            int k=j;
            while(i<=k)
                swap(v[i++],v[k--]);
            i=j;
        }
    }
    return v;
}
/*EXAMPLE N=5 S=IDDI th ab v bnalo 1 2 3 4 5 ab agar I h th pgle se hi vector m humne increased m daale h ab agar D h th dekho ki D kitna chl rha h aage tk jese isme 2 idx tk chl rha h string m mtlb vector m 3 idx tk th 1 idx se lekr 3 idx tk reverse krdo jisse decreasing m aa jaenge.*/

```

```
vector<int> Solution::findPerm(const string A, int B) {
    vector<int> v;
    int l=1,r=B;
    int i=0;
    if(!A.length())return v;
    while(l<=r && v.size()!=A.length()+1)
    {
        if(A[i]=='I')
        {
            v.push_back(l);
            l++;
        }
        else
        {
            v.push_back(r);
            r--;
        }
        i++;
    }
    return v;
}
```

Maximum Sum Square SubMatrix

```
int window_slide(vector<int>&v, int B)
{
    int sum=0,i,j;
    for(j=0;j<B;j++)
    {
        sum+=v[j];
    }
    int max_sum=sum;
    while(j<v.size())
    {
        sum=sum-v[i++];
        sum+=v[j++];
        max_sum=max(sum,max_sum);
    }
    return max_sum;
}
int Solution::solve(vector<vector<int> > &matrix, int B) {
    int n=matrix.size(),ans=INT_MIN;
    for(int row1=0,x=0;row1<=n-B;row1++,x++)
    {
        vector<int>v(n,0);
        for(int row2=row1;row2<=B-1+x;row2++)
        {
            for(int col=0;col<n;col++)
            {
                v[col]+=matrix[row2][col];
            }
        }
        ans=max(ans,window_slide(v,B));
    }
    return ans;
}
//same as kadanes vala h but isme B fix krdia th kadanes se th B fix nhi rhta h isme B
//fix krdia th sliding window lo B size ki usse sum dekhte jao B size ka uska max lelo
```

Balanced Array

```
int Solution::solve(vector<int> &a) {
    int so=0,se=0,n=a.size();
    for(int i=0;i<n;i++)
    {
        if(i%2)
            so+=a[i];
        else
            se+=a[i];
    }
    int curr_odd=0,curr_even=0,cnt=0;
    //har element ko dekho ki kisi usko htane se th special nhi bn rhi agar koi element htaya th uske
    //baad ka even sum odd sum bn jaega and vice versa
    for(int i=0;i<n;i++)
    {
        //let suppose ith element is special element
        if(i%2==1)
        {
            int odd_sum=curr_odd+se-curr_even;
            int even_sum=curr_even+so-curr_odd-a[i];
            if(odd_sum==even_sum)
                cnt++;
            curr_odd+=a[i];
        }
        else
        {
            int odd_sum=curr_odd+se-curr_even-a[i];
            int even_sum=curr_even+so-curr_odd;
            if(odd_sum==even_sum)
                cnt++;
            curr_even+=a[i];
        }
    }
    return cnt;
}
```

Count number of triplets in an array having sum in the range [a, b]

```

/*Find count of triplets having a sum less than or equal to b. Let this count be x.
Find count of triplets having a sum less than a. Let this count be y.
Final result is x-y.*/
int countTripletsLessThan(int arr[], int n, int val)
{
    sort(arr, arr + n);
    int ans = 0;
    int j, k;
    int sum;
    for (int i = 0; i < n - 2; i++) {
        j = i + 1;
        k = n - 1;
        while (j != k) {
            sum = arr[i] + arr[j] + arr[k];
            if (sum > val)
                k--;
            else {
                ans += (k - j);
                j++;
            }
        }
    }
    return ans;
}
int countTriplets(int arr[], int n, int a, int b)
{int res;
// Find count of triplets having sum less
// than or equal to b and subtract count
// of triplets having sum less than or
// equal to a-1.
res = countTripletsLessThan(arr, n, b) -
      countTripletsLessThan(arr, n, a - 1);

return res;
}

```

Car Pooling

car's initial location.

Return `true` if it is possible to pick up and drop off all passengers for all the given trips, or `false` otherwise.

Example 1:

```
Input: trips = [[2,1,5],[3,3,7]],  
capacity = 4  
Output: false
```

Example 2:

```
Input: trips = [[2,1,5],[3,3,7]],  
capacity = 5  
Output: true
```

```
3 v  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
bool carPooling(vector<vector<int>>& trips, int capacity) {  
    map<int, int> m;  
    //map lia h kyuki max 1000 tk hi h value  
    for (auto &t : trips)  
        m[t[1]] += t[0], m[t[2]] -= t[0];  
    for (auto &v : m)  
        if ((capacity -= v.second) < 0)  
            return false;  
    return true;  
}
```

Shortest Unsorted Continuous Subarray

```
int findUnsortedSubarray(vector<int>& nums) {  
    int shortest = 0;  
  
    int left = 0, right = nums.size() - 1;  
    while (left < nums.size() - 1 && nums[left] <= nums[left + 1]) { left++; }  
    while (right > 0 && nums[right] >= nums[right - 1]) { right--; }  
  
    if (right > left) {  
        int vmin = INT_MAX, vmax = INT_MIN;  
        for (int i = left; i <= right; ++i) {  
            if (nums[i] > vmax) {  
                vmax = nums[i];  
            }  
            if (nums[i] < vmin) {  
                vmin = nums[i];  
            }  
        }  
  
        while (left >= 0 && nums[left] > vmin) { left--; }  
        while (right < nums.size() && nums[right] < vmax) { right++; }  
  
        shortest = right - left - 1;  
    }  
  
    return shortest;  
}
```

Maximum Points You Can Obtain from Cards

```

int maxScore(vector<int>&cardPoints, int k) {
    int sum = 0;
    int n = cardPoints.size();
    vector<int> cummulativeSumFromFront(n+1, 0);
    vector<int> cummulativeSumFromBehind(n+1, 0);
    sum = 0;
    for (int i=0; i<n; i++) {
        sum += cardPoints[i];
        cummulativeSumFromFront[i+1] = sum;
    }
    sum = 0;
    for (int i=n-1; i>=0; i--) {
        sum += cardPoints[i];
        cummulativeSumFromBehind[i] = sum;
    }
    // Reversing is optional. I reversed it so that it would be easy
    // to access sum of last (k-i) elements by just indexing at [k-i]
    // Otherwise, I would have had to index it at [n-k+i] which would
    // have made it difficult to read.
    reverse(cummulativeSumFromBehind.begin(), cummulativeSumFromBehind.end());

    int answer = 0;
    for(int i=0; i<=k; i++) {
        answer = max(answer,
                     cummulativeSumFromFront[i] // Sum of first 'i' cards.
                     + cummulativeSumFromBehind[k-i]); // Sum of last 'k-i' cards.
    }
    return answer;
}

```

```

// Approach 4: Sliding Window Approach

// 1. We must draw exactly k cards from the array in such a way that the score
// (sum of the cards) is maximized.

//      After drawing k cards from the array cardPoints.length - k cards will
// remain in the array.

// 2. Another way that we could view the problem is that our objective is to
// choose cards from the beginning or end of

//      array in such a way that the sum of the remaining cards is minimized.

// 3. We can use a sliding window to find the subarray of size cardPoints.length -
// k that has the minimal sum.

//      Subtracting this value from the total sum of all the cards will give us
// our answer.

// TC: O(n), SC: O(1)

```

```

int maxScore(vector<int>& A, int B) {
    //sliding window
    int sum = 0, n = A.size();
    for(int i = n - 1; i >= (n - B); i--) {
        sum += A[i];
    }
    if(B == n) return sum;
    int result = sum;
    int r = 0;
    int l = (n - B);
    while(r < B) {
        sum += A[r];
        sum -= A[l];
        l = (l + 1) % n;
        r = (r + 1) % n;
        result = max(result, sum);
    }
    return result;
}

```

Flip in subarrays to get Maximum One's in array

You are given a binary string **A**(i.e. with characters **0** and **1**) consisting of characters **A₁, A₂, ..., A_N**. In a single operation, you can choose two indices **L** and **R** such that $1 \leq L \leq R \leq N$ and flip the characters **A_L, A_{L+1}, ..., A_R**. By flipping, we mean change character **0** to **1** and vice-versa.

Your aim is to perform **AT MOST** one operation such that the final string number of **1s** is maximised.

```
vector<int> Solution::flip(string s) {
    // 0 ko 1 m krlo 1 ko -1 m ab hume esi subarray dekhni h
    // jisme max 0 ho mtlb jiska maximum sum ho th Kadanes lgado
    int n=s.length();
    vector<int>v(n);
    for(int i=0;i<n;i++)
    {
        v[i]=(s[i]=='0')?1:-1;
    }
    int max_so_far=INT_MIN,max_ending_here=0;
    int start=0,end=0,s1=0;
    for(int i=0;i<n;i++)
    {
        max_ending_here+=v[i];
        if(max_ending_here<0)
        {
            max_ending_here=0;
            s1=i+1;
        }
        if(max_ending_here>max_so_far)
        {
            start=s1;
            end=i;
            max_so_far=max_ending_here;
        }
    }
    if(max_so_far>0)
        return {start+1,end+1};
    else
        return {};
}
```

Maximum Absolute Difference

```
int Solution::maxArr(vector<int> &arr) {
    int n=arr.size();
    int max1 = INT_MIN, min1 = INT_MAX;
    int max2 = INT_MIN, min2 = INT_MAX;

    /*after solving all the absolute we get two cases same so at last we got we have to find
    maximum from the two cases i.e. =>(1.) (arr[i]+i)-(arr[j]+j)      (2.) (arr[i]-i)-(arr[j]-j)
    for case1 to be maximum we have to max(arr[i]+i)-min(arr[j]+j).Similarly for case2 to be
    maximum we have to find max(arr[i]-i)-min(arr[j]-j).*/
    for (int i = 0; i < n; i++) {
        // Updating max and min variables
        // as described in algorithm.
        max1 = max(max1, arr[i] + i);
        min1 = min(min1, arr[i] + i);
        max2 = max(max2, arr[i] - i);
        min2 = min(min2, arr[i] - i);
    }

    // Calculating maximum absolute difference.
    return max(max1 - min1, max2 - min2);
}
```

Partitions

```
bool canThreePartsEqualSum(vector<int>& a) {
    int sum = accumulate(a.begin(), a.end(), 0);

    if(sum%3!=0) return false;

    int subSum = sum/3;

    bool firstFound=false, secondFound = false, thirdFound = false;

    int prefixSum=0;
    for(auto ele : a)
    {
        prefixSum += ele;
        if(!firstFound && prefixSum==subSum) firstFound = true;
        else if(firstFound && !secondFound && prefixSum==subSum*2) secondFound = true;
        else if(firstFound && secondFound && prefixSum==subSum*3) thirdFound = true;
    }

    return (firstFound && secondFound && thirdFound);
}
```

Minimise the maximum difference between heights (Explanation)

```
int smallestRangeII(vector<int>& nums, int k) {
    sort(nums.begin(),nums.end());
    int n=nums.size();
    int result=nums[n-1]-nums[0];
    for(int i=0;i<n-1;i++)
    {
        int a=nums[i],b=nums[i+1];
        int maxs=max(a+k,nums[n-1]-k);
        int mins=min(b-k,nums[0]+k);
        result=min(result,maxs-mins);
    }
    return result;
}
```

Dutch National Flag Algorithm

```
void sortColors(vector<int>&a) {
    int n=a.size();
    int low=0,mid=0,high=n-1;
    while(mid<=high)
    {
        if(a[mid]==0)
        {
            swap(a[mid],a[low]);
            mid++;
            low++;
        }
        else if(a[mid]==1)
        {
            mid++;
        }
        else if(a[mid]==2)
        {
            swap(a[mid],a[high]);
            high--;
        }
    }
}
```

Maximum Sum Path in Two Arrays

Given two arrays, where if a value is present in both the array, you can switch b/w arrays, find the maximum sum

```
int maxPathSum(int ar1[], int ar2[], int m, int n)
{
    int i = 0, j = 0;
    int result = 0, sum1 = 0, sum2 = 0;
    // Below 3 loops are similar to merge in merge sort
    while (i < m && j < n)
    {
        if (ar1[i] < ar2[j])
            sum1 += ar1[i++];
        else if (ar1[i] > ar2[j])
            sum2 += ar2[j++];
        else // we reached a common point
        {
            result += max(sum1, sum2) + ar1[i];
            sum1 = 0;
            sum2 = 0;
            i++;
            j++;
        }
    }
    while (i < m)
        sum1 += ar1[i++];
    while (j < n)
        sum2 += ar2[j++];
    result += max(sum1, sum2);
    return result;
}
```

Picks up a random song from the playlist and plays it. (ZOMATO)

There is an audio player given that picks up a random song from the playlist and plays it. The song should not be repeated until all songs are played at least once. The sequence number of the songs is given in an array. Design such an audio player without using any extra space.

Write an algorithm for that.

```

#include<bits/stdc++.h>
#include <stdlib.h>
#include <time.h>
using namespace std;
void swap (int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
void randomize (int arr[], int n)
{
    // Use a different seed value so that
    // we don't get same result each time
    // we run this program
    srand (time(NULL));
    for (int i = n - 1; i > 0; i--)
    {
        int j = rand() % (i + 1);
        swap(&arr[i], &arr[j]);
    }
}
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
    int n = sizeof(arr) / sizeof(arr[0]);
    randomize (arr, n);
    printArray(arr, n);

    return 0;
}
Time Complexity: O(n), assuming that the function rand() takes O(1) time.

```

How to check if two given line segments intersect? (ZOMATO)

```

struct Point
{
    int x;
    int y;
};

// Given three collinear points p, q, r, the function checks if
// point q lies on line segment 'pr'
bool onSegment(Point p, Point q, Point r)
{
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
        return true;
    return false;
}

int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);
    if (val == 0) return 0; // collinear
    return (val > 0)? 1: 2; // clock or counterclock wise
}

// The main function that returns true if line segment 'p1q1'
// and 'p2q2' intersect.
bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
    // Find the four orientations needed for general and
    // special cases
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);
    // General case
    if (o1 != o2 && o3 != o4)

```

```
bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
    // Find the four orientations needed for general and
    // special cases
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);
    // General case
    if (o1 != o2 && o3 != o4)
        return true;
    // Special Cases
    // p1, q1 and p2 are collinear and p2 lies on segment p1q1
    if (o1 == 0 && onSegment(p1, p2, q1)) return true;
    // p1, q1 and q2 are collinear and q2 lies on segment p1q1
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;
    // p2, q2 and p1 are collinear and p1 lies on segment p2q2
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;
    // p2, q2 and q1 are collinear and q1 lies on segment p2q2
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;
    return false; // Doesn't fall in any of the above cases
}
int main()
{
    struct Point p1 = {1, 1}, q1 = {10, 1};
    struct Point p2 = {1, 2}, q2 = {10, 2};
    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";
    return 0;
}
```

COPY

Next Permutation

21

Step 1) Find the largest index k such that $\text{nums}[k] < \text{nums}[k+1]$.
If no such index exists, just reverse nums & done.

Step 2) Find the largest index $l > k$ such that $\text{nums}[k] < \text{nums}[l]$.

Step 3) Swap $\text{nums}[k]$ and $\text{nums}[l]$.

Step 4) Reverse the subarray $\text{nums}[k+1:n]$;

*Code

```
void nextPermutation (vector<int>&a)
```

```
{ int n = a.size(); int k, l;
```

```
for (k = n - 2; k >= 0; k--)
```

```
    if (a[k + 1] > a[k]) break;
```

```
    if (k == -1)
```

```
        sort(a.begin()); return;
```

```
    for (int i = n - 1; i > k; i++)
```

```
        if (a[i] > a[k]) break;
```

```
    swap(a[i], a[k]);
```

```
    reverse(a.begin() + k + 1, a.end());
```

19

SATURDAY OCTOBER

3. Count pairs with given sum

```
int getPairCount( int a[], int n, int k) {
    unordered_map<int, int> m; ans=0;
    for( int i=0; i<n; i++ )
        m[a[i]]++;
    for( int i=0; i<n; i++ )
        if ( m[a[i]+k] )
            ans += m[k-a[i]];
        if ( (k-a[i]) == a[i] ) (for i=6-6)
            ans--;
    }
    return ans/2;
}
```

3.

4. Count Inversion Using Merge Sort

```
int mergeSort( int arr[], int n )
{
    int temp[n];
    return _mergeSort( arr, temp, 0, n );
}
```

20 SUNDAY

```
int _mergeSort( int arr[], int temp[], int left, int right )
{
    int mid, inv_cnt = 0;
```

```

9 }   b(left < right)
mid = (left + right) / 2;
inv_cnt += mergeSort(aux, temp, left, mid);
inv_cnt += mergeSort(aux, temp, mid+1, right);
inv_cnt += merge(aux, temp, left, mid, right);
10 return inv_cnt;
11 }

12 } int mergeSort(int aux[], int temp[], int left, int mid, int right)
13 {
14     int i, j, k, inv_cnt = 0;
15     i = left;
16     j = mid+1;
17     k = left;
18
19     while (i <= mid && j <= right)
20     {
21         if (aux[i] < aux[j])
22             temp[k++] = aux[i++];
23         else
24         {
25             temp[k++] = aux[j++];
26             inv_cnt += mid - i + 1;
27         }
28     }
29
30     while (i <= mid)
31         temp[k++] = aux[i++];

```

```

WWK 42 + 2020 02/05
while (j <= right)
temp[k++] = aux[j++];

for (int i = left; i <= right; i++)
aux[i] = temp[i];
}
return inv_cnt;

```

7. Majority Element (more than $\lfloor n/3 \rfloor$) :-

```

vector<int> majority_element(vector<int> a)
{
    vector<int> ans;
    int num1 = -1, num2 = -1, c1 = 0, c2 = 0;
    for (int i = 0; i < n; i++)
    {
        if (a[i] == num1)
            c1++;
    }
}

```

12

SATURDAY OCTOBER

```

else if (a[i] == num2)
    c2++;
else if (c1 == 0)
    num1 = a[i];
    c1 = 1;
}
else if (c2 == 0)
    num2 = a[i];
    c2 = 1;
}

```

```

else
{
    c1--;
    c2--;
}
}
c1 = 0;
c2 = 0;

```

13 SUNDAY

```

for (i = 0; i < n)
{
    if (a[i] == num1) c1++;
    if (a[i] == num2) c2++;
}

```

OCTOBER 2019						
S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

```

if (c1 > n/3)
    ans.pb(num1);
if (c2 > n/3)
    ans.pb(num2);
return ans;
}

```

8. Longest Consecutive Subseq.

सबसे लंग्वेट कॉन्सेक्यूटिव शब्द
भी अक्षरों का सारा शब्द होता है।
प्रवृत्ति वाले अक्षरों का भी नहीं।
नहीं बल्कि जैसे यदि एक शब्द
(diff(P[i] - P[i-1]) > 1)

जो अक्षर आगे आ जाए
तो उसका ans+1 की जाए
res = max(res, ans) की जाए
जोर डफ़ ≤ 1 तो उसके
अन्त में जाओ और जाए तो
Break भाइ नहीं।
res = max(res, ans) की जाए।

end तो res return करो।

⑨ Max-pctt Subarray

```
int maxPctt (vector<int>a)
{
    int mnh=1, mnch=1, msf = INT-MIN,
    int n = a.size();
    for (int i=0, i<n; i++)
    {
        if (a[i] == 0)
            mnh=1, mnch=1;
        msf = max(0, msf);
        continue;
    }
    else
    {
        temp = mnch;
        mnch = max(temp * a[i], mnch * a[i], a[i]);
        mnh = min(temp * a[i], a[i], mnch * a[i]);
        msf = max(mnch, msf);
    }
    return msf;
}
```

⑩ Minimum Swaps and k together

```

int minSwap(int *a, int n, int k)
{
    int window_size=0;
    for(i=0, n)
        if(a[i] <= k) window_size++;
    int i=0, j=size-1, bad=0, ans=100005;
    for(int w=0; w <= j; w++)
    {
        if(a[w] > k) bad++;
        if(bad > ans) ans=bad;
        while(j < n-1)
        {
            if(a[i+1] > k)
                bad--;
            if(a[i+1] > k)
                bad++;
            if(bad < ans)
                ans=bad;
        }
        return ans;
    }
}

```

get window of length k
if And no > k & w <= k
then length of array at index i > 0
then window of bad length k
then if swap with k then window
slide one index & check if bad
and ans is less than ans
else ans is greater than ans

Maximum Subarray Sum with One Deletion

```

int maximumSum(vector<int>&nums) {
    int n=nums.size();
    vector<int>meh1(n,0);
    vector<int>meh2(n,0);
    meh1[0]=nums[0];
    int maxs=nums[0];
    for(int i=1;i<n;i++)
    {
        meh1[i]=max(meh1[i-1]+nums[i],nums[i]);
        if(meh1[i]>maxs)
            maxs=meh1[i];
    }
    meh2[n-1]=nums[n-1];
    for(int i=n-2;i>=0;i--)
    {
        meh2[i]=max(meh2[i+1]+nums[i],nums[i]);
    }
    for(int i=1;i<n-1;i++)
    {
        maxs=max(maxs,meh1[i-1]+meh2[i+1]);
    }
    return maxs;
}

```

ROTATE ARRAY TOWARDS LEFT BY K UNITS

```
void rotateArr(int arr[], int k, int n){  
    k=k%n  
    reverse(arr,arr+k);  
    reverse(arr+k,arr+n);  
    reverse(arr,arr+n);  
}
```

ROTATE ARRAY TOWARDS RIGHT BY K UNITS

```
void rotate(vector<int>& nums, int k) {  
    int n=nums.size();  
    k %= n;  
    reverse(nums.begin(),nums.end());  
    reverse(nums.begin(),nums.begin()+k);  
    reverse(nums.begin()+k,nums.end());  
}
```

BIT MANIPULATION

find the element X, Y in an array that occur once while the rest elements occur twice

```
//Given: An array A with all elements occuring twice except for x and y that occur once.  
//To Do: Find the x and y in O(1) space and O(N) time  
const int N = 1e6;  
int arr[N];  
  
bool hasBitSet(int n, int x) {  
    int tem = n & (1<<x);  
    return tem!=0;  
}  
  
int main() {  
    int n;  
    cin >> n;  
  
    int all = 0;  
    int ans[2] = {0,0};  
  
    for(int i=0; i<n; i++){  
        cin >> arr[i];  
        all ^= arr[i];  
    }  
  
    //assert(all != 0);  
    int k = 0;  
    while( hasBitSet(all, k) == 0 ) k++;  
  
    //partition array into two sets: kth bit on v/s off  
    for (int i=0; i<n; i++) {  
        ans[hasBitSet(arr[i], k)] ^= arr[i];  
    }  
  
    cout << ans[0] << " " << ans[1] << endl;  
    return 0;  
}
```

Repeat And Missing Number Array

```
bool hasSetBits(int n,int x)
{
    int temp=n & (1<<x);
    return (temp!=0);
}

vector<int> Solution::repeatedNumber(const vector<int> &arr) {
    int all=0;
    int n=arr.size();
    vector<int>ans(2,0);
    for(int i=0;i<n;i++)
        all=all^arr[i];
    for(int i=1;i<=n;i++)
        all^=i;
    int k=0;
    while(hasSetBits(all,k)==0)
        k++;
    for(int i=0;i<n;i++)
        ans[hasSetBits(arr[i],k)]^=arr[i];
    for(int i=1;i<=n;i++)
        ans[hasSetBits(i,k)]^=i;
    bool flag=false;
    for(int i=0;i<n;i++)
    {
        if(arr[i]==ans[0])
            flag=true;
    }
    if(flag==true)
        return ans;
    else
    {
        swap(ans[0],ans[1]);
        return ans;
    }
}
```

Every element appears three times except for one

har element 3 baar aa rha h th uski bit 3 baar aa rahi hogi th jo bit 3 se divisible nhi h mtlb vh ussi integer ki h jo ek baar aa rhi h th cnt array m sbki bit ka cnt lelia ab jiski bhi ith bit 3 se divide na ho rhi cnt array m mtlb vh usi number ki bit h.

```
int singleNumber(vector<int>& nums) {
    int cnt[32];
    long long ans=0;
    memset(cnt,0,sizeof cnt);
    for(int i=0;i<nums.size();i++)
    {
        for(int j=0;j<32;j++)
        {
            if(nums[i]&(1<<j))
                cnt[31-j]++;
        }
    }
    for(int i=31;i>=0;i--)
    {
        if(cnt[i]%3)
        {
            ans+=pow(2,31-i);
        }
    }
    return (int)ans;
}
```

Total Hamming Distance

```
int totalHammingDistance(vector<int>& nums) {
    int cnt[32],n=nums.size();
    long long ans=0;
    memset(cnt,0,sizeof cnt);
    for(int i=0;i<nums.size();i++)
    {
        for(int j=0;j<32;j++)
        {
            if(nums[i]&(1<<j))
                cnt[31-j]++;
        }
    }
    for(int i=31;i>=0;i--)
    {
        ans+=(n-cnt[i])*(cnt[i]); //pairs=(jитнo k same nh h)*(jитнo k same h)
    }
    return (int)ans;
}
```

Divide Two Integers

```
int divide(int dividend, int divisor) {
    if(dividend==INT_MIN&&divisor==-1)
        return INT_MAX;
    long dvd = labs(dividend), dvs = labs(divisor), ans = 0;
    int sign = dividend > 0 ^ divisor > 0 ? -1 : 1;
    while(dvd>=dvs)
    {
        long temp=dvs,m=1;
        while(temp<<1<=dvd)
        {
            temp<<=1;
            m<<=1;
        }
        dvd=dvd-temp;
        ans+=m;
    }
    return (sign*ans);
}
```

XOR of all subarray XORs

```
1 An efficient solution is based on the idea to enumerate all subarrays, we can count the frequency of each element that occurred totally in all subarrays, if the frequency of an element is odd then it will be included in the final result otherwise not.
2
3 Number at i-th index will have => (i + 1) * (N - i) frequency.
4
5 There are 4 cases possible:
6 Case 1: i is odd, N is odd
7 Let i = 2k+1, N = 2m+1
8 freq[i] = ((2k+1)+1)*((2m+1)-(2k+1)) = 4(m-k)(k+1) = even
9 Case 2: i is odd, N is even
10 Let i = 2k+1, N = 2m
11 freq[i] = ((2k+1)+1)*((2m)-(2k+1)) = 2(k+1)(2m-2k-1) = even
12 Case 3: i is even, N is odd
13 Let i = 2k, N = 2m+1
14 freq[i] = ((2k)+1)*((2m+1)-(2k)) = 2k(2m-2k+1)+(2m-2k)+1 = odd
15 Case 4: i is even, N is even
16 Let i = 2k, N = 2m
17 freq[i] = ((2k)+1)*((2m)-(2k)) = 2(m-k)(2k+1) = even
18
19 From this, we can conclude that if total no.of elements in the array is even, then frequency of element at any position is even. So total XOR will be 0. And if total no. of elements are odd, then frequency of elements at even positions are odd and frequency of elements at odd positions are even. So we need to find only the XOR of elements at even positions.
20
21 int Solution::solve(vector<int> &A) {
22     int n=A.size(),ans=0;
23     if(n%2==0)
24         return 0;
25     for(int i=0;i<n;i+=2)
26         ans^=A[i];
27     return ans;
28 }
29 }
```

Palindromic Binary Representation

```
int convertBinaryToInt(string s) {
    int n = s.length();
    int x = 0;
    for(int i=0; i<n; i++)
        x = (x*2) + (s[i]-'0');
    return x;
}
int Solution::solve(int A) {
    if(A == 1) return 1;
    A--;
    queue<string> q;
    q.push("11");
    while(!q.empty()) {
        string curr = q.front();
        q.pop();
        A--;
        if(A == 0) return convertBinaryToInt(curr);
        int n = curr.length();
        // if length is even, we have two choices
        if(n%2 == 0) {
            string s0 = curr, s1 = curr;
            s0.insert(n/2, "0");
            s1.insert(n/2, "1");
            q.push(s0);
            q.push(s1);
        }
        // if length is odd, we have one choice
        else {
            string temp(1, curr[n/2]);
            curr.insert(n/2, temp);
            q.push(curr);
        }
    }
}
```

STRINGS

Longest Palindromic Substring:

- The idea is to generate all even length and odd length palindromes and keep track of the longest palindrome seen so far.

```
void lp()
{
    ll maxlen=1;
    ll start=0;
    ll low,high;
    for(ll i=1;i<s.size();i++)
    {
        //for even length
        low=i-1;
        high=i;
        while(low>=0 && high<s.size() && s[low]==s[high])
        {
            if(high-low+1>maxlen)
            {
                start=low;
                maxlen=high-low+1;
            }
            low--;
            high++;
        }
        //for odd length
        low=i-1;
```

```
high=i+1;

while(low>=0 && high<s.size() && s[low]==s[high])

{

    if(high-low+1>maxlen)

    {

        start=low;

        maxlen=high-low+1;

    }

    low--;

    high++;

}

print(start,start+maxlen-1);

}
```

Print all the permutations of the given string.

```
void helper(string &s,int curr_index,vector<string>&ans)
{
    if(curr_index==s.size())
        ans.push_back(s);
    for(int i=curr_index;i<s.size();i++)
    {
        swap(s[i],s[curr_index]);
        helper(s,curr_index+1,ans);
        swap(s[i],s[curr_index]);
    }
}
vector<string>find_permutation(string s)
{
    int n=s.size();
    vector<string>ans;
    int idx=0;
    helper(s,0,ans);
    return ans;
}
```

RABIN KARP ALGO:

Rabin-Karp for pattern searching

(ii) search a pattern 's' in text 'T'

vector<int> Rabin_Karp(string s, string t)

```
{  
    int p=31, m: 1000000007, l=s.size(), T=t.size();  
    vector<int> p_pow(m+1);  
    p_pow[0]=1;  
    for (int i=1; i<p_pow.size(); i++)  
        p_pow[i] = (p*p_pow[i-1])%m;  
    vector<ll> h(l+1, 0);  
    h[0]=0;  
    for (int i=0; i<l; i++)  
        h[i+1] = (h[i] + (s[i]-97)*p_pow[i])%m;  
    if (h[l]==0)  
        h[l] = (h[l]+m)%m;  
}
```

```
if (h[s]==0);  
for (int i=0; i<l; i++)  
    h[s] = (h[s] + (s[i]-97)*p_pow[i])%m;
```

vector<int> occurrences;

```
for (int i=0; i+l-1<T; i++)
```

```
{  
    ll curr_h = (h[i+l] - h[i] + m)%m;
```

```
    if (curr_h == h[s]*p_pow[i])  
        occurrences.pb(i);
```

g.

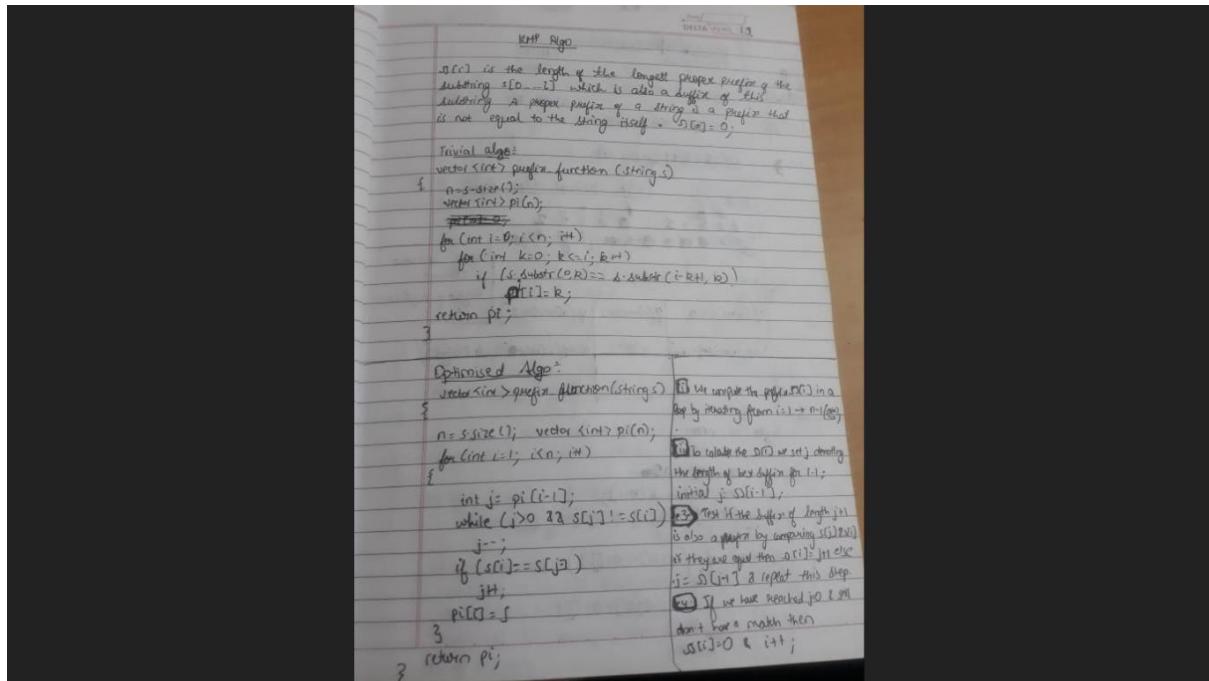
3.

$h[0]=0$, $h[1]=s[0]*p_pow[0]$, $h[2]=h[1]+s[1]*p_pow[1]$ where $p_pow[0]=1$ and $p_pow[1]=p$ and $p_pow[2]=p^2$.

KMP ALGO:

j=pi[j-1] while k andar image mai correction

h yh



Roman to Integer:::

The screenshot shows the LeetCode problem 13. Roman to Integer page. The code editor contains the following C++ solution:

```
class Solution {
public:
    int romanToInt(string s) {
        int n=s.size();
        unordered_map<char, int> m = { { 'I', 1 },
                                         { 'V', 5 },
                                         { 'X', 10 },
                                         { 'L', 50 },
                                         { 'C', 100 },
                                         { 'D', 500 },
                                         { 'M', 1000 } };

        int ans=m[s[n-1]];
        for(int i=n-2;i>=0;i--) {
            if(m[s[i]]>=m[s[i+1]]) {
                ans+=m[s[i]];
            }
            else {
                ans-=m[s[i]];
            }
        }
        return ans;
    }
};
```

The page includes a table of Roman numeral symbols and their values, and a detailed explanation of the Roman numeral system.

Number of flips to make binary string alternate::

count the number of replacements to convert the string in type 1 and store it in count then minimum replacement will be $\min(\text{count}, \text{len} - \text{count})$ where len is the length of the string. $\text{len} - \text{count}$ is the number of replacements to convert the string in type 2.

Minimum Swaps for Bracket Balancing:

You are given a string S of $2N$ characters consisting of N '[' brackets and N ']' brackets. A string is considered balanced if it can be represented in the form $S2[S1]$ where S1 and S2 are balanced strings. We can make an unbalanced string balanced by swapping adjacent characters. Calculate

the minimum number of swaps necessary to make a string balanced.
Note - Strings S1 and S2 can be empty.

Input : []][[]

Output : 2

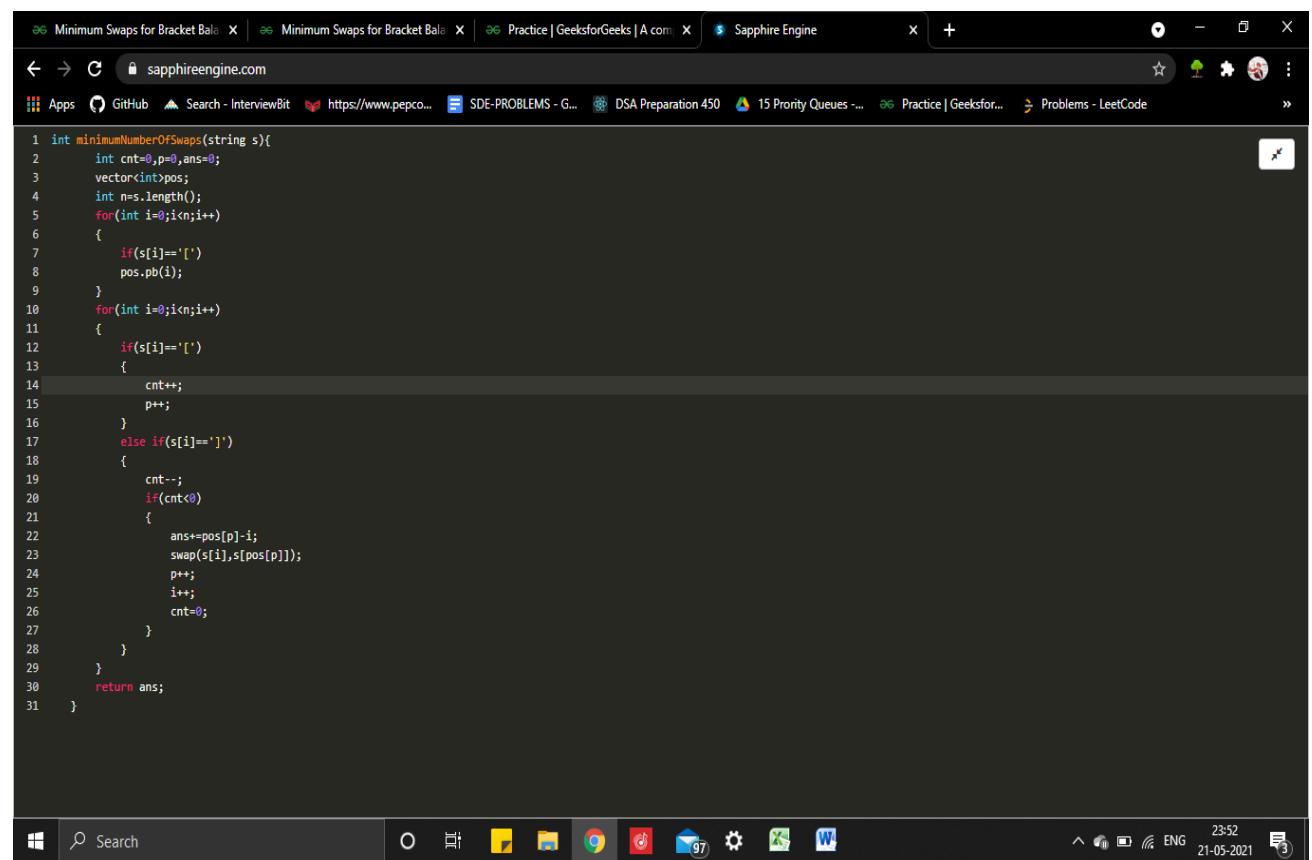
Explanation :

First swap: Position 3 and 4

[]][[]

Second swap: Position 5 and 6

[]][[]]



The screenshot shows a browser window with multiple tabs open. The active tab displays a C++ code snippet for calculating the minimum number of swaps required to balance a string of brackets. The code uses a vector to store positions and iterates through the string to find pairs of brackets that need to be swapped. The browser interface includes a search bar, a navigation bar with back, forward, and refresh buttons, and a toolbar with various icons. The status bar at the bottom shows system information like battery level, signal strength, and the date/time.

```
1 int minimumNumberOfSwaps(string s){  
2     int cnt=0,p=0,ans=0;  
3     vector<int>pos;  
4     int n=s.length();  
5     for(int i=0;i<n;i++)  
6     {  
7         if(s[i]=='[')  
8             pos.pb(i);  
9     }  
10    for(int i=0;i<n;i++)  
11    {  
12        if(s[i]==']')  
13        {  
14            cnt++;  
15            p++;  
16        }  
17        else if(s[i]==')')  
18        {  
19            cnt--;  
20            if(cnt<0)  
21            {  
22                ans+=pos[p]-i;  
23                swap(s[i],s[pos[p]]);  
24                p++;  
25                i++;  
26                cnt=0;  
27            }  
28        }  
29    }  
30    return ans;  
31 }
```

Smallest distinct window:

Given a string 's'. The task is to find the **smallest** window length that contains all the characters of the given string at least one time.

```
1 // } Driver Code Ends
2
3 class Solution{
4 public:
5     string findSubString(string s)
6     {
7         bool visited[MAX_CHARS]={false};
8         int dist_cnt=0,n=s.size(),cnt=0;
9         for(int i=0;i<n;i++)
10        {
11            if(visited[s[i]]==false)
12            {
13                visited[s[i]]=true;
14                dist_cnt++;
15            }
16        }
17        int curr_cnt[MAX_CHARS]={0};
18        int min_length=INT_MAX,j=0;
19        for(int i=0;i<n;i++)
20        {
21            curr_cnt[s[i]]++;
22            if(curr_cnt[s[i]]==1)
23            {
24                cnt++;
25            }
26            if(cnt==dist_cnt)
27            {
28                while(curr_cnt[s[j]]>1)
29                {
30                    curr_cnt[s[i]]--;
31                    j++;
32                }
33                min_length=min(min_length,i-j+1);
34            }
35        }
36    }
37 };
38
39
40
41
42 };
43
44 // } Driver Code Ends
```

Given a string **s**, check if the letters can be rearranged so that two characters that are adjacent to each other are not the same.

Example 1:

Input: s = "aab"

Output: "aba"

The screenshot shows a browser-based code editor interface for a C++ solution to LeetCode problem 767. The code uses a priority queue and a map to rearrange characters in a string. It includes examples and notes.

```

string reorganizeString(string s) {
    priority_queue<pair<int,char>>pq;
    unordered_map<char,int>m;
    int n=s.size();
    for(int i=0;i<n;i++)
    {
        m[s[i]]++;
    }
    for(auto it=m.begin();it!=m.end();it++)
    {
        pq.push(make_pair(it->second,it->first));
    }
    string res="";
    while(pq.size()>1)
    {
        pair<int,char>top1=pq.top();pq.pop();
        pair<int,char>top2=pq.top();pq.pop();
        res+=top1.second;
        res+=top2.second;
        top1.first--;top2.first--;
        if(top1.first>0)
            pq.push(top1);
        if(top2.first>0)
            pq.push(top2);
    }
    if(!pq.empty())//in case all char have same freq then pq will be empty before reaching here
    {
        pair<int,char>top=pq.top();
        pq.pop();
        if(top.first>1)
            return "";
        else
            res+=top.second;
    }
}

```

Example 1:

Input: s = "aab"
Output: "aba"

Example 2:

Input: s = "aaab"
Output: ""

Note:

- s will consist of lowercase letters and have length in range [1, 500].

Your previous code was restored from your local storage. [Reset to default](#)

Print Anagrams Together:

N = 5

words[] = {act, god, cat, dog, tac}

Output:

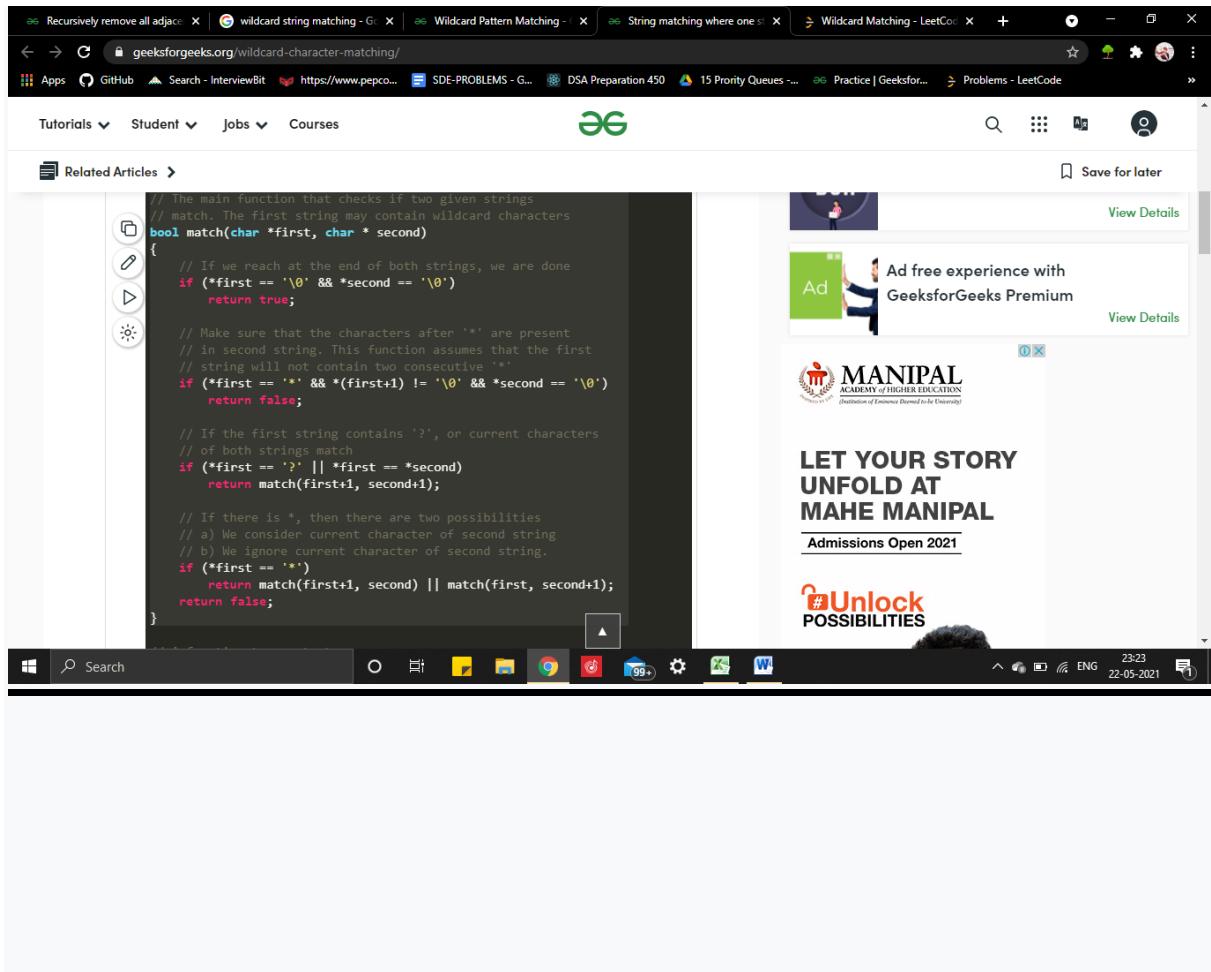
god dog

act cat tac

```
9 10 vector<vector<string>> Anagrams(vector<string>&s)
11 {
12     map<char,int>, vector<string>> my_map;
13     for(string str:s)
14     {
15         map<char,int>temp_map;
16         vector<string> temp_str;
17         for(int j=0;j<str.length();j++)
18             temp_map[str[j]]++;
19         auto it=my_map.find(temp_map);
20         if(it!=my_map.end())
21             it->second.push_back(str);
22         else
23         {
24             my_map[temp_map].push_back(str);
25         }
26     }
27     vector<vector<string>> ans;
28
29     for(auto it=my_map.begin();it!=my_map.end();it++)
30         ans.push_back(it->second);
31     }
32     return ans;
33 }
```

Wildcard Matching:

Isme ? K liye th dono string+1
krke recursion chlalado aur * k
liye
function(first+1,second) || (first,s
econd+1)



Recursively remove all adjacent duplicates:

Given a string, recursively remove adjacent duplicate characters from the string. The output string should not have any adjacent duplicates.

Input: azxxzy

Output: ay

```

1* void helper(string &s, int i, int n) {
2    if(i == n || n <= 1) return;
3
4    if(i+1 < n && s[i] == s[i+1]) {
5        if(i+2 < n && s[i] == s[i+2]) {
6            S.erase(i, 2);
7            return helper(s, i, n-1);
8        }
9        else {
10            S.erase(i, 2);
11            if(i > 0) i--;
12            return helper(s, i, n-2);
13        }
14    }
15    else {
16        return helper(s, i+1, n);
17    }
18}
19 class Solution {
20 public:
21     string removeDuplicates(string s) {
22         helper(s, 0, s.size());
23         return s;
24     }
25 };

```

Console ▾ Contribute i Run Code ▾ Submit

Minimum characters to be added at front to make string palindrome:

First we concat string by concatenating given string, a special character and reverse of given string then we will get lps array for this concatenated string, recall that each index of lps array represent longest proper prefix which is also suffix. We can use this lps array for solving the problem. Here we are only interested in the last value of this lps array because it shows us the largest suffix of the reversed string that matches the prefix of the original string i.e these many characters already satisfy the palindrome property. Finally minimum number of character needed to make the string a palindrome is length of the input string minus last entry of our lps array.

```

int getMinCharToAddedToMakeStringPalin(string str)
{
    string revStr = str;
    reverse(revStr.begin(), revStr.end());
    string concat = str + "$" + revStr;
    vector<int> lps = computeLPSArray(concat);
    return (str.length() - lps.back());
}

```

Minimum Window Substring:

Given two strings s and t of lengths m and n respectively, return *the minimum window in s which will contain all the characters in t* . If there is no such window in s that covers all characters in t , return *the empty string " "*.

Note that If there is such a window, it is guaranteed that there will always be only one unique minimum window in s .

The screenshot shows a LeetCode submission page for the 'Minimum Window Substring' problem. The code is written in C++ and uses a sliding window approach with character frequency arrays. It includes a helper function to check if all characters in t are present in the current window. The main function iterates over the string s , expanding the window by moving the end pointer i and shrinking it by moving the start pointer j until all characters of t are found. The window length is updated whenever a valid window is found. The code is annotated with line numbers from 1 to 45.

```
1 class Solution {
2 public:
3     int ms[128]={0};
4     int mt[128]={0};
5     bool getThemAll()
6     {
7         for(int i=0;i<128;i++)
8             if(mt[i]>ms[i])
9                 return false;
10        return true;
11    }
12    string minWindow(string s, string t) {
13        int S=s.size();
14        int m=s.size();
15        int i=0;
16        int j=0;
17        int char[128]={0};
18        for(;j<S;j++)
19            mt[t[j]]++;
20        int i=j;
21        int minlength=INT_MAX,start=-1;
22        while(i<s.size())
23        {
24            while(!getThemAll()&&i<s.size())
25            {
26                i++;
27                ms[s[i]]++;
28            }
29            if(getThemAll())
30            {
31                if(i-j<minlength)
32                {
33                    minlength=min(minlength,i-j+1);
34                    start=j;
35                }
36                ms[s[i]]--;
37                i++;
38            }
39        }
40        if(minlength==INT_MAX)
41            return "";
42        return (s.substr(start,minlength));
43    }
44};
```

Longest Substring Without Repeating Characters

```

int lengthOfLongestSubstring(string s) {
    int store[256]={0}; //array to store the occurrences of all the characters
    int l=0; //left pointer
    int r=0; //right pointer
    int ans=0; //initializing the required length as 0
    while(r<s.length()) //iterate over the string till the right pointer reaches the end of the string
    {
        store[s[r]]++; //increment the count of the character present in the right pointer
        while(store[s[r]]>1) //if the occurrence become more than 1 means the char is repeated
        {
            store[s[l]]--; //reduce the occurrence of temp as it might be present ahead also in the string
            l++; //contraction of the present window till the occurrence of the 't' char becomes 1
        }
        ans = max(ans,r-l+1); //As the index starts from 0 , ans will be (right pointer-left pointer + 1)
        r++; // now will increment the right pointer
    }
    return ans;
}

```

Transform One String to Another using Minimum Number of Given Operation:

Given two strings A and B, the task is to convert A to B if possible. The only operation allowed is to put any character from A and insert it at front. Find if it's possible to convert the string. If yes, then output minimum no. of operations required for transformation.

The screenshot shows a browser window with the URL [geeksforgeeks.org/transform-one-string-to-another-using-minimum-number-of-given-operation/](https://www.geeksforgeeks.org/transform-one-string-to-another-using-minimum-number-of-given-operation/). The page displays a C++ code snippet for calculating the minimum number of operations required to transform string A into string B. The code uses a sliding window approach with character counts to determine the minimum operations needed. To the right of the code, there is a sidebar with related articles and a small image of a person working on a laptop.

```

int mindps(string A, string B)
{
    int m = A.length(), n = B.length();

    // This part checks whether conversion is
    // possible or not
    if (n > m)
        return -1;

    int count[256];
    memset(count, 0, sizeof(count));
    for (int i=0; i<m; i++) // count characters in A
        count[A[i]]++;

    for (int i=0; i<n; i++) // subtract count for
                           // every character in B
        count[B[i]]++;

    for (int i=0; i<256; i++) // check if all counts become 0
        if (count[i])
            return -1;

    // This part calculates the number of operations required
    int res = 0;
    for (int i=n-1, j=m-1; i>=0; )
    {
        // If there is a mismatch, then keep incrementing
        // result 'res' until B[j] is not found in A[0..i]
        while ((i>=0 && A[i] != B[j]))
        {
            i--;
            res++;
        }

        // If A[i] and B[j] match
        if (i >= 0)
        {
            i--;
            j--;
        }
    }
    return res;
}

// Driver program
int main()
{
    string A = "EACBD";
    string B = "EABCD";
    cout << "Minimum number of operations required is " << mindps(A, B);
    return 0;
}

```

```
string idToShortURL(long int n)
{
    char map[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
                  "GHIJKLMNOPQRSTUVWXYZ0123456789";
    string shorturl;
    while (n)
    {
        shorturl.push_back(map[n%62]);
        n = n/62;
    }
    reverse(shorturl.begin(), shorturl.end());
    return shorturl;
}
long int shortURLtoID(string shortURL)
{
    long int id = 0;
    for (int i=0; i < shortURL.length(); i++)
    {
        if ('a' <= shortURL[i] && shortURL[i] <= 'z')
            id = id*62 + shortURL[i] - 'a';
        if ('A' <= shortURL[i] && shortURL[i] <= 'Z')
            id = id*62 + shortURL[i] - 'A' + 26;
        if ('0' <= shortURL[i] && shortURL[i] <= '9')
            id = id*62 + shortURL[i] - '0' + 52;
    }
    return id;
}
```

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
int strStr(string t, string s) {
    int m=t.size(),n=s.size();
    if(n==0)
        return 0;
    string ss=s+"#"+t;
    vector<int>lps=prefix_function(ss);
    for(int i=n+1;i<ss.length();i++)
    {
        if(lps[i]==n)
        {
            return (i-2*n);
        }
    }
    return -1;
}
```

Minimum Insertion Steps to Make a String Palindrome

Palindrome

```
int findlongestpalindromicsubsequence(string s,string ss)
{
    int n=s.length();
    int m=ss.length();
    int dp[n+1][m+1];
    memset(dp,0,sizeof(dp));
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            if(s[i-1]==ss[j-1])
                dp[i][j]=dp[i-1][j-1];
            else
                dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
        }
    }
    return dp[n][m];
}
int minInsertions(string s) {
    string ss=s;
    reverse(ss.begin(),ss.end());
    int lps=findlongestpalindromicsubsequence(s,ss);
    return (s.length()-lps);
}
```

Valid Palindrome after deleting at most One Character

```
bool validPalindrome(string &s, int lo, int hi, int count) {
    if (count > 1) return false;
    while (lo < hi) {
        if (s[lo] == s[hi]) {
            lo++; hi--;
        }
        else {
            return validPalindrome(s, lo + 1, hi, count + 1) ||
                   validPalindrome(s, lo, hi - 1, count + 1);
        }
    }
    return true;
}
bool validPalindrome(string s) {
    int lo = 0, hi = s.size() - 1;
    return validPalindrome(s, lo, hi, 0);
}
```

Multiply Strings

```
string multiply(string num1, string num2) {
    string sum(num1.size() + num2.size(), '0');

    for (int i = num1.size() - 1; 0 <= i; --i) {
        int carry = 0;
        for (int j = num2.size() - 1; 0 <= j; --j) {
            int tmp = (sum[i + j + 1] - '0') + (num1[i] - '0') * (num2[j] - '0') + carry;
            sum[i + j + 1] = tmp % 10 + '0';
            carry = tmp / 10;
        }
        sum[i] += carry;
    }

    size_t startpos = sum.find_first_not_of("0");
    if (string::npos != startpos) {
        return sum.substr(startpos);
    }
    return "0";
}
```

Vowel and consonant Substring

```
int Solution::solve(string s) {
    long long vtn=0,ctn=0,ans=0,mod=10000000007;
    //where vtn=>vowels_till_now && ctn=>consonants_till_now
    for(int i=0;i<s.size();i++)
    {
        if(s[i]=='a'||s[i]=='e'||s[i]=='i'||s[i]=='o'||s[i]=='u')
        {
            vtn++;
            ans=(ans+ctn)%mod;
        }
        else
        {
            ctn++;
            ans=(ans+vtn)%mod;
        }
    }
    return (int)ans;
}
```

MATRIX

Spiral traversal on a Matrix:

minr,mic,maxr,maxc lelo aur har for loop k baad minr++ fr next for m
maxc-- fr next m maxr-- fr next m minc++ kro

Search an element in a matrix:

Simply bsearch lgao jhaan mid_value=matrix[mid/m][mid%m] m is column;

Find row with maximum no. of 1's:(TC: O(m+n))

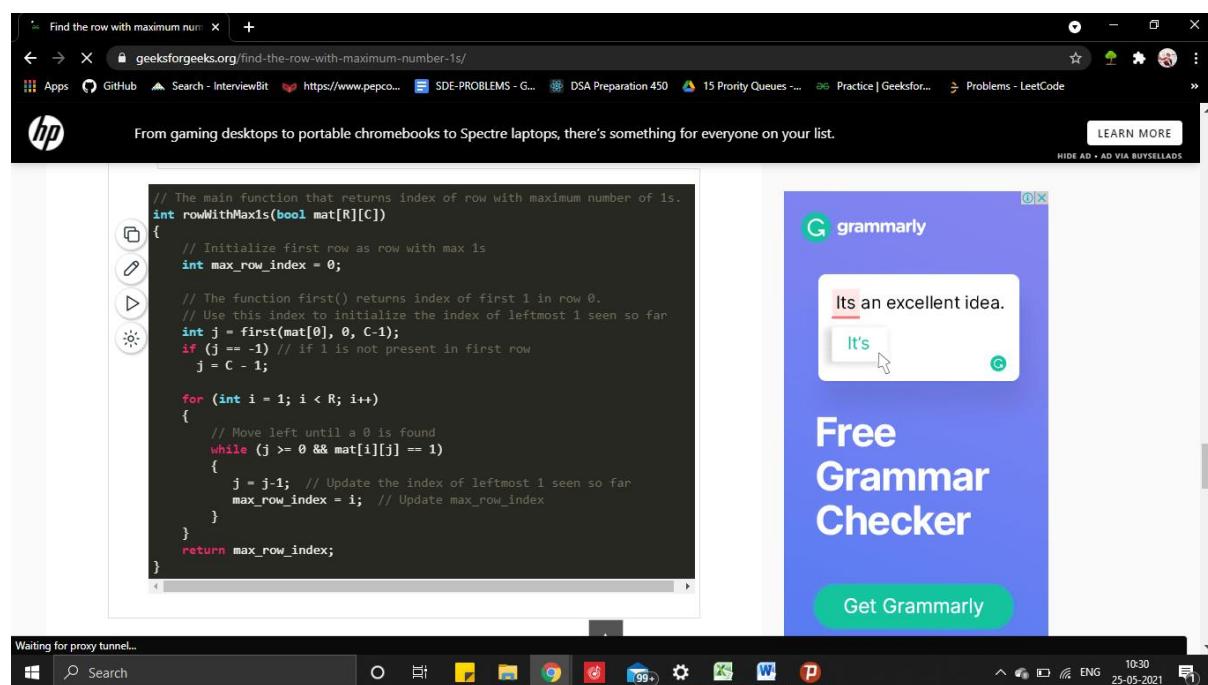
Step1: Get the index of first (or leftmost) 1 in the first row.

Step2: Do following for every row after the first row

...IF the element on left of previous leftmost 1 is 0, ignore this row.

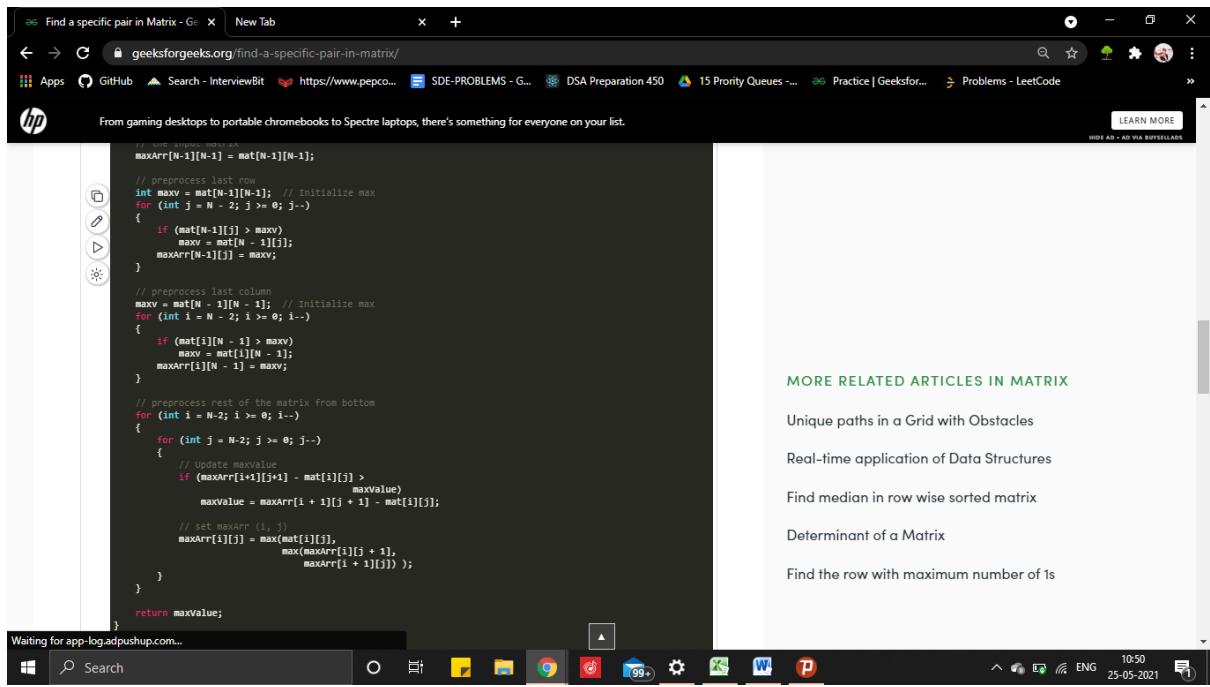
...ELSE Move left until a 0 is found. Update the leftmost index to this index and max_row_index to be the current row.

The time complexity is $O(m+n)$ because we can possibly go as far left as we came ahead in the first step.



Find a specific pair in matrix:

An **efficient solution** uses extra space. We pre-process the matrix such that index(i, j) stores max of elements in matrix from (i, j) to (N-1, N-1) and in the process keeps on updating maximum value found so far. We finally return the maximum value.



The screenshot shows a browser window with the URL geeksforgeeks.org/find-a-specific-pair-in-matrix/. The page content displays a C++ code snippet for solving the problem. The code initializes a matrix `mat` and a max array `maxArr`. It preprocesses the last row and column to find the maximum value in those sections. Then, it iterates through the rest of the matrix from bottom-left to top-right, updating the `maxValue` variable whenever a larger value is found. Finally, it returns the `maxValue`.

```
// C++ program to find a specific pair in Matrix
int maxArr[N-1][N-1] = mat[N-1][N-1];

// preprocess last row
int maxv = mat[N-1][N-1]; // Initialize max
for (int j = N - 2; j >= 0; j--)
{
    if (mat[N-1][j] > maxv)
        maxv = mat[N-1][j];
    maxArr[N-1][j] = maxv;
}

// preprocess last column
maxv = mat[N-1][N-1]; // Initialize max
for (int i = N - 2; i >= 0; i--)
{
    if (mat[i][N-1] > maxv)
        maxv = mat[i][N-1];
    maxArr[i][N-1] = maxv;
}

// preprocess rest of the matrix from bottom
for (int i = N-2; i >= 0; i--)
{
    for (int j = N-2; j >= 0; j--)
    {
        // Update maxValue
        if ((maxArr[i+1][j+1] - mat[i][j]) > maxValue)
            maxValue = maxArr[i+1][j+1] - mat[i][j];

        // set maxArr[i][j]
        maxArr[i][j] = max(maxArr[i][j],
                           max(maxArr[i][j+1],
                               maxArr[i+1][j]));
    }
}
return maxValue;
}
```

Maths:

Reach a Number

```
1 int reachNumber(int target) {
2     if(target<0){
3         target=-target;
4     }
5     int step=0;
6     int sum=0;
7     while(true)
8     {
9         step++;
10        sum=sum+step;
11        if(sum==target)
12        {
13            return step;
14        }
15        else if(sum>target && (sum-target)%2==0)
16        {
17            return step;
18        }
19    }
20 }
```

SIEVE OF ERATOSTHENES

```
for( int i= 2; i < sqrt(N); i++)
{
    if (prime[i] == true)
    {
        for( int j = i*i; j <= N; j = j+i)
            prime[j] = false;
    }
}
```

Numbers of length N and value less than K

```
int Solution::solve(vector<int> &A, int B, int C) {
    int n = A.size();
    if(n == 0 || C == 0) return 0;
    vector<int> temp;
    int x = C;
    while(x) {
        temp.push_back(x%10);
        x /= 10;
    }
    int m = temp.size();
    if(B > m) {
        return 0;
    }
    else if(B < m) {
        if(B == 1) return n;
        if(A[0] == 0) return (n-1)*pow(n, B-1);
        else return pow(n, B);
    }
    else {
        if(m == 1) return (lower_bound(A.begin(), A.end(), C) - A.begin());
        int ans = 0;
        for(int i=m-1; i>=0; i--) {
            int idx=(lower_bound(A.begin(), A.end(), temp[i]) - A.begin());
            if(i==m-1&&A[0]==0)
            {
                idx--;
            }
            ans+=idx*pow(n,i);
            if(!binary_search(A.begin(),A.end(),temp[i]))
                return ans;
        }
        return ans;
    }
}
```

Excel Column Title

Example 1:

```
Input: columnNumber = 1
Output: "A"
```

Example 2:

```
Input: columnNumber = 28
Output: "AB"
```

Example 3:

```
Input: columnNumber = 701
Output: "ZY"
```

Example 4:

```
Input: columnNumber = 2147483647
Output: "FXSHRXW"
```

```
1 class Solution {
2     public:
3         string convertToTitle(int n) {
4             string res;
5             char tmp;
6             while(n){
7                 n -= 1;
8                 tmp = 'A' + n % 26;
9                 res = tmp + res;
10                n /= 26;
11            }
12            return res;
13        }
14    };
```

Next Smallest Palindrome!

```
bool allNine(string s)
{
    for(int i=0; i<s.size(); i++)
    {
        if(s[i]!='9')
            return false;
    }
    return true;
}
string Solution::solve(string s) {
    string str="";
    if(allNine(s))
    {
        str.push_back('1');
        for(int i=0; i<s.size()-1; i++)
            str.push_back('0');
        str.push_back('1');
        return str;
    }
    else
    {
        int n=s.size();
        int mid=n/2;
        int i=mid-1;
        int j=(n%2==0)?mid:mid+1;
        bool leftSmaller=false;
        while(i>=0 && s[i]==s[j])
        {
            i--;j++;
        }
        if(i<0 || s[i]-'0'<s[j]-'0')
            leftSmaller=true;
        while(i>=0)
        {
            s[j]=s[i];
            i--;
            j++;
        }
        if(leftSmaller==true)
        {
            i=mid-1;
            int carry=1;
            if(n%2==1)
            {
                int x=(s[mid]-'0')+carry;
                carry=x/10;
                x=x%10;
                s[mid]=x+'0';
                j=mid+1;
            }
            else
            {
                j=mid;
                while(i>=0)
                {
                    int x=(s[i]-'0')+carry;
                    carry=x/10;
                    x=x%10;
                    s[i]=x+'0';
                    s[j++]=s[i--];
                }
            }
        }
        return str;
    }
}
```

GCD

```
int Solution::gcd(int A, int B) {
    if(B>A)
        return gcd(B,A);
    if(A==0)
        return B;
    else if(B==0)
        return A;
    while(A%B!=0)
    {
        int rem=A%B;
        A=B;
        B=rem;
    }
    return B;
}
```

Sorted Permutation Rank

```
int mod = 1000003;
int fact(int n)
{
    if(n==0)
        return 1 ;
    else
        return (n*fact(n-1)) %mod;
}
int Solution::findRank(string s) {
    string ss=s;
    sort(ss.begin(),ss.end());
    int n=s.size();
    long long ans=0;
    while(ss.size()>0)
    {
        if(s[0]==ss[0])
        {
            ss=ss.substr(1);
            s=s.substr(1);
        }
        else
        {
            int idx=ss.find(s[0]);
            ans=(ans+(idx*fact(ss.size()-1))%mod)%mod;
            ss.erase(idx,1);
            s=s.substr(1);
        }
    }
    return ans+1;
}
```

Sorted Permutation Rank with Repetits

```
long long modInverse(long long x, long long n, long long mod) {
    long long ans = 1;
    while(n) {
        if(n&1) ans = (ans*x) % mod;

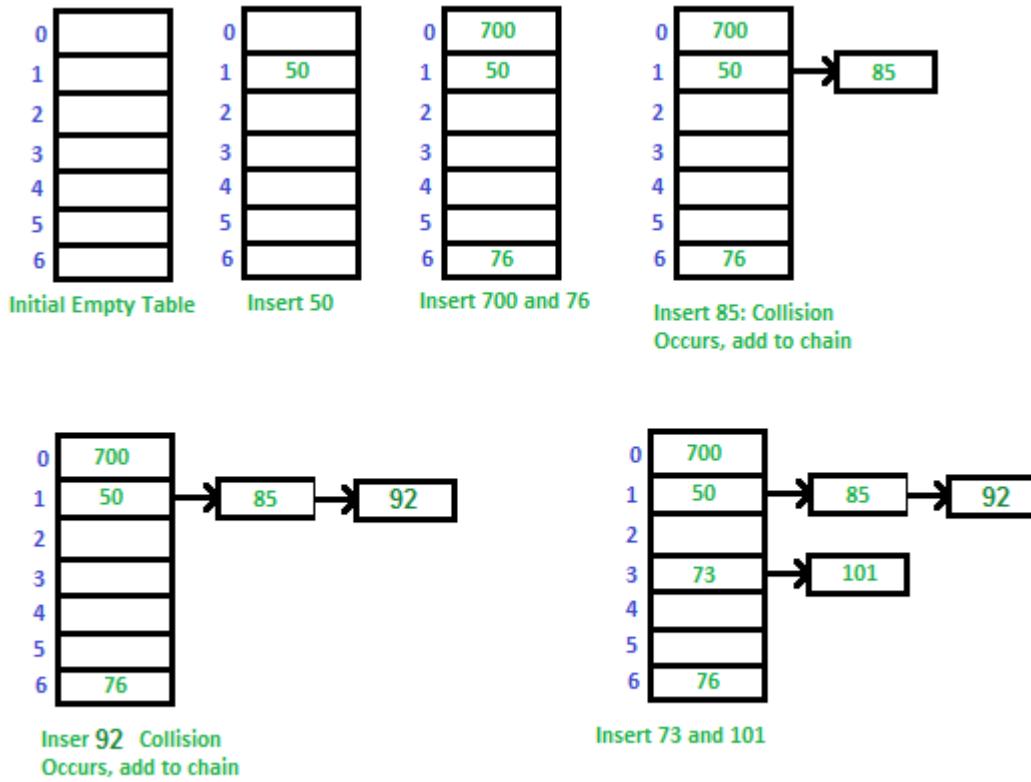
        x = (x*x) % mod;
        n >>= 1;
    }
    return ans;
}
int Solution::findRank(string A) {
    long long ans = 1, mod = 10000003;
    for(int i=0; i<A.length(); i++) {
        long long cnt = 0; // elements less than A[i]
        for(int j=i+1; j<A.length(); j++) {
            if(A[i] > A[j]) cnt++;
        }
        long long freq[256] = {0};
        for(int j=i; j<A.length(); j++) {
            freq[A[j]]++;
        }
        // removing permutations of repetitive characters
        long long repeat = 1;
        for(int j=0; j<256; j++) {
            repeat = (repeat * fact(freq[j], mod)) % mod;
        }
        repeat = modInverse(repeat, mod-2, mod); // taking modular inverse using Fermat's Little theorem
        long long temp = (fact(A.length()-i-1, mod) * cnt) % mod; // add permutations of total chars smaller than A[i]
        temp = (temp * repeat) % mod; // remove repeated permutations
        ans = (ans + temp) % mod;
    }
    return (int)ans;
}
```

HASHING:

Collision Handling Techniques

Separate Chaining

The idea is to make each cell of the hash table point to a linked list of records that have the same hash function value. Here we maintain an **Array of Linked Lists.**



Advantages:

- 1) Simple to implement.
- 2) Hash table never fills up, we can always add more elements to the chain.
- 3) Less sensitive to the hash function or load factors.
- 4) It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.

Disadvantages:

- 1) Cache performance of chaining is not good as keys are stored using a linked list. Open addressing provides better cache performance as everything is stored in the same table.
- 2) Wastage of Space (Some Parts of hash table are never used)
- 3) If the chain becomes long, then search time can become $O(n)$ in the worst case.
- 4) Uses extra space for links.

Data Structures For Storing Chains:

- Linked lists

- Search: $O(l)$ where l = length of linked list
 - Delete: $O(l)$
 - Insert: $O(l)$
 - Not cache friendly
- Dynamic Sized Arrays (Vectors in C++)
 - Search: $O(l)$ where l = length of array
 - Delete: $O(l)$
 - Insert: $O(l)$
 - Cache friendly
- Self Balancing BST (AVL Trees, Red Black Trees)
 - Search: $O(\log(l))$
 - Delete: $O(\log(l))$
 - Insert: $O(l)$
 - Not cache friendly
 - Java 8 onwards use this for HashMap

Chaining

Performance

m = No. of slots in Hash Table

n = No. of keys to be inserted.

Load Factor $\alpha = n/m$

Expected chain length = α

Expected Time to Search = $O(1+\alpha)$

Expected Time to Insert/Delete = $O(1+\alpha)$

GeeksforGeeks
A computer science portal for geeks

samarth.gupta@geeksforgeeks.org

4:27 -148 2x

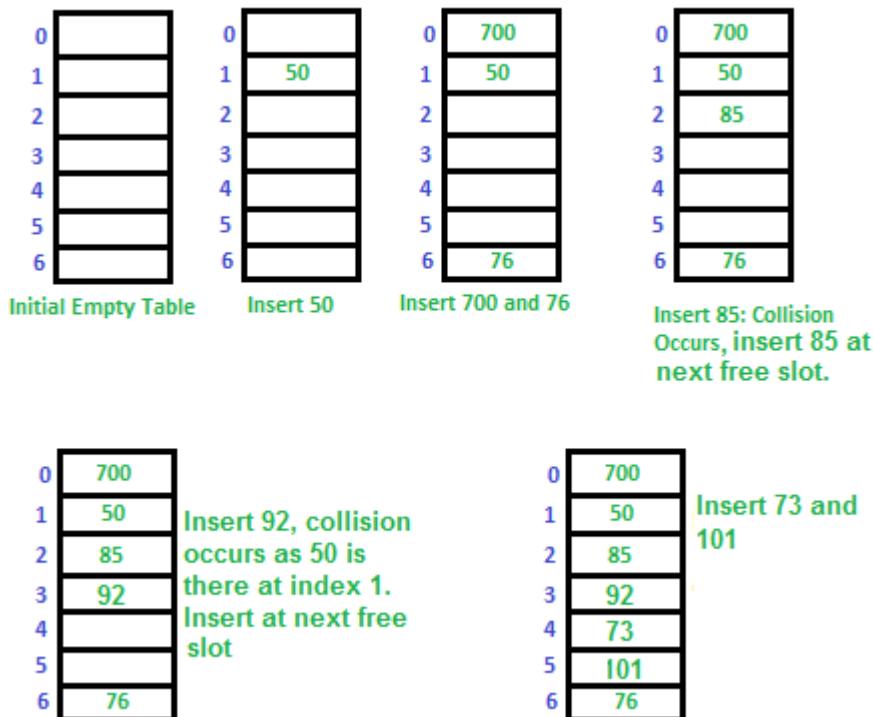
Open Addressing

In Open Addressing, all elements are stored in the hash table itself. So at any point, the size of the table must be greater than or equal to the total number of

=>No. of slots in Hash Table >= No. of keys to be inserted.

=>Cache Friendly.

keys (Note that we can increase table size by copying old data if needed).



Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k .

Search(k): Keep probing until the slot's key doesn't become equal to k or an empty slot is reached.

Delete(k): **Delete operation is interesting.** If we simply delete a key, then the search may fail. So **slots** of deleted keys are marked specially as "**deleted**".

The insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.

Challenges in Linear Probing :

1. **Primary Clustering(In Linear Probing):** One of the problems with linear probing is Primary clustering, many consecutive elements form groups and it starts taking time to find a free slot or to search for an element.
2. **Secondary Clustering(In Quadratic Probing):** Secondary clustering is less severe, two records only have the same collision chain (Probe Sequence) if their initial position is the same.

b) Quadratic Probing We look for i^2 'th slot in i 'th iteration.

```
let hash(x) be the slot index computed using hash function.  
If slot hash(x) % S is full, then we try (hash(x) + 1*1) % S  
If (hash(x) + 1*1) % S is also full, then we try (hash(x) + 2*2) % S  
If (hash(x) + 2*2) % S is also full, then we try (hash(x) + 3*3) % S  
.....  
.....
```

The problem here is that as it is checking quadratically there might be conditions where it doesn't find empty slots even if they are there.

If load factor<0.5 && m is prime then only it guarantees to find empty slots.

c) Double Hashing We use another hash function $\text{hash2}(x)$ and look for $i*\text{hash2}(x)$ slot in i 'th rotation.

```
let hash(x) be the slot index computed using hash function.  
If slot hash(x) % S is full, then we try (hash(x) + 1*\text{hash2}(x)) % S  
If (hash(x) + 1*\text{hash2}(x)) % S is also full, then we try (hash(x) +  
2*\text{hash2}(x)) % S  
If (hash(x) + 2*\text{hash2}(x)) % S is also full, then we try (hash(x) +  
3*\text{hash2}(x)) % S  
.....
```

Open Addressing

Double Hashing :- $\text{hash}(\text{key}, i) = (\text{h}_1(\text{key}) + i\text{h}_2(\text{key})) \% m$

- If $\text{h}_2(\text{key})$ is relatively prime to m , then it always find a free slot if there is one.
- Distributes keys more uniformly than linear probing and quadratic hashing).
- No clustering.



Open Addressing

$$\text{hash}(\text{key}, i) = (\text{h}_1(\text{key}) + i\text{h}_2(\text{key})) \% m$$

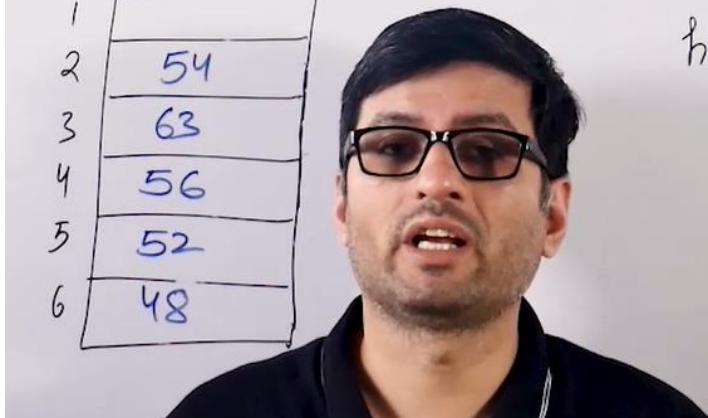
49, 63, 56, 52, 54, 48

$$m = 7$$

$$\text{h}_1(\text{key}) = (\text{key} \% 7)$$

$$\text{h}_2(\text{key}) = 6 - (\text{key} \% 6)$$

0	49
1	
2	54
3	63
4	56
5	52
6	48



Open Addressing

$$\text{hash}(\text{key}, i) = (\text{h}_1(\text{key}) + i \cdot \text{h}_2(\text{key})) \% m$$

49, 63, 56, 52, 54, 48

0	49
1	
2	54
3	63
4	56
5	52
6	48

$$m = 7$$

$$\text{h}_1(\text{key}) = (\text{key} \% 7)$$

$$\text{h}_2(\text{key}) = 6 - (\text{key} \% 6)$$

Why $\text{h}_2(\text{key})$ and m should be relatively prime?

$$\begin{array}{ll} (1 \times 6) \% 7 & = 6 \\ (2 \times 6) \% 7 & = 5 \\ (3 \times 6) \% 7 & = 4 \\ (4 \times 6) \% 7 & = 3 \\ (5 \times 6) \% 7 & = 2 \\ (6 \times 6) \% 7 & = 1 \end{array}$$

$\text{h}_2(\text{key}) = 6$

Colorful Number

```
int Solution::colorful(int n) {
    vector<int>v;
    v.clear();
    unordered_map<int,int>m;
    int n1=n;
    while(n1>0)
    {
        int x=n1%10;
        v.push_back(x);
        if(m.find(x)!=m.end())
            return false;
        m[x]++;
        n1=n1/10;
    }
    reverse(v.begin(),v.end());
    n=v.size();
    if(n==1)
        return true;
    for(int i=0;i<n;i++)
    {
        if(v[i]==1)
            return false;
    }
    vector<int>p(n+1,1);
    for(int i=1;i<=n;i++)
    {
        p[i]=p[i-1]*v[i-1];
    }
    for(int i=2;i<=n-1;i++)
    {
        for(int j=i;j<=n;j++)
        {
            int x=p[j]/p[j-i];
            if(m.find(x)!=m.end())
                return false;
            m[p[j]/p[j-i]]++;
        }
    }
    return true;
}
```

Length of the longest subarray having count of 1's one more than count of 0's

```
int Solution::solve(vector<int> &arr) {
    int n=arr.size();
    unordered_map<int, int> um;
    int sum = 0, maxLen = 0;
    for (int i = 0; i < n; i++) {
        sum += arr[i] == 0 ? -1 : 1;
        if (sum == 1)
            maxLen = i + 1;
        else if (um.find(sum) == um.end())
            um[sum] = i;
        if (um.find(sum - 1) != um.end())
            if (maxLen < (i - um[sum - 1]))
                maxLen = i - um[sum - 1];
    }
    return maxLen;
}
```

An Increment Problem(Stream of integers)

```
vector<int> Solution::solve(vector<int> &A) {
    int n=A.size();
    unordered_map<int, set<int> > mp;
    int k=1;
    vector<int> sol;
    for (int x : A)
        sol.push_back(x);

    for (int i = 0; i < n; i++) {
        if (mp.find(sol[i]) == mp.end()
            || mp[sol[i]].size() == 0) {
            mp[sol[i]].insert(i);
        }
        else {
            auto idxx = mp[sol[i]].begin();
            int idx = *idxx;
            mp[sol[i]].erase(idxx);
            sol[idx] += k;
            mp[sol[idx]].insert(idx);
            mp[sol[i]].insert(i);
        }
    }
    return sol;
}
```

Count Subarrays with given XOR

A screenshot of a Windows desktop environment. At the top, there is a taskbar with several pinned icons: Apps, Stringdock - Google Chrome, GitHub, Search - InterviewBit, https://www.pepcoding.com, Progress.xlsx - Google Sheets, SDE-PROBLEMS - Google Sheets, DSA Preparation 450, 15 Priority Queues, and a Reading list icon. Below the taskbar is a browser window with the address bar showing "sapphireengine.com". The main content area of the browser displays a C++ code editor with the following code:

```
1 int Solution::solve(vector<int> &arr, int m) {
2     int n=arr.size();
3     int ans = 0;
4     int* xorArr = new int[n];
5     unordered_map<int, int> mp;
6     xorArr[0] = arr[0];
7     for (int i = 1; i < n; i++)
8         xorArr[i] = xorArr[i - 1] ^ arr[i];
9     for (int i = 0; i < n; i++) {
10         int tmp = m ^ xorArr[i];
11         ans = ans + ((int)mp[tmp]);
12         if (xorArr[i] == m)
13             ans++;
14         mp[xorArr[i]]++;
15     }
16     return ans;
17 }
18 }
```

18:38
10-08-2021

Unique Numbers 2

```
//Given: An array A with all elements occuring twice except for x and y that occur once.  
//To Do: Find the x and y in O(1) space and O(N) time  
  
#include <bits/stdc++.h>  
using namespace std;  
const int N = 1e6;  
int arr[N];  
bool hasBitSet(int n, int x) {  
    int tem = n & (1<<x);  
    return tem!=0;  
}  
int main() {  
    int n;  
    cin >> n;  
    int all = 0;  
    int ans[2] = {0,0};  
    for(int i=0; i<n; i++){  
        cin >> arr[i];  
        all ^= arr[i];  
    }  
    int k = 0;  
    while( hasBitSet(all, k) == 0 ) k++;  
    //partition array into two sets: kth bit on v/s off  
    for (int i=0; i<n; i++) { //dividing whole array in two sets one set having kth bit as 1 and other having kth bit as 0  
        ans[hasBitSet(arr[i], k)] ^= arr[i];  
    }  
  
    cout << ans[0] << " " << ans[1] << endl;  
    return 0;  
}
```

Two out of Three

```
const int MAX = 100001;  
vector<int> Solution::solve(vector<int> &A, vector<int> &B, vector<int> &C) {  
    int hs[3][MAX];  
    memset(hs, 0, sizeof hs);  
    for(int i = 0; i < MAX; i++) {  
        if(i < A.size())  
            hs[0][A[i]]++;  
        if(i < C.size())  
            hs[1][C[i]]++;  
        if(i < B.size())  
            hs[2][B[i]]++;  
    }  
    vector<int> ans;  
    for(int i = 0; i < MAX; i++) {  
        if((hs[0][i] and hs[1][i]) or (hs[0][i] and hs[2][i]) or (hs[2][i] and hs[1][i]))  
            ans.push_back(i);  
    }  
    return ans;  
}
```

Points on the same line

```
int gcd(int a, int b) {
    while (b) {
        a = a % b;
        swap(a, b);
    }
    return a;
}
int maxPoints(vector<vector<int>>& points) {
    int n=points.size();
    if(n<3)
        return n;
    int ans=1;
    for(int i=0;i<n-1;i++)
    {
        unordered_map<string,int>m;
        int dup=1;
        for(int j=i+1;j<n;j++)
        {
            if (points[j][0] == points[i][0] && points[j][1] == points[i][1])
                dup++;
            else
            {
                int dx=points[j][0]-points[i][0];
                int dy=points[j][1]-points[i][1];
                int g = gcd(dx, dy);
                m[to_string(dx / g) + '_' + to_string(dy / g)]++;
            }
        }
        ans=max(ans,dup);
        for(auto it:m)
        {
            //ans=max(ans,it.second+1);
            ans=max(ans,it.second+dup);
        }
    }
    return ans;
}
```

Fraction to recurring decimal

Basically yhaa p jo element recur hone ki vajah hum **reminder** jo recurr ho rha h uska **mapping** kr rhe h uske **idx** k sath

```
string fractionToDecimal(int numerator, int denominator) {
    if (!numerator)
        return "0";
    string ans;
    if (numerator > 0 ^ denominator > 0) {
        ans += '-';
    }
    long n = labs(numerator), d = labs(denominator), r = n % d;
    ans += to_string(n / d);
    if (!r) {
        return ans;
    }
    ans += '.';
    unordered_map<long, int> rs;
    while(r)
    {
        if(rs.count(r))
        {
            ans+= ')';
            ans.insert(rs[r],"(");
            break;
        }
        rs[r]=ans.size();
        r=r*10;
        ans+= to_string(r/d);
        r=r%d;
    }
    return ans;
}
```

Equal

```
vector<int> solution::equal(vector<int> &arr) {
    int n=arr.size();
    unordered_map<int, pair<int, int>>Hash;
    vector<int>v;
    for (int i = 0; i < n; ++i)
    {
        for (int j = i + 1; j < n; ++j)
        {
            int sum = arr[i] + arr[j];
            if (Hash.find(sum) == Hash.end())
                Hash[sum] = make_pair(i, j);
            else
            {
                pair<int, int> pp = Hash[sum];
                if(v.size()==0&&pp.first!=pp.second&&pp.first!=i&&pp.first!=j&&pp.second!=i&&pp.second!=j)
                    v={pp.first,pp.second, i,j};
                else if(v.size()&&pp.first!=pp.second&&pp.first!=i&&pp.first!=j&&pp.second!=i&&pp.second!=j)
                {
                    //we have stored in such a way that v[0]<v[1]<v[2]<v[3]
                    bool flag=false;
                    if(pp.first<v[0])
                        flag=true;
                    else if(pp.first==v[0])
                    {
                        if(pp.second<v[1])
                            flag=true;
                        else if(pp.second==v[1])
                        {
                            if(i<v[2])
                                flag=true;
                            else if(i==v[2])
                            {
                                if(j<v[3])
                                    flag=true;
                            }
                        }
                    }
                    if(flag==true)
                        v={pp.first,pp.second,i,j};
                }
            }
        }
    }
}
```

Group Shifted String

```

11 n;cin>>n;
vector<string>v(n);
for0(i,n)
{
    cin>>v[i];
}
unordered_map<string,vector<string>>m;
for0(i,n)
{
    string ss="";
    int sz=v[i].size();
    string s=v[i];
    for0(i,sz-1)
    {
        int x=(s[i+1]-s[i]+26)%26;
        string temp=to_string(x);
        ss+=temp;
    }
    m[ss].push_back(s);
}
for(auto it=m.begin();it!=m.end();it++)
{
    vector<string>v1=it->second;
    for0(i,v1.size())
    {
        cout<<v1[i]<<" ";
    }
    cout<<endl;
}

```

SEARCHING AND SORTING:

```
int search(vector<int>& nums, int target) {
    int pivot=-1;
    int low=0,high=nums.size()-1;
    while(low<=high)
    {
        int mid=(low+high)/2;
        if(mid<high && nums[mid]>nums[mid+1])
        {
            pivot=mid;
            break;
        }
        if(mid>low && nums[mid]<nums[mid-1])
        {
            pivot=mid-1;
            break;
        }
        if(nums[low]>nums[mid])
            high=mid-1;
        else if(nums[low]<=nums[mid])
            low=mid+1;
    }
    low=0;
    high=nums.size()-1;
```

```

30     if(nums[low]>nums[mid])
31     {
32         high=mid-1;
33     }
34     else if(nums[low]<=nums[mid])
35     {
36         low=mid+1;
37     }
38 }
39 if(pivot==1)
40 {
41     if(nums[pivot]==target)
42     {
43         return pivot;
44     }
45     else if(target>nums[0])
46     {
47         high=pivot-1;
48     }
49     else
50     {
51         low=pivot+1;
52     }
53 }
54 while(low<=high)
55 {
56     int mid=(low+high)/2;
57     if(nums[mid]==target)
58     {
59         return mid;
60     }
61     if(nums[mid]>target)
62     {
63         low=mid+1;
64     }
65     else if(nums[mid]<target)
66     {
67         high=mid-1;
68     }
69 }
70 return -1;
71 }
72 }
73 if(!flag)
74 {
75     prl(-1);
76 }

```

Your previous code was restored from your local storage. [Reset to default](#)

Console ▾ Contribute i Run Code ▾ Submit

Windows Search 20:48 25-05-2021

Find Pair Given Difference:

two pointers $i=0$ and $j=1$ se fr diff compare krte rho

```

54     vector<int>a(n);
55     for(i,n)
56     cin>a[i];
57     sort(a.begin(),a.end());
58     int i=0,j=1,flag=0;
59     while(j<n&&i<n)
60     {
61         int x=a[j]-a[i];
62         if(x==diff&&i!=j)
63         {
64             prl(1);
65             flag=1;
66             break;
67         }
68         else if(x>diff)
69             i++;
70         else
71             j++;
72     }
73     if(!flag)
74     prl(-1);
75 }
76

```

Problems Courses Get Hired Events </> Problem of the Day Practice

Oracle
Samsung
Adobe
Synopsys
Infosys
Cisco
Wipro
Ola-Cabs
Morgan-Stanley
Goldman-Sachs
show more

Popular Topic Tags
Maths
Array
Dynamic-Programming
Greedy-Algorithm
Hashing
Tree
Bit-Algorithm

Trial Now
Coding & Math with WhiteHat Jr
Make the most of your kid's time, enroll in online math and coding classes at WhiteHat Jr.
code.whitehatjr.com

Learn More

Weekly Monthly Overall

Windows Search 22:17 25-05-2021

4 Sum:

The screenshot shows the LeetCode problem page for "4Sum". The problem description and examples are visible. A C++ code editor is open with the following code:

```
13 *
14     int target2=target-a[i]-a[j];
15     int first=j+1;
16     int last=n-1;
17     while(first<last)
18     {
19         if(a[first]+a[last]>target2)
20             {
21                 last--;
22             }
23         else if(a[first]+a[last]<target2)
24             {
25                 first++;
26             }
27         else
28         {
29             vector<int>temp;
30             temp.pb(a[i]);
31             temp.pb(a[j]);
32             temp.pb(a[first]);
33             temp.pb(a[last]);
34             ans.push_back(temp);
35             first++;
36             if(first==last && a[first]==temp[2])
37                 break;
38             while(first<last && a[last]==temp[3])
39                 last--;
40         }
41         while(j+1<n && a[j]==a[j+1])
42             j++;
43     }
44     while(i<n && a[i]==a[i+1])
45         i++;
46 }
47 return ans;
48 }
```

A blue star is drawn over the code editor area.

Product of Array Except Self:

Input: nums = [1,2,3,4]

Output: [24,12,8,6]

The screenshot shows the LeetCode problem page for "Product of Array Except Self". The problem description and examples are visible. A C++ code editor is open with the following code:

```
1+ class Solution {
2+ public:
3+     vector<int> productExceptSelf(vector<int>& nums) {
4+         int n=nums.size();
5+         vector<int>pdt(n);
6+         int temp=1;
7+         for(int i=0;i<n;i++)
8+         {
9+             temp*=nums[i];
10+            pdt[i]=temp;
11+        }
12+        //abhi tk bss left subarray ka pdt hua h right ka temp m store kराएंगे
13+        temp=1;
14+        for(int i=n-1;i>0;i--)
15+        {
16+            pdt[i]=pdt[i-1]*temp;
17+            temp*=nums[i];
18+        }
19+        pdt[0]=temp;
20+        return pdt;
21+    }
22+}
```

Your previous code was restored from your local storage. [Reset to default](#)

Sort by Set Bit Count:

sort lgado direct ek comp function k acc jese priority queue m lgate hai

```
int comp(int a,int b)
{
    int x= __builtin_popcount(a);
    int y= __builtin_popcount(b);
    return(x>y);
}

class Solution{
public:

void sortBySetBitCount(int arr[], int n)
{
    stable_sort(arr,arr+n,comp);
}
};
```

Allocate minimum number of pages:

You are given **N** number of books. Every i^{th} book has A_i number of pages. You have to allocate books to **M** number of students. There can be many ways or permutations to do so. In each permutation, one of the **M** students will be allocated the maximum number of pages. Out of all these permutations, the task is to find that particular permutation in which the maximum number of pages allocated to a student is minimum of those in all the other permutations and print this minimum value.

Each book will be allocated to exactly one student. Each student has to be allocated at least one book.

Example 1:

Input:

N = 4

A[] = {12, 34, 67, 90}

M = 2

Output:

113

Explanation:

Allocation can be done in following ways:

{12} and {34, 67, 90} Maximum Pages = 191

{12, 34} and {67, 90} Maximum Pages = 157

{12, 34, 67} and {90} Maximum Pages = 113

Therefore, the minimum of these cases is

113, which is selected as the output.

```
1 bool f(ll mid)
2 {
3     ll reqd=1, curr_sum=0;
4     for(i, n)
5     {
6         if(a[i]>mid)
7             return false;
8         if(a[i]+curr_sum>mid)
9         {
10            reqd++;
11            curr_sum=a[i];
12            if(reqd>stu)
13                return false;
14        }
15        else
16            curr_sum+=a[i];
17    }
18    return true;
19 }
20 ll bsearch()
21 {
22     ll l=0, h=sum, ans=INT_MAX;
23     while(l<h)
24     {
25         ll mid=(h-l)/2;
26         if(f(mid))
27         {
28             h=mid-1;
29             ans=min(ans, mid);
30         }
31         else
32             l=mid+1;
33     }
34     return ans;
35 }
```

Painter Partition Problem

```
#define mod 10000003
bool check(int n,int m,long long mid,vector<int>&a,int B)
{
    int no_of_reqd=1;
    long long allocated=0;
    for(int i=0;i<n;i++)
    {
        if(a[i]>mid)
            return false;
        allocated=(allocated+a[i])%mod;
        if(allocated>mid)
        {
            no_of_reqd++;
            allocated=a[i];
            if(no_of_reqd>m)
                return false;
        }
    }

    return (no_of_reqd<=m);
}
int Solution::paint(int m, int B, vector<int> &a) {
    long long sum=0;int n=a.size();
    for(int i=0;i<n;i++)
        sum=(sum+a[i])%mod;
    long long low=0,high=sum,ans=-1;
    while(low<=high)
    {
        long long mid=low + (high - low)/2;
        if(check(n,m,mid,a,B)==true)
        {
            ans=mid;
            high=(mid-1);
        }
        else
            low=(mid+1);
    }
    if(ans==INT_MAX)
        ans=-1;
    return (int)((ans*B)%mod);
}
```

Single Element in a Sorted Array

isme bass yh dhyan rakhna ki while k andar first if m yh check nhi krna ki whether mid>0 and mid<n instead yh check krna h ki whether **mid>low** and **mid<high**

```
int search(int low,int high,vector<int>&nums)//low is initially 0 and high=n-1
{
    while(low<=high)
    {
        int mid=low+(high-low)/2;
        if(low<mid&&mid<high)
        {
            if(nums[mid]!=nums[mid-1]&&nums[mid]!=nums[mid+1])
                return nums[mid];
        }
        if(mid==low)
        {
            if(nums[mid]!=nums[mid+1])
                return nums[mid];
        }
        if(mid==high)
        {
            if(nums[mid]!=nums[mid-1])
                return nums[mid];
        }
        if(mid%2==0)
        {
            if(nums[mid]==nums[mid+1])
                low=mid+2;
            else
                high=mid-1;
        }
        else
        {
            if(nums[mid]==nums[mid+1])
                high=mid-1;
            else
                low=mid+1;
        }
    }
    return 0;
}
```

Missing Number in AP:

```
1 findMissingUtil(int arr[], int low,
2                               int high, int diff)
3 {
4     if (high <= low)
5         return INT_MAX;
6     int mid = low + (high - low) / 2;
7
8     if (arr[mid + 1] - arr[mid] != diff)
9         return (arr[mid] + diff);
10
11    if (mid > 0 && arr[mid] - arr[mid - 1] != diff)
12        return (arr[mid - 1] + diff);
13    if (arr[mid] == arr[0] + mid * diff)
14        return findMissingUtil(arr, mid + 1,
15                               high, diff);
16    return findMissingUtil(arr, low, mid - 1, diff);
17 }
18 int findMissing(int arr[], int n)
19 {
20     int diff = (arr[n - 1] - arr[0]) / n;
21     return findMissingUtil(arr, 0, n - 1, diff);
22 }
```

Smallest number with at least n trailing zeroes in factorial:

Trailing 0s in $x!$ = Count of 5s in prime factors of $x! =$
 $= \lfloor x/5 \rfloor + \lfloor x/25 \rfloor + \lfloor x/125 \rfloor + \dots$

$l=0$, $h=5*n$ fr har mid p check kro agar aata hai th $h=mid-1$ kro jisse yh pta chlega ki minimum kispe aaega

The Double HeLiX:

Phle th dono m common elements dekhlo fr vhaa tk ka sum kiska jada h vh prefix sum array se dekhlo.

The screenshot shows a Windows desktop environment. At the top, there is a taskbar with several pinned icons: Apps, GitHub, Search - InterviewBit, https://www.pepcoding.com, SDE-PROBLEMS - G..., DSA Preparation 450, 15 Priority Queues -..., Practice | Geeksfor..., and Problems - LeetCode. Below the taskbar is a browser window titled "SPOJ.com - Problem ANARC05B" and "Sphere Online Judge (SPOJ) - Sta". The main content of the browser is a C++ code editor with the following code:

```
1 void f(n)
2 {
3     int n,s1=1,s2=1,ans=0;
4     int arr[n];
5     cin>>arr[1];
6     cin>>s1;
7     int brr[n];
8     for(i=1,n);
9     cin>>brr[1];
10    cin>>n;
11
12    int pref[n+1];
13    pref[0]=0;
14    pref[1]=arr[1];
15    for(int i=2;i<=n;i++)
16        pref[i]=pref[i-1]+arr[i-1];
17
18    int cref[n+1];
19    cref[0]=0;
20    cref[1]=brr[1];
21    cref[2]=brr[2];
22
23
24    for(int i=3;i<=n;i++)
25        cref[i]=cref[i-1]+brr[i-1];
26
27    for(int i=0;i<n;i++)
28    {
29        if(binary_search(arr,arr+s1,brr[1]))
30        {
31            int idx=lower_bound(arr,arr+s1,brr[1])-arr;
32            int val=pref[idx]-pref[idx-1];
33            s1+=val;
34            cref[s2]=cref[s2-1];
35            ans+=max(val,y);
36        }
37        s1+=idx;
38        s2+=idx;
39
40        int x=pref[n]-pref[s1-1];
41        int y=cref[n]-cref[s2-1];
42        ans+=max(x,y);
43        cout<<ans<<endl;
44        cin>>n;
45    }
46 }
```

The status bar at the bottom of the browser window shows the date and time: 26-05-2021 21:41. The system tray icons include a battery icon, signal strength, ENG, and a date/time icon.

minimum no. of swaps required to sort the array:

```
int minSwaps(vector<int>&nums)
{
    int n=nums.size(),ans=0;
    vector<pair<int,int>>arrPos(n);
    for(int i=0;i<n;i++)
    {
        arrPos[i]={nums[i],i};
    }
    sort(arrPos.begin(),arrPos.end());
    vector<bool>visited(n,false);
    for(int i=0;i<n;i++)
    {
        if(visited[i]==true || nums[i]==arrPos[i].first)
            continue;
        else
        {
            int j=i,cnt=0;
            while(!visited[j]&&arrPos[i].first!=nums[i])
            {
                visited[j]=true;
                j=arrPos[j].second;
                cnt++;
            }
            if(cnt>0)
                ans+=cnt-1;
        }
    }
    return ans;
}
```

Median of Two Sorted Arrays

```
double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
    if(nums1.size()>nums2.size())
        return findMedianSortedArrays(nums2,nums1);
    int low=0,high=nums1.size(),n=nums1.size(),m=nums2.size();
    while(low<=high)
    {
        int cut1=(high+low)/2;
        int cut2=(n+m+1)/2-cut1;
        int left1=(cut1==0)?INT_MIN:nums1[cut1-1];
        int left2=(cut2==0)?INT_MIN:nums2[cut2-1];
        int right1=(cut1==n)?INT_MAX:nums1[cut1];
        int right2=(cut2==m)?INT_MAX:nums2[cut2];

        if(left1<=right2&&left2<=right1)
        {
            if((n+m)%2==0)
            {
                return (max(left1,left2)+min(right1,right2))/2.0;
            }
            else
            {
                return max(left1,left2);
            }
        }
        else if(left1>right2)
        {
            high=cut1-1;
        }
        else
            low=cut1+1;
    }
    return 0.0;
}
```

Median in row wise sorted matrix

```
int Solution::findMedian(vector<vector<int> > &v) {
    int low=INT_MIN;
    int high=INT_MAX;
    int m=v[0].size();
    int n=v.size();
    while(low<=high)
    {
        int mid=(low+high)/2;
        int cnt=0;
        for(int i=0;i<v.size();i++)
        {
            vector<int>temp;
            temp=v[i];
            cnt+=upper_bound(temp.begin(),temp.end(),mid)-temp.begin();
        }
        // if(cnt==(n*m)/2 +1)
        // return mid;
        if (cnt <= (n * m) / 2)
            low = mid + 1;
        else
            high=mid-1;
    }
    return low;
}
```

yh aise bhi kar sakte h ki maan lo ki K sorted arrays h ab median hga sabka minimum the Kth smallest element in K sorted arrays min head se ho jaega

Container With Most Water

Start by evaluating the **widest container**, using the first and the last line. All other possible containers are less wide, so to hold more water, they need to be higher.

Thus, after evaluating that widest container, skip lines at both ends that don't support a higher height. Then evaluate that new container we arrived at. Repeat until there are no more possible containers left.

```
int maxArea(vector<int>& height) {
    int i=0,j=height.size()-1,ans=-1;
    while(i<j)
    {
        int mins=min(height[j],height[i]);
        ans=max(ans,(j-i)*mins);
        while(i<j&&height[i]<=mins)
            i++;
        while(i<j&&height[j]<=mins)
            j--;
    }
    return ans;
```

Trapping Rainwater Problem

```
int trap(vector<int>&height) {
    int n=height.size();
    if(n==0||n==1||n==2)
        return 0;
    vector<int>a(n);
    vector<int>b(n);
    a[0]=height[0];
    b[n-1]=height[n-1];
    for(int i=1;i<n;i++)
    {
        a[i]=max(height[i],a[i-1]);
    }
    for(int i=n-2;i>=0;i--)
    {
        b[i]=max(height[i],b[i+1]);
    }
    // for(int i=0;i<n;i++)
    //     cout<<a[i]<<" ";
    // cout<<endl;
    // for(int i=0;i<n;i++)
    //     cout<<b[i]<<" ";
    int profit=0;
    for(int i=0;i<n;i++)
    {
        profit+=min(a[i],b[i])-height[i];
    }
    return profit;
}
```

Valid Triangle Number

```
int triangleNumber(vector<int>& nums) {
    int res = 0, n = nums.size();

    sort(nums.begin(), nums.end());

    for (int i = n-1; i >= 0; i--) {
        int lo = 0, hi = i-1;

        while (lo < hi) {
            if (nums[lo] + nums[hi] > nums[i]) {
                res += hi - lo;
                hi--;
            }
            else lo++;
        }
    }

    return res;
}
```

[Remove Duplicates from Sorted Array II](#)

```
int removeDuplicates(vector<int>& nums) {
    int n=nums.size();
    if(n==0||n==1||n==2)
        return n;
    int cnt=1,i=0;
    for(int j=1;j<nums.size();j++)
    {
        if(nums[j-1]==nums[j])
            cnt++;
        else
            cnt=1;
        if(cnt<=2)
            nums[++i]=nums[j];
    }
    return ++i;
}
```

[Max Consecutive Ones III](#)

Given a binary array `nums` and an integer `k`,
return *the maximum number of consecutive 1's
in the array if you can flip at most k 0's.*

```

int longestOnes(vector<int>& nums, int k) {
    int i=0,j,k1=k,n=nums.size(),ans=-1;
    if(k>=nums.size())
        return nums.size();
    for(j=0;j<n;j++)
    {
        if(nums[j]==0)
            k1--;
        while(k1<0)
        {
            if(nums[i]==0)
            {
                i++;
                k1++;
            }
            else
                i++;
        }
        ans=max(ans,j-i+1);
    }
    return ans;
}

```

Count Number of Nice Subarrays

```

int atmost(vector<int>& nums, int k)
{
    int i=0;
    int n=nums.size();
    int k1=k,cnt=0;
    for(int j=0;j<n;j++)
    {
        if(nums[j]%2==1)
        {
            k1--;
        }
        while(k1<0)
        {
            if(nums[i]%2==1)
            {
                k1++;
                i++;
            }
            else
                i++;
        }
        if(k1>=0)
        {
            cnt+=j-i+1;
        }
    }
    return cnt;
}
int numberOfSubarrays(vector<int>& nums, int k) {
    //exactly K=atmost(k)-atmost(k-1)
    return (atmost(nums,k)-atmost(nums,k-1));
}

```

Subarray Product Less Than K

(ese ques h hmesha while loop phle lgao cnt m add krne se)

```
int numSubarrayProductLessThanK(vector<int>& nums, int k) {
    int n=nums.size();
    if(k<=1)
        return 0;
    int pdt=1,i=0,cnt=0;
    for(int j=0;j<n;j++)
    {
        pdt=pdt*nums[j];
        while(pdt>=k)
        {
            pdt=pdt/nums[i++];
        }
        cnt+=j-i+1;
    }
    return cnt;
}
```

Subarrays with K Different Integers

```
int atmost(vector<int>& nums, int k)
{
    int i=0, n=nums.size(), cnt=0;
    unordered_map<int,int>m;
    for(int j=0;j<n;j++)
    {
        if(m.count(nums[j])==0)
        {
            k--;
        }
        m[nums[j]]++;
        while(k<0)
        {
            m[nums[i]]--;
            if(m[nums[i]]==0)
            {
                m.erase(nums[i]);
                k++;
            }
            i++;
        }
        cnt+=j-i+1;
    }
    return cnt;
}
int subarraysWithKDistinct(vector<int>& nums, int k) {
    int a=atmost(nums,k);
    int b=atmost(nums,k-1);
    return (a-b);
}
```

Pow(x,n)

```
double myPow(double x, int n) {
    double ans=1.0;
    long long absN=abs(n);
    while(absN>0)
    {
        if((absN&1)==1)
        {
            ans=ans*x;
            absN--;
        }
        else
        {
            absN>>=1;
            x*=x;
        }
    }
    return (n<0)?(double)1.0/ans :ans;
}
```

BUBBLE SORT

Bubble Sort

Date / DELTA Pg No. / 108

Sort 'n' elements $\in \mathbb{R}$ at ' $n-1$ ' iterations until $\in \mathbb{R}$
 At i^{th} iteration # swap first element last at
 Pch jata $\in \mathbb{R}$
 i^{th} iteration # i^{th} ante swap adjacent elements compare them

Q1

1.1

5 9 8 2 1
 ↙ no swap

1.2

5 9 8 9 2 1
 ↙ swap

1.3

5 8 9 2 9 1
 ↙ swap

Worst Case $T.C. \geq O(n^2)$

Best Case $\Rightarrow O(n)$ when all are sorted

avg $\log n^2$

MHb Bb Sorts

Stable \Rightarrow Yes

Boundary left \Rightarrow when elements are sorted.

1.4

5 8 2 9 2 9 \Rightarrow 5 8 2 1 [9]
 ↙ swap next item for this array

Q2

2.1

5 8 2 1 [9]

2.2

5 8 2 8 1 [9]
 ↙ swap

2.3

5 2 8 1 9 [9] \Rightarrow 5 2 1 [8 9]
 ↙ swap next for this array

```
void bubbleSort(int arr[], int n)
{
    int m=n-1;
    int cnt=1;
    while(m)
    {
        bool swap=false;
        for(int i=0;i<n-cnt;i++)
        {
            if(arr[i]>arr[i+1])
            {
                int temp=arr[i+1];
                arr[i+1]=arr[i];
                arr[i]=temp;
                swap=true;
            }
        }
        if(swap==false)
            break;
        m--;
        cnt++;
    }
}
```

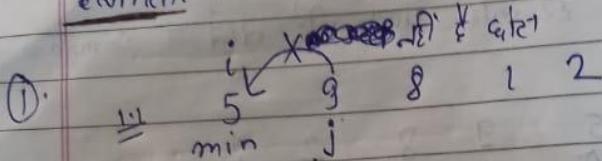
Selection Sort

How

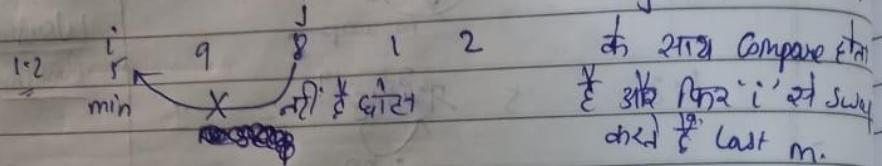
Selection Sort

DELTA Pg No.

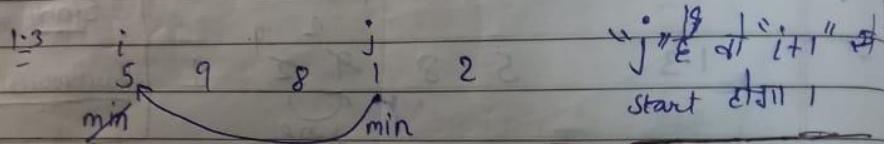
अगर n elements हैं तो $n-1$ iterations होती है। क्योंकि 42
में सबसे छोटा element होता है और उसको First
element के बाय Swap करते हैं।



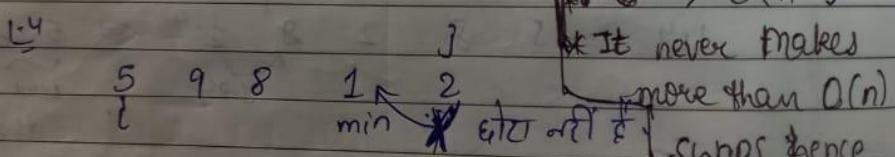
j का Min



के साथ Compare कर
और अगर i का swap
चर्हा हो तो last m.



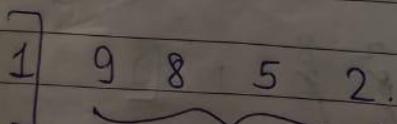
" j " का "i+1" का
start होगा।



$T.C. \Rightarrow O(n^2)$

* It never makes
more than $O(n)$
swaps hence
memory efficient
* Stable

In last swap i & min



next iteration for this.

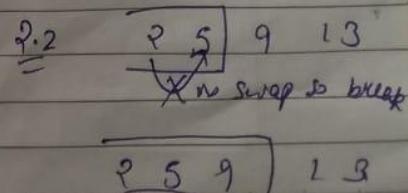
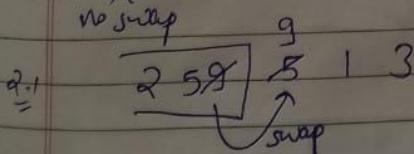
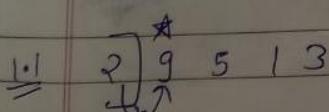
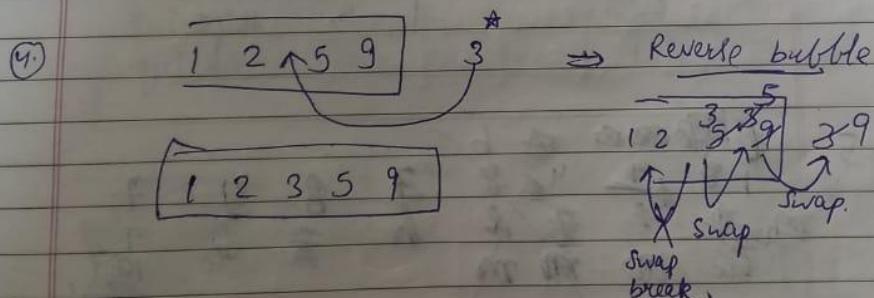
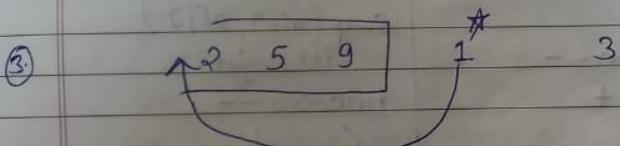
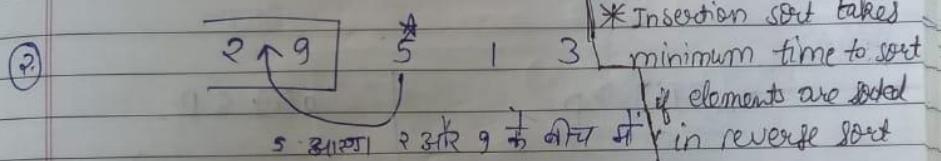
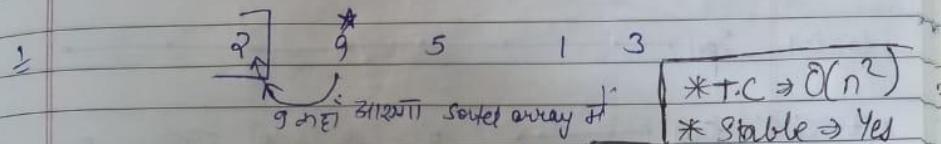
```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;
    for (i = 0; i < n-1; i++)
    {
        min_idx = i;
        for (j = i+1; j < n; j++)
        {
            if (arr[j] < arr[min_idx])
                min_idx = j;
        }
        swap(&arr[min_idx], &arr[i]);
    }
}
```

Insertion Sort

Insertion Sort

Date _____
DELTA Pg No. 110

n elements \Rightarrow $n-1$ iterations के लिए प्रक्रिया
element को sorted array में bcz 1 element को sorted
array के 3rd next element को compare करने के बाद $i-1$ तक की तरह
sorted array के 3rd next element को compare करता है,



```
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Partition an array

Date _____
 DELTA Pg No. _____

Partition an array

[$pvt = 5$]

$j < i \{ \text{array} \}$ $j \leftarrow$
 ↓
 7 9 4 8 3 6 2 1

i to end unknown

j to i-1 $> pvt$

0 to j-1 $< pvt$

$a[i] > p$

i++

Unknown --

> +

$a[i] \leq p$

swap($a[i], a[j]$)

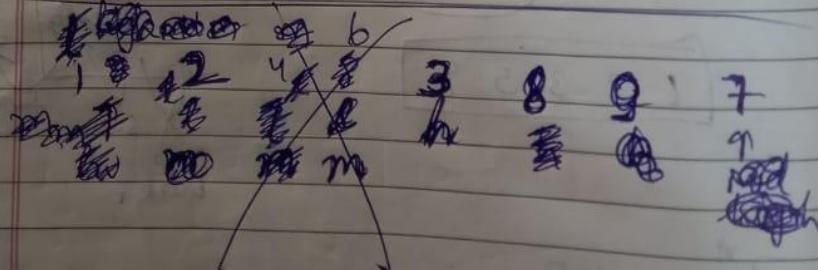
i++, j++:

unknow --

<= +

> Shift

$pvt = 5$



```
vector<int>helper(vector<int>&a,int pvt)
{
    int i=0,j=0,n=a.length();
    //0->j-1=><=pvt
    //j->i-1=> >pvt
    //i->n-1=> unknown
    while(i<n&&j<n)
    {
        if(a[i]>pvt)
            i++//bcz i-1 tk th greater than pivot rhnge
        else
        {
            swap(a[i],a[j]);
            i++;j++;//bcz j-1 tk th smaller than equal to pivot rhnge
        }
    }
    return a;
}
```

Quick Sort

Quick Sort

Date _____
DELTA Pg No. 112

8 5 1 3 7 2 9 6
After Pivot partitioning ↓

Pass it into put function

8 5 1 3 2 6 8 7 9 स्टेप Recursion
↑
स्टेप Recursion उन्हें element तो sort हो जाएगा।
मिलाओ।

o (initially) n-1 (initially)

Void quicksort (int arr[], int lo, int hi)

{
if (lo >= hi)
return;

T.C \Rightarrow worst \Rightarrow $O(n^2)$
Avg \Rightarrow $O(n \log n)$
Stable \Rightarrow No

int pivot = arr[hi];
int pvt_idn = partition (arr, lo, hi, pivot);
quicksort (arr, lo, pvt_idn - 1);
quicksort (arr, pvt_idn + 1, hi);

}

Because array ने extra space कैसे किया?

Quick Sort

Preferred over Merge in Arrays

Merge

Sort preferred in Linked List

Not linked list नहीं

मध्य में जो O(1) नोड

नहीं प्रोटोकॉल नहीं

Count Sort

Count Sort

यह तरल अलगे जाने वाले elements का अंतर नहीं है।
But count vary का करे MHB range का बढ़ावा देता है।

e.g.: Jee Rank को sort करके Marks को Ace to Slab convert
But Marks $0 \rightarrow 340$ के बीच में एक नहीं है।

अब यह traversal करने में 8 steps होंगे।
 $\frac{3}{3} \quad \frac{9}{9}$
freq. array का size के बराबर हो।

$$(3-3+1) = 7$$

T.C $\Rightarrow O(n)$	stable
Space \Rightarrow Range	

Original array $\Rightarrow 9^*, 6, 3, 5, 3, 4, 3, 9^*, 6, 4, 6, 5, 8^*, 9^*$
 $\downarrow \quad \downarrow \quad \downarrow$
 $0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad 12 \quad 13 \quad 14$

Stand for freq. of 3 value
 freq. index $\Rightarrow 0[3] \quad 1[4] \quad 2[5] \quad 3[6] \quad 4[7] \quad 5[8] \quad 6[9]$
 $\Rightarrow \boxed{3 \quad 2 \quad 2 \quad 3 \quad 0 \quad 1 \quad 4}$

अब freq. array की Prefix sum लिखाना।

3	5	7	10	10	11	15
---	---	---	----	----	----	----

पहली 3 पर 3rd position
 4th 3rd 5th position
 तक 4th 5th position

pref[i]-1 से 10 तक 14 तक 15

2	4	6	9	9	10	14	.
---	---	---	---	---	----	----	---

3rd 3rd 3rd 8th 3rd 9th 14th
 2nd position 4th 10th position 9th 14th
 3rd 3rd 3rd 3rd 3rd 3rd 3rd

Traverse original array from right to left.

अब हमें यही जाना होता कि एक सिर्फ $a[n-1]$ के ग' तो
आपका prefix array में ऐसा आया 14th idm ans array के g'
14th idm में आवरण 14 को -1 करदे तो 13 मैंलग अब 14
ग' आया तो वह 13th idm पर आया।

Direct Prefix में freq. store करके बड़ी कमी करा

Bcz Current sort जो stable रहना दिता है

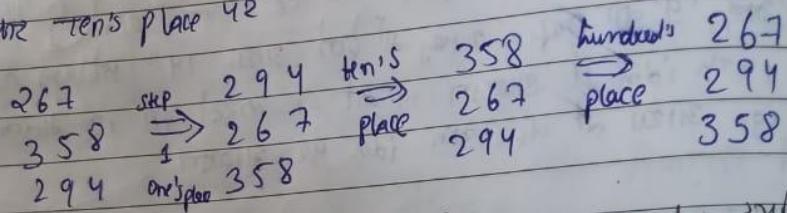
ज) अब DELTA traverse करते ही पहले g''' 3rd 1st 1st 2nd 3rd 4th
लिए g'' 2nd 3rd 4'.
.....

```
public static void countSort(int[] arr, int min, int max) {  
    //write your code here  
    int range = max - min + 1;  
    int[] farr = new int[range];  
  
    for(int i = 0; i < arr.length; i++){  
        int idx = arr[i] - min;  
        farr[idx]++;  
    }  
  
    for(int i = 1; i < farr.length; i++){  
        farr[i] = farr[i] + farr[i - 1];  
    }  
  
    int[] ans = new int[arr.length];  
  
    for(int i = arr.length - 1; i >= 0; i--){  
        int val = arr[i];  
        int pos = farr[val - min];  
        int idx = pos - 1;  
        ans[idx] = val;  
        farr[val - min]--;  
    }  
  
    for(int i = 0; i < ans.length; i++){  
        arr[i] = ans[i];  
    }  
}
```

Radix Sort

Radix Sort

सबसे पहले Unit's place के Basis पर sort करते।
पहले Ten's place पर



* * ① पहले One's place, Ten's place, Hundred's place का precedence क्या है।
मित्र वह hundred's place की precedence jada है।

* * ② Reason कि मानव Ten's place पर sort करते।
Hence hundred's place का stability रखना कठिन है।
जिसके बाद 267, 358, 294 को Ten's place का stability रखना होता है।
267 को 294 से 358 की ओर लाना होता है।

$$n > t > o \quad (\text{precedence})$$

अपने को hundred place पर बराकर करते हैं।

तो पुराना Order preserve करते हैं।

5934

$$4 \rightarrow (5934/1) \% 10$$

$$3 \rightarrow (5934/10) \% 10$$

$$9 \rightarrow (5934/100) \% 10$$

$$5 \rightarrow (5934/1000) \% 10$$

```
public static void radixSort(int[] arr) {  
    // write code here    I  
    int max = Integer.MIN_VALUE;  
    for(int val: arr){  
        if(val > max){  
            max = val;  
        }  
    }  
  
    int exp = 1;  
    while(exp <= max){  
        countSort(arr, exp);  
        exp = exp * 10;  
    }  
}  
  
  
public static void countSort(int[] arr, int exp) {  
    int[] ans = new int[arr.length];  
    //make frequency arr  
    int[] farr = new int[10];  
    for(int i = 0 ; i < arr.length; i++){  
        farr[arr[i] / exp % 10]++;  
    }  
    //convert it into prefix sum array  
    for(int i = 1 ; i < farr.length; i++){  
        farr[i] += farr[i - 1];  
    }  
    //stable sorting(filling ans array)  
    for(int i = arr.length - 1; i >= 0; i--){  
        int pos = farr[arr[i] / exp % 10] - 1;  
        ans[pos] = arr[i];  
        farr[arr[i] / exp % 10]--;  
    }  
    //filling original array with the help of ans array  
    for(int i = 0 ; i < arr.length; i++){  
        arr[i] = ans[i];  
    }  
}
```



Linked Lists:

Reverse a Linked List in group of Given Size:

```
1 struct node *reverse (struct node *head, int k)
2 {
3     if(head==NULL)
4         return NULL;
5     int cnt=0;
6     node*curr=head,*prev=NULL,*next1=NULL;
7     while(curr!=NULL&&cnt<k)
8     {
9
10        next1=curr->next;
11        curr->next=prev;
12        prev=curr;
13        curr=next1;
14        cnt++;
15    }
16    if(next1!=NULL)
17        head->next=reverse(next1,k);
18    return prev;
19 }
```

Reverse Alternate K Nodes

```
ListNode* KAltReverse(ListNode* head, int k, bool flag)
{
    if(flag==true)
    {
        int cnt=0;
        ListNode* curr=head, *prev=NULL, *next1=NULL;
        while(curr!=NULL&&cnt<k)
        {
            next1=curr->next;
            curr->next=prev;
            prev=curr;
            curr=next1;
            cnt++;
        }
        if(next1!=NULL)
            head->next=KAltReverse(next1,k,false);
        return prev;
    }
    else if(flag==false)
    {
        int cnt=0;
        ListNode* curr=head, *prev=NULL, *next1=NULL;
        while(curr!=NULL&&cnt<k)
        {
            next1=curr->next;
            prev=curr;
            curr=curr->next;
        }
        if(next1!=NULL)
            prev->next=KAltReverse(next1,k,true);
        return head;
    }
}
ListNode* Solution::solve(ListNode* head, int k) {
    return KAltReverse(head,k,true);
}
```

Intersection Point of two Linked Lists.

Ek method toh h ki bdi string ko diff of length se start krlo fr two pointers se address match kro jhaan hue match vhi intersection point h vrna M2. k liye striver ki video dekho

The screenshot shows a LeetCode problem titled '(1) Intersection of Two Linked Lists'. The code is written in C++ and defines a singly-linked list structure and a Solution class with a getIntersectionNode method. The code iterates through both lists until it finds a common node or reaches NULL.

```
1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode(int x) : val(x), next(NULL) {}
7  * };
8 */
9
10 class Solution {
11 public:
12     ListNode *getIntersectionNode(ListNode *head1, ListNode *head2) {
13         if(head1==NULL||head2==NULL)
14             return NULL;
15         ListNode*a=head1;
16         ListNode*b=head2;
17         while(a!=b)
18         {
19             a=(a==NULL)?head2:a->next;
20             b=(b==NULL)?head1:b->next;
21         }
22         if(a==b&&a==NULL)
23             return NULL;
24         return a;
25     }
26 };

```

Your previous code was restored from your local storage. [Reset to default](#)

Console ▾ Contribute i Run Code ▾ Submit

Windows Search Taskbar 15:49 27-05-2021

Merge Sort For Linked lists.[Very Important]:

The screenshot shows a Sapphire Engine browser window displaying C++ code for merging two linked lists. The code uses a recursive approach to merge the lists. It includes a findMid function to find the middle of the list using the Tortoise and Hare approach.

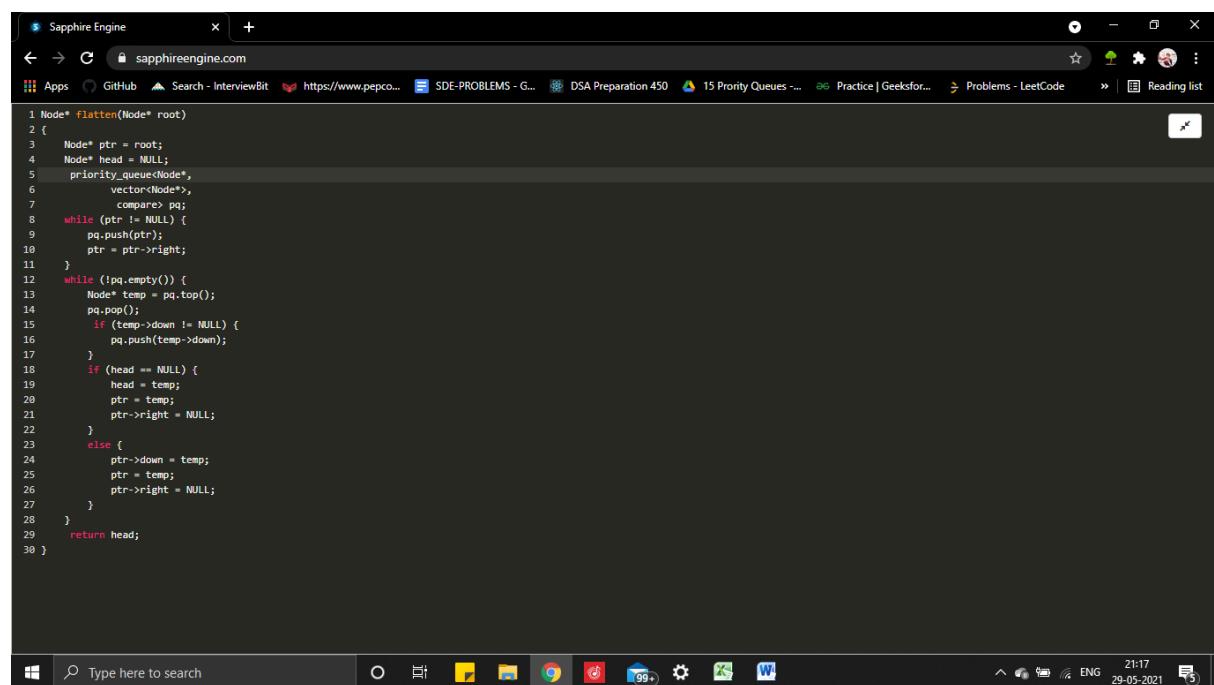
```
1 static Node mergeSort(Node head) {
2     if (head.next == null)
3         return head;
4
5     Node mid = findMid(head);
6     Node head1 = mid.next;
7     mid.next = null;
8
9     Node newHead1 = mergeSort(head);
10    Node newHead2 = mergeSort(head1);
11    Node finalHead = merge(newHead1, newHead2);
12
13    return finalHead;
14}
15 static Node merge(Node head1, Node head2) {
16 {
17     Node merged = new Node(-1);
18     Node temp = merged;
19     while (head1 != null && head2 != null) {
20         if (head1.data < head2.data) {
21             temp.next = head1;
22             head1 = head1.next;
23         } else {
24             temp.next = head2;
25             head2 = head2.next;
26         }
27         temp = temp.next;
28     }
29     while (head1 != null) {
30         temp.next = head1;
31         head1 = head1.next;
32         temp = temp.next;
33     }
34     while (head2 != null) {
35         temp.next = head2;
36         head2 = head2.next;
37         temp = temp.next;
38     }
39 }
40 return merged.next;
41 }
42 // Find mid using The Tortoise and The Hare approach
43 static Node findMid(Node head) {
44 {
45     Node slow = head, fast = head.next;
46     while (fast != null && fast.next != null) {
47         slow = slow.next;
48         fast = fast.next.next;
49     }
50 }
51 return slow;
52 }
```

Windows Search Taskbar 11:18 28-05-2021

Split a Circular linked list into two halves:

```
1 void splitList(Node *head, Node **head1_ref,
2                               Node **head2_ref)
3 {
4     Node *slow_ptr = head;
5     Node *fast_ptr = head;
6     if(head == NULL)
7         return;
8     while(fast_ptr->next != head &&
9            fast_ptr->next->next != head)
10    {
11        fast_ptr = fast_ptr->next->next;
12        slow_ptr = slow_ptr->next;
13    }
14    if(fast_ptr->next->next == head)
15        fast_ptr = fast_ptr->next;
16    *head1_ref = head;
17    if(head->next != head)
18        *head2_ref = slow_ptr->next;
19    fast_ptr->next = slow_ptr->next;
20    slow_ptr->next = head;
21 }
```

Flatten a Linked List:



```
1 Node* flatten(Node* root)
2 {
3     Node* ptr = root;
4     Node* head = NULL;
5     priority_queue<Node*>;
6     vector<Node*>;
7     compare(pq);
8     while (ptr != NULL) {
9         pq.push(ptr);
10        ptr = ptr->right;
11    }
12    while (!pq.empty()) {
13        Node* temp = pq.top();
14        pq.pop();
15        if (temp->down != NULL) {
16            pq.push(temp->down);
17        }
18        if (head == NULL) {
19            head = temp;
20            ptr = temp;
21            ptr->right = NULL;
22        }
23        else {
24            ptr->down = temp;
25            ptr = temp;
26            ptr->right = NULL;
27        }
28    }
29    return head;
30 }
```

```

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;
    return res -> bottom;
}

Node *flatten(Node *root)
{
    if (root == NULL || root->next == NULL)
        return root;
    root->next = flatten(root->next);
    root = mergeTwoLists(root, root->next);
    return root;
}

```

Clone a linked list with next and random pointer:

```

1 Node*copy_next(Node*head,unordered_map<Node*,Node*>&m)
2 {
3     if(!head)
4         return NULL;
5     Node*clone_node=new Node(head->data);
6     m[head]=clone_node;
7     clone_node->next=copy_next(head->next,m);
8     return clone_node;
9 }
10
11 Node *copyList(Node *head) {
12     if(head==NULL)
13         return NULL;
14     Node*clone_head=NULL;
15     unordered_map<Node*,Node*>m;
16     clone_head=copy_next(head,m);
17     Node*temp=clone_head;
18     Node*temp1=head;
19     while(temp1!=NULL)
20     {
21         temp->arb=m[temp1->arb];
22         temp1=temp1->next;
23         temp=temp->next;
24     }
25     return clone_head;
26 }

```

Delete nodes having greater value on right

phle reverse krdo puri ko fr jo head aaya reverse hone k baad usko max lelo ab iterate kro agar temp->data>max th yh node remove nhi hga aur max ko update hga ab agar yh less than max hua th yh node remove krdo aur pure iteration k baad vps jo linked list aai ko reverse krdo original bnnane k liye.

```

1 |
2 Node *compute(Node *head)
3 {
4     if(head==NULL)
5         return NULL;
6     head=reverse(head);
7     int max=head->data;
8     Node*prev=head;
9     Node*curr=head->next;
10    while(curr!=NULL)
11    {
12        if(curr->data>=max)
13        {
14            max=curr->data;
15            prev=curr;
16            curr=curr->next;
17        }
18        else
19        {
20            prev->next=curr->next;
21            Node*x=curr;
22            curr=curr->next;
23            x->next=NULL;
24            free(x);
25        }
26    }
27    return reverse(head);
28 }

```

Program for n'th node from the end of a Linked List:

Maintain two pointers - reference pointer and main pointer. Initialize both reference and main pointers to head. First, move the reference pointer to n nodes from head. Now move both pointers one by one until the reference pointer reaches the end. Now the main pointer will point to nth node from the end. Return the main pointer.

First non-repeating character in a stream

Input: A = "aabc"

Output: "a#bb"

Explanation: For every character first non

repeating character is as follow-

"a" - first non-repeating character is 'a'

"aa" - no non-repeating character so '#'

"aab" - first non-repeating character is 'b'

"aabc" - first non-repeating character is 'b'

```
1 string FirstNonRepeating(string s){
2     int arr[256]={0};
3     queue<char>q;
4     string ans="";
5     ans+=s[0];
6     q.push(s[0]);
7     arr[s[0]]++;
8     for(int i=1;i<s.length();i++)
9     {
10         arr[s[i]]++;
11         if(arr[s[i]]==1){
12             q.push(s[i]);
13         }
14         while(q.size()>0&&arr[q.front()]>1)
15             q.pop();
16         if(q.size()==0)
17             ans+='#';
18         else
19             ans+=q.front();
20     }
21     return ans;
22 }
```

Inception Sort List

```
ListNode* insertionSortList(ListNode* head) {
    if(head==NULL||head->next==NULL)
        return head;
    ListNode*temp=head->next;
    head->next=NULL;
    while(temp!=NULL)
    {
        ListNode*q=head;
        ListNode*temp_next=temp->next;
        if(temp->val<q->val)
        {
            temp->next=q;
            head=temp;
        }
        else
        {
            while(q->next!=NULL&&q->next->val<=temp->val)
                q=q->next;
            if(q->next==NULL)
            {
                q->next=temp;
                temp->next=NULL;
            }
            else
            {
                ListNode*q_next=q->next;
                q->next=temp;
                temp->next=q_next;
            }
        }
        temp=temp_next;
    }
    return head;
}
```

Reorder List

mid se break krdo phle th fr dusri ko LL bni
usko reverse krdo fr bss merge krte rho

```
void reorderList(ListNode* head) {
    if(head==NULL||head->next==NULL)
        return;
    ListNode*slow=head;ListNode*fast=head;
    while(fast->next!=NULL&&fast->next->next!=NULL)
    {
        slow=slow->next;
        fast=fast->next->next;
    }
    ListNode*head1=slow->next;
    slow->next=NULL;
    ListNode*head2=reverse(head1);
    head1=head;
    while(head2!=NULL)
    {
        ListNode*temp1=head1->next;
        ListNode*temp2=head2->next;
        head1->next=head2;
        head2->next=temp1;
        head1=temp1;
        head2=temp2;
    }
}
```

Remove Nth Node From End of List

```
ListNode* removeNthFromEnd(ListNode* head, int key) {
    ListNode *temp;
    ListNode *first = head;
    ListNode *second = head;
    for (int i = 0; i < key; i++)
    {
        // If count of nodes in the given
        // linked list is <= N
        if (second->next == NULL)
        {
            // If count = N i.e.
            // delete the head node
            if (i == key - 1){
                temp = head;
                head = head->next;
                //free (temp);
            }
            return head;
        }
        second = second->next;
    }
    while (second->next != NULL)
    {
        first = first->next;
        second = second->next;
    }
    temp = first->next;
    first->next = first->next->next;
    //free (temp);
    return head;
}.
```

Remove Duplicates from Sorted List II

```
ListNode* Solution::deleteDuplicates(ListNode* head) {
    ListNode* current = head;
    ListNode* prev=NULL;
    while (current != NULL && current->next != NULL) {
        if (current->next->val == current->val) {
            int value=current->val;
            while(current!=NULL&&current->val==value)
                current=current->next;
            if(prev!=NULL)
                prev->next = current;
            else
                head=current;
        } else {
            prev=current;
            current = current->next;
        }
    }
    return head;
}
```

Even Reverse

```
ListNode* t;
int n;
void reverse(ListNode* a,int i){
    if(a==NULL) return;
    reverse(a->next,i+1);
    if(i%2 || i*2<=n) return;
    int x = t->val;
    t->val=a->val;
    a->val=x;
    t=t->next->next;
}
ListNode* Solution::solve(ListNode* a) {
    int sz=0;
    auto temp = a;
    while(temp){
        sz++;
        temp=temp->next;
    }
    if(sz<=3) return a;
    t=a->next;
    n=sz;
    reverse(a,1);
    return a;
}
```

Kth Node From Middle

```
int Solution::solve(ListNode* head, int k) {
    if(head==NULL||head->next==NULL)
        return -1;
    ListNode*slow=head,*fast=head;
    while(k>0)
    {
        if(fast->next&&fast->next->next)
            slow=slow->next;fast=fast->next->next;
        else if(fast->next!=NULL&&fast->next->next==NULL)
            slow=slow->next;fast=fast->next;
        else
            break;
        k--;
    }
    cout<<k<<endl;
    if(k>0)
        return -1;
    else
    {
        ListNode*ans=head;
        while(fast->next&&fast->next->next)
        {
            slow=slow->next;
            fast=fast->next->next;
            ans=ans->next;
        }
        if(fast->next)
        {
            slow=slow->next;
            fast=fast->next;
            ans=ans->next;
        }
        return ans->val;
    }
}
```

GREEDY:

Minimum Platforms Problem:

```
1 int findPlatform(int arr[], int dep[], int n)
2 {
3     sort(arr,arr+n);
4     sort(dep,dep+n);
5     int p_need=1;
6     int ans=-1;
7     int i=1,j=0;
8     while(i<n&&j<n)
9     {
10         if(dep[j]>=arr[i])
11         {
12             p_need++;
13             i++;
14         }
15         else if(arr[i]>dep[j])
16         {
17             p_need--;
18             j++;
19         }
20         ans=max(ans,p_need);
21     }
22     return ans;
23 }
```

Minimum Cost to cut a board into squares:

Sort krdo irrespective horizontal h ya vertical fr max vale lete jaao MAX agar horizontal h th vertical cuts se multiply kro agar vertical array m h th horizontal cuts se multiply kro.

```

1 int minimumCostOfBreaking(int X[], int Y[], int m, int n)
2 {
3     int res = 0;
4     sort(X, X + m, greater<int>());
5     sort(Y, Y + n, greater<int>());
6     int hzntl = 1, vert = 1;
7     int i = 0, j = 0;
8     while (i < m && j < n)
9     {
10         if (X[i] > Y[j])
11         {
12             res += X[i] * vert;
13             hzntl++;
14             i++;
15         }
16         else
17         {
18             res += Y[j] * hzntl;
19             vert++;
20             j++;
21         }
22     }
23     int total = 0;
24     while (i < m)
25         total += X[i++];
26     res += total * vert;
27     total = 0;
28     while (j < n)
29         total += Y[j++];
30     res += total * hzntl;
31
32     return res;
33 }
```

Minimize Cash Flow among a given set of friends who have borrowed money from each other:(TC: O(N²))

```
1 int getMin(int arr[])
2 {
3     int minInd = 0;
4     for (int i=1; i<N; i++)
5         if (arr[i] < arr[minInd])
6             minInd = i;
7     return minInd;
8 }
9 int getMax(int arr[])
10 {
11     int maxInd = 0;
12     for (int i=1; i<N; i++)
13         if (arr[i] > arr[maxInd])
14             maxInd = i;
15     return maxInd;
16 }
17 int minOf2(int x, int y)
18 {
19     return (x<y)? x: y;
20 }
21 void minCashFlowRec(int amount[])
22 {
23     int mxCredit = getMax(amount), mxDebit = getMin(amount);
24     if (amount[mxCredit] == 0 && amount[mxDebit] == 0)
25         return;
26     int min = minOf2(-amount[mxDebit], amount[mxCredit]);
27     amount[mxCredit] -= min;
28     amount[mxDebit] += min;
29     cout << "Person " << mxDebit << " pays " << min
30     << " to " << "Person " << mxCredit << endl;
31     minCashFlowRec(amount);
32 }
33 void minCashFlow(int graph[][N])
34 {
35     int amount[N] = {0};
36     for (int p=0; p<N; p++)
37         for (int l=0; l<N; l++)
38             amount[p] += (graph[l][p] - graph[p][l]);
39     minCashFlowRec(amount);
40 }
```

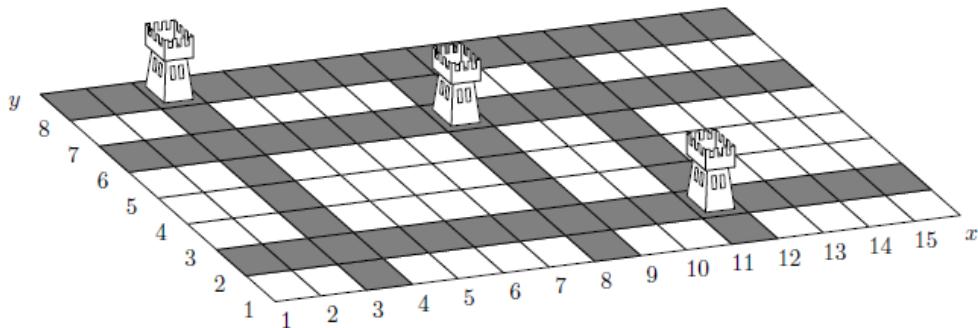
DEFKIN -Defense of a Kingdom:

DEFKIN - Defense of a Kingdom

no tags

Theodore implements a new strategy game "Defense of a Kingdom". On each level a player defends the Kingdom that is represented by a rectangular grid of cells. The player builds crossbow towers in some cells of the grid. The tower defends all the cells in the same row and the same column. No two towers share a row or a column.

The penalty of the position is the number of cells in the largest undefended rectangle. For example, the position shown on the picture has penalty 12.



Help Theodore write a program that calculates the penalty of the given position.

Input

The first line of the input file contains the number of test cases.

Each test case consists of a line with three integer numbers: w — width of the grid, h — height of the grid and n —

```

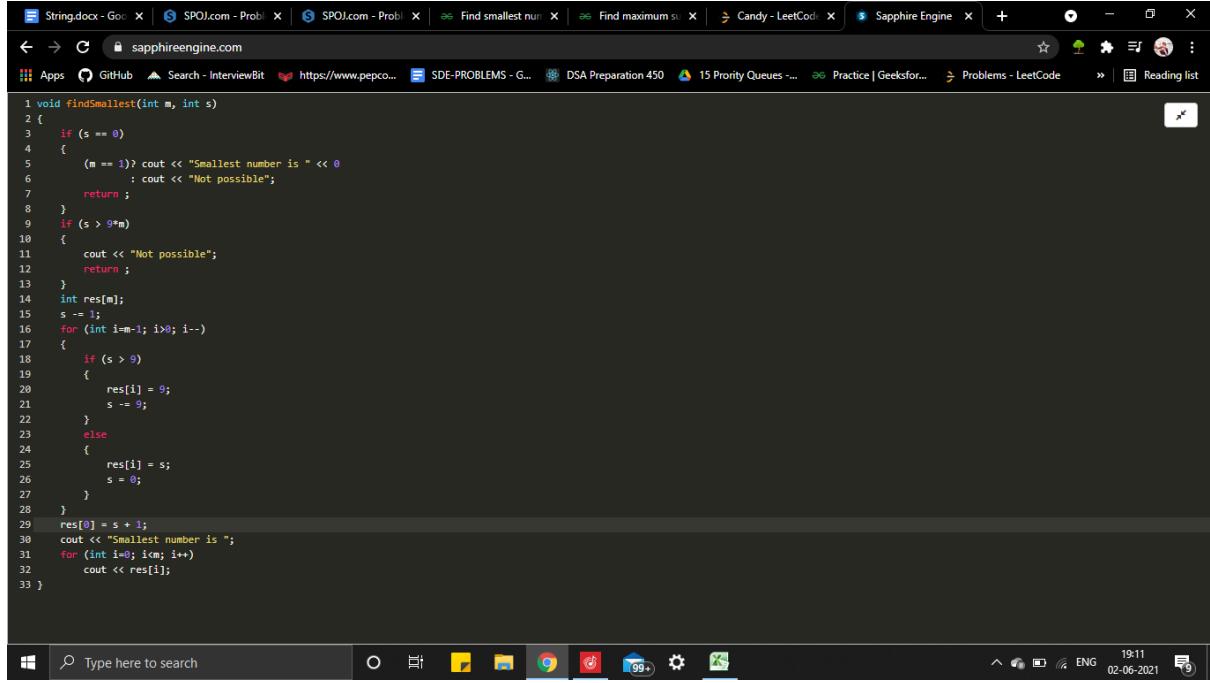
1 t
2 {
3     ll x,y,n;
4     cin>>x>>y>>n;
5     if(n==0)
6         prl(x*y);
7     else
8     {
9         vector<int>vx;
10        vector<int>vy;
11        for(i,n)
12        {
13            ll p,q;
14            cin>>p>>q;
15            vx.pb(p);
16            vy.pb(q);
17        }
18        vx.pb(x+1);
19        vy.pb(y+1);
20        sort(all(vx));
21        sort(all(vy));
22        ll maxx=vx[0]-1,maxy=vy[0]-1;
23        for(int i=1;i<=n;i++)
24        {
25            maxx=max(maxx,vx[i]-vx[i-1]-1);
26            maxy=max(maxy,vy[i]-vy[i-1]-1);
27        }
28        prl(maxx*maxy);
29    }
30}

```

GERGOVIA - Wine trading in Gergovia:

ek infinite while loop lgao fr uska andar i point krega frst buy ko aur j point krega frst sell ko fr sauda krke a[i] m kam jada krke next p jao fr vhi i j check kro naya.

Find smallest number with given number of digits and sum of digits:



```
1 void findSmallest(int m, int s)
2 {
3     if (s == 0)
4     {
5         (m == 1)? cout << "Smallest number is " << 0
6         : cout << "Not possible";
7         return ;
8     }
9     if (s > 9*m)
10    {
11        cout << "Not possible";
12        return ;
13    }
14    int res[m];
15    s -= 1;
16    for (int i=m-1; i>0; i--)
17    {
18        if (s > 9)
19        {
20            res[i] = 9;
21            s -= 9;
22        }
23        else
24        {
25            res[i] = s;
26            s = 0;
27        }
28    }
29    res[0] = s + 1;
30    cout << "Smallest number is ";
31    for (int i=0; i<m; i++)
32        cout << res[i];
33 }
```

Candy:

jab bhi neighbour vgerah bola ho th ek baar right iterate kro aur ek baar left iterate kro mostly usse ho jata h.

```

1 v
2
3 v
4 public:
5     int candy(vector<int>& ratings) {
6         int size=ratings.size();
7         if(size<1)
8             return size;
9         vector<int> num(size,1);
10        //DIVIDE INTO SUBPROBLEMS
11        //LEFT VALA DEKHO
12        for (int i = 1; i < size; i++)
13        {
14            if(ratings[i]>ratings[i-1])
15                num[i]=num[i-1]+1;
16        }
17        //RIGHT VALA DEKHO
18        for (int i= size-1; i>0 ; i--)
19        {
20            if(ratings[i-1]>ratings[i])
21                num[i-1]=max(num[i]+1,num[i-1]);
22        }
23        int result=0;
24        for (int i = 0; i < size; i++)
25        {
26            result+=num[i];
27        }
28    }
29 };

```

Your previous code was restored from your local storage. [Reset to default](#)

Problems Pick One 135/1884 Console Contribute i Run Code ^ Submit

Gas station

```

int canCompleteCircuit(vector<int>& A, vector<int>& B) {
    int sumGas = 0, sumCost = 0, start = 0, tank = 0;
    for(int i=0; i<A.size(); i++) {
        sumGas += A[i];
        sumCost += B[i];
        tank += A[i] - B[i];
        if(tank < 0) {
            start = i+1;
            tank = 0;
        }
    }

    if(sumGas < sumCost) return -1;
    else return start;
}

```

Double Switches on IV

```
int minFlips(string s) {
    int n=s.length();
    string ss="";
    for(int i=0;i<n;i++)
        ss+='0';
    int cnt=0,temp=0;//temp store kr rha h ki aage vale
    //saare kya hnge initially sb 0 h islie 0
    for(int i=0;i<n;i++)
    {
        if(temp==0&&s[i]=='1')
        {
            cnt++;
            temp=1;
        }
        else if(temp==1&&s[i]=='0')
        {
            cnt++;
            temp=0;
        }
    }
    return cnt;
}
```

Largest Permutations after at most K Swaps

```
static bool compare(pair<int,int>a,pair<int,int>b)
{
    return (a.first>b.first);
}
void KswapPermutation(int a[], int n, int k)
{
    vector<pair<int,int>>arrPos(n);
    for(int i=0;i<n;i++)
    {
        arrPos[i]={a[i],i};
    }
    sort(arrPos.begin(),arrPos.end(),compare);
    for(int i=0;i<n&&k>0;i++)
    {
        if(a[i]!=arrPos[i].first)
        {
            int position=arrPos[i].second;
            int correct_idx_of_curr_element=n-a[i];
            swap(a[i],a[position]);
            arrPos[i].second=i;
            arrPos[correct_idx_of_curr_element].second=position;
            k--;
        }
    }
}
```

BACKTRACKING:

N-Queen Problem:

A screenshot of a Windows desktop showing a Microsoft Edge browser window. The title bar says "Printing all solutions in N-Queen". The address bar shows "sapphireengine.com". The page content displays C++ code for solving the N-Queen problem. The code uses backtracking to place queens on an NxN board. It includes a helper function `issafe` to check if a queen can be placed at a given position without attacking others. The main function `solveNQUtil` tries to place queens column by column, using a vector `v` to keep track of valid rows for each column. If a solution is found, it's printed. If no solution is found, it backtracks. The code is well-commented and follows standard C++ conventions.

```
1 bool issafe(vector<vector<int>> &board,
2             int row, int col)
3 {
4     int i, j;
5     int N = board.size();
6     for (i = 0; i < col; i++)
7         if (board[row][i])
8             return false;
9     for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
10        if (board[i][j])
11            return false;
12     for (i = row, j = col; j >= 0 && i < N; i++, j--)
13        if (board[i][j])
14            return false;
15     return true;
16 }
17 bool solveNQUtil(vector<vector<int>> &board, int col)
18 {
19     int N = board.size();
20     if (col == N) {
21         vector<int> v;
22         for (int i = 0; i < N; i++) {
23             for (int j = 0; j < N; j++) {
24                 if (board[i][j] == 1)
25                     v.push_back(j + 1);
26             }
27         }
28         result.push_back(v);
29         return true;
30     }
31     bool res = false;
32     for (int i = 0; i < N; i++) {
33         if (issafe(board, i, col)) {
34             board[i][col] = 1;
35             res = solveNQUtil(board, col + 1) || res;
36             board[i][col] = 0; // BACKTRACK
37         }
38     }
39     return res;
40 }
```

Remove Invalid Parentheses:

tc: O(n^n)

sbse phle yh dekhlo stack ki help se ki minimum kitne remove krne h balanced k liye ab loop m iterate kro ki agar ith htado th dekho ki baat bnegi ki nhi agar ban jati h th shi h

```

1 vector<string> removeInvalidParentheses(string s)
2 {
3     int getmin(string s)
4     {
5         stack<char> st;
6         int ans_size();
7         for(int i=0;i<s.size();i++)
8         {
9             if(s[i]== '(')
10                 st.push(s[i]);
11             else if(s[i]== ')')
12             {
13                 if(st.size()==0)
14                     st.push(')');
15                 else if(st.top()=='(')
16                     st.pop();
17                 else if(st.top()=='')
18                     st.push(')');
19             }
20         }
21         return st.size();
22     }
23     void solve(string s,int min)
24     {
25         if(min==0)
26         {
27             int minnow=getmin(s);
28             if(minnow==0)
29             {
30                 if(std::find(ans.begin(),ans.end(),s)==ans.end())
31                     ans.push_back(s);
32             }
33         }
34         for(int i=0;i<s.size();i++)
35         {
36             string lefts=s.substr(0,i);
37             string rights=s.substr(i+1);
38             string news=lefts+rights;
39             solve(news,min-1);
40         }
41     }
42     vector<string> removeInvalidParentheses(string s) {
43         int min=getmin(s);
44         solve(s,min);
45         return ans;
46     }

```

Sudoku Solver:

tc: O($9^{m \times n}$)

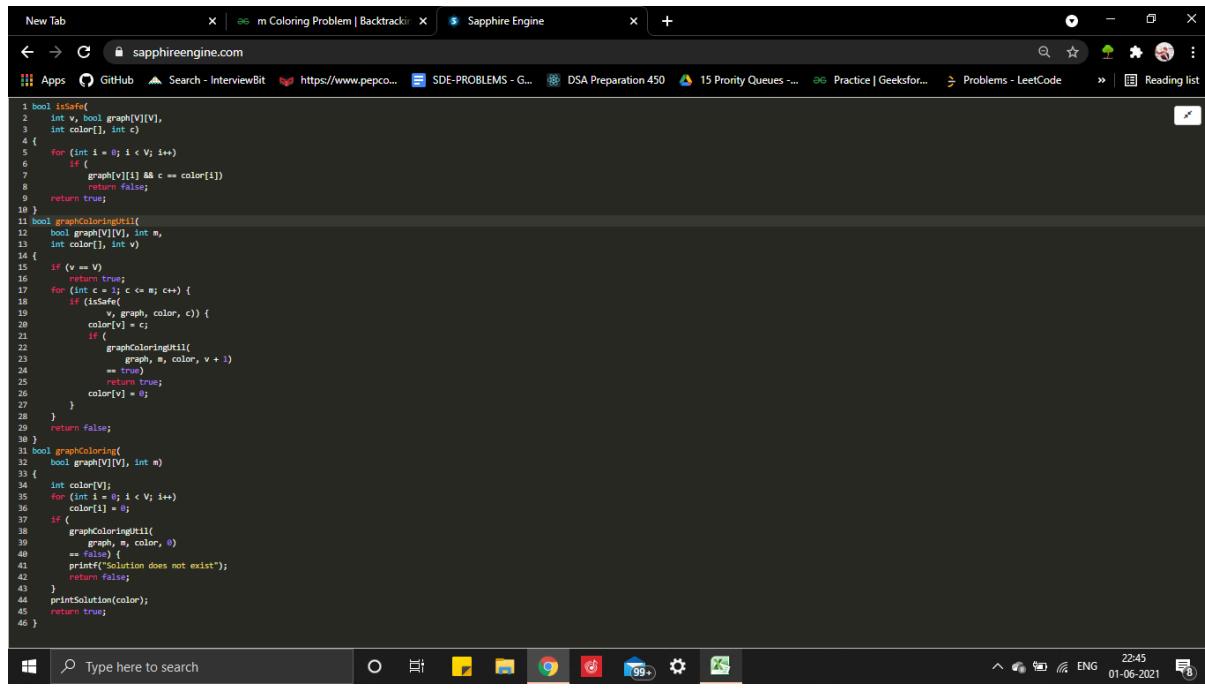
```

1 bool isvalid(vector<vector<char>>&board,int row,int col)
2 {
3     for(int j=0;j<9;j++)
4     {
5         if(board[row][j]=='.')
6             return false;
7     }
8     for(int i=0;i<9;i++)
9     {
10        if(board[i][col]=='.')
11            return false;
12    }
13    int num=board[row][col];
14    int rowi=row;
15    int coli=col;
16    for(int i=0;i<9;i++)
17    {
18        for(int j=0;j<9;j++)
19        {
20            if(board[i+rowi][j+coli]=='.')
21                return false;
22        }
23    }
24    return true;
25 }
26 bool allassigned=false;
27 void solve(vector<vector<char>>&board,int row,int col)
28 {
29     if(row==9)
30     {
31         allassigned=true;
32     }
33     else
34     {
35         if(col==9)
36         {
37             ni=row+1;
38             nj=0;
39         }
40         else
41         {
42             ni=row;
43             nj=col+1;
44         }
45         if(board[ni][nj]=='.')
46             solve(board,nj,0);
47         else
48             for(int num=1;num<10;num++)
49             {
50                 if(isvalid(board,num,ni,nj))
51                 {
52                     board[ni][nj]=num+'0';
53                     solve(board,ni,nj);
54                     if(allassigned)
55                         return;
56                     board[ni][nj]='.';
57                 }
58             }
59     }
60 }

```

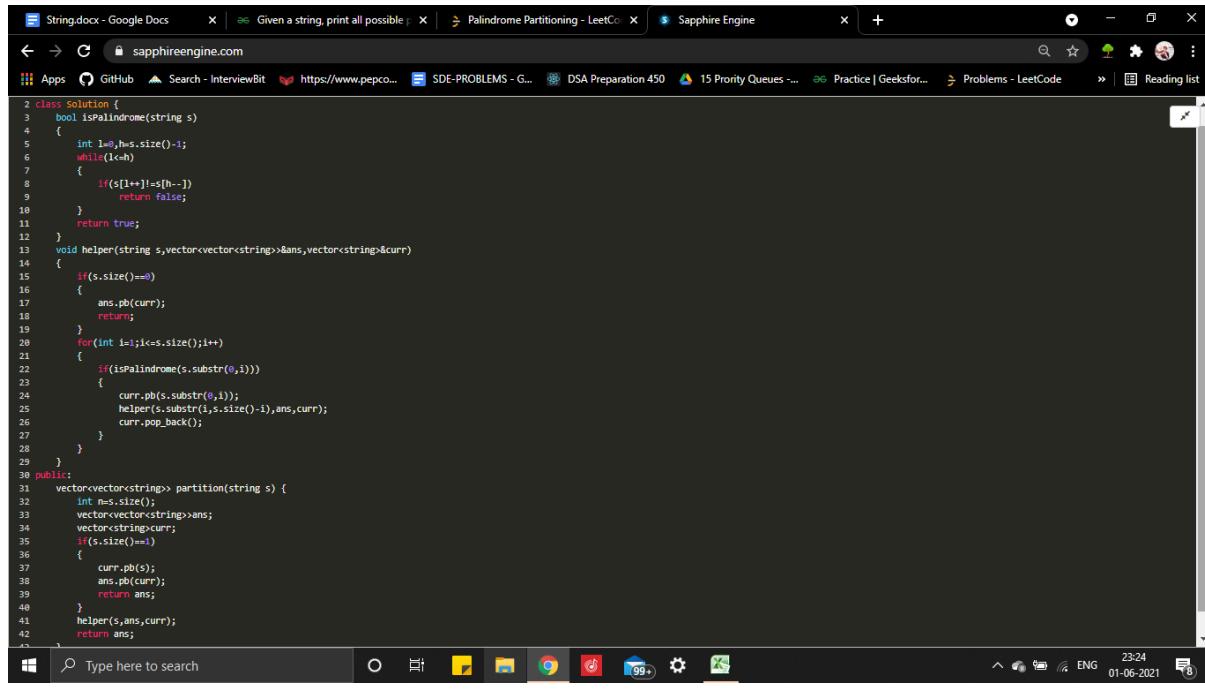
M Coloring Problem:

har ek possible color check krlo



```
1 bool isSafe(
2     int v, bool graph[V][V],
3     int color[], int c)
4 {
5     for (int i = 0; i < V; i++)
6         if (
7             graph[v][i] && c == color[i])
8                 return false;
9     return true;
10 }
11 bool graphColoringHilf(
12     bool graph[V][V], int m,
13     int color[], int v)
14 {
15     if (v == V)
16         return true;
17     for (int c = 1; c <= m; c++) {
18         if (isSafe(
19             v, graph, color, c)) {
20             color[v] = c;
21             if (
22                 graphColoringHilf(
23                     graph, m, color, v + 1)
24                     == true)
25                 return true;
26             color[v] = 0;
27         }
28     }
29     return false;
30 }
31 bool graphColoring(
32     bool graph[V][V], int m)
33 {
34     int color[V];
35     for (int i = 0; i < V; i++)
36         color[i] = 0;
37     if (
38         graphColoringHilf(
39             graph, m, color, 0)
40             == false) {
41         printf("solution does not exist");
42         return false;
43     }
44     printSolution(color);
45     return true;
46 }
```

Print all palindromic partitions of a string:



```
2 class Solution {
3     bool isPalindrome(string s)
4     {
5         int l=0,h=s.size()-1;
6         while(l<h)
7         {
8             if(s[l++]!=s[h--])
9                 return false;
10        }
11        return true;
12    }
13    void helper(string s,vector<vector<string>>&ans,vector<string>&curr)
14    {
15        if(s.size()==0)
16        {
17            ans.pb(curr);
18            return;
19        }
20        for(int i=1;i<=s.size();i++)
21        {
22            if(isPalindrome(s.substr(0,i)))
23            {
24                curr.pb(s.substr(0,i));
25                helper(s.substr(i,s.size()-i),ans,curr);
26                curr.pop_back();
27            }
28        }
29    }
30 public:
31    vector<vector<string>> partition(string s) {
32        int n=s.size();
33        vector<vector<string>>ans;
34        vector<string>curr;
35        if(s.size()==1)
36        {
37            curr.pb(s);
38            ans.pb(curr);
39            return ans;
40        }
41        helper(s,ans,curr);
42        return ans;
43    }
}
```

Tug Of War:

```

1 int mins=INT_MAX;
2 void solve(vector<ll>&v1,vector<ll>&v2,ll sv1,ll sv2,vector<ll>&v,ll idx)
3 {
4     if(idx==v.size())//base case
5     {
6         ll delta=abs(sv2-sv1);
7         if(delta<mins)
8         {
9             mins=delta;
10            ans.clear();
11            ans.pb(v1);
12            ans.pb(v2);
13        }
14    }
15    if(v1.size()<(v.size()+1)/2)
16    {
17        v1.pb(v[idx]);
18        solve(v1,v2,sv1+v[idx],sv2,v,idx+1);
19        v1.pop_back();
20    }
21    if(v2.size()<(v.size()+1)/2)
22    {
23        v2.pb(v[idx]);
24        solve(v1,v2,sv1,sv2+v[idx],v,idx+1);
25        v2.pop_back();
26    }
27 }

```

Partition to K Equal Sum Subsets:

```

1 bool helper(vector<int>& nums, int k, bool visited[], int target, int curr, int nextIndexToCheck)
2 {
3     if(k==0)
4         return true;
5     if(curr==target)
6     {
7         return helper(nums,k-1,visited,target,0,0);
8     }
9     for(int i=nextIndexToCheck;i<nums.size();i++)
10    {
11        if(!visited[i]&&nums[i]+curr<=target)
12        {
13            visited[i]=true;
14            if(helper(nums,k,visited,target,curr+nums[i],i))
15                return true;
16            visited[i]=false;
17        }
18    }
19    return false;
20 }
21 class Solution {
22 public:
23     bool canPartitionKSubsets(vector<int>& nums, int k) {
24         int n=nums.size();
25         int maxs=1;
26         int sum=0;
27         for(int i=0;i<n;i++)
28         {
29             sum+=nums[i];
30             maxs=max(maxs,nums[i]);
31         }
32         if(sum<k||maxs>k)
33             return false;
34         if(maxs>sum/k)
35             return false;
36         bool visited[n];
37         memset(visited,false,sizeof(visited));
38         return helper(nums,k,visited,sum/k,0,0);
39     }
40 };

```

Type here to search 23:54 01-06-2021 ENG

Combination Sum:

39. Combination Sum

Medium 6198 154 Add to List Share

Given an array of **distinct** integers candidates and a target integer target, return a list of all **unique combinations** of candidates where the chosen numbers sum to target. You may return the combinations in **any order**.

The **same** number may be chosen from candidates an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

It is **guaranteed** that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

Example 1:

```

Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.

```

```

int n=0;
vector<vector<int>>ans;
#define pb push_back
void solve(vector<int>&v,int curr_sum,int idx,int target,vector<int>&set)
{
    if(curr_sum==target)
    {
        if(std::find(ans.begin(),ans.end(),set)==ans.end())
        ans.pb(set);
        return;
    }
    if(curr_sum>target||idx==v.size())
    return;
    set.pb(v[idx]);
    solve(v,curr_sum+v[idx],idx,target,set);
    set.pop_back();
    solve(v,curr_sum,idx+1,target,set);
}
class Solution {
public:
    vector<vector<int>> combinationSum(vector<int>&v, int target) {
        int curr_sum=0;
        vector<int>v;
        set<vector<int>>set;
        set.clear();
        sort(v.begin(),v.end());
        solve(v,curr_sum,0,target,set);
        return ans;
    }
};

```

Your previous code was restored from your local storage. Reset to default

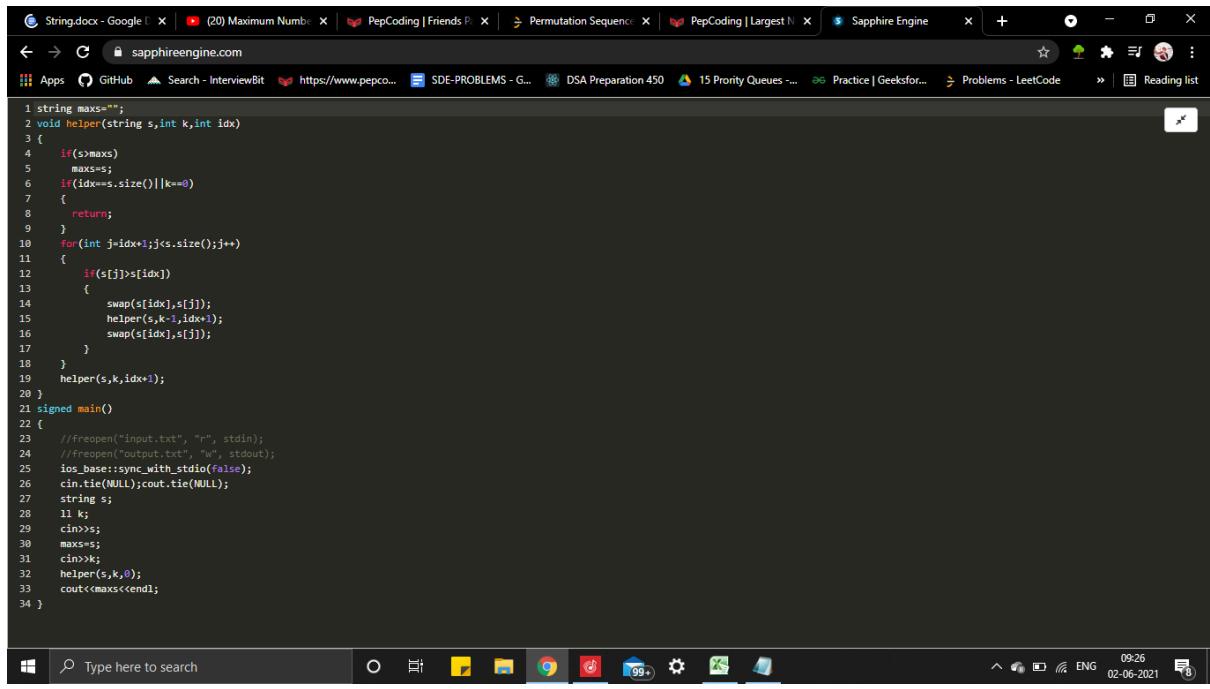
Problems Pick One < Prev 39/1883 Next > Console Contribute i Run Code Submit

Type here to search 00:03 02-06-2021 ENG

Combination Sums II

```
vector<vector<int>>ans;
#define pb push_back
void solve(vector<int>v,int curr_sum,int idx,int target,vector<int>&set)
{
    if(curr_sum==target)
    {
        //if(std::find(ans.begin(),ans.end(),set)==ans.end())
        ans.pb(set);
        return;
    }
    if(curr_sum>target||idx==v.size())
        return;
    set.pb(v[idx]);
    solve(v,curr_sum+v[idx],idx+1,target,set);
    set.pop_back();
    while(idx+1<v.size()&&v[idx]==v[idx+1])
        idx++;
    solve(v,curr_sum,idx+1,target,set);
}
vector<vector<int>> combinationSum2(vector<int>& v, int target) {
    int curr_sum=0;
    ans.clear();
    vector<int>set;
    set.clear();
    sort(v.begin(),v.end());
    solve(v,curr_sum,0,target,set);
    //sort(ans.begin(),ans.end());
    return ans;
}
```

Largest Number Possible After At Most K Swaps:

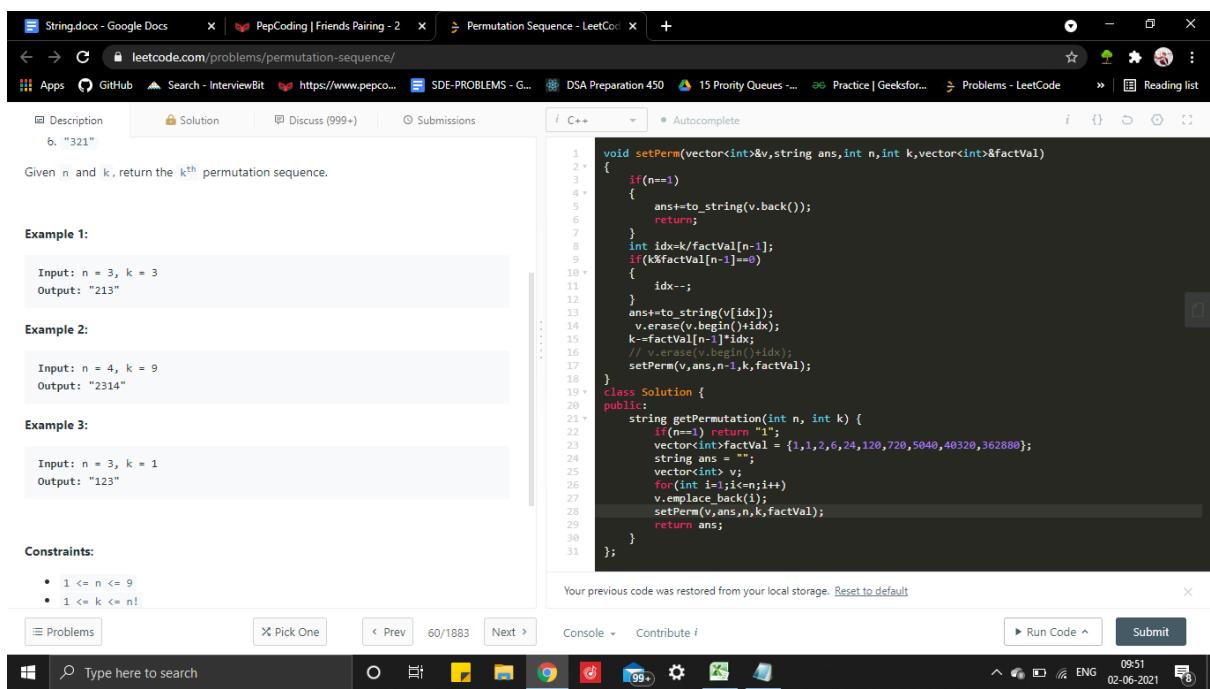


```

1 string maxs="";
2 void helper(string s,int k,int idx)
3 {
4     if(s>maxs)
5         maxs=s;
6     if(idx==s.size()||k==0)
7     {
8         return;
9     }
10    for(int j=idx+1;j<s.size();j++)
11    {
12        if(s[j]>s[idx])
13        {
14            swap(s[idx],s[j]);
15            helper(s,k-1,idx+1);
16            swap(s[idx],s[j]);
17        }
18    }
19    helper(s,k,idx+1);
20 }
21 signed main()
22 {
23     //freopen("input.txt", "r", stdin);
24     //freopen("output.txt", "w", stdout);
25     ios_base::sync_with_stdio(false);
26     cin.tie(NULL);cout.tie(NULL);
27     string s;
28     ll k;
29     cin>>s;
30     maxs=s;
31     cin>>k;
32     helper(s,k,0);
33     cout<<maxs<<endl;
34 }

```

Permutation Sequence



Problem Statement:

Given n and k , return the k^{th} permutation sequence.

Example 1:

Input: $n = 3$, $k = 3$
Output: "213"

Example 2:

Input: $n = 4$, $k = 9$
Output: "2314"

Example 3:

Input: $n = 3$, $k = 1$
Output: "123"

Constraints:

- $1 \leq n \leq 9$
- $1 \leq k \leq n!$

```

void setPerm(vector<int>&v,string ans,int n,int k,vector<int>&factVal)
{
    if(n==1)
    {
        ans+=to_string(v.back());
        return;
    }
    int idx=k/factVal[n-1];
    if(k%factVal[n-1]==0)
    {
        idx--;
    }
    ans+=to_string(v[idx]);
    v.erase(v.begin()+idx);
    k-=factVal[n-1]*idx;
    // v.erase(v.begin()+idx);
    setPerm(v,ans,n-1,k,factVal);
}
class Solution {
public:
    string getPermutation(int n, int k) {
        if(n==1) return "1";
        vector<int>factVal = {1,1,2,6,24,120,720,5040,40320,362880};
        string ans = "";
        vector<int> v;
        for(int i=1;i<n;i++)
        v.emplace_back(i);
        setPerm(v,ans,n,k,factVal);
        return ans;
    }
};

Your previous code was restored from your local storage. Reset to default

```

Run Code Submit

String Permutation

```
vector<vector<int>>ans;
int n=0;
void solve(vector<int>nums, string curr,int idx)
{
    if(idx==n)
    {
        if(curr.size()==n)
        {
            vector<int>temp;
            temp.clear();
            for(int i=0;i<n;i++)
                temp.push_back(curr[i]-'0');
            ans.push_back(temp);
        }
        return;
    }
    char x=nums[idx]+'
```

char z=nums[0]+'

```
+ '0';
string curr="";
curr+=z;
solve(nums,curr,1);
return ans;
}
class Solution {
public:
    vector<vector<int>> permute(vector<int>& nums) {
        ans.clear();
        n=nums.size();
        char z=nums[0]+'
```

+ '0';

```
string curr="";
curr+=z;
solve(nums,curr,1);
return ans;
};
```

Friends Pairing

```
1 void helper(11 n,11 i,bool used[],string ans)
2 {
3     if(i>n)
4     {
5         cout<<ans<<endl;
6         return;
7     }
8     if(used[i]==true)
9     helper(n,i+1,used,ans);
10    else
11    {
12        used[i]=true;
13        helper(n,i+1,used,ans+(to_string(i)+" "+));
14        for(11 j=i+1;j<=n;j++)
15        {
16            if(used[j]==false)
17            {
18                used[j]=true;
19                helper(n,i+1,used,ans+(to_string(i)+" "+to_string(j)+" "+));
20                used[j]=false;
21            }
22        }
23        used[i]=false;
24    }
25 }
26 signed main()
27 {
28     11 n;
29     cin>>n;
30     bool used[n+1]={false};
31     11 i=1;
32     helper(n,i,used,"");
33 }
```

Generate Parenthesis

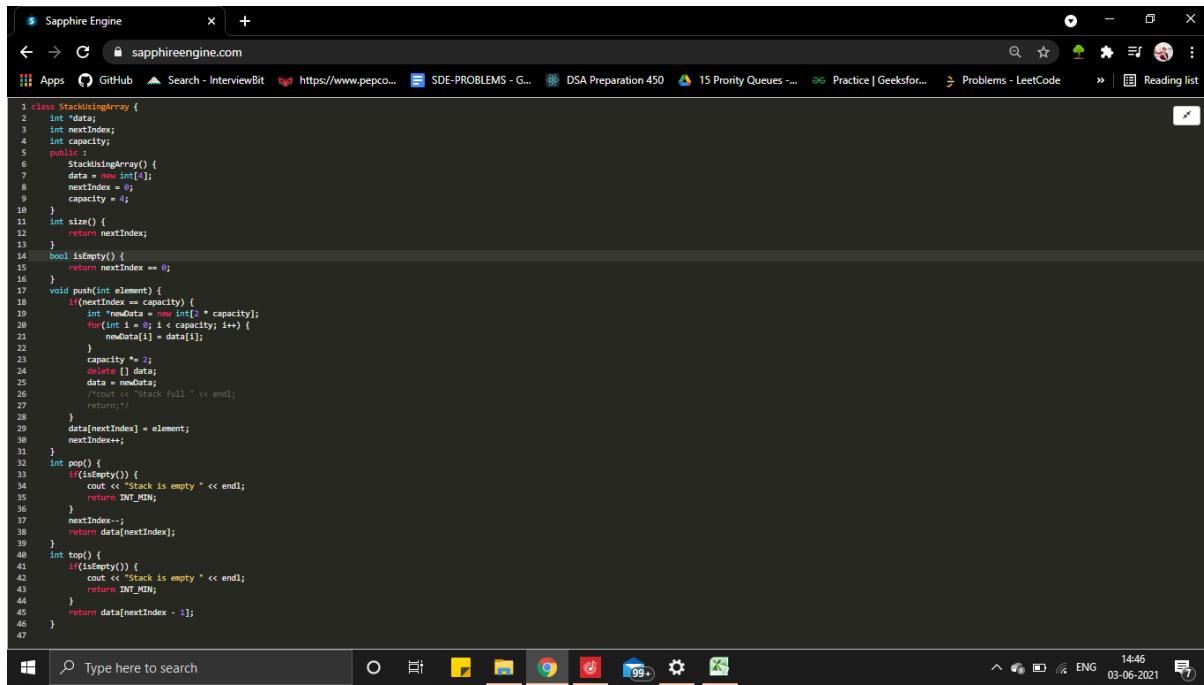
```
bool isValid(string s)
{
    stack<char>st;
    for(int i=0;i<s.size();i++)
    {
        if(s[i]== '(')
            st.push('(');
        else
        {
            if(st.size()>0&&st.top()=='(')
                st.pop();
            else
                return false;
        }
    }
    return (st.size()==0);
}
void helper(int n,vector<string>&ans,string temp)
{
    if(n==0)
    {
        if(isValid(temp))
            ans.pb(temp);
        return;
    }

    helper(n-1,ans,temp+"(");
    helper(n-1,ans,temp+")");
    helper(n-1,ans,temp+"("));
    helper(n-1,ans,temp+"))");
}
vector<string> generateParenthesis(int n) {
    if(n==1)
        return {"()"};
    vector<string>ans;
    string temp="";
    helper(n,ans,temp);
    return ans;
}
```

STACKS AND QUEUES:

(AGAR PREFIX M HAI PHLE TO RIGHT TO LEFT HI TRAVERSE KRO AUR AGAR PHLE POSTFIX HAI TH LEFT TO RIGHT)

Implement Stack from Scratch:



The screenshot shows a Windows desktop environment. In the center is a Microsoft Edge browser window titled "Sapphire Engine". The address bar shows the URL "sapphireengine.com". The main content area of the browser displays the following C++ code:

```
1 class StackUsingArray {
2     int *data;
3     int nextIndex;
4     int capacity;
5 public:
6     StackUsingArray() {
7         data = new int[4];
8         nextIndex = 0;
9         capacity = 4;
10    }
11    int size() {
12        return nextIndex;
13    }
14    bool isEmpty() {
15        return nextIndex == 0;
16    }
17    void push(int element) {
18        if(nextIndex == capacity) {
19            int *newData = new int[2 * capacity];
20            for(int i = 0; i < capacity; i++) {
21                newData[i] = data[i];
22            }
23            capacity *= 2;
24            delete [] data;
25            data = newData;
26            //cout << "Stack full " << endl;
27        }
28        data[nextIndex] = element;
29        nextIndex++;
30    }
31    int pop() {
32        if(isEmpty()) {
33            cout << "Stack is empty " << endl;
34            return INT_MIN;
35        }
36        nextIndex--;
37        return data[nextIndex];
38    }
39    int top() {
40        if(isEmpty()) {
41            cout << "Stack is empty " << endl;
42            return INT_MIN;
43        }
44        return data[nextIndex - 1];
45    }
46 }
```

The browser window has a dark theme. At the bottom of the screen, the taskbar is visible with icons for File Explorer, Google Chrome, Task View, and others. The system tray shows the date and time as "03-06-2021 14:46".

Implementation Overview from Scratch

```
class QueueUsingArray {
    T *data;
    int nextIndex;
    int firstIndex;
    int size;
    int capacity;
public :
    QueueUsingArray(int s) {
        data = new T[s];
        nextIndex = 0;
        firstIndex = -1;
        size = 0;
        capacity = s;
    }
    int getSize() {
        return size;
    }
    bool isEmpty() {
        return size == 0;
    }
    void enqueue(T element) {
        if(size == capacity) {
            cout << "Queue Full ! " << endl;
            return;
        }
        data[nextIndex] = element;
        nextIndex = (nextIndex + 1) % capacity ;
        if(firstIndex == -1) {
            firstIndex = 0;
        }
        size++;
    }

    T front() {
        if(isEmpty()) {
            cout << "Queue is empty ! " << endl;
            return 0;
        }
        return data[firstIndex];
    }
    T dequeue() {
        if(isEmpty()) {
            cout << "Queue is empty ! " << endl;
            return 0;
        }
        T ans = data[firstIndex];
        firstIndex = (firstIndex + 1) % capacity;
        size--;
        if(size == 0) {
            firstIndex = -1;
            nextIndex = 0;
        }
        return ans;
    }
}
```

Implement two stacks in an array

Method 1 (Divide the space in two halves)

A simple way to implement two stacks is to divide the array in two halves and assign the half space to two stacks, i.e., use arr[0] to arr[n/2] for stack1, and arr[(n/2) + 1] to arr[n-1] for stack2 where arr[] is the array to be used to implement two stacks and size of array be n.

The problem with this method is inefficient use of array space. A stack push operation may result in stack overflow even if there is space available in arr[]. For example, say the array size is 6 and we push 3 elements to stack1 and do not push anything to second stack2. When we push 4th element to stack1, there will be overflow even if we have space for 3 more elements in array.

Method 2 (A space efficient implementation)

This method efficiently utilizes the available space. It doesn't cause an overflow if there is space available in arr[]. The idea is to start two stacks from two extreme corners of arr[]. stack1 starts from the leftmost element, the first element in stack1 is pushed at index 0. The stack2 starts from the rightmost corner, the first element in stack2 is pushed at index (n-1). Both stacks grow (or shrink) in opposite direction. To check for overflow, all we need to check is for space between top elements of both stacks. This check is highlighted in the below code.

```

class twoStacks {
    int* arr;
    int size;
    int top1, top2;
public:
    twoStacks(int n) // constructor
    {
        size = n;arr = new int[n];
        top1 = -1;
        top2 = size;
    }
    void push1(int x)
    {
        if (top1 < top2 - 1) {
            top1++;arr[top1] = x;
        }
        else {
            cout << "Stack Overflow";exit(1);
        }
    }
    void push2(int x)
    {
        if (top1 < top2 - 1) {
            top2--; arr[top2] = x;
        }
        else {
            cout << "Stack Overflow"; exit(1);
        }
    }
    int pop1()
    {
        if (top1 >= 0) {
            int x = arr[top1];
            top1--;
            return x;
        }
        else {
            cout << "Stack UnderFlow"; exit(1);
        }
    }
    int pop2()
    {
        if (top2 < size) {
            int x = arr[top2];
            top2++;return x;
        }
        else {
            cout << "Stack UnderFlow";exit(1);
        }
    }
};

```

Stack that supports getMin() in O(1) time and O(1) extra space:

Agar curr>min h th ese hi push krdo stack m vrna stack m push hga
 $2 * curr - min$ element aur min=curr kro ese hi jb pop kro th agar
 $s.top() > min$ h th ese hi pop krdo vrna min= $2 * min - s.top()$ krdo

```

1 void pop()
2 {
3     if (s.empty())
4     {
5         cout << "Stack is empty\n";
6         return;
7     }
8     cout << "Top Most Element Removed: ";
9     int t = s.top();
10    s.pop();
11    if (t < minEle)
12    {
13        cout << minEle << "\n";
14        minEle = 2*minEle - t;
15    }
16    else
17        cout << t << "\n";
18 }
19 void push(int x)
20 {
21     if (s.empty())
22     {
23         minEle = x;
24         s.push(x);
25         cout << "Number Inserted: " << x << "\n";
26         return;
27     }
28     if (x < minEle)
29     {
30         s.push(2*x - minEle);
31         minEle = x;
32     }
33     else
34         s.push(x);
35     cout << "Number Inserted: " << x << "\n";
36 }

```

Infix to Postfix:

Chapter 10 Infix to Postfix AlgorithmHindi

Algorithm

- Push '**(**' onto the STACK and add '**)**' to the end of Q
- Scan Q from left to right and repeat steps 3 to 6 for each element of Q until the STACK is empty.
- If an **operand** is encountered add it to P.
- If a **left parenthesis** is encountered, push it onto STACK.
- If an **operator** is encountered, then:
 - Repeatedly pop from STACK and add to P each operator which has the same precedence as or higher precedence than **O**
 - Add **O** to STACK
- If a **right parenthesis** is encountered, then
 - Repeatedly pop from the STACK and add to P each operator until a left parenthesis is encountered.
 - Remove the left parenthesis.
- Exit

6:09 / 16:55

Sarabjeet Singh Sir

Infix To Prefix :

infix ko phle reverse krdo fr jo new string aai ko postfix m krlo fr jo new string aai usko reverse krdo vhi ans h. jab frst tym reverse kroge th (' isko ') isme convert krna and vice versa.

The Celebrity Problem:

0 se n tk saare elements daldo stack m fr top 2 niaklo agar M[i][j]==1 th mtlb i j ko janta h th i celebrity nhi ho skta th j ko push krdo vps else i j ko nhi janta th j celebrity nhi ho skta th i ko vps push krdo jo last m bcha vh celebrity h ab bss jo e

```
1 int celebrity(vector<vector<int> & M, int n) {
2     stack<int>s;
3     for(int i=0;i<n;i++)
4         s.push(i);
5     while(s.size()>=2)
6     {
7         int i=s.top();
8         s.pop();
9         int j=s.top();
10        s.pop();
11        if(M[i][j]==1)
12        {
13            //i knows j =>i cant be celebrity
14            s.push(j);
15        }
16        else
17        {
18            //j is not known by i =>j cant be celebrity
19            s.push(i);
20        }
21    }
22    int pot=s.top();
23    for(int i=0;i<n;i++)
24    {
25        if(i!=pot)
26        {
27            if(M[i][pot]!=1||M[pot][i]!=0)
28                return -1;
29        }
30    }
31    return pot;
32 }
```

Evaluation of Postfix Expression:

```
1 int evaluatePostfix(char* exp)
2 {
3     struct Stack* stack = createStack(strlen(exp));
4     int i;
5     if (!stack) return -1;
6     for (i = 0; exp[i]; ++i)
7     {
8         if (isdigit(exp[i]))
9             push(stack, exp[i] - '0');
10        else
11        {
12            int val1 = pop(stack);
13            int val2 = pop(stack);
14            switch (exp[i])
15            {
16                case '+': push(stack, val2 + val1); break;
17                case '-': push(stack, val2 - val1); break;
18                case '*': push(stack, val2 * val1); break;
19                case '/': push(stack, val2/val1); break;
20            }
21        }
22    }
23    return pop(stack);
24 }
```

Implement a method to insert an element at its bottom without using any other data structure:

Use recursion phle jbtk empty nhi ho jato stack tbtk khaali krte jaao fr if(s.empty()) ho jae th stack m element push krdo fr baaki recursion jb return krega th khud push krti rhega nikle hue elements ko

Reverse a stack using recursion:

Phle reverse function ko call krlo fr uske andar insert at the bottom ko call karo.

```
1 char insert_at_bottom(char x)
2 {
3     if(st.size() == 0)
4         st.push(x);
5     else
6     {
7         char a = st.top();
8         st.pop();
9         insert_at_bottom(x);
10        st.push(a);
11    }
12 }
13 char reverse()
14 {
15     if(st.size()>0)
16     {
17         char x = st.top();
18         st.pop();
19         reverse();
20         insert_at_bottom(x);
21     }
22 }
```

SORT A STACK USING RECURSION:

```
1 #include <bits/stdc++.h>
2 #include <iostream>
3 using namespace std;
4 stack<int>s;
5 void sorted_insert(int x)
6 {
7     if(s.size()==0 || x>s.top())
8     {
9         s.push(x);
10        return;
11    }
12    int temp=s.pop();
13    sorted_insert(x);
14    s.push(temp);
15 }
16 void sortStack()
17 {
18     if(s.size()==0)
19         return;
20     int x=s.top();
21     s.pop();
22     sortStack();
23     sorted_insert(x);
24 }
```

Largest Rectangle in Histogram:

Ek baar next smaller index on right nikalo dusri baar next small index on left nikalo fr
ans hga ans=max(ans,a[i]*(nsr[i]-nsl[i]-1))

```
2 public:
3     int largestRectangleArea(vector<int>&a) {
4         stack<int>s;
5         int n=a.size();
6         int nsr[n],nsl[n];
7         memset(nsr,0,sizeof(nsr));
8         memset(nsl,0,sizeof(nsl));
9         nsr[n-1]=n;
10        s.push(n-1);
11        for(int i=n-2;i>=0;i--) {
12            while(s.size()>0&&a[s.top()]>=a[i])
13                s.pop();
14            if(s.size()==0)
15                nsr[i]=n;
16            else
17                nsr[i]=s.top();
18            s.push(i);
19        }
20        nsl[0]=1;
21        while(!s.empty())
22            s.pop();
23        s.push(0);
24        for(int i=1;i<n;i++)
25        {
26            while(s.size()>0&&a[s.top()]>=a[i])
27                s.pop();
28            if(s.size()==0)
29                nsl[i]=1;
30            else
31                nsl[i]=s.top();
32            s.push(i);
33        }
34        int ans=0;
35        for(int i=0;i<n;i++)
36        {
37            ans=max(ans,a[i]*(nsr[i]-nsl[i]-1));
38        }
39        return ans;
40    }
41 }
42 }
```

Longest Valid Parentheses:

```
public int longestValidParentheses(String s) {  
    int left = 0, right = 0, maxlen = 0;  
    for (int i = 0; i < s.length(); i++) {  
        if (s.charAt(i) == '(') {  
            left++;  
        } else {  
            right++;  
        }  
        if (left == right) {  
            maxlen = Math.max(maxlen, 2 * right);  
        } else if (right >= left) {  
            left = right = 0;  
        }  
    }  
    left = right = 0;  
    for (int i = s.length() - 1; i >= 0; i--) {  
        if (s.charAt(i) == '(') {  
            left++;  
        } else {  
            right++;  
        }  
        if (left == right) {  
            maxlen = Math.max(maxlen, 2 * left);  
        } else if (left >= right) {  
            left = right = 0;  
        }  
    }  
    return maxlen;  
}
```

The screenshot shows a browser window with multiple tabs open. The active tab is titled "Sapphire Engine" and contains a C++ code snippet for finding the length of the longest valid parentheses substring. The code uses a stack to keep track of indices. A note in the code indicates that if the stack is empty, it means there's no matching opening parenthesis for the current closing one. The code is annotated with comments explaining its logic.

```
1 int longestValidParentheses(string s) {
2     stack<int>st;
3     st.push(-1);
4     int ans=0;
5     for(int i=0;i<s.length();i++)
6     {
7         if(s[i]== '(')
8             st.push(i);
9         else
10        {
11            st.pop();
12            //ab agar st khaali nhi h th mtlb closing bracket k liye koi n koi open bracket th h pair krne vala
13            if(!st.empty())
14            {
15                //max islie lena h kyuki ek baar st empty ho gaya aur usme ab closing index vale bracket ka index push ho gaya h vhaa break aa jaega vhaa se
16                ans=max(ans,i-st.top());
17            }
18            else
19                st.push(i);
20        }
21    }
22    return ans;
23 }
```

Implement Stack using Deque:

push in stack=inser_last in deque pop in st=remove_last in dq enqueue q =insert_last in dq and dequeue in q=remove_first from dq.

Stack Permutations:

A stack permutation is a permutation of objects in the given input queue which is done by transferring elements from input queue to the output queue with the help of a stack and the built-in push and pop functions.

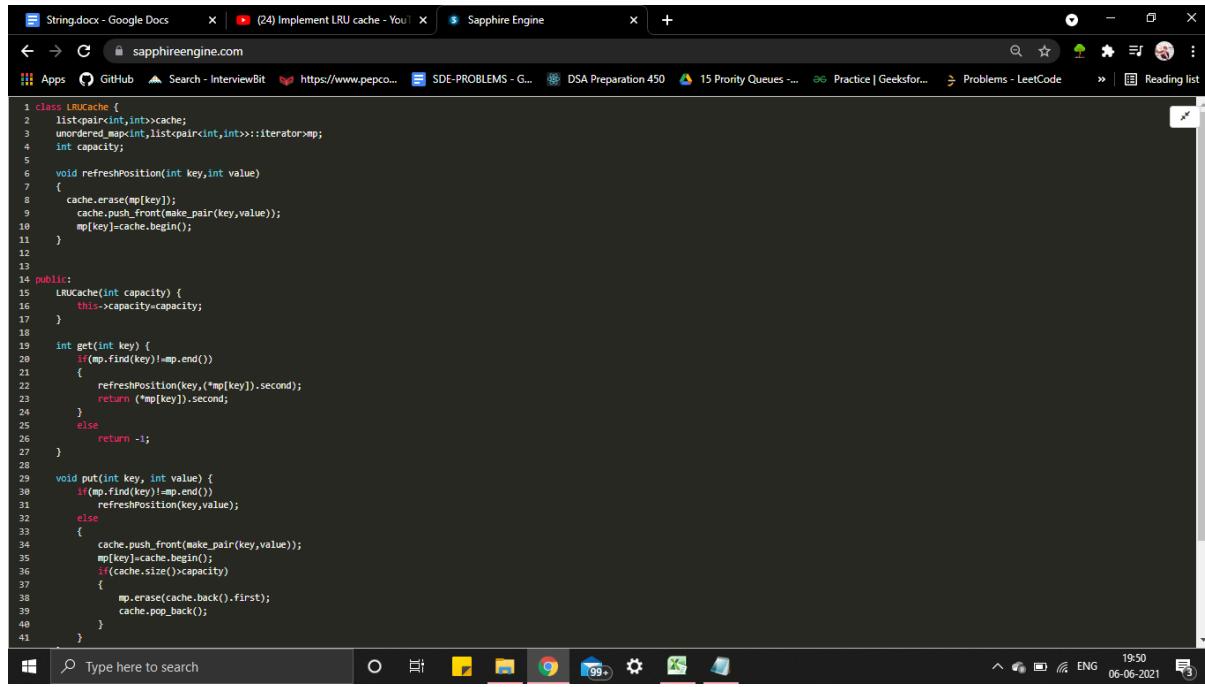
The well defined rules are:

1. Only dequeue from the input queue.
2. Use inbuilt push, pop functions in the single stack.
3. Stack and input queue must be empty at the end.
4. Only enqueue to the output queue.

yh dekho ki top p kya h oput queue k jbtk vh stack k top p nhi milta tab tk input queue se element uthao aur output queue m dalte rho jese hi vh mil gya usko output queue m daaldo fr ab hume ouput queue ka jo dusra element h usko stack k top p lana h fr.

```
bool checkStackPermutation(int ip[], int op[], int n)
{
    queue<int> input;
    for (int i=0;i<n;i++)
        input.push(ip[i]);
    queue<int> output;
    for (int i=0;i<n;i++)
        output.push(op[i]);
    stack <int> tempStack;
    while (!input.empty())
    {
        int ele = input.front();
        input.pop();
        if (ele == output.front())
        {
            output.pop();
            while (!tempStack.empty())
            {
                if (tempStack.top() == output.front())
                {
                    tempStack.pop();
                    output.pop();
                }
                else
                    break;
            }
        }
        else
            tempStack.push(ele);
    }
    return (input.empty()&&tempStack.empty());
}
```

LRU CACHE:



```
1 class LRUCache {
2     list<pair<int,int>>cache;
3     unordered_map<int,list<pair<int,int>>::iterator>mp;
4     int capacity;
5
6     void refreshPosition(int key,int value)
7     {
8         cache.erase(mp[key]);
9         cache.push_front(make_pair(key,value));
10        mp[key]=cache.begin();
11    }
12
13
14 public:
15     LRUCache(int capacity) {
16         this->capacity=capacity;
17     }
18
19     int get(int key) {
20         if(mp.find(key)!=mp.end())
21         {
22             refreshPosition(key,(mp[key]).second);
23             return (*mp[key]).second;
24         }
25         else
26             return -1;
27     }
28
29     void put(int key, int value) {
30         if(mp.find(key)!=mp.end())
31             refreshPosition(key,value);
32         else
33         {
34             cache.push_front(make_pair(key,value));
35             mp[key]=cache.begin();
36             if(cache.size()>capacity)
37             {
38                 mp.erase(cache.back().first);
39                 cache.pop_back();
40             }
41         }
42     }
43 }
```

Interleave the first half of the queue with second half:

stack m th frst half ese ki esa aa jaega (step 1-4) se

Following are the steps to solve the problem:

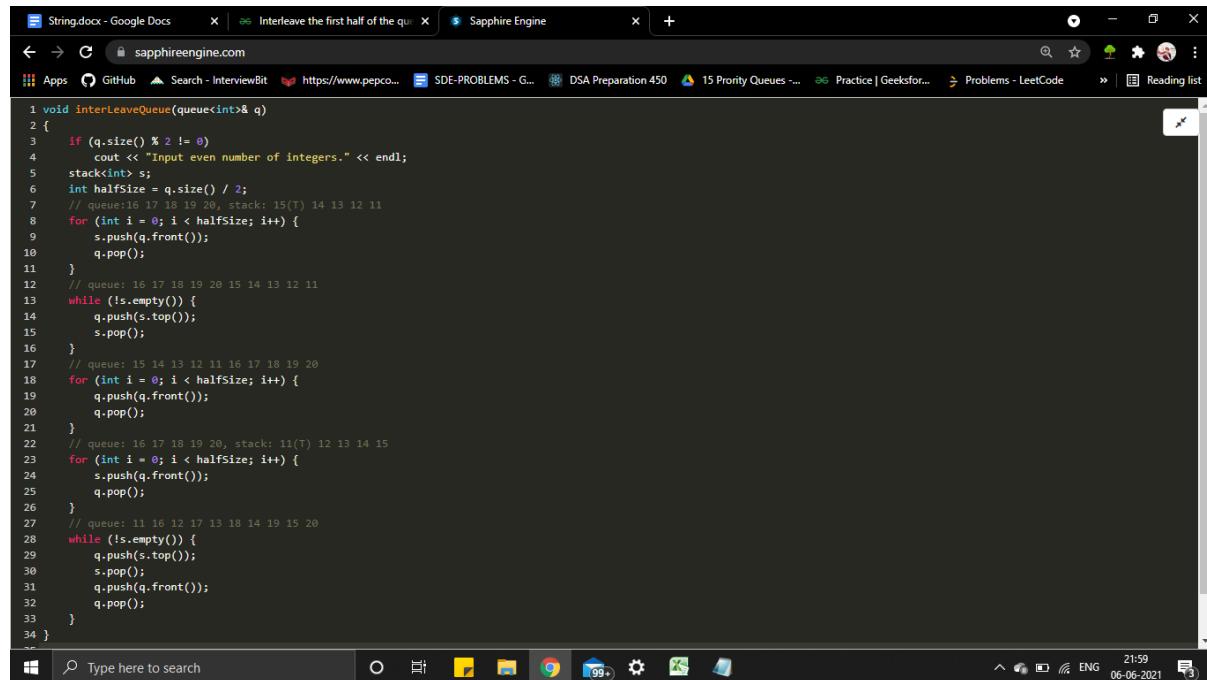
- 1.Push the first half elements of queue to stack.

2.Enqueue back the stack elements.

3.Dequeue the first half elements of the queue and enqueue them back.

4.Again push the first half elements into the stack.

5.Interleave the elements of queue and stack.



```
String.docx - Google Docs | Interleave the first half of the que... | Sapphire Engine
apps GitHub Search - InterviewBit https://www.pepcoding.com/ SDE-PROBLEMS - G... DSA Preparation 450 15 Priority Queues ... Practice | Geeksfor... Problems - LeetCode > | Reading list
void interLeaveQueue(queue<int>& q)
{
    if (q.size() % 2 != 0)
        cout << "Input even number of integers." << endl;
    stack<int> s;
    int halfSize = q.size() / 2;
    // queue: 16 17 18 19 20, stack: 15(T) 14 13 12 11
    for (int i = 0; i < halfSize; i++) {
        s.push(q.front());
        q.pop();
    }
    // queue: 16 17 18 19 20 15 14 13 12 11
    while (!s.empty()) {
        q.push(s.top());
        s.pop();
    }
    // queue: 15 14 13 12 11 16 17 18 19 20
    for (int i = 0; i < halfSize; i++) {
        q.push(q.front());
        q.pop();
    }
    // queue: 16 17 18 19 20, stack: 11(T) 12 13 14 15
    for (int i = 0; i < halfSize; i++) {
        s.push(q.front());
        q.pop();
    }
    // queue: 11 16 12 17 13 18 14 19 15 20
    while (!s.empty()) {
        q.push(s.top());
        s.pop();
        q.push(q.front());
        q.pop();
    }
}
```

Rotten Oranges:

```
1 int orangesRotting(vector<vector<int>>& grid) {
2     vector<int> dir={-1,0,1,0,-1}; //used for finding all 4 adjacent coordinates
3 
4     int m=grid.size();
5     int n=grid[0].size();
6 
7     queue<pair<int,int>> q;
8     int fresh=0; //to keep track of all fresh oranges left
9     for(int i=0;i<m;i++)
10    {
11        for(int j=0;j<n;j++)
12        {
13            if(grid[i][j]==2)
14                q.push({i,j});
15            if(grid[i][j]==1)
16                fresh++;
17        }
18    }
19    int ans=-1; //initialised to -1 since after each step we increment the time by 1 and initially all rotten oranges started at 0.
20    while(!q.empty())
21    {
22        int sz=q.size();
23        while(sz--)
24        {
25            pair<int,int> p=q.front();
26            q.pop();
27            for(int i=0;i<4;i++)
28            {
29                int r=p.first+dir[i];
30                int c=p.second+dir[i+1];
31                if(r<0 && c<0 && c>=m && r>=n && grid[r][c]==1)
32                {
33                    grid[r][c]=2;
34                    q.push({r,c});
35                    fresh--;
36                }
37            }
38            ans++; //Incremented after each minute passes
39        }
40    }
41    if(fresh>0) return -1; //if fresh>0 that means there are fresh oranges left
42    if(ans==1) return 0; //we initialised with -1, so if there were no oranges it'd take 0 mins.
43    return ans;
44 }
```

Distance of nearest cell having 0 in a binary matrix:

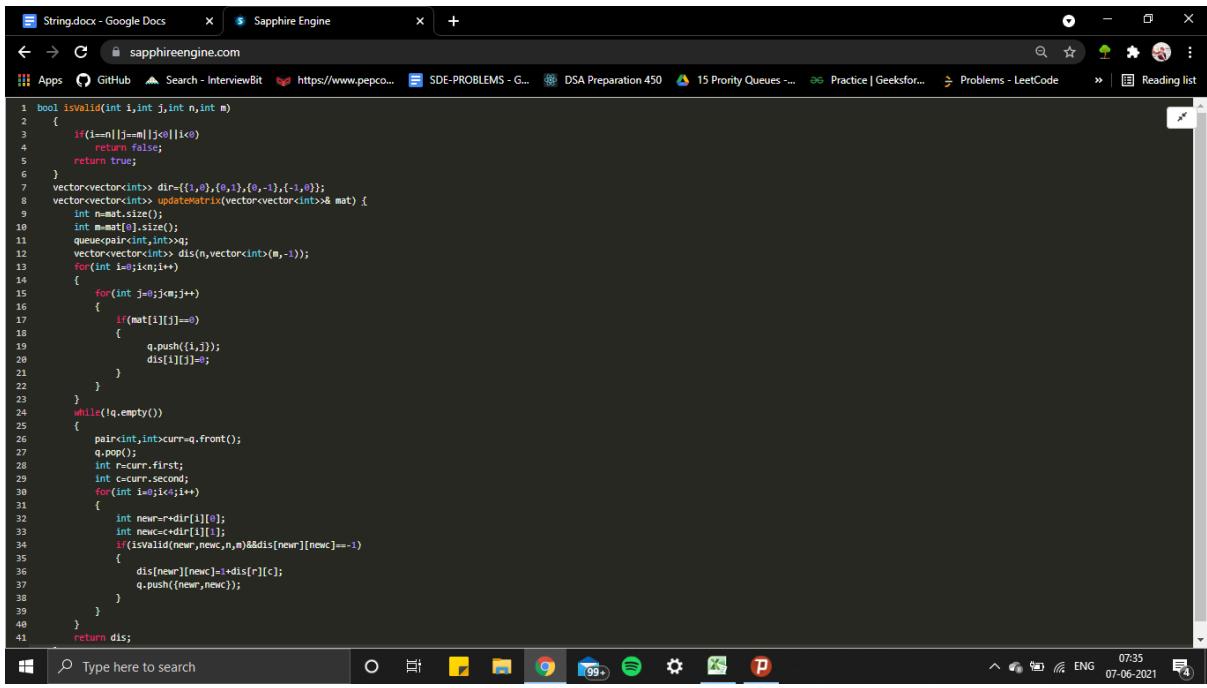
matrix:

sbse phle saare 0 queue m dalkr uska dist

update kro fr 0 ko queue se uthao uske next

ko update kro ese hi update krte rho +1 krke

dist ko



```
1 bool isValid(int i,int j,int n,int m)
2 {
3     if(i==n||j==m||j<0||i<0)
4         return false;
5     return true;
6 }
7 vector<vector<int>> dir={{1,0},{0,1},{0,-1},{-1,0}};
8 vector<vector<int>> updateMatrix(vector<vector<int>>& mat) {
9     int n=mat.size();
10    int m=mat[0].size();
11    queue<pair<int,int>>q;
12    vector<vector<int>> dis(n,vector<int>(m,-1));
13    for(int i=0;i<n;i++)
14    {
15        for(int j=0;j<m;j++)
16        {
17            if(mat[i][j]==0)
18            {
19                q.push({i,j});
20                dis[i][j]=0;
21            }
22        }
23    }
24    while(!q.empty())
25    {
26        pair<int,int> curr=q.front();
27        q.pop();
28        int r=curr.first;
29        int c=curr.second;
30        for(int i=0;i<4;i++)
31        {
32            int newr=r+dir[i][0];
33            int newc=c+dir[i][1];
34            if(isValid(newr,newc,n,m)&&dis[newr][newc]==-1)
35            {
36                dis[newr][newc]=dis[r][c];
37                q.push({newr,newc});
38            }
39        }
40    }
41    return dis;
42 }
```

Sliding Window Maximum:

The screenshot shows a LeetCode problem page for "Sliding Window Maximum" (Problem 239). The problem statement and example 1 are visible on the left, while the code editor on the right contains a C++ implementation using a deque.

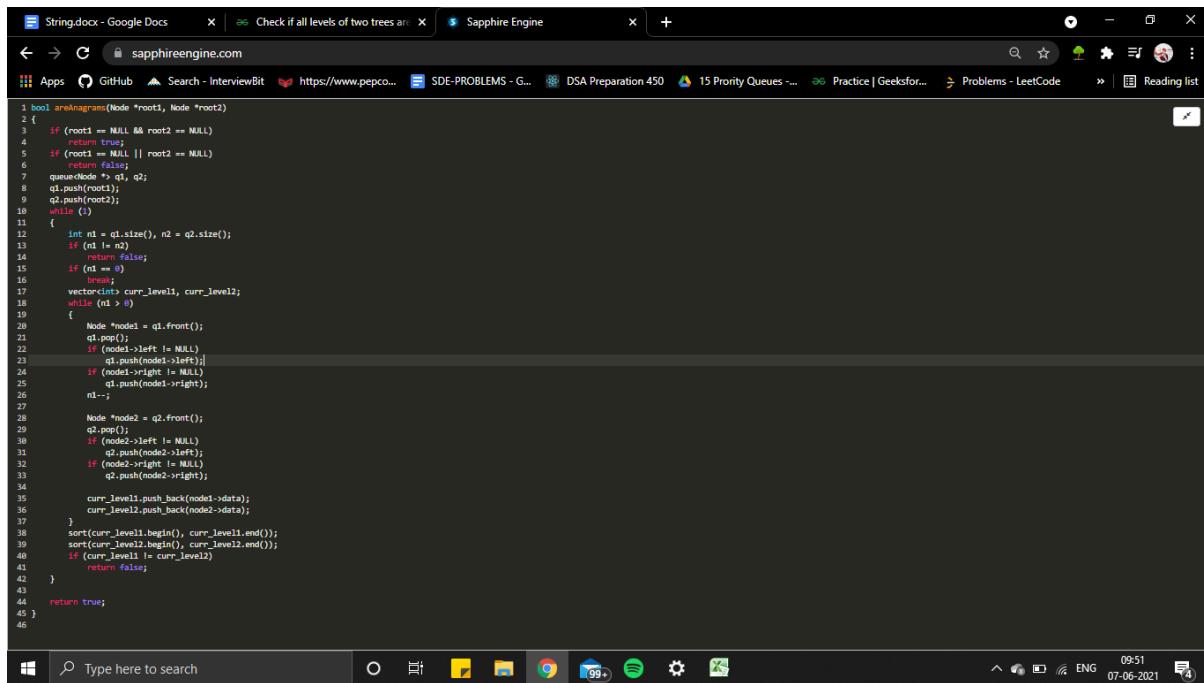
```
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        deque<int> dq(k);
        int i;
        for(i=0;i<k;i++)
        {
            while((dq.empty())&&nums[i]>=nums[dq.back()])
                dq.pop_back();
            dq.push_back(i);
        }
        vector<int> ans;
        for(;i<nums.size();i++)
        {
            ans.push_back(nums[dq.front()]);
            while((dq.empty())&&dq.front()<=i-k)
                dq.pop_front();
            while((dq.empty())&&nums[i]>=nums[dq.back()])
                dq.pop_back();
            dq.push_back(i);
        }
        ans.push_back(nums[dq.front()]);
        return ans;
    }
};
```

Longest Contiguous Subarray With Absolute Diff Less Than or Equal to Limit(Sliding window maximum and minimum)

```
public:
    int longestSubarray(vector<int>& nums, int limit) {
        //har subarray kaminimum aur maximum dekhna hai th sliding window maximum jesa hi ho gaya
        //islie dequeue use kia h
        int n=nums.size();
        int i=0,j=0,min,max,ans=0;
        deque<int>maxd;
        deque<int>mind;
        maxd.push_back(nums[0]);
        mind.push_back(nums[0]);
        while(j<nums.size())
        {
            maxs=maxd.front();
            mins=mind.front();
            if(maxs-mins<=limit)
            {
                j++;
                ans=max(ans,j-i);
                if(j<n)
                {
                    while(!mind.empty() and mind.back()>nums[j])
                        mind.pop_back();
                    mind.push_back(nums[j]);
                    while(!maxd.empty() and maxd.back()<nums[j])
                        maxd.pop_back();
                    maxd.push_back(nums[j]);
                }
            }
            else
            {
                if(nums[i]==mins)
                {
                    mind.pop_front();
                }
                if(nums[i]==maxs)
                {
                    maxd.pop_front();
                }
                i++;
            }
        }
        return ans;
    }
```

Check if all levels of two trees are anagrams or not:

1. Write a recursive program for level order traversal of a tree.
2. Traverse each level of both the trees one by one and store the result of traversals in 2 different vectors, one for each tree.
3. Sort both the vectors and compare them iteratively for each level, if they are same for each level then return true else return false.



```

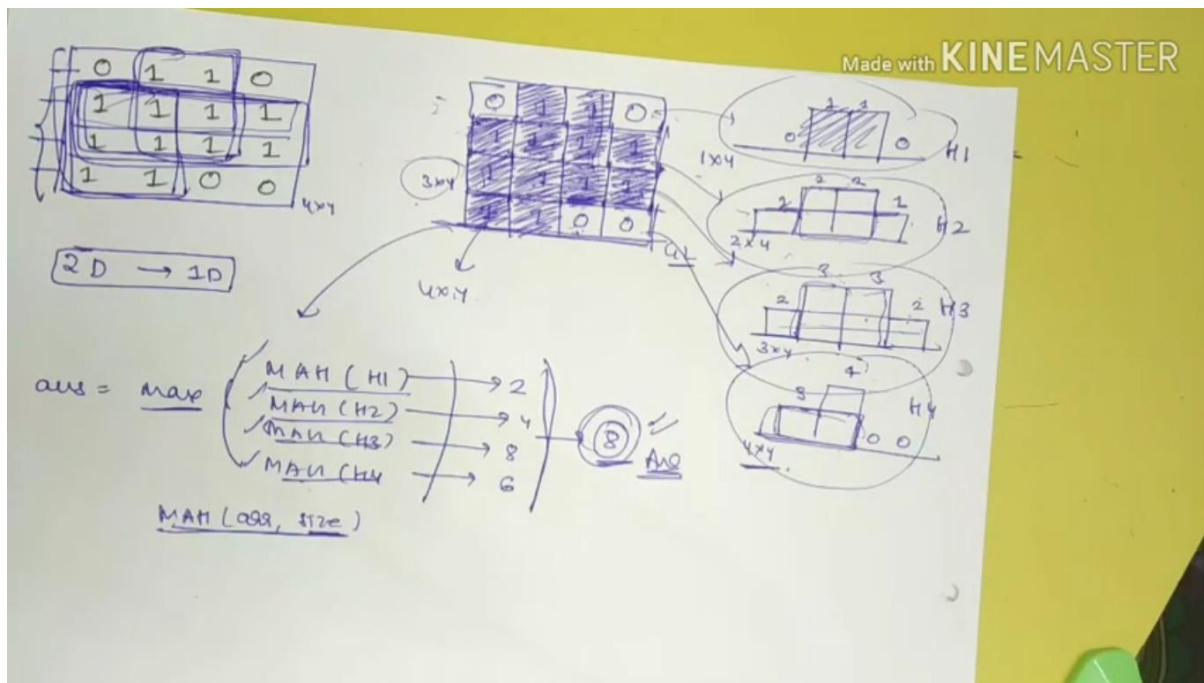
1 bool areNagograms(Node *root1, Node *root2)
2 {
3     if (root1 == NULL && root2 == NULL)
4         return true;
5     if (root1 == NULL || root2 == NULL)
6         return false;
7     queue<Node*> q1, q2;
8     q1.push(root1);
9     q2.push(root2);
10    while (!)
11    {
12        int n1 = q1.size(), n2 = q2.size();
13        if (n1 != n2)
14            return false;
15        if (n1 == 0)
16            break;
17        vector<int> curr_level1, curr_level2;
18        while (n1 > 0)
19        {
20            Node *node1 = q1.front();
21            q1.pop();
22            if (node1->left != NULL)
23                q1.push(node1->left);
24            if (node1->right != NULL)
25                q1.push(node1->right);
26            n1--;
27
28            Node *node2 = q2.front();
29            q2.pop();
30            if (node2->left != NULL)
31                q2.push(node2->left);
32            if (node2->right != NULL)
33                q2.push(node2->right);
34
35            curr_level1.push_back(node1->data);
36            curr_level2.push_back(node2->data);
37        }
38        sort(curr_level1.begin(), curr_level1.end());
39        sort(curr_level2.begin(), curr_level2.end());
40        if (curr_level1 != curr_level2)
41            return false;
42    }
43
44    return true;
45 }

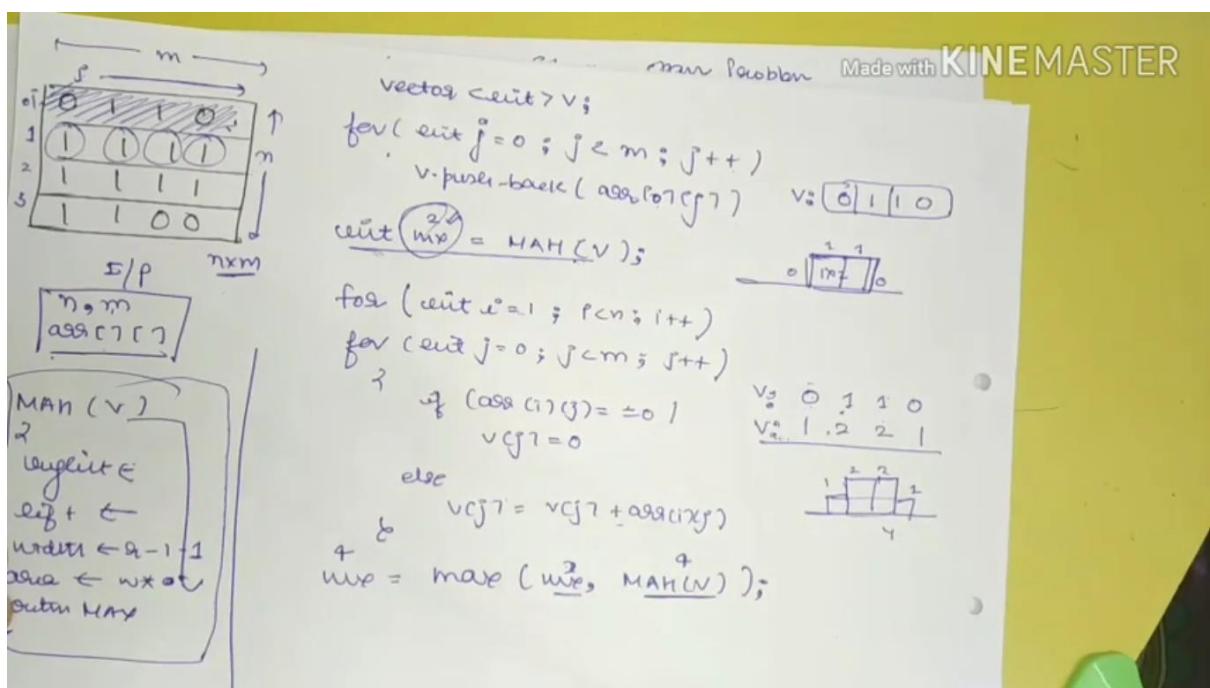
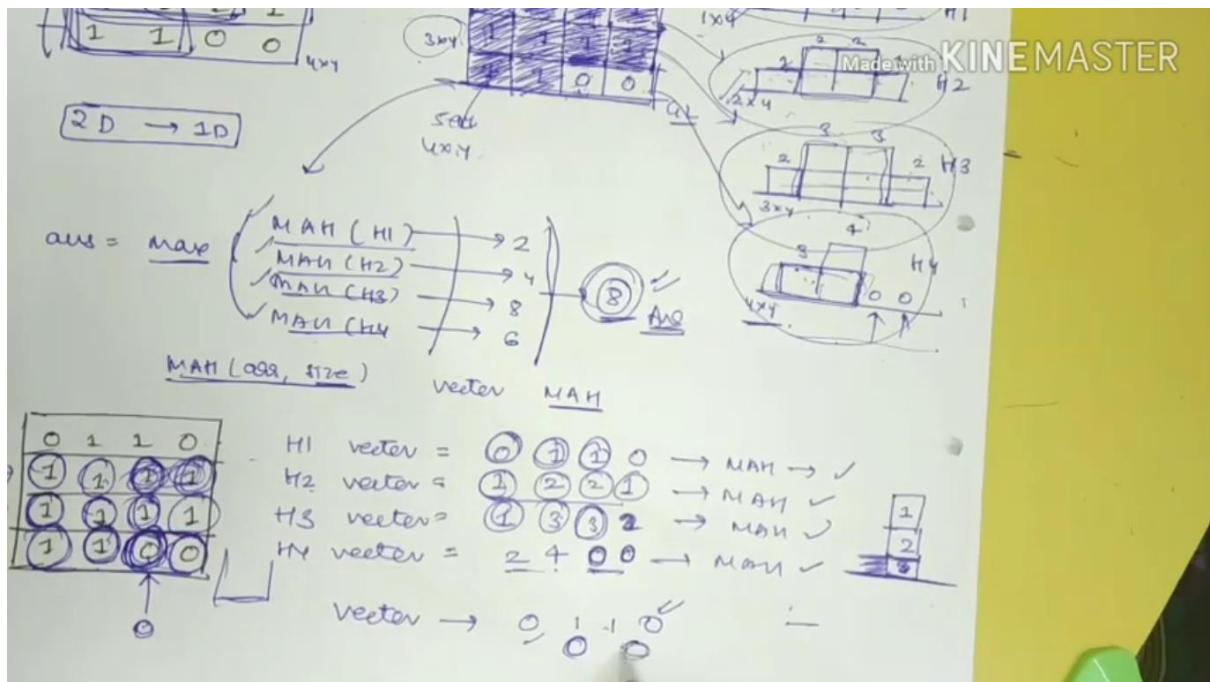
```

Sum of minimum and maximum elements of all subarrays of size k:

```
1 int SumOfKsubArray(int arr[] , int n , int k)
2 {
3     int sum = 0;
4     deque< int > S(k), G(k);
5     int i = 0;
6     for (i = 0; i < k; i++)
7     {
8         while ( (!S.empty()) && arr[S.back()] >= arr[i])
9             S.pop_back();
10        while ( (!G.empty()) && arr[G.back()] <= arr[i])
11            G.pop_back();
12        G.push_back(i);
13        S.push_back(i);
14    }
15    for ( ; i < n; i++)
16    {
17        sum += arr[S.front()] + arr[G.front()];
18        while ( !S.empty() && S.front() <= i - k)
19            S.pop_front();
20        while ( !G.empty() && G.front() <= i - k)
21            G.pop_front();
22        while ( (!S.empty()) && arr[S.back()] >= arr[i])
23            S.pop_back();
24        while ( (!G.empty()) && arr[G.back()] <= arr[i])
25            G.pop_back();
26        G.push_back(i);
27        S.push_back(i);
28    }
29    sum += arr[S.front()] + arr[G.front()];
30    return sum;
31 }
```

Maximal Rectangle in 0-1 Matrix





Simplify Path

agar string ka last element '/' nhi hai th last m '/' add krdo aur fr $n=n+1$ krdo ab $i=1$ se iterate kro.

Hume bss vhi element lene h jo ‘/’ and next ‘/’ k beech m h . Jo temp string m aaya dono / k beech m dekho agar . hai th continue ya temp empty string h th continue ya “..” hai th st.pop() krdo vrna stack m temp ko dalo aur temp ko empty krdo. Ab stack m jo bhi string h vh aa chuki h th bss while loop chalao jab tk stack empty na hota fr

```
while(!s.empty()){

ans=/+st.top()+ans;

s.pop();

}
```

```

string simplifyPath(string s) {
    int n=s.size();
    stack<string>st;
    if(s[n-1]!='/')
        {s+="/";n+=1;};
    string temp="";
    string ans="";
    for(int i=1;i<n;i++)
    {
        if(s[i]=='/')
        {
            if(temp==".")||temp=="")
            {
                temp="";
                continue;
            }
            else if(temp=="..")
            {
                if(!st.empty())
                    st.pop();
            }
            else
            {
                st.push(temp);
            }
            temp="";
        }
        else
        {
            temp.push_back(s[i]);
        }
    }
    while(!st.empty())
    {
        ans="/"+st.top()+ans;
        st.pop();
    }
    if(ans.length()==0)
        ans("/");
    return ans;
}

```

HEAPS:

Heaps follow **CBT** property and also **heap order** property.

Up heapify=> Time Complexity **O(logn)** bcz BST h th height k equal hi upr jaega.

Down Heapify=> Time Complexity **O(logn)** bcz BST h th height k equal hi neeche jaega.

Get=> **O(1)**

Heap Sort => Time Complexity $O(n \log n)$ bcz ek ki logn aur n elements h th nlogn.

Kth smallest element in a sorted matrix:

same as we have given n sorted and we have to find kth element so insert first element of each row in priority queue.

```
class Solution {
public:
    struct compare
    {
        bool operator()(const pair<int,pair<int, int> & a, const pair<int,pair<int, int> & b)
        {
            return a.first>b.first;
        }
    };
    int kthSmallest(vector<vector<int>>& arr, int k) {
        int n=arr.size(),m=arr[0].size();

        priority_queue< pair<int,pair<int, int> >, vector<pair<int, pair<int, int> > , compare > p;

        for(int i=0;i<n;i++)
            p.push(make_pair(arr[i][0],make_pair(i,0)));

        int x=k,ans;
        while(x--)
        {
            int e=p.top().first;
            int i=p.top().second.first;
            int j=p.top().second.second;
            ans=e;
            p.pop();
            if(j!=m-1)
                p.push(make_pair(arr[i][j+1],make_pair(i,j+1)));
        }
        return ans;
    }
};
```

INSERT AND REMOVE MIN FROM PRIORITY

QUEUE:

```
1+ class PriorityQueue {
2+     vector<int> pq;
3+ public:
4+     priorityQueue() {
5+     }
6+     bool isEmpty() {
7+         return pq.size() == 0;
8+     }
9+     int getSize() {
10+    return pq.size();
11+ }
12+ int getMin() {
13+    if(isEmpty()) {
14+        return 0;      // Priority Queue is empty
15+    }
16+    return pq[0];
17+ }
18+ void insert(int element) {
19+    pq.push_back(element);
20+    int childIndex = pq.size() - 1;
21+    while(childIndex > 0) {
22+        int parentIndex = (childIndex - 1) / 2;
23+        if(pq[childIndex] < pq[parentIndex]) {
24+            swap(pq[childIndex], pq[parentIndex]);
25+            pq[childIndex] = pq[parentIndex];
26+            pq[parentIndex] = temp;
27+        }
28+        else {
29+            break;
30+        }
31+        childIndex = parentIndex;
32+    }
33+ }
34+ int removeMin() {
35+    if(isEmpty()) {
36+        return 0;      // Priority Queue is empty
37+    }
38+    int ans = pq[0];
39+    pq[0] = pq[pq.size() - 1];
40+    pq.pop_back();
41+    int parentIndex = 0;
42+    int leftChildIndex = 2 * parentIndex + 1;
43+    int rightChildIndex = 2 * parentIndex + 2;
44+    while(leftChildIndex < pq.size()) {
45+        int minIndex = parentIndex;
46+        if(pq[minIndex] > pq[leftChildIndex]) {
47+            minIndex = leftChildIndex;
48+        }
49+        if(rightChildIndex < pq.size() && pq[rightChildIndex] < pq[minIndex]) {
50+            minIndex = rightChildIndex;
51+        }
52+        if(minIndex == parentIndex) {
53+            break;
54+        }
55+        int temp = pq[minIndex];
56+        pq[minIndex] = pq[leftChildIndex];
57+        pq[leftChildIndex] = temp;
58+        parentIndex = minIndex;
59+        leftChildIndex = 2 * parentIndex + 1;
60+        rightChildIndex = 2 * parentIndex + 2;
61+    }
62+    return ans;
63+ }
```

INPLACE HEAP SORT:

```
1 void inplaceHeapSort(int pq[], int n){
2     //to create min heap from input array without extra vector(space)
3     for(int i=0;i<n;i++)
4     {
5         int childIndex=i;
6         while(childIndex > 0) {
7             int parentIndex = (childIndex - 1) / 2;
8
9             if(pq[childIndex] < pq[parentIndex]) {
10                 int temp = pq[childIndex];
11                 pq[childIndex] = pq[parentIndex];
12                 pq[parentIndex] = temp;
13             }
14             else {
15                 break;
16             }
17             childIndex = parentIndex;
18         }
19     }
20     //remove elements
21     int size=n;
22     while(size > 1) {
23         int temp = pq[0];
24         pq[0] = pq[size - 1];
25         pq[size-1] = temp;
26
27         size--;
28
29         int parentIndex = 0;
30         int leftChildIndex = 2 * parentIndex + 1;
31         int rightChildIndx = 2 * parentIndex + 2;
32
33         while(leftChildIndex < size) {
34             int minIndex = parentIndex;
35             if(pq[minIndex] > pq[leftChildIndex]) {
36                 minIndex = leftChildIndex;
37             }
38             if(rightChildIndx < size && pq[rightChildIndx] < pq[minIndex]) {
39                 minIndex = rightChildIndx;
40             }
41             if(minIndex == parentIndex) {
42                 break;
43             }
44             int temp = pq[minIndex];
45             pq[minIndex] = pq[parentIndex];
46             pq[parentIndex] = temp;
47
48             parentIndex = minIndex;
49             leftChildIndex = 2 * parentIndex + 1;
50             rightChildIndx = 2 * parentIndex + 2;
51         }
52     }
53 }
```

K-th Largest Sum Contiguous Subarray:

Given an array of integers. Write a program to find the K-th largest sum of contiguous subarray within the array of numbers which has negative and positive numbers.

Examples:

```
Input: a[] = {20, -5, -1}
      k = 3
Output: 14
Explanation: All sum of contiguous
subarrays are (20, 15, 14, -5, -6, -1)
so the 3rd largest sum is 14.

Input: a[] = {10, -10, 20, -40}
      k = 6
Output: -10
Explanation: The 6th largest sum among
sum of all contiguous subarrays is -10.
```

Red Hat
Summit

Virtual Experience
June 15–16, 2021

TUNE-IN
for the latest in future technologies

Register

WHAT'S NEW

Ek prefix sum array bnalo ab subarray ka

sum hga pref[j]-pref[i] isko pq m daaldo k

size ka rkhna bss min priority queue

```
1 int kthLargestSum(int arr[], int n, int k)
2 {
3     int sum[n + 1];
4     sum[0] = 0;
5     sum[1] = arr[0];
6     for (int i = 2; i <= n; i++)
7         sum[i] = sum[i - 1] + arr[i - 1];
8     priority_queue<int, vector<int>, greater<int> > Q;
9     for (int i = 1; i <= n; i++)
10    {
11        for (int j = i; j <= n; j++)
12        {
13            int x = sum[j] - sum[i - 1];
14            if (Q.size() < k)
15                Q.push(x);
16            else
17            {
18                if (Q.top() < x)
19                {
20                    Q.pop();
21                    Q.push(x);
22                }
23            }
24        }
25    }
26    return Q.top();
27 }
```

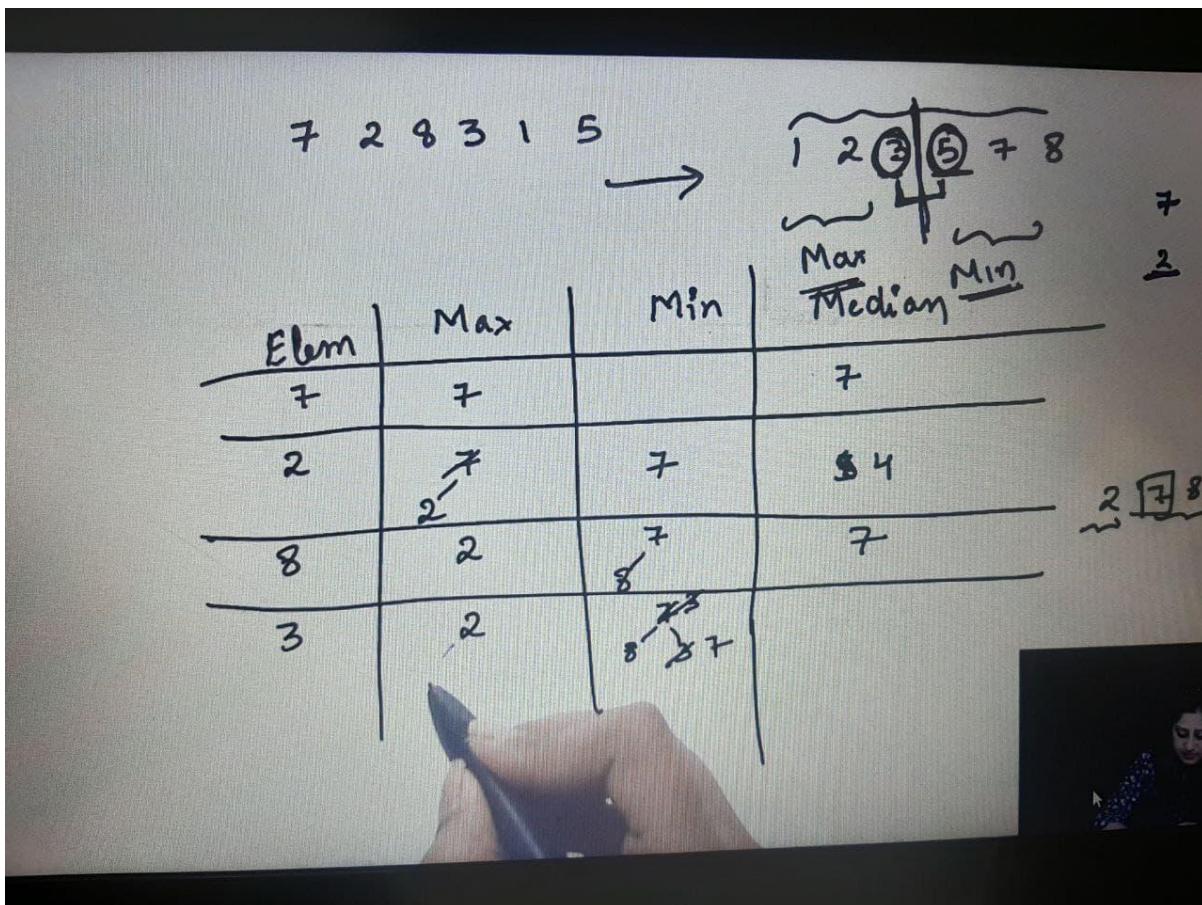
Convert BST to Min Heap:

1. Create an array **arr[]** of size **n**, where **n** is the number of nodes in the given BST.
2. Perform the inorder traversal of the BST and copy the node values in the **arr[]** in sorted order.

3. Now perform the preorder traversal of the tree.
4. While traversing the root during the preorder traversal, one by one copy the values from the array **arr[]** to the nodes.

```
void inorderTraversal(Node *root, vector<int>& arr)
{
    if (root == NULL)
        return;
    inorderTraversal(root->left, arr);
    arr.push_back(root->data);
    inorderTraversal(root->right, arr);
}
void BSTToMinHeap(Node *root, vector<int> arr, int *i)
{
    if (root == NULL)
        return;
    root->data = arr[++*i];
    BSTToMinHeap(root->left, arr, i);
    BSTToMinHeap(root->right, arr, i);
}
void convertToMinHeapUtil(Node *root)
{
    vector<int> arr;
    int i = -1;
    inorderTraversal(root, arr);
    BSTToMinHeap(root, arr, &i);
}
void preorderTraversal(Node *root)
{
    if (!root)
        return;
    cout << root->data << " ";
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}
```

Running Median:



```

1 #include<queue>
2 void findMedian(int a[], int n){
3     priority_queue<int, vector<int>, greater<int>>minheap;
4     priority_queue<int>maxheap;
5     maxheap.push(a[0]);
6     cout<<a[0]<<endl;
7     for(int i=1;i<n;i++)
8     {
9         if(a[i]<maxheap.top())
10        {
11            maxheap.push(a[i]);
12        }
13        else
14        {
15            minheap.push(a[i]);
16        }
17        if(abs((int)maxheap.size()-(int)minheap.size())>1)
18        {
19            if(maxheap.size()>minheap.size())
20            {
21                int w=maxheap.top();
22                maxheap.pop();
23                minheap.push(w);
24            }
25            else
26            {
27                int w=minheap.top();
28                minheap.pop();
29                maxheap.push(w);
30            }
31        }
32        if(maxheap.size()>minheap.size())
33            cout<<maxheap.top()<<endl;
34        else if(maxheap.size()<minheap.size())
35            cout<<minheap.top()<<endl;
36        else
37            cout<<(minheap.top()+maxheap.top())/2<<endl;
38    }
39 }
```

Maximum Sum Combinations

Approach 1 (Naive Algorithm) :

We can use Brute force through all the possible combinations that can be made by taking one element from array A and another from array B and inserting them to a max heap. In a max heap maximum element is at the root node so whenever we pop from max heap we get the maximum element present in the heap. After inserting all the sum combinations we take out K elements from max heap and display it.

Time Complexity : O(N²)

This will not pass under given constraints.

Approach 2 (Sorting, Max heap, Map) :

Instead of brute forcing through all the possible sum combinations we should find a way to limit our search space to possible candidate sum combinations.

1. Sort both arrays array A and array B.
2. Create a max heap i.e priority_queue in C++ to store the sum combinations along with the indices of elements from both arrays A and B which make up the sum. Heap is ordered by the sum.
3. Initialize the heap with the maximum possible sum combination i.e $(A[N - 1] + B[N - 1]$ where N is the size of array) and with the indices of elements from both arrays (N – 1, N – 1). The tuple inside max heap will be $(A[N-1] + B[N - 1], N - 1, N - 1)$. Heap is ordered by first value i.e sum of both elements.
4. Pop the heap to get the current largest sum and along with the indices of the element that make up the sum. Let the tuple be (sum, i, j).
5. Next insert $(A[i - 1] + B[j], i - 1, j)$ and $(A[i] + B[j - 1], i, j - 1)$ into the max heap but make sure that the pair of indices i.e $(i - 1, j)$ and $(i, j - 1)$ are not already present in the max heap. To check this we can use set in C++.
6. Go back to 4 until K times.

Time Complexity : O(NlogN) assuming C <= N

Another approach where no need of set:

```

We here use the concept of merging k - sorted linked lists.
Let the given vectors after sorting be
A : 16, 11, 7, 1
B : 15, 10, 9, 2

Now we know that 4 * 4 different sums are possible for this combination and we need to
find the largest C out of these. Let the pairs be:
(16,15), (16,10), (16,9), (16,2),
(11,15), (11,10), (11,9), (11,2),
(7,15), (7,10), (7,9), (7,2),
(1,15), (1,10), (1,9), (1,2),

Note that the numbers (pair of the sums) in every row are sorted in descending order. Now we
need to merge these sorted LinkedLists.

vector<int> Solution::solve(vector<int> &A, vector<int> &B, int C)
{
    sort(A.rbegin(), A.rend());
    sort(B.rbegin(), B.rend()); //sorting both the lists in descending order.
    priority_queue<pair<int, pair<int, int>>> maxh; //max heap for storing (sum, (i - index, j - index))

    for (int i = 0; i < C; i++) //pushing the heads of all the linked lists.
    {
        maxh.push(make_pair(B[0] + A[i], make_pair(0, i)));
    }

    std::vector<int> ans;
    while(ans.size() < C)
    {
        auto maxele = maxh.top(); maxh.pop();
        ans.emplace_back(maxele.first); //pushing the top to ans and inserting head -> next.
        maxh.push(make_pair(B[maxele.second.first + 1] + A[maxele.second.second], make_pair(maxele.second.first + 1, maxele.second.second)));
    }
    return ans;
}

```

DP:

Nth catalan number

Catalan numbers are a sequence of natural numbers that occurs in many interesting counting problems like following.

1. Count the number of expressions containing n pairs of parentheses which are correctly matched. For $n = 3$, possible expressions are $((())), (())(), ()()()$.
2. Count the number of possible Binary Search Trees with n keys (See [this](#))
3. Count the number of full binary trees (A rooted binary tree is full if every vertex has either two children or no children) with $n+1$ leaves.
4. Given a number n , return the number of ways you can draw n chords in a circle with $2 \times n$ points such that no 2 chords intersect.

PAINTING THE FENCE:

Paint Fence (Leetcode) Dynamic Programming | Explanation with Code

	1	2	3	4	5	
i	-	3 nn bb	6 nss bb sns	18 nbb bs sns	48	$b \leq n$
j	-	6 nb bb gn bg	18 nns bb sns nsb nbs	48	132	$k \otimes (k-1)$
total		9 nn bb ns nb bg gn gb	24	66	180	$\text{Extend } \oplus (k-1)$

Paint Fence (Leetcode) Dynamic Programming | Explanation with Code

QUESTION EDITOR JAVA HISTORY SOLUTION STATS

```

1 import java.io.*;
2 import java.util.*;
3
4 public class Main {
5
6     public static void main(String[] args) throws Exception {
7         Scanner scn = new Scanner(System.in);
8
9         int n = scn.nextInt();
10        int k = scn.nextInt();
11
12        long same = k * 1;
13        long diff = k * (k - 1);
14        long total = same + diff;
15
16        for(int i = 3; i <= n; i++){
17            same = diff * 1;
18            diff = total * (k - 1);
19            total = same + diff;
20        }
21
22        System.out.println(total);
23    }
24 }
```

Add your own input. Your code will run again on test case.

Distinct Subsequences

```
int numDistinct(string s, string t) {
    int n=s.size(),m=t.size();
    if((n<m) || (n==0) || (m==0))
        return 0;
    long long int dp[n+1][m+1];
    for(int i=0;i<=n;++i)
        dp[i][0]=1;
    for(int j=1;j<=m;++j)
        dp[0][j]=0;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            if(i<j)
                dp[i][j]=0;
            else if(s[i-1]==t[j-1])
                dp[i][j]=dp[i-1][j-1]/*include kia*/+dp[i-1][j]/*nhi kia include*/;
            else
                dp[i][j]=dp[i-1][j]//include nhi krne k alava koi option hi nhi h;
        }
    }
    return dp[n][m];
}
```

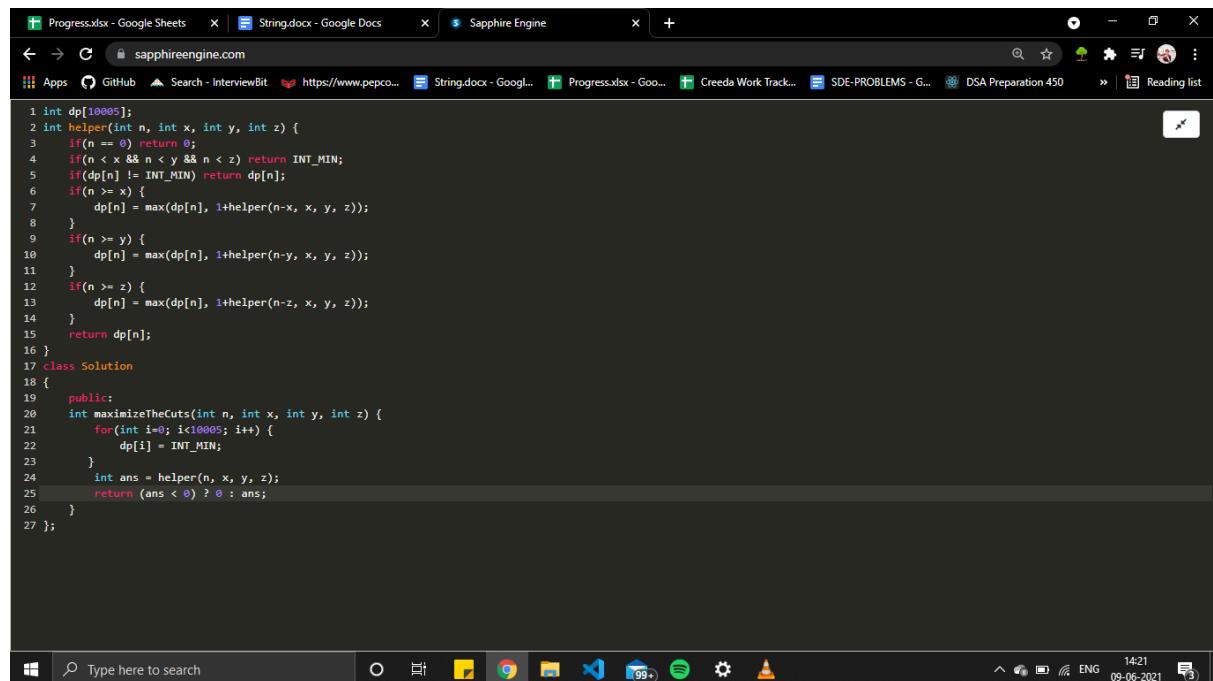
Wildcard Matching

```
bool isMatch(string s, string p) {
    int n=s.length();
    int m=p.length();
    if(m==0)
        return (n==0);
    bool dp[n+1][m+1];
    memset(dp, false, sizeof(dp));
    dp[0][0]=true;
    for(int i=1;i<=m;i++)
    {
        if(p[i-1]=='*')
            dp[0][i]=dp[0][i-1];
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            if(p[j-1]=='*')
                dp[i][j]=dp[i][j-1]||dp[i-1][j];
            else if(s[i-1]==p[j-1]||p[j-1]=='?')
                dp[i][j]=dp[i-1][j];
        }
    }
    return dp[n][m];
}
```

Assembly Line Scheduling:

```
1 int carAssembly(int a[][NUM_STATION],
2                  int t[][NUM_STATION],
3                  int *e, int *x)
4 {
5     int T1[NUM_STATION], T2[NUM_STATION], i;
6
7     // time taken to leave first station in line 1
8     T1[0] = e[0] + a[0][0];
9
10    // time taken to leave first station in line 2
11    T2[0] = e[1] + a[1][0];
12
13    // Fill tables T1[] and T2[] using the
14    // above given recursive relations
15    for (i = 1; i < NUM_STATION; ++i)
16    {
17        T1[i] = min(T1[i - 1] + a[0][i],
18                    T2[i - 1] + t[1][i] + a[0][i]);
19        T2[i] = min(T2[i - 1] + a[1][i],
20                    T1[i - 1] + t[0][i] + a[1][i]);
21    }
22
23    // Consider exit times and return minimum
24    return min(T1[NUM_STATION - 1] + x[0],
25               T2[NUM_STATION - 1] + x[1]);
26 }
27
```

Maximize The Cut Segments:



```
1 int dp[10005];
2 int helper(int n, int x, int y, int z) {
3     if(n == 0) return 0;
4     if(n < x && n < y && n < z) return INT_MIN;
5     if(dp[n] != INT_MIN) return dp[n];
6     if(n >= x) {
7         dp[n] = max(dp[n], 1+helper(n-x, x, y, z));
8     }
9     if(n >= y) {
10        dp[n] = max(dp[n], 1+helper(n-y, x, y, z));
11    }
12    if(n >= z) {
13        dp[n] = max(dp[n], 1+helper(n-z, x, y, z));
14    }
15    return dp[n];
16 }
17 class Solution
18 {
19     public:
20     int maximizeTheCuts(int n, int x, int y, int z) {
21         for(int i=0; i<10005; i++) {
22             dp[i] = INT_MIN;
23         }
24         int ans = helper(n, x, y, z);
25         return (ans < 0) ? 0 : ans;
26     }
27 };
```

A Space Optimized Solution of LCS:

```
1 int lcs(string &X, string &Y)
2 {
3     int m = X.length(), n = Y.length();
4     int L[2][n + 1];
5     bool bi;
6     for (int i = 0; i <= m; i++)
7     {
8         bi = i & 1;
9         for (int j = 0; j <= n; j++)
10        {
11            if (i == 0 || j == 0)
12                L[bi][j] = 0;
13
14            else if (X[i-1] == Y[j-1])
15                L[bi][j] = L[1 - bi][j - 1] + 1;
16
17            else
18                L[bi][j] = max(L[1 - bi][j],
19                                L[bi][j - 1]);
20        }
21    }
22    return L[bi][n];
23 }
```

Printing Longest Common Subsequence:

24

WEDNESDAY APRIL

WEU - 11429

Print lcs of the 2 strings

APR • 2019						
S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

	v	a	b	c	d	a	f
0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1
2	0	1	1	2	2	2	2
3	0	1	2	2	2	2	3
4	0	1	2	3	3	3	4
5	0	1	2	3	3	3	4
6	0	1	2	3	3	3	4

fcba

reverse

. abcfa

31312 1-1 & 5-1 equal to 7 Diagonal on left
 21311 92nd 3rd row Max on right side

```
int i=m; int j=n;
```

```
String ss="";
```

```
while(i>0 && j>0)
```

```
{
```

```
if (a[i-1]==b[j-1])
```

```
{ ss+=a[i-1];
```

```
i--, j--;
```

```
g
```

```
else
```

```
{
```

```
if (a[i][j-1] > a[i-1][j])
```

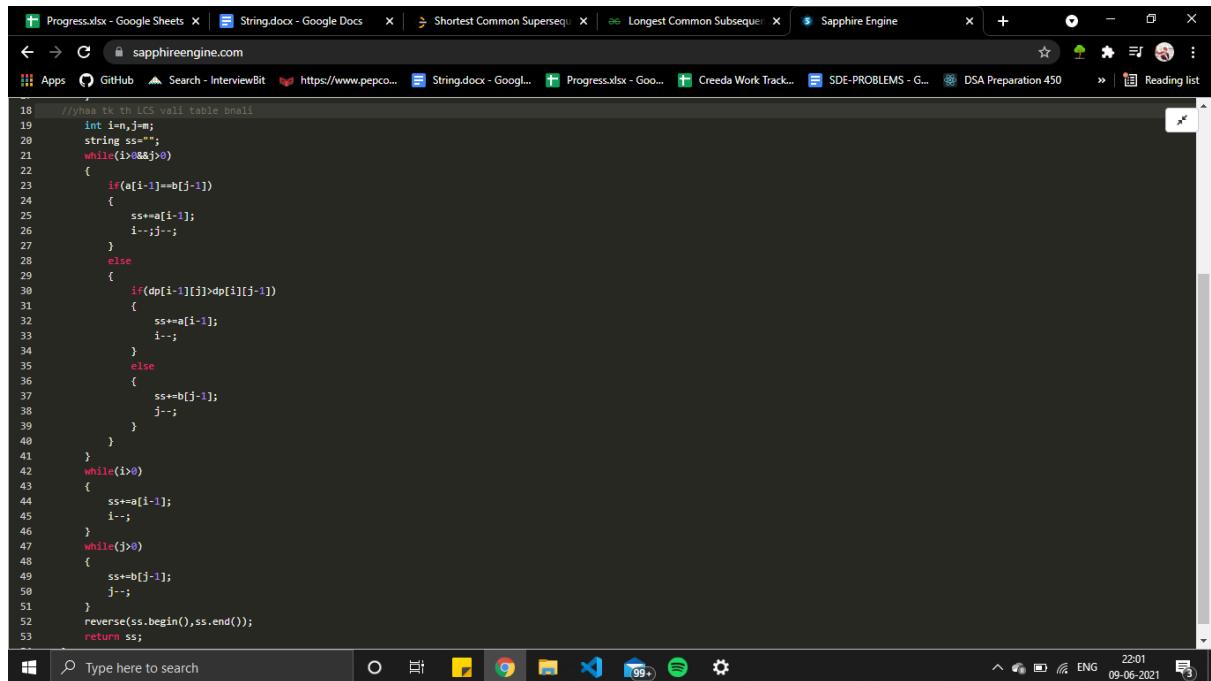
```
j--;
```

```
else
```

```
i--
```

reverse(all(ss)); return

Print Shortest Common Supersequence:



```
18 //yhaa tk th LCS vali table bnali
19 int i=n,j=m;
20 string ss="";
21 while(i>0&&j>0)
22 {
23     if(a[i-1]==b[j-1])
24     {
25         ss+=a[i-1];
26         i--;
27         j--;
28     }
29     else
30     {
31         if(dp[i-1][j]>dp[i][j-1])
32         {
33             ss+=a[i-1];
34             i--;
35         }
36         else
37         {
38             ss+=b[j-1];
39             j--;
40         }
41     }
42     while(i>0)
43     {
44         ss+=a[i-1];
45         i--;
46     }
47     while(j>0)
48     {
49         ss+=b[j-1];
50         j--;
51     }
52     reverse(ss.begin(),ss.end());
53     return ss;
54 }
```

MATRIX CHAIN MULTIPLICATION:

- 1.) Find i and j.
- 2.) Base Condition
- 3.) Run loop of k

4.) Temp ans function.

```
10+ class Solution{
11  int dp[101][101];
12  int solve(int n,int a[],int i,int j)
13  {
14      if(i>=j)
15          return 0;
16      if(dp[i][j]!=-1)
17          return dp[i][j];
18      int ans=INT_MAX;
19      int left,right;
20      for(int k=i;k<=j-1;k++)
21      {
22          if(dp[i][k]!=-1)
23              left=dp[i][k];
24          else
25          {
26              left=solve(n,a,i,k);
27              dp[i][k]=left;
28          }
29          if(dp[k+1][j]!=-1)
30              right=dp[k+1][j];
31          else
32          {
33              right=solve(n,a,k+1,j);
34              dp[k+1][j]=right;
35          }
36          int temp=left+right+(a[i-1]*a[k]*a[j]);
37          if(temp<ans)
38              ans=temp;
39      }
40      return dp[i][j]=ans;
41  }
42  public:
43  int matrixMultiplication(int n, int a[])
44  {
45      memset(dp,-1,sizeof(dp));
46      return solve(n,a,1,n-1);
47  }
48 };
49 } // } Driver Code Ends
```

Palindromic partitioning:

Given a string str, a partitioning of the string is a *palindrome partitioning* if every sub-string of the partition is a palindrome. Determine the fewest cuts needed for palindrome partitioning of given string.

Example 1:

Input: str = "ababbabbababa"

Output: 3

Explanation: After 3 partitioning substrings are "a", "babbbab", "b", "ababa".

```
23     return true;
24 }
25 int dp[501][501];
26 int solve(int n,string a,int i,int j)
27 {
28     if(i>=j||isPalindrome(a,i,j))
29         return 0;
30     if(dp[i][j]!=-1)
31         return dp[i][j];
32     int ans=INT_MAX;
33     int left,right;
34     for(int k=i;k<=j-1;k++)
35     {
36         if(dp[i][k]!=-1)
37             left=dp[i][k];
38         else
39         {
40             left=solve(n,a,i,k);
41             dp[i][k]=left;
42         }
43         if(dp[k+1][j]!=-1)
44             right=dp[k+1][j];
45         else
46         {
47             right=solve(n,a,k+1,j);
48             dp[k+1][j]=right;
49         }
50         int temp=left+right+1;
51         if(temp<ans)
52             ans=temp;
53     }
54     return dp[i][j]=ans;
55 }
56 int palindromicPartition(string s)
57 {
58     memset(dp,-1,sizeof(dp));
59     return solve(s.size(),s,0,s.size()-1);
60 }
61 };
62
63 } // } Driver Code Ends
```

EVALUATE EXPRESSION TO TRUE:

Given a boolean expression S of length N with following symbols.

Symbols

'T' ---> true

'F' ---> false

and following operators filled between symbols

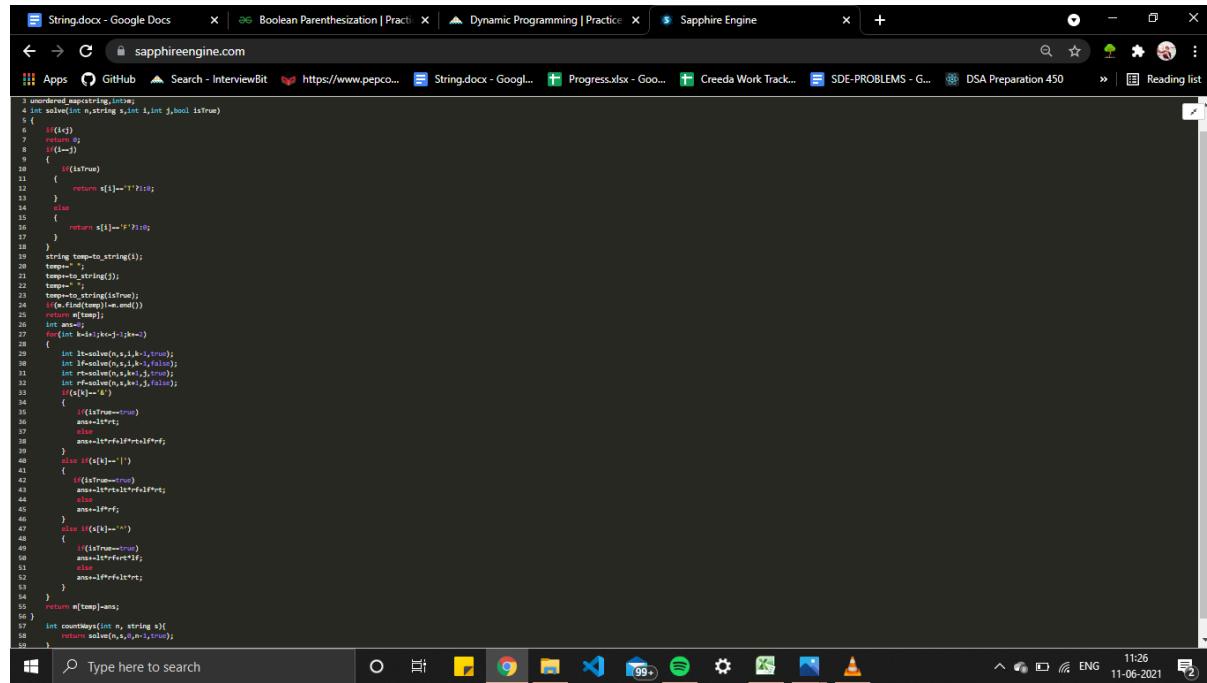
Operators

& ---> boolean AND

| ---> boolean OR

^ ---> boolean XOR

Count the number of ways we can parenthesize the expression so that the value of expression evaluates to true.



The screenshot shows a browser window with multiple tabs open. The active tab displays a C++ code snippet for solving the Boolean Parenthesization problem. The code uses dynamic programming to calculate the number of ways to parenthesize a string of length n such that the resulting expression evaluates to true. It includes functions for calculating powers of 2, counting ways, and solving the main problem. The browser interface includes a search bar at the bottom and various system icons.

```
3 unordered_map<string, int>;
4 int solve(int n, string s, int i, int j, bool lIsTrue)
5 {
6     if(i>j)
7         return 0;
8     if(i==j)
9     {
10         if(lIsTrue)
11             {
12                 return s[i]==''?1:0;
13             }
14         else
15             {
16                 return s[i]==''?0:0;
17             }
18     }
19     string temp=to_string(i);
20     temp+='-';
21     temp+=to_string(j);
22     temp+=')';
23     temp+=to_string(lIsTrue);
24     if(m.find(temp)==m.end())
25         return a[temp];
26     for(int k=i;k<=j;k++)
27     {
28         int lIsTrue=(n,i,k-1,true);
29         int lF=(n,i,k-1,false);
30         int rF=solve(n,j,k+1,true);
31         int rIs=solve(n,j,k+1,false);
32         int rFOr=solve(n,j,k+1,j,rF);
33         int rFAnd=solve(n,j,k+1,j,rIs);
34         if(s[k]=='^')
35         {
36             if(lIsTrue==true)
37                 ans+=lF*rF;
38             else
39                 ans+=lF*rFOr+rF*rF;
40         }
41         else if(s[k]== '|')
42         {
43             if(lIsTrue==true)
44                 ans+=lF+rF+rFOr+rF;
45             else
46                 ans+=lF+rF;
47         }
48         else if(s[k]== '&')
49         {
50             if(lIsTrue==true)
51                 ans+=lF*rFOr+rF;
52             else
53                 ans+=lF*rF+lFOr+rF;
54         }
55     }
56     return a[temp]=ans;
57 }
58 int countways(int n, string s){
59     return solve(n,n-1,true);
60 }
```

EDIT DISTANCE.

A screenshot of a Windows operating system desktop. At the top, there is a taskbar with several pinned icons: Apps, GitHub, Search - InterviewBit, https://www.pepc..., String.docx - Google..., Progress.xlsx - Google..., Creeda Work Track..., SDE-PROBLEMS - G..., DSA Preparation 450, and Reading list. Below the taskbar is a window titled "Sapphire Engine" which contains a C++ code editor. The code is a solution for the Edit Distance problem, using dynamic programming with a 2D array dp. The code includes comments explaining the logic for handling matches, mismatches, and insertions. The code editor has syntax highlighting for C++ and a dark theme. The bottom right corner of the screen shows the date and time as 11-06-2021 11:40.

```
1 int n=s.size();
2 int m=t.size();
3 int dp[n+1][m+1];
4 for(int i=0;i<n;i++)
5 {
6     for(int j=0;j<m;j++)
7     {
8         if(i==0)
9             dp[i][j]=j;
10        else if(j==0)
11            dp[i][j]=i;
12        else
13        {
14            if(s[i-1]==t[j-1])
15                dp[i][j]=dp[i-1][j-1];
16            else
17            {
18                dp[i][j]=1+min((dp[i][j-1],dp[i-1][j]));
19            }
20        }
21    }
22 }
23 return dp[n][m];
```

LCS OF THREE STRINGS:

```
1 int lcsOf3( string X, string Y, string Z, int m,
2                           int n, int o)
3 {
4     int L[m+1][n+1][o+1];
5     for (int i=0; i<=m; i++)
6     {
7         for (int j=0; j<=n; j++)
8         {
9             for (int k=0; k<=o; k++)
10            {
11                if (i == 0 || j == 0 || k == 0)
12                    L[i][j][k] = 0;
13                else if (X[i-1] == Y[j-1] && X[i-1] == Z[k-1])
14                    L[i][j][k] = L[i-1][j-1][k-1] + 1;
15                else
16                    L[i][j][k] = max(max(L[i-1][j][k],
17                                         L[i][j-1][k]),
18                                         L[i][j][k-1]);
19            }
20        }
21    }
22    return L[m][n][o];
23 }
```

Count all subsequences having product less than K:

Subsequences pucha hai th fr **dp** and **subarrays** pucha hota th **sliding window**

Given a non-negative array, find the number of subsequences having a product smaller than K.

Examples:

Input : [1, 2, 3, 4]

k = 10

Output :11

The subsequences are {1}, {2}, {3}, {4},

{1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4},

```
{1, 2, 3}, {1, 2, 4}
```

```
1 int productSubSeqCount(vector<int> &a, int k)
2 {
3     int n = a.size();
4     int dp[n + 1][k + 1];
5     memset(dp, 0, sizeof(dp));
6
7     for (int i = 1; i <= n; i++) {
8         for (int j = 1; j <= k; j++) {
9             if(a[i-1]>j||a[i-1]==0)
10                 dp[i][j]=dp[i-1][j];
11             else
12                 dp[i][j]=(dp[i-1][j])+(1+dp[i-1][j/a[i-1]]);
13         }
14     }
15     return dp[n][k];
16 }
17 int main()
18 {
19     vector<int> A;
20     A.push_back(4);
21     A.push_back(8);
22     A.push_back(7);
23     A.push_back(2);
24     int k = 50;
25     cout << productSubSeqCount(A, k) << endl;
26 }
27
```

Maximum difference of zeros and ones in binary string | Set 2 (O(n) time):

Idea behind that if we convert all zeros into 1 and all ones into -1.now our problem reduces to find out the maximum sum sub_array Using Kadane's Algorithm.

```
1 int findLength(string str, int n)
2 {
3     int current_sum = 0;
4     int max_sum = 0;
5     for (int i = 0; i < n; i++) {
6         current_sum += (str[i] == '0' ? 1 : -1);
7         if (current_sum < 0)
8             current_sum = 0;
9         max_sum = max(current_sum, max_sum);
10    }
11    return max_sum == 0 ? -1 : max_sum;
12 }
```

Maximum sum of pairs with specific difference:

Input : arr[] = {3, 5, 10, 15, 17, 12, 9}, K = 4

Output : 62

Explanation:

Then disjoint pairs with difference less than K are, (3, 5), (10, 12), (15, 17)

So maximum sum which we can get is $3 + 5 + 12 + 10 + 15 + 17 = 62$

Note that an alternate way to form disjoint pairs is, (3, 5), (9, 12), (15, 17), but this pairing produces lesser sum.



The screenshot shows a browser window with multiple tabs open. The active tab is "ide.geeksforgeeks.org/Eky1pJ". The IDE interface has a sidebar on the left with language selection (C, C++, C++14, C#, Java, Perl, PHP, Python, Python 3, Scala, HTML & JS) and file operations (Upload, Download). The main area displays C++ code:

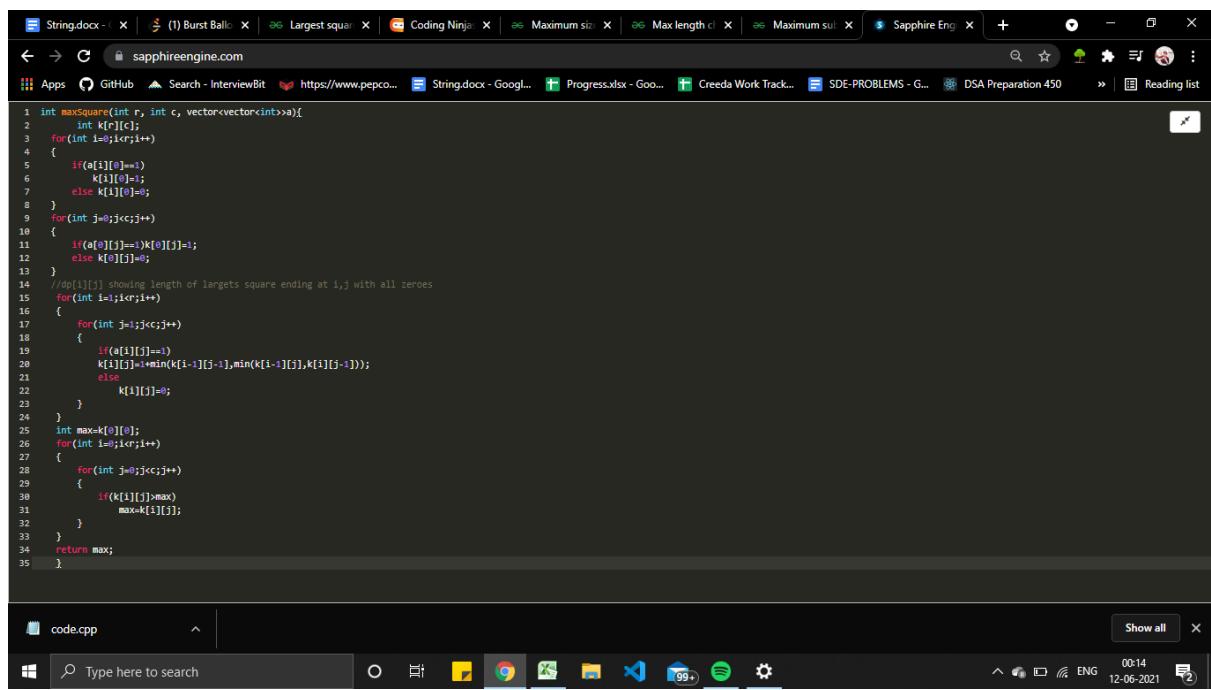
```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     int t,n,k,arr[100000],sum[100000];
7     cin >> t;
8     while(t--)
9     {
10         cin >> n;
11         for (int i = 1; i <= n; ++i)
12             cin >> arr[i];
13         cin >> k;
14         sort(arr+1,arr+n+1);
15         sum[0] = sum[1] = 0;
16         for (int i = 2; i <= n; ++i)
17             if (arr[i]-arr[i-1] < k)
18                 sum[i] = sum[i-2] + arr[i]+arr[i-1];
19             else
20                 sum[i] = sum[i-1];
21         cout << sum[n] << endl;
22     }
23     return 0;
24 }

```

The right side of the IDE includes a "Report Bug" button, a sidebar with "GeeksforGeeks" and "Explore Free Courses" links, and a taskbar at the bottom with various icons.

Maximum size square sub-matrix with all 1s



The screenshot shows a browser window with multiple tabs open. The active tab is "sapphireengine.com". The IDE interface has a sidebar on the left with language selection (code.cpp) and file operations (Upload, Download). The main area displays C++ code:

```

1 int maxsquare(int r, int c, vector<vector<int>>){ 
2     int k[r][c];
3     for(int i=0;i<r;i++)
4     {
5         if(a[i][0]==1)
6             k[i][0]=1;
7         else k[i][0]=0;
8     }
9     for(int j=0;j<c;j++)
10    {
11        if(a[0][j]==1)k[0][j]=1;
12        else k[0][j]=0;
13    }
14    //dp[i][j] showing length of largest square ending at i,j with all zeroes
15    for(int i=1;i<r;i++)
16    {
17        for(int j=1;j<c;j++)
18        {
19            if(a[i][j]==1)
20                k[i][j]=min(k[i-1][j-1],min(k[i-1][j],k[i][j-1]));
21            else
22                k[i][j]=0;
23        }
24    }
25    int max=k[0][0];
26    for(int i=0;i<r;i++)
27    {
28        for(int j=0;j<c;j++)
29        {
30            if(k[i][j]>max)
31                max=k[i][j];
32        }
33    }
34    return max;
35 }

```

The right side of the IDE includes a "Show all" button, a taskbar at the bottom with various icons, and a status bar indicating "00:14 12-06-2021".

Maximum subsequence sum such that no three are consecutive:

`sum[i]` : Stores result for subarray `arr[0..i]`, i.e.,
maximum possible sum in subarray `arr[0..i]`
such that no three elements are consecutive.

`sum[0] = arr[0]`

// Note : All elements are positive

`sum[1] = arr[0] + arr[1]`

// We have three cases

// 1) Exclude `arr[2]`, i.e., `sum[2] = sum[1]`

// 2) Exclude `arr[1]`, i.e., `sum[2] = sum[0] + arr[2]`

// 3) Exclude `arr[0]`, i.e., `sum[2] = arr[1] + arr[2]`

`sum[2] = max(sum[1], arr[0] + arr[2], arr[1] + arr[2])`

In general,

// We have three cases

// 1) Exclude `arr[i]`, i.e., `sum[i] = sum[i-1]`

// 2) Exclude `arr[i-1]`, i.e., `sum[i] = sum[i-2] + arr[i]`

// 3) Exclude `arr[i-2]`, i.e., `sum[i-3] + arr[i] + arr[i-1]`

`sum[i] = max(sum[i-1], sum[i-2] + arr[i],`

`sum[i-3] + arr[i] + arr[i-1])`

```

1 int maxSumW03Consec(int arr[], int n)
2 {
3     // Stores result for subarray arr[0..i], i.e.,
4     // maximum possible sum in subarray arr[0..i]
5     // such that no three elements are consecutive.
6     int sum[n];
7
8     // Base cases (process first three elements)
9     if (n >= 1)
10        sum[0] = arr[0];
11
12    if (n >= 2)
13        sum[1] = arr[0] + arr[1];
14
15    if (n > 2)
16        sum[2] = max(sum[1], max(arr[1] +
17                                arr[2], arr[0] + arr[2]));
18
19    // Process rest of the elements
20    // We have three cases
21    // 1) Exclude arr[i], i.e., sum[i] = sum[i-1]
22    // 2) Exclude arr[i-1], i.e., sum[i] = sum[i-2] + arr[i]
23    // 3) Exclude arr[i-2], i.e., sum[i-3] + arr[i] + arr[i-1]
24    for (int i = 3; i < n; i++)
25        sum[i] = max(max(sum[i - 1], sum[i - 2] + arr[i]),
26                      arr[i] + arr[i - 1] + sum[i - 3]);
27
28    return sum[n - 1];
29 }

```

Count Balanced Binary Trees of Height h:

Since the difference between the heights of left and right subtree is not more than one, possible heights of left and right part can be one of the following:

1. (h-1), (h-2)
2. (h-2), (h-1)
3. (h-1), (h-1)

A screenshot of a Windows desktop environment. At the top, there is a taskbar with several pinned and open application icons, including Microsoft Word, Microsoft Excel, Microsoft PowerPoint, Microsoft OneNote, Microsoft Edge, Spotify, and File Explorer. Below the taskbar is a window titled "sapphireengine.com" which contains a block of C++ code. The code defines a function `countBT` that calculates the number of binary trees with height `h`. It uses dynamic programming with a modulus of 1000000007. The code includes base cases for `h=0` and `h=1`, and a loop from `i=2` to `h` that updates the dp array. The code is color-coded with syntax highlighting. The system tray at the bottom right shows the date and time as "12-06-2021 21:48".

```
1 count(h) = count(h-1) * count(h-2) +
2         count(h-2) * count(h-1) +
3         count(h-1) * count(h-1)
4     = 2 * count(h-1) * count(h-2) +
5         count(h-1) * count(h-1)
6     = count(h-1) * (2*count(h - 2) +
7                     count(h - 1))
8
9 long long int countBT(int h) {
10
11     long long int dp[h + 1];
12     //base cases
13     dp[0] = dp[1] = 1;
14     for(int i = 2; i <= h; i++) {
15         dp[i] = (dp[i - 1] * ((2 * dp[i - 2])%mod + dp[i - 1])%mod) % mod;
16     }
17     return dp[h];
18 }
```

Minimum Difference Subsets

```
int Solution::solve(vector<int> &v) {
    int sum=0,n=v.size();
    for(int i=0;i<n;i++)
        sum+=v[i];
    bool dp[sum+1];
    memset(dp, false, sizeof(dp));
    dp[0]=true;
    for(int i=0;i<n;i++)
    {
        for(int j=v[i];j<=sum;j++)
        {
            dp[j]=dp[j-v[i]] || dp[j];
        }
    }
    int ans=sum;
    for(int i=sum/2;i>=0;i--)
    {
        if(dp[i]==true)
        {
            ans=(sum-2*i);
            break;
        }
    }
    return ans;
}
```

Word Break

```
bool wordBreak(string s, vector<string>& wordDict) {
    int n=s.size();
    int m=wordDict.size();
    int dp[n+1];
    memset(dp,0,sizeof(dp));
    dp[0]=1;
    for(int i=0;i<n;i++)
    {
        string ss="";
        for(int j=i;j>=0;j--)
        {
            ss=s[j]+ss;
            if(find(wordDict.begin(),wordDict.end(),ss)!=wordDict.end())
            {
                dp[i+1]=dp[j];
            }
            if(dp[i+1]==1)
                break;
        }
    }
    return dp[n];
}
```

Ways to arrange Balls such that adjacent balls are of different types

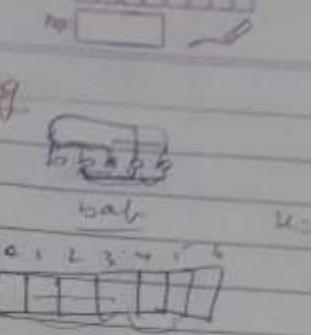
```
1 // table to store to store results of subproblems
2 int dp[MAX][MAX][MAX][3];
3
4 // Returns count of arrangements where last placed ball is
5 // 'last'. 'last' is 0 for 'p', 1 for 'q' and 2 for 'r'
6 int countWays(int p, int q, int r, int last)
7 {
8     // if number of balls of any color becomes less
9     // than 0 the number of ways arrangements is 0.
10    if (p<0 || q<0 || r<0)
11        return 0;
12
13    // If last ball required is of type P and the number
14    // of balls of P type is 1 while number of balls of
15    // other color is 0 the number of ways is 1.
16    if (p==1 && q==0 && r==0 && last==0)
17        return 1;
18    if (p==0 && q==1 && r==0 && last==1)
19        return 1;
20    if (p==0 && q==0 && r==1 && last==2)
21        return 1;
22    if (dp[p][q][r][last] != -1)
23        return dp[p][q][r][last];
24    if (last==0)
25        dp[p][q][r][last] = countWays(p-1,q,r,1) + countWays(p-1,q,r,2);
26    else if (last==1)
27        dp[p][q][r][last] = countWays(p,q-1,r,0) + countWays(p,q-1,r,2);
28    else //last==2
29        dp[p][q][r][last] = countWays(p,q,r-1,0) + countWays(p,q,r-1,1);
30    return dp[p][q][r][last];
31 }
32 int countUtil(int p, int q, int r)
33 {
34     memset(dp, -1, sizeof(dp));
35     return countWays(p, q, r, 0) + // Last required balls is type P
36             countWays(p, q, r, 1) + // Last required balls is type Q
37             countWays(p, q, r, 2); // Last required balls is type R
38 }
```

Length of longest Palindromic Substring

```
1 int dp[1001][1001];
2 int ans=1;
3 bool isPalindrome(string s,int i,int j)
4 {
5     while(i<=j)
6     {
7         if(s[i]==s[j])
8         {
9             i++;j--;
10        }
11        else
12            return false;
13    }
14    return true;
15 }
16 int solve(string s,int i,int j)
17 {
18     if(i>j)
19         return 0;
20     else if(isPalindrome(s,i,j))
21         return (j-i+1);
22     else if(i==j)
23         return 1;
24     for(int k=i;k<=j-1;k++)
25     {
26         int temp=max(solve(s,i,k),solve(s,k+1,j));
27         ans=max(ans,temp);
28     }
29     return ans;
30 }
31 int longestPalindrome(string s) {
32     memset(dp,-1,sizeof(dp));
33     return solve(s,0,s.size()-1);
34 }
```

Longest Palindromic Substring

		b	b	a	b	b
b	1	1	8	0	1	
① b	0	1	0	1	0	
② a	0	0	1	0	0	
b	0	0	0	1	1	
5	0	0	0	0	1.	



void lp()

```
ll n = s.length();
bool dp[n][n];
ll start=0, maxlen=1;
maxlen(dp, false, 11209(dp));
for(i=0; i<n; i++)
    dp[i][i]=true;
for (int i=0; i<(n-1); i++) // length of 2
{
    if (s[i]==s[i+1])
        {
            dp[i][i+1]=true;
            start=i;
            maxlen=2;
        }
}
for (ll l=3; l<=n; l++) // length of 3
{
    for (ll i=0; i<n-l; i++)
    {
        j = i+l-1;
        if (s[i]==s[j] && dp[i+1][j-1])
            dp[i][j]=true;
    }
}
PrintSubstr(s, start, maxlen);
cout<<endl;
```

Burst Balloons

```
1 const int N = 5e2 + 5;
2
3 class Solution {
4 public:
5
6     int dp[N][N];
7
8     int rec(vector<int>& nums, int x, int y)
9     {
10         if (x == y)
11             return 0;
12         int &ret = dp[x][y];
13         if (~ret)
14             return ret;
15         int ans = 0;
16         for (int k = x; k < y; ++k)
17         {
18             int temp = rec(nums, x, k) + rec(nums, k + 1, y) +
19                         nums[x-1] * nums[k] * nums[y];
20             ans=max(ans,temp);
21         }
22         ret = ans;
23     }
24
25     int maxCoins(vector<int>& nums) {
26         nums.push_back(1);
27         reverse(nums.begin(), nums.end());
28         nums.push_back(1);
29         reverse(nums.begin(), nums.end());
30         int n = nums.size();
31         memset(dp, -1, sizeof(dp));
32         return rec(nums, 1, n - 1);
33     }
34 };
```

EGG DROPPING PROBLEM:

```
1 int solve(int e,int f)
2 {
3     if(e==1)
4         return f;
5     if(f==0||f==1)
6         return f;
7     if(dp[e][f]!=-1)
8         return dp[e][f];
9     int mn=INT_MAX;
10    for(int k=1;k<=f;k++)
11    {
12        int low,high;
13        if(dp[e-1][k-1]!=-1)
14            low=dp[e-1][k-1];
15        else
16        {
17            low=solve(e-1,k-1);
18            dp[e-1][k-1]=low;
19        }
20        if(dp[e][f-k]!=-1)
21            high=dp[e][f-k];
22        else
23        {
24            high=solve(e,f-k);
25            dp[e][f-k]=high;
26        }
27        int temp=1+max(low,high); //max islie kyuki worst case ka pucha h
28        mn=min(mn,temp);
29    }
30    return dp[e][f]=mn;
31 }
```

Longest alternating subsequence:

```
1 int zzis(int arr[], int n)
2 {
3     /*las[i][0] = Length of the longest
4      alternating subsequence ending at
5      index i and last element is greater
6      than its previous element
7      las[i][1] = Length of the longest
8      alternating subsequence ending
9      at index i and last element is
10     smaller than its previous element */
11    int las[n][2];
12    for(int i = 0; i < n; i++)
13        las[i][0] = las[i][1] = 1;
14    int res = 1;
15    for(int i = 1; i < n; i++)
16    {
17        for(int j = 0; j < i; j++)
18        {
19            // If arr[i] is greater, then
20            // check with las[j][1]
21            if (arr[j] < arr[i] &&
22                las[i][0] < las[j][1] + 1)
23                las[i][0] = las[j][1] + 1;
24            // If arr[i] is smaller, then
25            // check with las[j][0]
26            if(arr[j] > arr[i] &&
27                las[i][1] < las[j][0] + 1)
28                las[i][1] = las[j][0] + 1;
29        }
30        if (res < max(las[i][0], las[i][1]))
31            res = max(las[i][0], las[i][1]);
32    }
33    return res;
```

Weighted Job Scheduling

har index k liye 2 option h ya th lo ya mtlo jese **Knapsack** m hota h

```
1 int latestNonConflict(Job arr[], int i)
2 {
3     for (int j=i-1; j>=0; j--)
4     {
5         if (arr[j].finish <= arr[i].start)
6             return j;
7     }
8     return -1;
9 }
10 int findMaxProfit(Job arr[], int n)
11 {
12     sort(arr, arr+n, jobComparataor);
13     int *table = new int[n];
14     table[0] = arr[0].profit;
15     for (int i=1; i<n; i++)
16     {
17         // Find profit including the current job
18         int inclProf = arr[i].profit;
19         int l = latestNonConflict(arr, i);
20         if (l != -1)
21             inclProf += table[l];
22         table[i] = max(inclProf, table[i-1]);
23     }
24     int result = table[n-1];
25     delete[] table;
26     return result;
27 }
```

Coin game winner where every player has three choices

```
1 bool findWinner(int x, int y, int n)
2 {
3     int dp[n + 1];
4     dp[0] = false;
5     dp[1] = true;
6     for (int i = 2; i <= n; i++) {
7         if (i - 1 >= 0 and !dp[i - 1])
8             dp[i] = true;
9         else if (i - x >= 0 and !dp[i - x])
10            dp[i] = true;
11        else if (i - y >= 0 and !dp[i - y])
12            dp[i] = true;
13        else
14            dp[i] = false;
15    }
16    return dp[n];
17 }
```

Maximum profit by buying and selling a share at most twice

```
int helper(vector<int>&prices,int day,int transaction_left,vector<vector<int>>&dp)
{
    if(day==prices.size() || transaction_left==0)
        return 0;

    if(dp[day][transaction_left]!=-1)
        return dp[day][transaction_left];

    //no transaction
    int ans1=helper(prices,day+1,transaction_left,dp);

    int ans2=0;
    //transaction buy sell buy sell
    if(transaction_left%2==0)//buy
    {
        ans2=-prices[day]+helper(prices,day+1,transaction_left-1,dp);
    }
    else //sell
    {
        ans2=prices[day]+helper(prices,day+1,transaction_left-1,dp);
    }
    return dp[day][transaction_left]=max(ans1,ans2);
}
int maxProfit(vector<int>& prices) {
    vector<vector<int>>dp(prices.size(),vector<int>(5,-1));
    return helper(prices,0,4,dp);
}
```

Best Time to Buy and Sell Stock IV

```
int helper(vector<int>&prices,int day,int transaction_left,vector<vector<int>>&dp)
{
    if(day==prices.size() || transaction_left==0)
        return 0;

    if(dp[day][transaction_left]!=-1)
        return dp[day][transaction_left];

    //no transaction
    int ans1=helper(prices,day+1,transaction_left,dp);

    int ans2=0;
    //transaction buy sell buy sell
    if(transaction_left%2==0)//buy
    {
        ans2=-prices[day]+helper(prices,day+1,transaction_left-1,dp);
    }
    else //sell
    {
        ans2=prices[day]+helper(prices,day+1,transaction_left-1,dp);
    }
    return dp[day][transaction_left]=max(ans1,ans2);
}
int maxProfit(int k, vector<int>& prices) {
    int trans=2*k;
    vector<vector<int>>dp(prices.size(),vector<int>(2*k+1,-1));
    return helper(prices,0,2*k,dp);
}
```

Best Time to Buy and Sell Stock with Cooldown

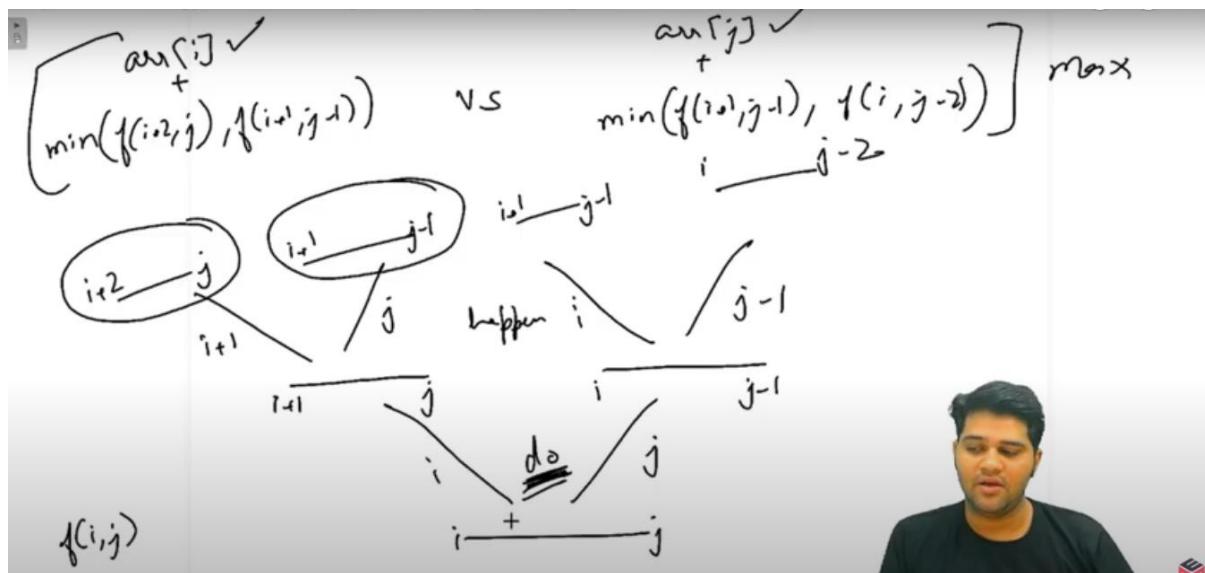
```
int helper(vector<int>&prices,int day,vector<vector<int>>&dp,bool buy)
{
    if(day>=prices.size())
        return 0;

    if(dp[day][buy]!=-1)
        return dp[day][buy];

    //no transaction
    int ans1=helper(prices,day+1,dp,buy);

    int ans2=0;
    //transaction buy sell buy sell
    if(buy==true)//buy
    {
        ans2=-prices[day]+helper(prices,day+1,dp,1-buy);
    }
    else //sell
    {
        ans2=prices[day]+helper(prices,day+2,dp,1-buy);
    }
    return dp[day][buy]=max(ans1,ans2);
}
int maxProfit(vector<int>& prices) {
    vector<vector<int>>dp(prices.size(),vector<int>(2,-1));
    bool buy=true;
    return helper(prices,0,dp,buy);
}
```

Optimal Strategy For A Game



```
public static void optimalStrategy(int[] arr) {
    int[][] dp = new int[arr.length][arr.length];

    for(int g = 0; g < dp.length; g++){
        for(int i = 0, j = g; j < dp.length; i++, j++){
            if(g == 0){
                dp[i][j] = arr[i];
            } else if(g == 1){
                dp[i][j] = Math.max(arr[i], arr[j]);
            } else {
                int val1 = arr[i] + Math.min(dp[i + 2][j], dp[i + 1][j - 1]);
                int val2 = arr[j] + Math.min(dp[i + 1][j - 1], dp[i][j - 2]);
                int val = Math.max(val1, val2);
                dp[i][j] = val;
            }
        }
    }

    System.out.println(dp[0][arr.length - 1]);
}
```

Find if a string is interleaved of two other strings

If done with two pointers then the test cases where the strings contain duplicates will fail.

For example, if we consider A = “AAB”, string B = “AAC” and string C = “AACCAAB”

```
int dp[501][501][501];
int n1,n2,n3;
bool helper(string s1,string s2,string s3,int n1,int n2,int n3)
{
    if(s1.size()==0)
    {
        if(s2==s3)
            return true;
        else
            return false;
    }
    else if(s2.size()==0)
    {
        if(s1==s3)
            return true;
        else
            return false;
    }
    if(dp[n1][n2][n3]>=0)
        return (dp[n1][n2][n3]==1)? true: false;
    bool a=false,b=false;
    if(s1[0]==s3[0]&&helper(s1.substr(1),s2,s3.substr(1),n1-1,n2,n3-1))
        a=true;
    if(s2[0]==s3[0]&&helper(s1,s2.substr(1),s3.substr(1),n1,n2+1,n3+1))
        b=true;
    return dp[n1][n2][n3]=(a||b);
}
bool isInterleave(string s1, string s2, string s3) {
    if(s1.size()+s2.size()!=s3.size())
        return false;
    memset(dp,-1,sizeof(dp));
    n1=s1.size();
    n2=s2.size();
    n3=s3.size();
    return helper(s1,s2,s3,n1,n2,n3);
}
```

Dice Throw:

```
int solve(int d,int f,int target)
{
    if(d==0&&target==0)
        return 1;
    else if(d==0| target==0)
        return 0;
    else if(d==1&&target<=f)
        return 1;
    else if(d==1&&target>f)
        return 0;
    if(dp[d][target]!=0)
        return dp[d][target]-1;
    int res=0;
    for(int i=1;i<=f&&i<target;i++)
        res=(res+solve(d-1,f,target-i)) % 1000000007;
    dp[d][target]=res+1;
    return res;
}
int numRollsToTarget(int d, int f, int target) {
    memset(dp,0,sizeof(dp));
    return solve(d,f,target);
}
```

For optimisation:

$$\begin{aligned} \text{mem}[i][j] &= \sum_{k=1}^{k=m} \text{mem}[i-1][j-k] \\ &= \cancel{\text{mem}[i-1][j-1]} + \cancel{\text{mem}[i-1][j-2]} + \cancel{\text{mem}[i-1][j-3]} - \\ &\quad \cancel{\text{mem}[i-1][j-m]} \\ \text{mem}[i-1][j-1] &= \cancel{\text{mem}[i-1][j-2]} + \cancel{\text{mem}[i-1][j-3]} - \\ &\quad \cancel{\text{mem}[i-1][j-m]} + \cancel{\text{mem}[i-1][j-m-1]} \\ \text{mem}[i][j] &= \cancel{\text{mem}[i-1][j-1]} + \cancel{\text{mem}[i][j-1]} - \cancel{\text{mem}[i-1][j-m-1]} \\ &\quad \text{if } j-m-1 \geq 0 \end{aligned}$$

```
long findWays(int f, int d, int s)
{
    long mem[d + 1][s + 1];
    memset(mem, 0, sizeof mem);
    mem[0][0] = 1;
    // Iterate over dices
    for (int i = 1; i <= d; i++)
    {
        // Iterate over sum
        for (int j = i; j <= s; j++)
        {
            mem[i][j] = mem[i][j - 1] + mem[i - 1][j - 1];
            if (j - f - 1 >= 0)
                mem[i][j] -= mem[i - 1][j - f - 1];
        }
    }
    return mem[d][s];
}
```

Box Stacking Problem:

```
//sort boxes in decreasing order of base area.
int compare (const void *a, const void * b)
{
    return ( (*(Box *)b).d * (*(Box *)b).w ) -
           ( (*(Box *)a).d * (*(Box *)a).w );
}
int maxStackHeight( Box arr[], int n )
{
    Box rot[3*n];
    int index = 0;
    for (int i = 0; i < n; i++)
    {
        rot[index].h = arr[i].h;rot[index].d = max(arr[i].d, arr[i].w);rot[index].w = min(arr[i].d, arr[i].w);
        index++;
        rot[index].h = arr[i].w;rot[index].d = max(arr[i].h, arr[i].d);rot[index].w = min(arr[i].h, arr[i].d);
        index++;
        rot[index].h = arr[i].d;rot[index].d = max(arr[i].h, arr[i].w);rot[index].w = min(arr[i].h, arr[i].w);
        index++;
    }
    n = 3*n;
    qsort (rot, n, sizeof(rot[0]), compare);
    /*msh[i] --> Maximum possible Stack Height with box i on top */
    int msh[n];
    for (int i = 0; i < n; i++ )
        msh[i] = rot[i].h;
    for (int i = 1; i < n; i++ )
        for (int j = 0; j < i; j++ )
            if ( rot[i].w < rot[j].w &&
                 rot[i].d < rot[j].d &&
                 msh[i] < msh[j] + rot[i].h
                )
            {
                msh[i] = msh[j] + rot[i].h;
            }
    return *mxelement(msh,msh+n);
}
```

House Robber II

Suppose there are n houses, since house 0 and $n - 1$ are now neighbors, we cannot rob them together and thus the solution is now the maximum of

1. **Rob houses 0 to $n - 2$;**
2. **Rob houses 1 to $n - 1$;**

```
int rob(vector<int>& nums) {
    int n=nums.size();
    if (n < 2) return n ? nums[0] : 0;
    int dp[n][2];
    //0 => current included;
    //1=> current not_included;
    memset(dp,0,sizeof(dp));
    dp[0][0]=nums[0];
    for(int i=1;i<n-1;i++)
    {
        dp[i][0]=nums[i]+dp[i-1][1];
        dp[i][1]=max(dp[i-1][0],dp[i-1][1]);
    }
    int ans=max(dp[n-2][0],dp[n-2][1]);
    memset(dp,0,sizeof(dp));
    dp[1][0]=nums[1];
    for(int i=2;i<n;i++)
    {
        dp[i][0]=nums[i]+dp[i-1][1];
        dp[i][1]=max(dp[i-1][0],dp[i-1][1]);
    }
    ans=max(ans,max(dp[n-1][0],dp[n-1][1]));
    return ans;
}
```

Maximum sum rectangle in a 2D matrix

```
int kadanes(vector<int>&a)
{
    int size=a.size();
    int max_so_far = INT_MIN, max_ending_here = 0;
    for (int i = 0; i < size; i++)
    {
        max_ending_here = max_ending_here + a[i];
        if (max_so_far < max_ending_here)
            max_so_far = max_ending_here;
        if (max_ending_here < 0)
            max_ending_here = 0;
    }
    return max_so_far;
}
int maximumSumRectangle(int n, int m, vector<vector<int>>&a) {
    int ans=INT_MIN;
    for(int r1=0;r1<n;r1++)
    {
        vector<int>temp(m,0);
        for(int r2=r1;r2<n;r2++)
        {
            for(int c=0;c<m;c++)
                temp[c]+=a[r2][c];
            ans=max(ans,kadanes(temp));
        }
    }
    return ans;
}.
```

Recursive Expression Matching

```
bool isMatch(string s, string p) {
    int n=s.size();
    int m=p.size();
    bool dp[n+1][m+1];
    memset(dp, false, sizeof(dp));
    dp[0][0]=true;
    for(int i=2;i<=n;i++)
    {
        if(p[i-1]=='*')
            dp[0][i]=dp[0][i-2];
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            if(s[i-1]==p[j-1]||p[j-1]=='.')
                dp[i][j]=dp[i-1][j-1];
            else if(p[j-1]=='*')
            {
                //4 cases cahr j-2==any char other than i-1
                //or j-2==i-1||j-2=='.'||j-2==i||same ho tb bhi j-2 krna h jese yh case
                //a
                //|aa*
                dp[i][j] = dp[i][j-2];
                if(p[j-2]==s[i-1]||p[j-2]=='.')
                    dp[i][j] = dp[i][j]||dp[i-1][j]||dp[i][j-1];
            }
        }
    }
    return dp[n][m];
}
```

BINARY TREES:

C++ STL Time Complexity and Under the hood Data Structure Implementation

1. unordered_map

DS : Hash Table

Search : $O(1)$

Insert : $O(1)$

Delete : $O(1)$

2. map

DS : Red Black Trees

Search : $O(\log n)$

Insert : $O(\log n)$

Delete : $O(\log n)$

3. priority_queue

DS : Heap

Insert : $O(\log n)$

Extract : $O(\log n)$

Top : $O(1)$

4. list

DS : doubly LinkedList

insert (front/back) : $O(1)$ | insert at : $O(n)$

delete : $O(1)$ | delete at : $O(n)$

search : $O(n)$



Home



My Network



Post



Notifications



Jobs



Search



4. list

DS : doubly LinkedList

insert (front/back) : O(1) | insert at : O(n)
delete : O(1) | delete at : O(n)
search : O(n)

5. vector

DS : dynamic array

insert (back) : O(1)
delete (back) : O(1)
search : O(n)

6. unordered_set

DS : Hash Table
Search : O(1)
Insert : O(1)
Delete : O(1)

7. set

DS : Red Black Trees
Search : O(log n)
Insert : O(log n)
Delete : O(log n)



Home



My Network



Post



Notifications



Jobs

Diameter of Binary Tree:

```
1 pair<int, int> heightDiameter(BinaryTreeNode<int>* root) {
2     if (root == NULL) {
3         pair<int, int> p;
4         p.first = 0;
5         p.second = 0;
6         return p;
7     }
8     pair<int, int> leftAns = heightDiameter(root->left);
9     pair<int,int> rightAns = heightDiameter(root->right);
10    int ld = leftAns.second;
11    int lh = leftAns.first;
12    int rd = rightAns.second;
13    int rh = rightAns.first;
14
15    int height = 1 + max(lh, rh);
16    int diameter = max(lh + rh, max(ld, rd));
17    pair<int, int> p;
18    p.first = height;
19    p.second = diameter;
20    return p;
21 }
```

Iterative Pre, Post and In-order traversal in Binary Tree:

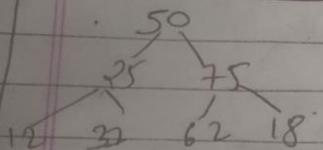
Iterative Pre, Post and In-order traversal in Binary Tree | Solution | Data Structures and Algorithm

```
90
91    while(st.size() > 0){
92        Pair top = st.peek();
93        if(top.state == 1){ // pre, s++, left
94            pre += top.node.data + " ";
95            top.state++;
96
97            if(top.node.left != null){
98                Pair lp = new Pair(top.node.left, 1);
99                st.push(lp);
100           }
101        } else if(top.state == 2){ // in, s++, right
102            in += top.node.data + " ";
103            top.state++;
104
105            if(top.node.right != null){
106                Pair rp = new Pair(top.node.right, 1);
107                st.push(rp);
108            }
109        } else { // post, pop
110            post += top.node.data + " ";
111            st.pop();
112        }
113    }
114
115 }
```

Iterative Pre, Post and In-order traversal in Binary Tree | Solution | Data Structures and Algorithm

```
82-
83     public static void iterativePrePostInTraversal(Node node) {
84         Stack<Pair> st = new Stack<>();
85         Pair rtp = new Pair(node, 1);
86         st.push(rtp);
87
88         String pre = "";
89         String in = "";
90         String post = "";
91
92         while(st.size() > 0){
93             Pair top = st.peek();
94             if(top.state == 1){ // pre, s++, left
95                 pre += top.node.data + " ";
96                 top.state++;
97
98                 if(top.node.left != null){
99                     Pair lp = new Pair(top.node.left, 1);
100                    st.push(lp);
101                }
102            } else if(top.state == 2){ // in, s++, right
103                in += top.node.data + " ";
104                top.state++;
105            } else { // post, pop
106                post += top.node.data + " ";
107                st.pop();
108            }
109        }
110    }
```

Iterative Pre Post & In Order Tree



if 1 \Rightarrow Pre, state++, left
if 2 \Rightarrow In, state++, Right
if 3 \Rightarrow Post, pop



Pre (state 1) \Rightarrow 50 25 12 37 75 62 18

In (state 2) \Rightarrow 12 25 37 50 62 75 18

Post (state 3) \Rightarrow 12 37 25 62 18 75 50

Left View of Binary Tree:

```
1 void view(Node*root,vector<int>&ans,int &max,int curr)
2 {
3     if(root==NULL)
4         return;
5     if(curr>max)
6     {
7         ans.push_back(root->data);
8         max=curr;
9     }
10    view(root->left,ans,max,curr+1);
11    view(root->right,ans,max,curr+1);
12 }
13 vector<int> leftView(Node *root)
14 {
15     vector<int>ans;
16     int max=0;
17     int curr=1;
18     view(root,ans,max,curr);
19     return ans;
20 }
```

Vertical Order Traversal of Binary Tree

```
vector<vector<int> > Solution::verticalOrderTraversal(TreeNode* A) {  
    vector<vector<int> >vect;  
    if(A==NULL)  
        return vect;  
    map<int, vector<int> >hash;  
    queue<pair<TreeNode*,int> >q;  
    q.push(make_pair(A,0));  
    while(!q.empty())  
    {  
        pair<TreeNode*,int>p1 = q.front();  
        q.pop();  
        TreeNode* temp = p1.first;  
        int count = p1.second;  
        hash[count].push_back(temp->val);  
        if(temp->left!=NULL)  
            q.push(make_pair(temp->left,count-1));  
        if(temp->right!=NULL)  
            q.push(make_pair(temp->right,count+1));  
        free(temp);  
    }  
    map<int,vector<int> >::iterator it;  
    for(it=hash.begin();it!=hash.end();it++)  
        vect.push_back(it->second);  
    return vect;  
}
```

Cousins in Binary Tree

```
vector<int> Solution::solve(TreeNode* A, int B) {
    queue<TreeNode*>q;
    vector<int>v;
    if(A==NULL)
        return v;
    q.push(A);
    bool flag=false;
    while(!q.empty())
    {
        int n=q.size();
        for(int i=0;i<n;i++)
        {
            TreeNode* temp=q.front();
            q.pop();
            if((temp->left&&temp->left->val==B)|| (temp->right&&temp->right->val==B))
                flag=true;
            else
            {
                if(temp->left)q.push(temp->left);
                if(temp->right)q.push(temp->right);
            }
        }
        if(flag)
        {
            while(!q.empty())
            {
                v.push_back(q.front()->val);
                q.pop();
            }
            return v;
        }
    }
    return v;
}
```

TOP VIEW OF BINARY TREE.

```
1 map<int,pair<int,int>>m;
2     void helper(Node*root,int depth,int hd)
3     {
4         if(root==NULL)
5             return;
6         if(m[hd].first)
7         {
8             pair<int,int>p=m[hd];
9             if(p.second>depth)
10             {
11                 m[hd]={root->data,depth};
12             }
13         }
14         else
15         {
16             m[hd]={root->data,depth};
17         }
18         helper(root->left,depth+1,hd-1);
19         helper(root->right,depth+1,hd+1);
20     }
21     vector<int> topView(Node *root)
22     {
23         vector<int>ans;
24         ans.clear();
25         if(root==NULL)
26             return ans;
27         helper(root,0,0);
28         for(auto it=m.begin();it!=m.end();it++)
29         {
30             ans.push_back(it->second.first);
31         }
32         return ans;
33     }
34 }
```

Check for Balanced Tree:

```
1 class Pair
2 {
3     public:
4     bool isBalance;
5     int height;
6 };
7 Pair helper(BinaryTreeNode<int> *root)
8 {
9     if(root==NULL)
10    {
11        Pair p;
12        p.isBalance=true;
13        p.height=0;
14        return p;
15    }
16    Pair leftans=helper(root->left);
17    Pair rightans=helper(root->right);
18    int h=1+max(leftans.height,rightans.height);
19    bool ans=leftans.isBalance and rightans.isBalance;
20    Pair p;
21    if(ans==false)
22        p.isBalance=ans;
23    else
24    {
25        if(abs(leftans.height-rightans.height)<=1)
26            p.isBalance=true;
27        else
28            p.isBalance=false;
29    }
30    p.height=h;
31    return p;
32 }
33 bool isBalanced(BinaryTreeNode<int> *root) {
34     return helper(root).isBalance;
```

DIAGONAL TRAVERSAL:

Diagonal Traversal

vector<int> diatree (root)

{

Queue<Node*>q;

q.push(root); vector<int> v;

while (!q.empty())

{

int p = q.size();

while (p -)

{

Treenode* front = q.front();

L-BPL;

while (front)

{

v.pb(front->val);

if (front->left)

q.push(left);

} } ;

return v; front = front->right;

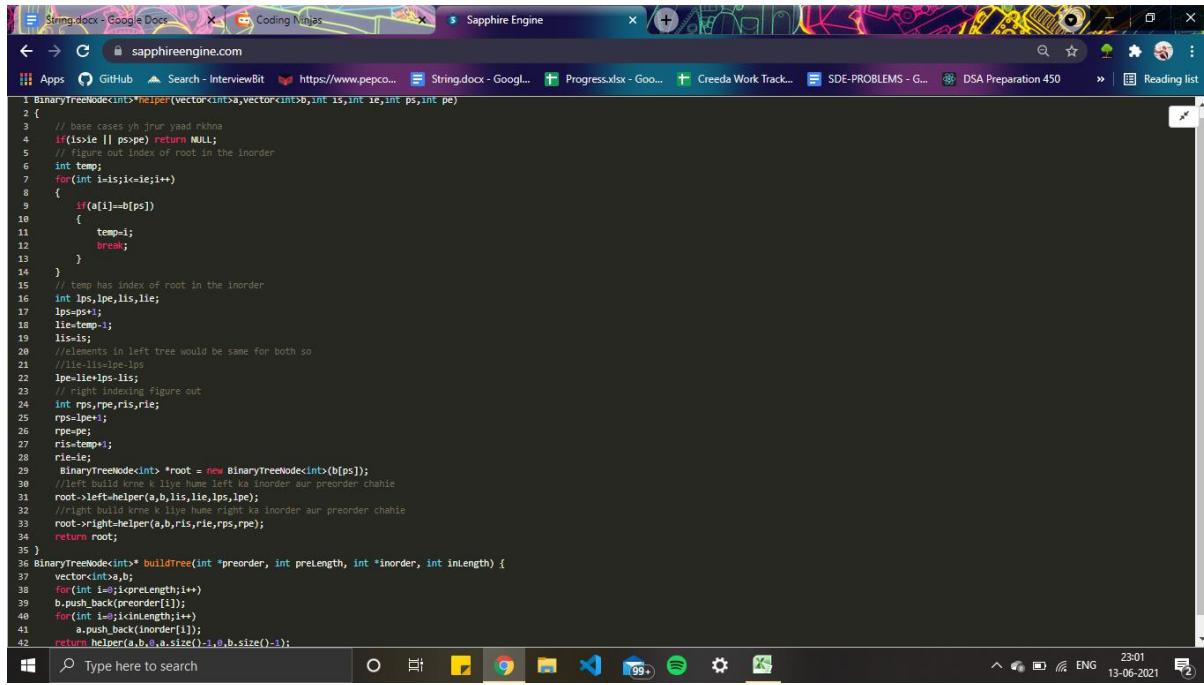
Boundary Traversal of binary tree:

```
5     printLeaves(root->left);
6     if (!(root->left) && !(root->right))
7         printf("%d ", root->data);
8     printLeaves(root->right);
9 }
10 void printBoundaryLeft(struct node* root)
11 {
12     if (root == NULL)
13         return;
14     if (root->left) {
15         printf("%d ", root->data);
16         printBoundaryLeft(root->left);
17     }
18     else if (root->right) {
19         printf("%d ", root->data);
20         printBoundaryLeft(root->right);
21     }
22 }
23 void printBoundaryRight(struct node* root)
24 {
25     if (root == NULL)
26         return;
27     if (root->right) {
28         printBoundaryRight(root->right);
29         printf("%d ", root->data);
30     }
31     else if (root->left) {
32         printBoundaryRight(root->left);
33         printf("%d ", root->data);
34     }
35 }
36 void printBoundary(struct node* root)
37 {
38     if (root == NULL)
39         return;
40     printf("%d ", root->data);
41     printBoundaryLeft(root->left);
42     printLeaves(root->left);
43     printLeaves(root->right);
44     printBoundaryRight(root->right);
45 }
```

Binary Tree to DLL

```
1 Node*head=NULL;
2     Node*tail=NULL;
3     void helper(Node*root)
4     {
5         if(root==NULL)
6             return;
7         if(root->left)
8             helper(root->left);
9         if(head==NULL)
10        {
11            head=root;
12            tail=root;
13        }
14        else{
15            tail->right=root;
16            root->left=tail;
17            tail=root;
18        }
19        if(root->right)
20            helper(root->right);
21    }
22    Node * bToDLL(Node *root)
23    {
24        if(root==NULL)
25            return NULL;
26        helper(root);
27        return head;
28    }.
```

Construct Binary tree from Inorder and preorder traversal:



The screenshot shows a Windows desktop environment with a browser window open at sapphireengine.com. The browser tab bar includes links for Apps, GitHub, Search - InterviewBit, https://www.pepc..., String.docx - Google..., Progress.xlsx - Google..., Creeda Work Track..., SDE-PROBLEMS - Google..., DSA Preparation 450, and Reading list. The main content area of the browser displays C++ code for constructing a binary tree from inorder and preorder traversal. The code uses vectors for inorder (a) and preorder (b) arrays, and pointers for lps (left pointer), lpe (left end), rps (right pointer), and rpe (right end). It identifies the root element in the preorder array and then recursively constructs left and right subtrees using helper functions. The code is annotated with comments explaining the logic for finding the root index in the inorder array and splitting it into left and right halves.

```
1 BinaryTreeNode<int>* helper(vector<int>a, vector<int>b, int ls, int le, int ps, int pe)
2 {
3     // base cases yh jirur yaad rkhn
4     if(ls>le || ps>pe) return NULL;
5     // Figure out index of root in the inorder
6     int temp;
7     for(int i=s;i<e;i++)
8     {
9         if(a[i]==b[ps])
10        {
11            temp=i;
12            break;
13        }
14    }
15    // temp has index of root in the inorder
16    int lps,lpe,lis,lie;
17    lps=ls;
18    lie=temp-1;
19    lis=s;
20    //elements in left tree would be same for both so
21    //lie-ls=lpe-lps
22    lpe=le-lps-lis;
23    // right indexing figure out
24    int rps,rpe,ris,rie;
25    rps=ps+1;
26    rpe=pe;
27    ris=temp+1;
28    rie=le;
29    BinaryTreeNode<int> *root = new BinaryTreeNode<int>(b[ps]);
30    //left build krene k liye hume left ka inorder aur preorder chahiye
31    root->left=helper(a,ls,lis,lie,lps,lpe);
32    //right build krene k liye hume right ka inorder aur preorder chahiye
33    root->right=helper(a,b,ris,rie,rps,rpe);
34    return root;
35 }
36 BinaryTreeNode<int>* buildTree(int *preorder, int preLength, int *inorder, int inLength) {
37     vector<int>a,b;
38     for(int i=0;i<preLength;i++)
39         b.push_back(preorder[i]);
40     for(int i=0;i<inLength;i++)
41         a.push_back(inorder[i]);
42     return helper(a,b,a.size()-1,b.size()-1);
}
```

Check if a given Binary Tree is SumTree:

```
1 int isSumTree(node* node)
2 {
3     int ls; // for sum of nodes in left subtree
4     int rs; // for sum of nodes in right subtree
5     if(node == NULL || isLeaf(node))
6         return 1;
7
8     if( isSumTree(node->left) && isSumTree(node->right))
9     {
10
11         // Get the sum of nodes in left subtree
12         if(node->left == NULL)
13             ls = 0;
14         else if(isLeaf(node->left))
15             ls = node->left->data;
16         else
17             ls = 2 * (node->left->data);
18
19         // Get the sum of nodes in right subtree
20         if(node->right == NULL)
21             rs = 0;
22         else if(isLeaf(node->right))
23             rs = node->right->data;
24         else
25             rs = 2 * (node->right->data);
26         return(node->data == ls + rs);
27     }
28     return 0;
29 }
```

Find Duplicate Subtrees:

```
1 string helper(TreeNode*root,unordered_map<string,int>&m,vector<TreeNode*>&res)
2 {
3     if(root==NULL)
4         return "#";
5     string s=to_string(root->val)+','+helper(root->left,m,res)+','+helper(root->right,m,res);
6     if(m[s]==1)
7         res.push_back(root);
8     m[s]++;
9     return s;
10 }
11 class Solution {
12 public:
13     vector<TreeNode*> findDuplicateSubtrees(TreeNode* root) {
14         vector<TreeNode*>res;
15         unordered_map<string,int>m;
16         helper(root,m,res);
17         return res;
18     }
19 };
```



Find largest subtree sum in a tree

A screenshot of a Windows desktop environment. The taskbar at the bottom shows various pinned icons and the date/time (14-06-2021 14:32). The main window is a code editor displaying C++ code for finding the largest subtree sum in a binary tree. The code uses a recursive utility function to calculate the sum of all nodes in a subtree rooted at a given node. The code editor has syntax highlighting and a status bar indicating the file is saved.

```
1 int findLargestSubtreeSumUtil(Node* root, int& ans)
2 {
3     if (root == NULL)
4         return 0;
5     int currSum = root->key +
6             findLargestSubtreeSumUtil(root->left, ans)
7             + findLargestSubtreeSumUtil(root->right, ans);
8     ans = max(ans, currSum);
9     return currSum;
10 }
```

Maximum sum of nodes in Binary tree such that no two are adjacent.

A screenshot of a web browser window titled "Maximum sum of nodes in Binary tree such that no two are adjacent.". The address bar shows "sapphireengine.com". The page content displays the following C++ code:

```
1 pair<int, int> maxSumHelper(Node *root)
2 {
3     if (root==NULL)
4     {
5         pair<int, int> sum(0, 0);
6         return sum;
7     }
8     pair<int, int> sum1 = maxSumHelper(root->left);
9     pair<int, int> sum2 = maxSumHelper(root->right);
10    pair<int, int> sum;
11    sum.first = sum1.second + sum2.second + root->data;
12    sum.second = max(sum1.first, sum1.second) +
13                  max(sum2.first, sum2.second);
14
15    return sum;
16 }
```

The browser interface includes a toolbar with various icons, a search bar at the bottom, and a taskbar at the very bottom showing other open applications like GitHub, Google Docs, and Microsoft Word.

Lowest Common Ancestor in a Binary Tree:

A screenshot of a web browser window titled "Lowest Common Ancestor in a Binary Tree". The address bar shows "sapphireengine.com". The page content displays the following C++ code:

```
1 Node* lca(Node* root ,int n1 ,int n2 )
2 {
3     if(root==NULL)
4         return NULL;
5     if(root->data==n2 || root->data==n1)
6         return root;
7     Node*left=lca(root->left,n1,n2);
8     Node*right=lca(root->right,n1,n2);
9     if(left and right)
10        return root;
11        return (left!=NULL)?left:right;
12 }
```

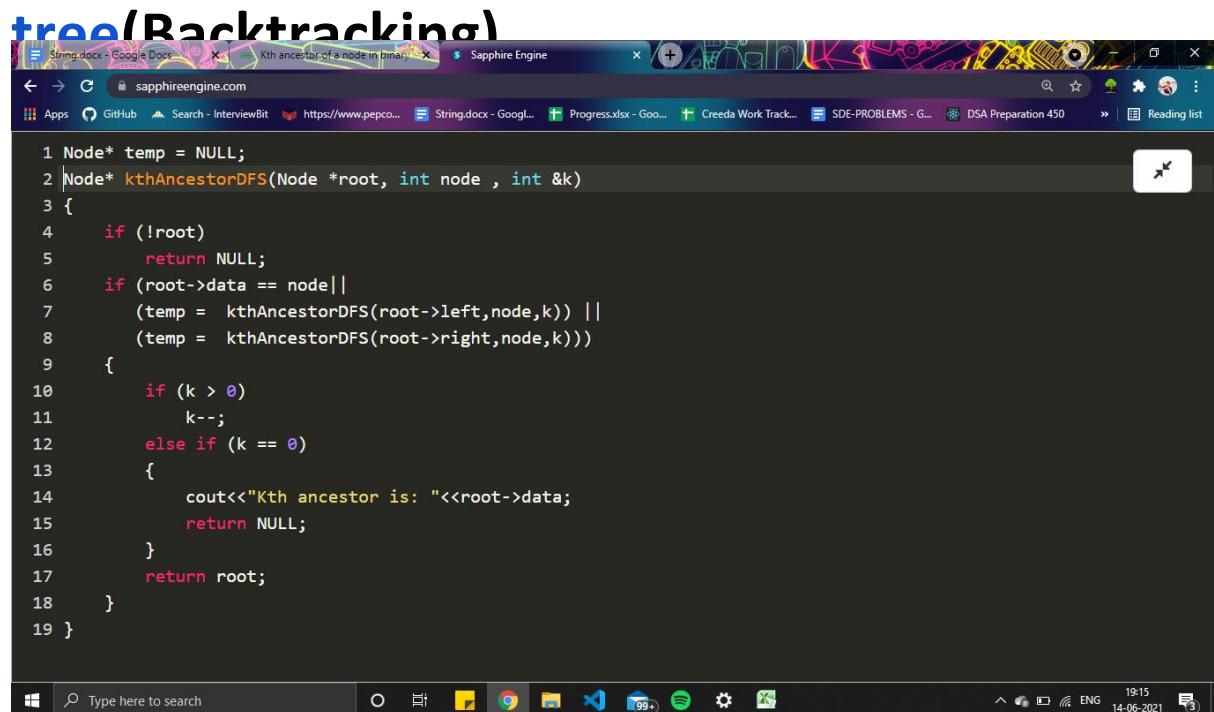
The browser interface includes a toolbar with various icons, a search bar at the bottom, and a taskbar at the very bottom showing other open applications like GitHub, Google Docs, and Microsoft Word.

Find distance between two nodes of a Binary Tree :

```
String.docx - Google Docs | Find distance between two nodes | (16) WhatsApp | Sapphire Engine | Apps GitHub Search - InterviewBit https://www.pepcoding.com String.docx - Google Sheets Progress.xlsx - Google Sheets Creeda Work Track... SDE-PROBLEMS - Google Sheets DSA Preparation 450 | Reading list
1 int findLevel(Node *root, int k, int level)
2 {
3     // Base Case
4     if (root == NULL)
5         return -1;
6     if (root->key == k)
7         return level;
8
9     int l = findLevel(root->left, k, level+1);
10    return (l != -1)? l : findLevel(root->right, k, level+1);
11 }
12 Node *findDistUtil(Node* root, int n1, int n2, int &d1,
13                      int &d2, int &dist, int lvl)
14 {
15     // Base case
16     if (root == NULL) return NULL;
17     if (root->key == n1)
18     {
19         d1 = lvl;
20         return root;
21     }
22     if (root->key == n2)
23     {
24         d2 = lvl;
25         return root;
26     }
27     Node *left_lca = findDistUtil(root->left, n1, n2,
28                                    d1, d2, dist, lvl+1);
29     Node *right_lca = findDistUtil(root->right, n1, n2,
30                                    d1, d2, dist, lvl+1);
31     if (left_lca && right_lca)
32     {
33         dist = d1 + d2 - 2*lvl;
34         return root;
35     }
36     return (left_lca != NULL)? left_lca: right_lca;
37 }
38 int findDistance(Node *root, int n1, int n2)
39 {
40     int d1 = -1, d2 = -1, dist;
41     Node *lca = findDistUtil(root, n1, n2, d1, d2,
42                             dist, 1);
43     if (d1 != -1 && d2 != -1)
44         return dist;
45     if (d1 != -1)
46     {
47         dist = findLevel(lca, n2, 0);
48         return dist;
49     }
50     if (d2 != -1)
51     {
52         dist = findLevel(lca, n1, 0);
53         return dist;
54     }
55     return -1;
56 }
```

```
String.docx - Google Docs | Find distance between two nodes | (16) WhatsApp | Sapphire Engine | Apps GitHub Search - InterviewBit https://www.pepcoding.com String.docx - Google Sheets Progress.xlsx - Google Sheets Creeda Work Track... SDE-PROBLEMS - Google Sheets DSA Preparation 450 | Reading list
23     {
24         d2 = lvl;
25         return root;
26     }
27     Node *left_lca = findDistUtil(root->left, n1, n2,
28                                    d1, d2, dist, lvl+1);
29     Node *right_lca = findDistUtil(root->right, n1, n2,
30                                    d1, d2, dist, lvl+1);
31     if (left_lca && right_lca)
32     {
33         dist = d1 + d2 - 2*lvl;
34         return root;
35     }
36     return (left_lca != NULL)? left_lca: right_lca;
37 }
38 int findDistance(Node *root, int n1, int n2)
39 {
40     int d1 = -1, d2 = -1, dist;
41     Node *lca = findDistUtil(root, n1, n2, d1, d2,
42                             dist, 1);
43     if (d1 != -1 && d2 != -1)
44         return dist;
45     if (d1 != -1)
46     {
47         dist = findLevel(lca, n2, 0);
48         return dist;
49     }
50     if (d2 != -1)
51     {
52         dist = findLevel(lca, n1, 0);
53         return dist;
54     }
55     return -1;
56 }
```

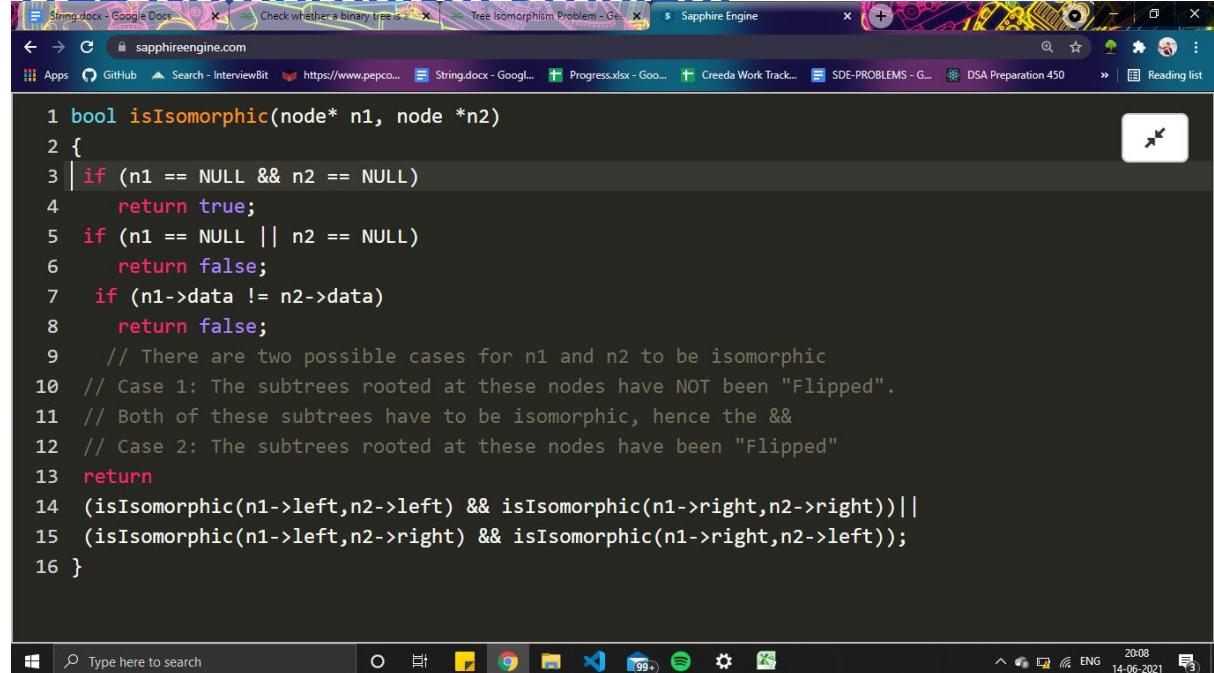
Kth ancestor of a node in binary tree(Backtracking)



The screenshot shows a Windows desktop environment. In the center is a Microsoft Edge browser window displaying a C++ program. The code implements a Depth-First Search (DFS) algorithm to find the Kth ancestor of a given node in a binary tree. The code uses a recursive function `kthAncestorDFS` that takes a `Node*`, an `int node`, and an `int &k`. It checks if the current node is the target node. If not, it recursively searches the left and right subtrees. If a match is found in either subtree, it updates the `temp` pointer and decrements `k`. Once `k` reaches zero, it prints the current node's data and returns `NULL`. Finally, it returns the root node. The browser window has several tabs open, including "String.docx - Google Docs", "Kth ancestor of a node in binary", and "Sapphire Engine". The taskbar at the bottom shows icons for File Explorer, Task View, Start, Taskbar settings, and other system icons. The system tray indicates the date as 14-06-2021 and the time as 19:15.

```
1 Node* temp = NULL;
2 Node* kthAncestorDFS(Node *root, int node , int &k)
3 {
4     if (!root)
5         return NULL;
6     if (root->data == node|| 
7         (temp = kthAncestorDFS(root->left,node,k)) ||
8         (temp = kthAncestorDFS(root->right,node,k)))
9     {
10        if (k > 0)
11            k--;
12        else if (k == 0)
13        {
14            cout<<"Kth ancestor is: "<<root->data;
15            return NULL;
16        }
17        return root;
18    }
19 }
```

Tree Isomorphism Problem:



```
1 bool isIsomorphic(node* n1, node *n2)
2 {
3     if (n1 == NULL && n2 == NULL)
4         return true;
5     if (n1 == NULL || n2 == NULL)
6         return false;
7     if (n1->data != n2->data)
8         return false;
9     // There are two possible cases for n1 and n2 to be isomorphic
10    // Case 1: The subtrees rooted at these nodes have NOT been "Flipped".
11    // Both of these subtrees have to be isomorphic, hence the &&
12    // Case 2: The subtrees rooted at these nodes have been "Flipped"
13    return
14    (isIsomorphic(n1->left,n2->left) && isIsomorphic(n1->right,n2->right)) ||
15    (isIsomorphic(n1->left,n2->right) && isIsomorphic(n1->right,n2->left));
16 }
```

Check whether a binary tree is a complete tree or not:

Let us first define the term ‘Full Node’. A node is ‘Full Node’ if both left and right children are not empty (or not NULL).

The approach is to do a level order traversal starting from the root. In the traversal, once a node is found which is NOT a Full Node, all the following nodes must be leaf nodes.

Also, one more thing needs to be checked to handle the case below: If a node has an empty left child, then the right child must be empty.

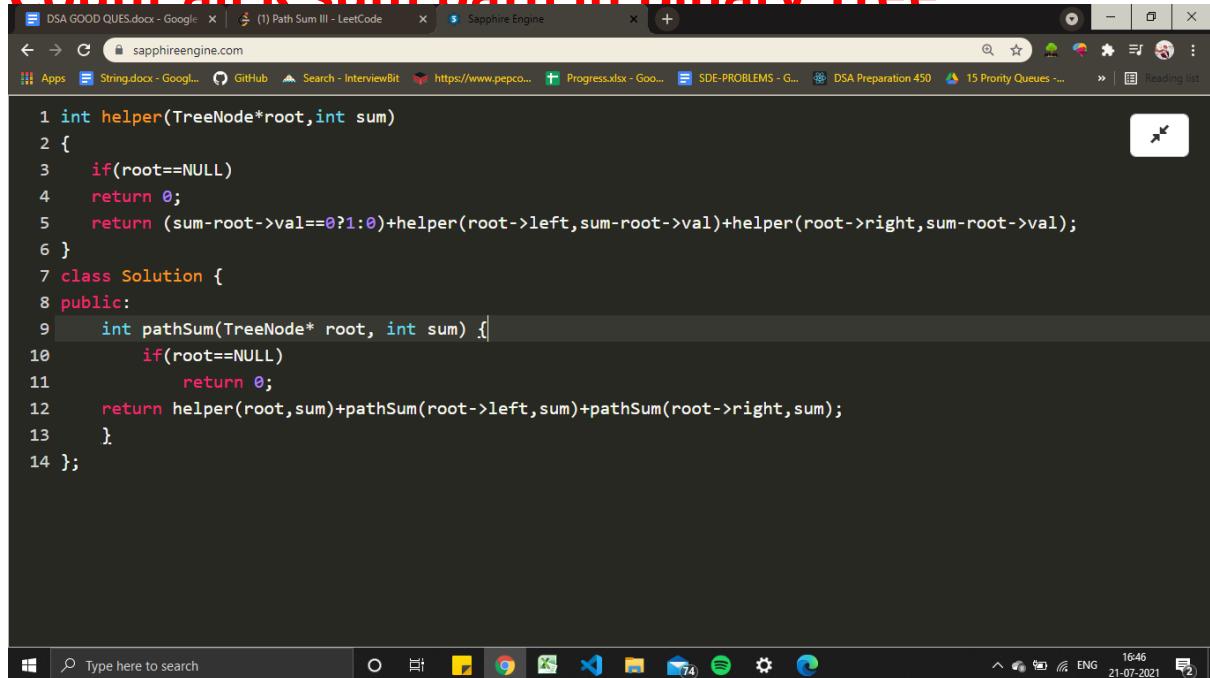
```
1 bool isCompleteBT(node* root)
2 {
3     if (root == NULL)
4         return true;
5     queue<node *> q;
6     q.push(root);
7     bool flag = false;
8     while(!q.empty())
9     {
10         node *temp = q.front();
11         q.pop();
12         if(temp->left)
13         {
14             // If we have seen a non full node,
15             // and we see a node with non-empty
16             // left child, then the given tree is not
17             // a complete Birary Tree
18             if (flag == true)
19                 return false;
20             q.push(temp->left); // Enqueue Left Child
21         }
22         else // If this a non-full node, set the flag as true
23             flag = true;
24     /* Check if right child is present*/
25         if(temp->right)
26         {
27             // If we have seen a non full node,
28             // and we see a node with non-empty
29             // right child, then the given tree is not
30             // a complete Binary Tree
31             if(flag == true)
32                 return false;
33             q.push(temp->right); // Enqueue Right Child
34         }
35         else // If this a non-full node, set the flag as true
36             flag = true;
37     }
38     // If we reach here, then the
39     // tree is complete Binary Tree
40     return true;
41 }
```

Sum of nodes on the longest path from root to leaf node

```
class Pair
{
public:
    int sum;
    int maxLength;
};

Pair helper(Node*root)
{
    if(root==NULL)
    {
        Pair ans;
        ans.sum=0;
        ans.maxLength=0;
        return ans;
    }
    Pair a=helper(root->left);
    Pair b=helper(root->right);
    Pair p1;
    if(a.maxLength>b.maxLength)
    {
        p1.maxLength=1+a.maxLength;
        p1.sum=root->data+a.sum;
    }
    else if(b.maxLength>a.maxLength)
    {
        p1.maxLength=1+b.maxLength;
        p1.sum=root->data+b.sum;
    }
    else
    {
        p1.maxLength=1+b.maxLength;
        p1.sum=root->data+max(a.sum,b.sum);
    }
    return p1;
}
int sumOfLongRootToLeafPath(Node* root)
{
    if(root==NULL)
        return 0;
    return helper(root).sum;
}
```

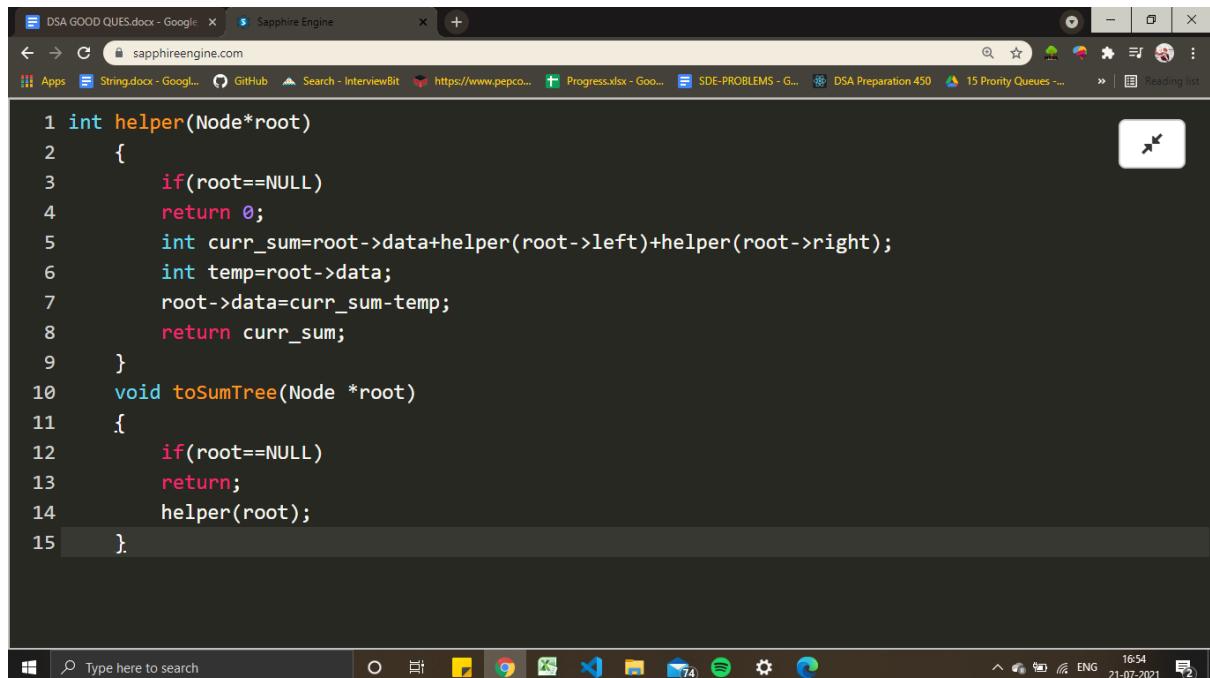
Count all K sum path in Binary Tree



A screenshot of a Windows desktop environment. In the center is a Microsoft Edge browser window displaying a C++ code snippet for finding all paths in a binary tree that sum up to a given value K. The code uses a helper function to traverse the tree and a main function to calculate the total number of such paths. The browser's address bar shows 'sapphireengine.com'. The taskbar at the bottom has several pinned icons, including File Explorer, Task View, Edge, File History, Task Scheduler, File Explorer, Mail, Spotify, Settings, and Microsoft Edge. The system tray shows the date as 21-07-2021 and the time as 16:46.

```
1 int helper(TreeNode*&root,int sum)
2 {
3     if(root==NULL)
4         return 0;
5     return (sum-root->val==0?1:0)+helper(root->left,sum-root->val)+helper(root->right,sum-root->val);
6 }
7 class Solution {
8 public:
9     int pathSum(TreeNode* root, int sum) {
10         if(root==NULL)
11             return 0;
12         return helper(root,sum)+pathSum(root->left,sum)+pathSum(root->right,sum);
13     }
14 };
```

Transform to Sum Tree



A screenshot of a Windows desktop environment. In the center is a Microsoft Edge browser window displaying a C++ code snippet for transforming a binary tree into a 'Sum Tree'. The code defines two functions: 'helper' which calculates the current sum of the tree rooted at 'root' and updates the data of each node to be its original data plus the current sum, and 'toSumTree' which performs this transformation on the entire tree starting from the root. The browser's address bar shows 'sapphireengine.com'. The taskbar at the bottom has several pinned icons, including File Explorer, Task View, Edge, File History, Task Scheduler, File Explorer, Mail, Spotify, Settings, and Microsoft Edge. The system tray shows the date as 21-07-2021 and the time as 16:54.

```
1 int helper(Node*&root)
2 {
3     if(root==NULL)
4         return 0;
5     int curr_sum=root->data+helper(root->left)+helper(root->right);
6     int temp=root->data;
7     root->data=curr_sum-temp;
8     return curr_sum;
9 }
10 void toSumTree(Node *root)
11 {
12     if(root==NULL)
13         return;
14     helper(root);
15 }
```

Binary Tree Maximum Path Sum

```
int helper(TreeNode*&root,int &res)//return type yaad rkhna
{
    if(root==NULL)
        return 0;
    int l=helper(root->left,res);
    int r=helper(root->right,res);
    int max_single=max(root->val,root->val+max(l,r));
    int max_top=max(max_single,root->val+l+r);
    res=max(res,max_top);
    return max_single;
}
class Solution {
public:
    int maxPathSum(TreeNode* root) {
        int res=INT_MIN;
        helper(root,res);
        return res;
    }
};
```

Serialise and Deserialise Binary Tree

```
string serialize(TreeNode* root) {
    if(!root) return "#";

    return to_string(root->val) + "," + serialize(root->left) + "," + serialize(root->right);
}

TreeNode* helper(string &s) {
    if(s[0] == '#') {
        if(s.length() == 1) s = "";
        else s = s.substr(2);

        return NULL;
    }
    int i = 0;
    while(s[i] != '#' && s[i] != ',') {
        i++;
    }
    int x = stoi(s.substr(0, i));
    i++;
    s = s.substr(i);
    TreeNode* root = new TreeNode(x);
    root->left = helper(s);
    root->right = helper(s);
    return root;
}
TreeNode* deserialize(string data) {
    return helper(data);
}
```

Mirror Tree

```
TreeNode* helper(TreeNode*root)
{
    if(root==NULL)
        return NULL;
    TreeNode*a=root->left;
    TreeNode*b=root->right;
    if(a==NULL&&b==NULL)
        return root;
    root->right=helper(a);
    root->left=helper(b);
    return root;
}
TreeNode* Solution::invertTree(TreeNode* root) {
    if(root==NULL)
        return NULL;
    return helper(root);
}
```

Populating Next Right Pointers in Each Node

II

```
Node* connect(Node* root) {
    Node*curr=root;//current node of current level
    Node*head=NULL;//head of the next level
    Node*prev=NULL;//the leading node on the next level
    //level order traversal without queue
    while(curr!=NULL)//jis level p ho uske
        //next level k saare next assign krdo
        {
            while(curr!=NULL)
            {
                if(curr->left!=NULL)
                {
                    if(prev!=NULL)
                        prev->next=curr->left;
                    else
                        head=curr->left;
                    prev=curr->left;
                }
                if(curr->right!=NULL)
                {
                    if(prev!=NULL)
                        prev->next=curr->right;
                    else
                        head=curr->right;
                    prev=curr->right;
                }
                curr=curr->next;
            }
            curr = head;
            head = NULL;
            prev = NULL;
        }
        return root;
}
```

[Flatten Binary Tree to Linked List](#)

```
void flatten(TreeNode* root) {
    if(root==NULL)
        return;
    if(root->left)
    {
        flatten(root->left);
        TreeNode*temp=root->right;
        root->right=root->left;
        root->left=NULL;
        TreeNode*t=root->right;
        while(t->right)
            t=t->right;
        t->right=temp;
    }
    flatten(root->right);
}
```

Recover Binary Search Tree

Since inorder traversal of BST is always a sorted array, the problem can be reduced to a problem where two elements of a sorted array are swapped. There are two cases that we need to handle:

1. The swapped nodes are not adjacent in the inorder traversal of the BST.

For example, Nodes 5 and 25 are swapped in {3 5 7 8 10 15 20 25}.

The inorder traversal of the given tree is 3 25 7 8 10 15 20 5

If we observe carefully, during inorder traversal, we find node 7 is smaller than the previous visited node 25. Here save the context of node 25 (previous

node). Again, we find that node 5 is smaller than the previous node 20. This time, we save the context of node 5 (the current node). Finally, swap the two node's values.

2. The swapped nodes are adjacent in the inorder traversal of BST.

For example, Nodes 7 and 8 are swapped in {3 5 7 8 10 15 20 25}.

The inorder traversal of the given tree is 3 5 8 7 10 15 20 25

Unlike case #1, here only one point exists where a node value is smaller than the previous node value. e.g. node 7 is smaller than node 8.

How to Solve? We will maintain three-pointers, *first*, *middle*, and *last*. When we find the first point where the current node value is smaller than the previous node value, we update the *first* with the previous node & the *middle* with the current node. When we find the second point where the current node value is smaller than the previous node value, we update the *last* with the current node. In the case of #2, we will never find the second point. So, the *last* pointer will not be updated. After processing, if the *last* node value is null, then two swapped nodes of BST are adjacent.

For reference of soln refer [this](#) video

```
void help(TreeNode* root,TreeNode* &first,TreeNode* &mid,TreeNode* &last,TreeNode* &prevv){
    if(root==NULL) return;
    help(root->left,first,mid,last,prevv);
    if(prevv==NULL) prevv=root;
    else
    {
        if(root->val<prevv->val)
        {
            if(mid==NULL)
            {
                first=prevv;
                mid=root;
            }
            else
            {
                if(root->val<mid->val)
                last=root;
            }
        }
        prevv=root;
        help(root->right,first,mid,last,prevv);
    }
}
vector<int> Solution::recoverTree(TreeNode* A) {
    TreeNode*first=NULL;
    TreeNode*middle=NULL;
    TreeNode*last=NULL;
    TreeNode*prevv=NULL;
    help(A,first,middle,last,prevv);
    vector<int>v;
    if(first&&last)
    v={first->val,last->val};
    else
    v={first->val,middle->val};
    sort(v.begin(),v.end());
    return(v);
}
```

Remove Half Nodes

```
 */
TreeNode* Solution::solve(TreeNode* root) {
    if (root==NULL)
        return NULL;

    root->left = solve(root->left);
    root->right = solve(root->right);

    if (root->left==NULL && root->right==NULL)
        return root;
    if (root->left==NULL)
    {
        TreeNode *new_root = root->right;
        free(root);
        return new_root;
    }
    if (root->right==NULL)
    {
        TreeNode *new_root = root->left;
        free(root);
        return new_root;
    }

    return root;
}
```

Maximum Edge Removal

Har Node ka count nikal lo agar uske subtree m odd number of nodes h th uske sir k upr vali node krna possible h vrna nhi h.

```

void dfs(int node,vector<int>&cnt,vector<int>adj[],vector<bool>visited)
{ //to count all nodes children below it mtlb children k bhi children sab cnt kro bss neche aane chahiye uske
    visited[node]=true;
    for(auto it:adj[node])
    {
        if(!visited[it])
        {
            dfs(it,cnt,adj,visited);
            cnt[node]+=cnt[it]+1;
        }
    }
    cnt[node]+=0;
}
int Solution::solve(int A, vector<vector<int>>&B) {
    vector<int>adj[A];
    for(int i=0;i<B.size();i++)
    {
        adj[B[i][0]-1].push_back(B[i][1]-1);
        adj[B[i][1]-1].push_back(B[i][0]-1);
    }
    vector<bool>vis(A,false);
    vector<int>cnt(A,0);
    dfs(0,cnt,adj,vis);
    int ans=0;
    for(int it=0;it<A;it++)
    {
        if(cnt[it]%2!=0 && it!=0)ans++; //ab agar uss node k neeche no. of nodes odd h th uske upr
        | //vali edge remove kr skte h islie it!=0 kia h bcz 0 k upr th koi edge hi na hgi
    }return ans;
}

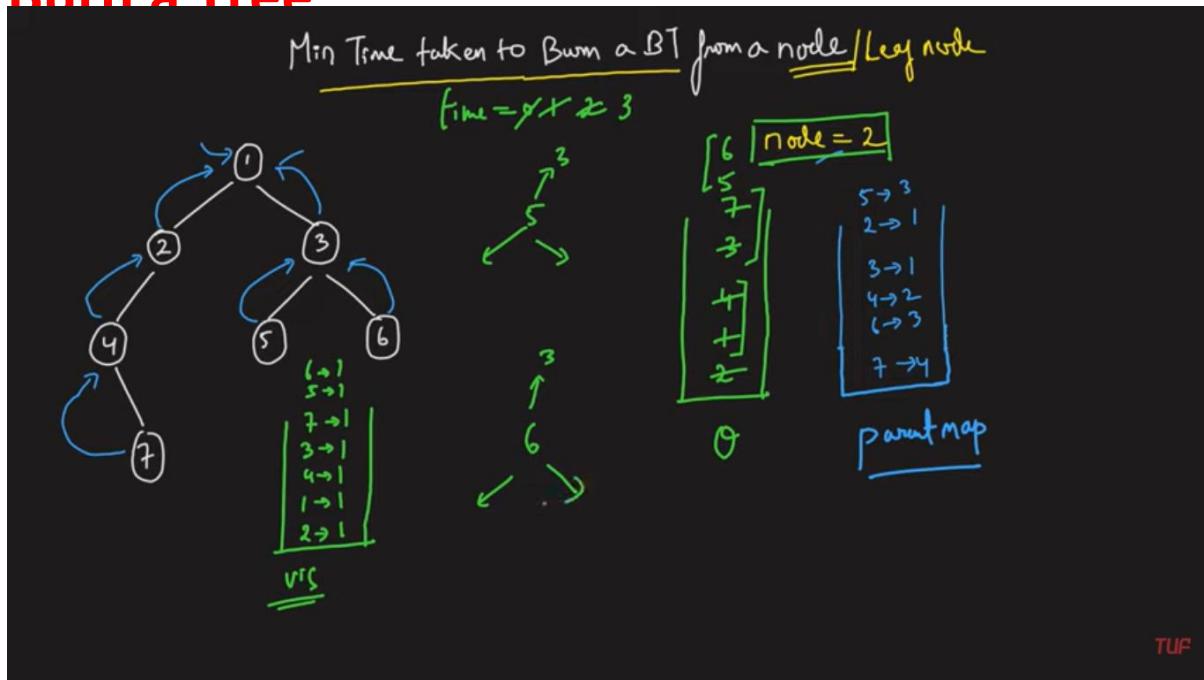
```

Inorder Traversal of Cartesian Tree

```
int max(vector<int>&nums, int l, int r) {
    int max_i = l;
    for (int i = l+1; i <=r; i++) {
        if (nums[max_i] < nums[i])
            max_i = i;
    }
    return max_i;
}
TreeNode*helper(vector<int>&nums,int l,int r)
{
    if (l >r)
        return NULL;
    int max_i = max(nums, l, r);
    TreeNode*root = new TreeNode(nums[max_i]);
    root->left = helper(nums, l, max_i-1);
    root->right = helper(nums, max_i + 1, r);
    return root;
}
TreeNode* Solution::buildTree(vector<int>&v) {
    int n=v.size();
    return helper(v,0,n-1);
}
```

```
TreeNode* constructMaximumBinaryTree(vector<int>& nums) {
    stack<TreeNode*>s;
    //agar ith idx i+1 se chota h th i+1 vala parent hga aur ith vala uska left child
    //agar ith bda h th i+1 uska right child hga kyuki inorder traversal h
    for(int i=0;i<nums.size();i++)
    {
        TreeNode*curr=new TreeNode(nums[i]);
        while(!s.empty()&&nums[i]>s.top()->val)
        {
            curr->left=s.top();
            s.pop();
        }
        if(s.empty())
        {
            s.push(curr);
        }
        else if(!s.empty())
        {
            TreeNode*temp=s.top();
            temp->right=curr;
            s.push(curr);
        }
    }
    while(s.size()>1)
        s.pop();
    return s.top();
}.
```

Burn a Tree



```

int findMaxDistance(map<BinaryTreeNode<int>*, BinaryTreeNode<int>*> &mpp, BinaryTreeNode<int>* target) {
    queue<BinaryTreeNode<int>*> q;
    q.push(target);
    map<BinaryTreeNode<int>*, int> vis;
    vis[target] = 1;
    int maxi = 0;
    while(!q.empty()) {
        int sz = q.size();
        int fl = 0;
        for(int i = 0;i<sz;i++) {
            auto node = q.front();
            q.pop();
            if(node->left && !vis[node->left]) {
                fl = 1;
                vis[node->left] = 1;
                q.push(node->left);
            }
            if(node->right && !vis[node->right]) {
                fl = 1;
                vis[node->right] = 1;
                q.push(node->right);
            }
        }
        if(mpp[node] && !vis[mpp[node]]) {
            fl = 1;
            vis[mpp[node]] = 1;
            q.push(mpp[node]);
        }
    }
    if(fl) maxi++;
}
return maxi;
}

BinaryTreeNode<int>* bfsToMapParents(BinaryTreeNode<int>* root,
                                     map<BinaryTreeNode<int>*, BinaryTreeNode<int>*> &mpp, int start) {
    queue<BinaryTreeNode<int>*> q;
    q.push(root);
    BinaryTreeNode<int>* res;
    while(!q.empty()) {
        BinaryTreeNode<int>* node = q.front();
        if(node->data == start) res = node;
        q.pop();
        if(node->left) {
            mpp[node->left] = node;
            q.push(node->left);
        }
        if(node->right) {
            mpp[node->right] = node;
            q.push(node->right);
        }
    }
    return res;
}
int timeToBurnTree(BinaryTreeNode<int>* root, int start)
{
    map<BinaryTreeNode<int>*, BinaryTreeNode<int>*> mpp;
    BinaryTreeNode<int>* target = bfsToMapParents(root, mpp, start);
    int maxi = findMaxDistance(mpp, target);
    return maxi;
}

```

Binary Search Trees:

Delete a node in BST

```
TreeNode* deleteNode(TreeNode* root, int key) {
    if (root == NULL) {
        return NULL;
    }
    if(key>root->val)
    {
        root->right=deleteNode(root->right,key);return root;}
    else if(key<root->val)
    {
        root->left=deleteNode(root->left,key); //yh left m root->left m hi store krana h
        //return k baad jo aaega glti ho skti h
        return root;}
    else
    {
        if(root->left==NULL and root->right==NULL)
        {
            delete root;return NULL; }
        else if(root->left==NULL)
        {
            TreeNode*temp=root->right;
            root->right=NULL;
            delete root;return temp;}
        else if(root->right==NULL)
        {
            TreeNode* temp=root->left;
            root->left=NULL;
            delete root;return temp;}
        else
        {
            //yh vala imp hai
            TreeNode* rightmin=root->right;
            while(rightmin->left!=NULL)
                rightmin=rightmin->left;
            root->val=rightmin->val;
            root->right=deleteNode(root->right,rightmin->val);
            return root;}
    }
}
```

Check is BST

```

class IsBSTReturn {
public:
    bool isBST;
    int minimum;
    int maximum;
};

IsBSTReturn isBST2(BinaryTreeNode<int>* root) {
    if (root == NULL) {
        IsBSTReturn output;
        output.isBST = true;
        output.minimum = INT_MAX;
        output.maximum = INT_MIN;
        return output;
    }
    IsBSTReturn leftOutput = isBST2(root->left);
    IsBSTReturn rightOutput = isBST2(root->right);
    int minimum = min(root->data, min(leftOutput.minimum, rightOutput.minimum));
    int maximum = max(root->data, max(leftOutput.maximum, rightOutput.maximum));
    bool isBSTFinal = (root->data > leftOutput.maximum) && (root->data <= rightOutput.minimum) &&
        leftOutput.isBST && rightOutput.isBST;
    IsBSTReturn output;
    output.minimum = minimum;output.maximum = maximum;output.isBST = isBSTFinal;
    return output;
}
bool isBST3(BinaryTreeNode<int>* root, int min = INT_MIN, int max = INT_MAX) {
    if (root == NULL) {
        return true;
    }
    if (root->data < min || root->data > max) {
        return false;
    }
    bool isLeftOk = isBST3(root->left, min, root->data - 1);
    bool isRightOk = isBST3(root->right, root->data, max);
    return isLeftOk && isRightOk;
}

```

BST ITERATOR

```
stack<TreeNode*> s;
BSTIterator::BSTIterator(TreeNode *root) {
    while(!s.empty()) {
        s.pop();
    }

    while(root) {
        s.push(root);
        root = root->left;
    }
}

/** @return whether we have a next smallest number */
bool BSTIterator::hasNext() {
    if(s.empty()) return false;

    return true;
}

/** @return the next smallest number */
int BSTIterator::next() {
    TreeNode* curr = s.top();
    s.pop();
    int ans = curr->val;

    curr = curr->right;
    while(curr) {
        s.push(curr);
        curr = curr->left;
    }

    return ans;
}
```

Populate Inorder Successor of root

```
void helper(struct node*&root,struct node*&next1)
{
    if(root==NULL)
        return;
    helper(root->right,next1);
    root->next=next1;
    next1=root;
    helper(root->left,next1);
}
void populateNext(struct node* root)
{
    struct node*next=NULL;
    return helper(root,next);
}
```

Construct Binary Search Tree from Preorder Traversal

```
TreeNode*helper(vector<int>&preorder,int &i,int ubound)
{
    if(i==preorder.size()||preorder[i]>ubound)
        return NULL;
    TreeNode*root=new TreeNode(preorder[i++]);
    root->left=helper(preorder,i,root->val);
    root->right=helper(preorder,i,ubound);
    return root;
}
TreeNode* bstFromPreorder(vector<int>& preorder) {
    int i=0;
    return helper(preorder,i,INT_MAX);
}
```

```
TreeNode*helper(vector<int>& preorder,int &idx,int key,int size,int max,int min)
{
    if(idx>=size)
        return NULL;
    TreeNode*root=NULL;
    if(key>min and key<max)
    {
        root=new TreeNode(key);
        idx++;
        if(idx<size)
        {
            root->left=helper(preorder,idx,preorder[idx],size,key,min);
            //root->right=helper(preorder,idx,preorder[idx],size,max,key);
            //isse andar nhi rkhe skte kyuki iske liye alag se check
            //krna pdega for eg:[4,2]iss case k liye
        }
        if(idx<size)
        {
            root->right=helper(preorder,idx,preorder[idx],size,max,key);
        }
    }
    return root;
}
TreeNode* bstFromPreorder(vector<int>& preorder) {
    int size=preorder.size();
    int idx=0;
    return helper(preorder,idx,preorder[0],size,INT_MAX,INT_MIN);
}
```

Merge two BST's

```
vector<int>ans;
void merge_helper(Node*root1,Node*root2)
{
    if(!root1 && !root2)
        return ;
    else if(!root1)
    {
        inorder(root2);
        return;
    }
    else if(!root2)
    {
        inorder(root1);
        return;
    }
    else
    {
        Node*temp1=root1;
        Node*prev1=NULL;
        Node*temp2=root2;
        Node*prev2=NULL;
        while(temp1->left)
        {
            prev1=temp1;
            temp1=temp1->left;
        }
        while(temp2->left!=NULL)
        {
            prev2=temp2;
            temp2=temp2->left;
        }
    }
}
```

```

        }
        if(temp1->data>=temp2->data)
        {
            ans.push_back(temp2->data);
            if(prev2==NULL)
                merge_helper(root1,root2->right);
            else
            {
                prev2->left=temp2->right;
                merge_helper(root1,root2);
            }
        }
        else
        {
            ans.push_back(temp1->data);
            if(prev1==NULL)
                merge_helper(root1->right,root2);
            else
            {
                prev1->left=temp1->right;
                merge_helper(root1,root2);
            }
        }
    }
    vector<int> merge(Node *root1, Node *root2)
    {
        ans.clear();
        merge_helper(root1,root2);
        return ans;
    }
};

```

Kth largest Element

Done using **REVERSE INORDER TRAVERSAL**

```
Node*ans=NULL;
void helper(Node*root,int &k)
{
    if(root==NULL)
        return;
    helper(root->right,k);
    k--;
    if(k==0)
    {
        ans=root;
        return;
    }
    helper(root->left,k);

}
int kthLargest(Node *root, int k)
{
    if(root==NULL)
        return -1;
    helper(root,k);
    if(ans==NULL)
        return -1;
    else
        return ans->data;}
```

Count pairs from two BSTs whose sum is equal to a given value x

```
int countPairs(Node* root1, Node* root2, int x)
{
    if (root1 == NULL || root2 == NULL)
        return 0;
    stack<Node*> st1, st2;
    Node* top1, *top2;
    int count = 0;
    while (1) {
        while (root1 != NULL) {
            st1.push(root1);
            root1 = root1->left;
        }
        while (root2 != NULL) {
            st2.push(root2);
            root2 = root2->right;
        }
        if (st1.empty() || st2.empty())
            break;
        top1 = st1.top();
        top2 = st2.top();
        if ((top1->data + top2->data) == x) {
            count++;
            st1.pop();
            st2.pop();
            root1 = top1->right;
            root2 = top2->left;
        }
        else if ((top1->data + top2->data) < x) {
            st1.pop();
            root1 = top1->right;
        }
        else {
            st2.pop();
            root2 = top2->left;
        }
    }
    return count;
}
```

Morris Inorder Traversal

```
void MorrisTraversal(struct tNode* root)
{
    struct tNode *current, *pre;
    if (root == NULL)
        return;
    current = root;
    while (current != NULL) {
        if (current->left == NULL) {
            printf("%d ", current->data);
            current = current->right;
        }
        else {
            pre = current->left;
            while (pre->right != NULL
                   && pre->right != current)
                pre = pre->right;
            if (pre->right == NULL) {
                pre->right = current;
                current = current->left;
            }
            else {
                pre->right = NULL;
                printf("%d ", current->data);
                current = current->right;
            }
        }
    }
}
```

Find Median in BST

```
int counNodes(struct Node *root)
{
    struct Node *current, *pre;
    int count = 0;
    if (root == NULL)
        return count;
    current = root;
    while (current != NULL)
    {
        if (current->left == NULL)
        {
            count++;
            current = current->right;
        }
        else
        {
            pre = current->left;
            while (pre->right != NULL &&
                   pre->right != current)
                pre = pre->right;
            if(pre->right == NULL)
            {
                pre->right = current;
                current = current->left;
            }
            else
            {
                pre->right = NULL;
                count++;
                current = current->right;
            }
        }
    }
    return count;
}
```

```
int findMedian(struct Node *root)
{
    if (root == NULL)
        return 0;
    int count = countNodes(root);
    int currCount = 0;
    struct Node *current = root, *pre, *prev;
    while (current != NULL)
    {
        if (current->left == NULL)
        {
            currCount++;
            if (count % 2 != 0 && currCount == (count+1)/2)
                return current->data;
            else if (count % 2 == 0 && currCount == (count/2)+1)
                return (prev->data + current->data)/2;
            prev = current;
            current = current->right;
        }
        else
        {
            pre = current->left;
            while (pre->right != NULL && pre->right != current)
                pre = pre->right;
            if (pre->right == NULL)
            {
                pre->right = current;
                current = current->left;
            }
            else
            {
                pre->right = NULL;
                prev = pre;
                currCount++;
                if (count % 2 != 0 && currCount == (count+1)/2 )
                    return current->data;
                else if (count%2==0 && currCount == (count/2)+1)
                    return (prev->data+current->data)/2;
                prev = current;
                current = current->right;
            }
        }
    }
}
```

Replace every element with the least greater element on its right

```
Node* insert(Node* node, int data, Node*& succ)
{
    if (node == NULL)
        return node = newNode(data);
    if (data < node->data) {
        succ = node;
        node->left = insert(node->left, data, succ);
    }
    else if (data > node->data)
        node->right = insert(node->right, data, succ);
    return node;
}
void replace(int arr[], int n)
{
    Node* root = NULL;
    for (int i = n - 1; i >= 0; i--) {
        Node* succ = NULL;
        root = insert(root, arr[i], succ);
        if (succ)
            arr[i] = succ->data;
        else
            arr[i] = -1;
    }
}
```

Valid BST from Preorder

```
bool helper(int &i, vector<int>&v, int min, int max, int n, int key)
{
    if(i>=n)
        return true;
    if(v[i]>max || v[i]<min)
        return false;
    i++;
    return helper(i, v, min, key, n, v[i]) || helper(i, v, key, max, n, v[i]);
}
int Solution::solve(vector<int> &v) {
    int i=0;
    return helper(i, v, INT_MIN, INT_MAX, v.size(), v[0]);
}.
```

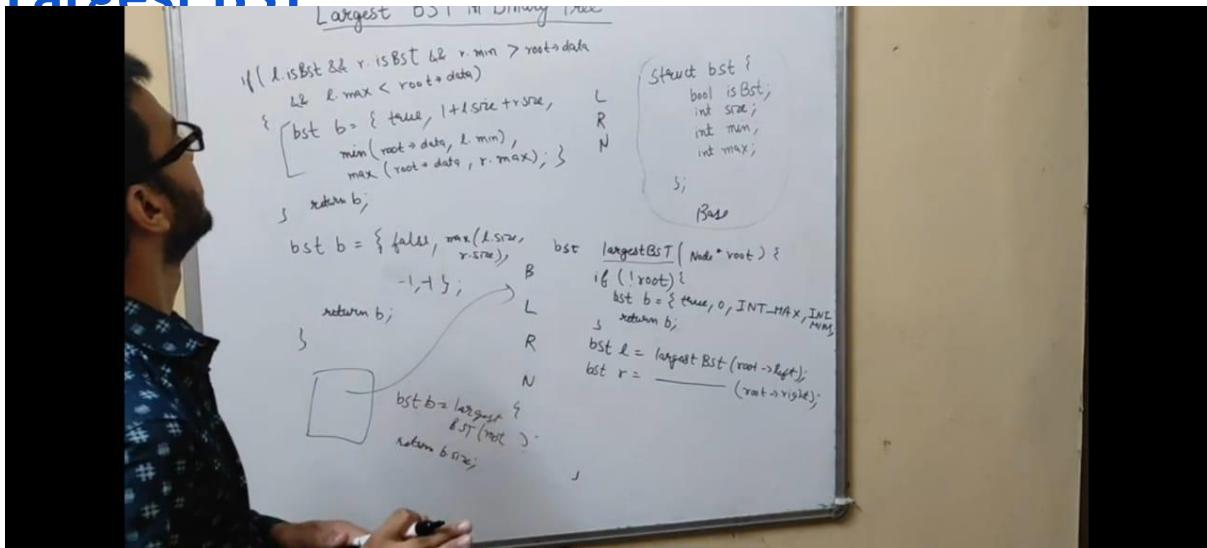
```
1 bool canRepresentBST(int pre[], int n)
2 {
3     stack<int> s;
4     int root = INT_MIN;
5     for (int i=0; i<n; i++)
6     {
7         if (pre[i] < root)
8             return false;
9         while (!s.empty() && s.top()<pre[i])
10        {
11            root = s.top();
12            s.pop();
13        }
14        s.push(pre[i]);
15    }
16    return true;
17 }
```

Check whether BST contains Dead End or not

If we take a closer look at the problem, we can notice that we basically need to check if there is a leaf node with value x such that $x+1$ and $x-1$ exist in BST with the exception of $x = 1$. For $x = 1$, we can't insert 0 as the problem statement says BST contains positive integers only.

```
bool deadEnd(Node* root, int min=1, int max=INT_MAX)
{
    if (!root)
        return false;
    if (min == max)
        return true;
    return deadEnd(root->left, min, root->data - 1) ||
           deadEnd(root->right, root->data + 1, max);
}
```

Largest BST



Sorted Array to BST

A screenshot of a browser window showing a C++ code snippet for converting a sorted array into a balanced BST. The code is as follows:

```
1 TreeNode*>helper(vector<int>input,int l,int r)
2 {
3     if(l>r)
4     {
5         return NULL;
6     }
7     int mid = (l+r)/2;
8     TreeNode* root=new TreeNode(input[mid]);
9     root->left=helper(input,l,mid-1);
10    root->right=helper(input,mid+1,r);
11    return root;
12 }
13 TreeNode*> Solution::sortedArrayToBST(const vector<int> &A) {
14     return helper(A,0,A.size()-1);
15 }
```

The browser window also shows several tabs open, including "DSA GOOD QUES.docx - Google Docs" and "Sapphire Engine". The status bar at the bottom indicates the date and time as 04-08-2021 18:05.

GRAPHS:

Detect cycle in an undirected graph

```
bool isCyclicUtil(int v,bool visited[],int parent,vector<int>g[])
{
    visited[v]=true;
    for(auto i=g[v].begin();i!=g[v].end();i++)
    {
        if (!visited[*i])
        {
            if (isCyclicUtil(*i, visited, v,g))
                return true;
        }
        else if(*i!=parent)
            return true;
    }
    return false;
}

bool isCyclic(vector<int> g[], int V)
{
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;
    for (int u = 0; u < V; u++)
    {
        if (!visited[u])
        {
            if (isCyclicUtil(u, visited, -1,g))
                return true;
        }
    }
    return false;
}
```

Detect Cycle in a Directed Graph

```
private boolean checkCycle(int node, ArrayList<ArrayList<Integer>> adj, int vis[], int dfsVis[]) {  
    vis[node] = 1;  
    dfsVis[node] = 1;  
  
    for(Integer it: adj.get(node)) {  
        if(vis[it] == 0) {  
            if(checkCycle(it, adj, vis, dfsVis) == true) {  
                return true;  
            }  
        } else if(dfsVis[it] == 1) {  
            return true;  
        }  
    }  
    dfsVis[node] = 0;  
    return false;  
}  
  
public boolean isCyclic(int N, ArrayList<ArrayList<Integer>> adj) {  
    int vis[] = new int[N];  
    int dfsVis[] = new int[N]; |  
  
    for(int i = 0;i<N;i++) {  
        if(vis[i] == 0) {  
            if(checkCycle(i, adj, vis, dfsVis) == true) return true;  
        }  
    }  
    return false;  
}
```

Minimum Steps by Knight:

```
bool isvalid(int x, int y, int N)
{
    if (x >= 1 && x <= N && y >= 1 && y <= N)
        return true;
    return false;
}
int a[8][2]={{-2,-1},{-1,-2},{1,-2},{2,-1},{-2,1},{-1,2},{1,2},{2,1}};
ll minsteps(ll sx,ll sy,ll dx,ll dy, ll n)
{

    bool visited[n+1][n+1];
    for (ll i = 1; i <= n; i++)
        for (ll j = 1; j <= n; j++)
            visited[i][j] = false;
    visited[sx][sy]=1;
    queue<pair<pair<ll, ll>, ll>>q;
    q.push(make_pair(make_pair(sx,sy),0));
    while(!q.empty())
    {
        pair<pair<ll, ll>, ll>front=q.front();
        q.pop();
        if(front.fi.fi==dx && front.fi.se==dy)
            return front.se;
        for(ll i=0;i<8;i++)
        {
            ll newx=front.fi.fi+a[i][0];
            ll newy=front.fi.se+a[i][1];
            if(isvalid(newx,newy,n) and !visited[newx][newy])
            {
                visited[newx][newy]=true;
                q.push(make_pair(make_pair(newx,newy),front.se+1));
            }
        }
    }
}
```

Making wired Connections:

```
private:
    void dfs(vector<vector<int>> &adj, vector<bool> &visited, int src)
    {
        visited[src] = true;
        for(int i : adj[src])
            if(!visited[i])
                dfs(adj, visited, i);
    }
public:
    int makeConnected(int n, vector<vector<int>>& connections) {
        vector<vector<int>> adj(n);
        for(auto v : connections)
        {
            adj[v[0]].push_back(v[1]);
            adj[v[1]].push_back(v[0]);
        }
        int sum = 0;
        for (int i = 0 ; i < n; i++)
            sum += adj[i].size();
        sum/=2;
        if(sum<n-1)
            return -1;
        vector<bool> visited(n, false);
        int components = 0;
        for(int i=0; i<n; i++)
            if(!visited[i])
            {
                dfs(adj, visited, i);
                components++;
            }
        return components - 1;
    }
}
```

Clone a Graph

```
Node* cloneGraph(Node* root) {
    if(root==NULL)
        return NULL;
    Node*node=new Node(root->val);
    unordered_map<Node*,Node*>m;
    m[root]=node;
    queue<Node*>q;
    q.push(root);
    while(!q.empty())
    {
        Node*front=q.front();
        q.pop();
        vector<Node*>v=front->neighbors;
        for(int i=0;i<v.size();i++)
        {
            if(!m[v[i]])
            {
                Node*temp=new Node(v[i]->val);
                m[v[i]]=temp;
                q.push(v[i]);
            }
            m[front]->neighbors.push_back(m[v[i]]);
        }
    }
    return node;
}
```

Topological Sort DFS(Only possible for Directed Acyclic Graph):

```
1 // } Driver Code Ends
2 class Solution{
3     void findTopoSort(int node, vector<int> &vis, stack<int> &st, vector<int> adj[]) {
4         vis[node] = 1;
5         for(auto it : adj[node]) {
6             if(!vis[it]) {
7                 findTopoSort(it, vis, st, adj);
8             }
9         }
10        st.push(node);
11    }
12 public:
13     vector<int> topoSort(int N, vector<int> adj[]) {
14         stack<int> st;
15         vector<int> vis(N, 0);
16         for(int i = 0;i<N;i++) {
17             if(vis[i] == 0) {
18                 findTopoSort(i, vis, st, adj);
19             }
20         }
21         vector<int> topo;
22         while(!st.empty()) {
23             topo.push_back(st.top());
24             st.pop();
25         }
26         return topo;
27     }
28 };
29 }
```

Topo Sort (Using Kahn's algo): O(V+E)

```
vector<int> topoSort(int N, vector<int> adj[]) {
    queue<int> q;
    vector<int> indegree(N, 0);
    for(int i = 0;i<N;i++) {
        for(auto it: adj[i]) {
            indegree[it]++;
        }
    }

    for(int i = 0;i<N;i++) {
        if(indegree[i] == 0) {
            q.push(i);
        }
    }
    vector<int> topo;
    while(!q.empty()) {
        int node = q.front();
        q.pop();
        topo.push_back(node);
        for(auto it : adj[node]) {
            indegree[it]--;
            if(indegree[it] == 0) {
                q.push(it);
            }
        }
    }
    return topo;
}
```

Check Cycle using Kahn's Algo

```
bool isCyclic(int N, vector<int> adj[]) {  
    queue<int> q;  
    vector<int> indegree(N, 0);  
    for(int i = 0;i<N;i++) {  
        for(auto it: adj[i]) {  
            indegree[it]++;  
        }  
    }  
  
    for(int i = 0;i<N;i++) {  
        if(indegree[i] == 0) {  
            q.push(i);  
        }  
    }  
    int cnt = 0;  
    while(!q.empty()) {  
        int node = q.front();  
        q.pop();  
        cnt++;  
        for(auto it : adj[node]) {  
            indegree[it]--;  
            if(indegree[it] == 0) {  
                q.push(it);  
            }  
        }  
    }  
    if(cnt == N) return false;  
    return true;  
}
```

Shortest/Longest Path in Weighted Directed Acyclic Graph:

```
void findTopoSort(int node, int vis[], stack<int> &st, vector<pair<int,int>> adj[]) {
    vis[node] = 1;
    for(auto it : adj[node]) {
        if (vis[it.first]) {
            findTopoSort(it.first, vis, st, adj);
        }
    }
    st.push(node);
}

void shortestPath(int src, int N, vector<pair<int,int>> adj[])
{
    int vis[N] = {0};
    stack<int> st;
    for (int i = 0; i < N; i++)
        if (!vis[i])
            findTopoSort(i, vis, st, adj);

    int dist[N];
    for (int i = 0; i < N; i++)
        dist[i] = 1e9;
    dist[src] = 0;

    while(!st.empty())
    {
        int node = st.top();
        st.pop();

        if (dist[node] != INF) {
            for(auto it : adj[node]) {
                if(dist[node] + it.second < dist[it.first]) {
                    dist[it.first] = dist[node] + it.second;
                }
            }
        }
    }
}
```

Shortest Path in undirected graph of unit weight:

```
void BFS(vector<int> adj[], int N, int src)
{
    int dist[N];
    for(int i = 0;i<N;i++) dist[i] = INT_MAX;
    queue<int> q;

    dist[src] = 0;
    q.push(src);

    while(q.empty() == false)
    {
        int node = q.front();
        q.pop();

        for(auto it:adj[node]){
            if(dist[node] + 1 < dist[it]){
                dist[it]=dist[node]+1;
                q.push(it);
            }
        }
    }
    for(int i = 0;i<N;i++) cout << dist[i] << " ";
}
```

if weight is not 1 then we can use

Dijkstra Algo

to find **minimum distance** from source vertex to all other vertices in **undirected weighted graph**.

```
vector<pair<int,int>> g[n+1]; // 1-indexed adjacency list for of graph

int a,b,wt;
for(int i = 0; i<m ; i++){
    cin >> a >> b >> wt;
    g[a].push_back(make_pair(b,wt));
    g[b].push_back(make_pair(a,wt));
}

cin >> source;

// Dijkstra's algorithm begins from here
priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>> pq;// min-heap ; In pair => (dist
vector<int> distTo(n+1,INT_MAX); // 1-indexed array for calculating shortest paths;

distTo[source] = 0;
pq.push(make_pair(0,source)); // (dist,from)

while( !pq.empty() ){
    int dist = pq.top().first;
    int prev = pq.top().second;
    pq.pop();

    vector<pair<int,int>>::iterator it;
    for( auto it: g[prev]){
        int next = it.first;
        int nextDist = it.second;
        if( distTo[next] > distTo[prev] + nextDist){
            distTo[next] = distTo[prev] + nextDist;
            pq.push(make_pair(distTo[next], next));
        }
    }
}

cout << "The distances from source, " << source << ", are : \n";
for(int i = 1 ; i<=n ; i++) cout << distTo[i] << " ";
```

Word Ladder

```
int ladderLength(string begin, string end, vector<string>&List) {
    queue<string>q;
    unordered_set<string>s;
    for(int i=0;i<List.size();i++)
        s.insert(List[i]);
    q.push(begin);
    s.erase(begin);
    int ans=1;
    while(!q.empty())
    {
        int n=q.size();
        for(int i=0;i<n;i++)
        {
            string front=q.front();
            q.pop();
            if(front==end)
                return ans;
            for(int j=0;j<front.size();j++)
            {
                char c=front[j];
                for(int k=0;k<26;k++)
                {
                    front[j]='a'+k;
                    if(s.find(front)!=s.end())
                    {
                        q.push(front);
                        s.erase(front);
                    }
                }
                front[j]=c;
            }
        }
        ans++;
    }
    return 0;
}
```

Minimum time taken by each job to be completed given by a Directed Acyclic Graph:

same as Kahn's algo of topo sort

Approach: The job can be started only if all the jobs that are prerequisites of the job are done. Therefore, the idea is to use [Topological Sort](#) for the given network. Below are the steps:

1. Finish the jobs that are not dependent on any other job.
2. Create an array **inDegree[]** to store the count of the dependent node for each node in the given network.
3. Initialize a queue and push all the vertices whose **inDegree[]** is 0.
4. Initialize the timer to 1 and store the current queue size(say **size**) and do the following:
 - o Pop the node from the queue until the size is **0** and update the finishing time of this node to the **timer**.
 - o While popping the node(say node **U**) from the queue decrements the **inDegree** of every node connected to it.
 - o If **inDegree** of any node is **0** in the above step then insert that node in the queue.
 - o Increment the timer after all the above steps.
5. Print the finishing time of all the nodes after we traverse every node in the above step.

Alien Dictionary:

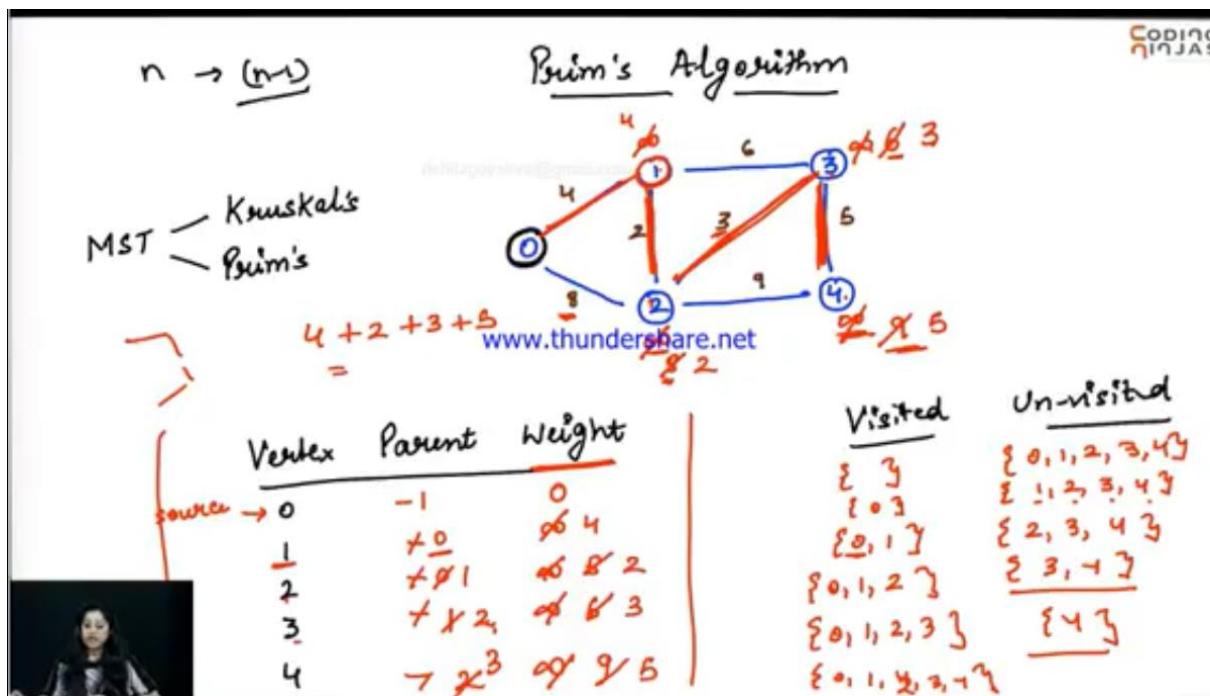
```
string findOrder(string dict[], int n, int k) {
    for(int i=0;i<n-1;i++)
    {
        string word1=dict[i];
        string word2=dict[i+1];
        for(int j=0;j<min(word1.size(),word2.size());j++)
        {
            if(word1[j]!=word2[j])
            {
                adj[word1[j]-'a'].push_back(word2[j]-'a');
                break;
            }
        }
    }
    topologicalSort();
    return ans;
}
```

Minimum Spanning Tree:

(for weighted undirected graph)

A spanning *tree* of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree. A minimum spanning tree has $(V - 1)$ edges where V is the number of vertices in the given graph.

Prim's Algo:



```

int parent[N];
int key[N];
bool mstSet[N];
for (int i = 0; i < N; i++)
    key[i] = INT_MAX, mstSet[i] = false;
priority_queue< pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>> pq;
key[0] = 0;
parent[0] = -1;
pq.push({0, 0});
while(!pq.empty())
{
    int u = pq.top().second;
    pq.pop();
    mstSet[u] = true;
    for (auto it : adj[u]) {
        int v = it.first;
        int weight = it.second;
        if (mstSet[v] == false && weight < key[v]) {
            parent[v] = u;
            key[v] = weight;
            pq.push({key[v], v});
        }
    }
}
for (int i = 1; i < N; i++)
    cout << parent[i] << " - " << i << "\n";
return 0;
}

```

Graph Colouring Problem

```
void Graph::greedyColoring()
{
    int result[V];
    result[0] = 0;
    for (int u = 1; u < V; u++)
        result[u] = -1; // no color is assigned to u
    bool available[V];
    for (int cr = 0; cr < V; cr++)
        available[cr] = false;
    for (int u = 1; u < V; u++)
    {
        list<int>::iterator i;
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
            if (result[*i] != -1)
                available[result[*i]] = true;
        int cr;
        for (cr = 0; cr < V; cr++)
            if (available[cr] == false)
                break;
        result[u] = cr; // Assign the found color
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
            if (result[*i] != -1)
                available[result[*i]] = false;
    }
}
```

Snake Ladder Problem

```
void getCoordinate(int n,int newx, int &r, int &c)
{
    r = n-1-(newx-1)/n;
    c = (newx-1)%n;
    if((n%2==1 && r%2==1) || (n%2==0 && r%2==0))
        c = n-1-c;
}
int snakesAndLadders(vector<vector<int>>& board)
{
    int n=board.size();
    vector<bool>visited(n*n+1,false);
    queue<pair<ll,ll>>q;
    q.push(mp(1,0));
    visited[1]=true;
    while(!q.empty())
    {
        pair<ll,ll>front=q.front();
        q.pop();
        int x=front.fi;
        int steps=front.se;
        if(x==n*n)
            return steps;
        for(ll i=1;i<=6 and x+i<=n*n;i++)
        {
            ll newx=x+i;
            ll r=0,c=0;
            getCoordinate(n,newx,r,c);
            newx=(board[r][c]==-1)?newx:board[r][c];
            if(!visited[newx])
            {
                q.push(mp(newx,steps+1));
                visited[newx]=true;
            }
        }
    }
    return -1;
}
```

Bridges in a Graph(O(V+E))

```
void dfs(int node, int parent, vector<int> &vis, vector<int> &tin, vector<int> &low, int &timer, vector<int> adj[]) {  
    vis[node] = 1;  
    tin[node] = low[node] = timer++;  
    for(auto it: adj[node]) {  
        if(it == parent) continue;  
  
        if(!vis[it]) {  
            dfs(it, node, vis, tin, low, timer, adj);  
            low[node] = min(low[node], low[it]);  
            if(low[it] > tin[node]) {  
                cout << node << " " << it << endl;  
            }  
        } else {  
            low[node] = min(low[node], tin[it]);  
        }  
    }  
}  
int main() {  
    int n, m;  
    cin >> n >> m;  
    vector<int> adj[n];  
    for(int i = 0;i<m;i++) {  
        int u, v;  
        cin >> u >> v;  
        adj[u].push_back(v);  
        adj[v].push_back(u);  
    }  
  
    vector<int> tin(n, -1);  
    vector<int> low(n, -1);  
    vector<int> vis(n, 0);  
    int timer = 0;  
    for(int i = 0;i<n;i++) {  
        if(!vis[i]) {  
            dfs(i, -1, vis, tin, low, timer, adj);  
        }  
    }  
  
    return 0;  
}
```

Articulation Point

```
void dfs(int node, int parent, vector<int> &vis, vector<int> &tin, vector<int> &low, int &timer, vector<int> adj[], vector<int> &isArticulation) {
    vis[node] = 1;
    tin[node] = low[node] = timer++;
    int child = 0;
    for(auto it: adj[node]) {
        if(it == parent) continue;

        if(!vis[it]) {
            dfs(it, node, vis, tin, low, timer, adj, isArticulation);
            low[node] = min(low[node], low[it]);
        }
        else {
            low[node] = min(low[node], tin[it]);
        }
    }

    if(parent == -1 && child > 1) {
        isArticulation[node] = 1;
    }
}
int main() {
    int n, m;
    cin >> n >> m;
    vector<int> adj[n];
    vector<int> tin(n, -1);
    vector<int> low(n, -1);
    vector<int> vis(n, 0);
    vector<int> isArticulation(n, 0);
    int timer = 0;
    for(int i = 0;i<n;i++) {
        if(!vis[i]) {
            dfs(i, -1, vis, tin, low, timer, adj, isArticulation);
        }
    }
    return 0;
}
```

Strongly Connected Components (Kosaraju's)

TC : O(N+E)

```
void dfs(int node, stack<int> &st, vector<int> &vis, vector<int> adj[]) {
    vis[node] = 1;
    for(auto it: adj[node]) {
        if(!vis[it]) {
            dfs(it, st, vis, adj);
        }
    }
    st.push(node);
}
void revDfs(int node, vector<int> &vis, vector<int> transpose[]) {
    cout << node << " ";
    vis[node] = 1;
    for(auto it: transpose[node]) {
        if(!vis[it]) {
            revDfs(it, vis, transpose);
        }
    }
}
int main() {
    stack<int> st;
    vector<int> vis(n, 0);
    for(int i = 0;i<n;i++) {
        if(!vis[i]) {
            dfs(i, st, vis, adj);
        }
    }
    vector<int> transpose[n];
    for(int i = 0;i<n;i++) {
        vis[i] = 0;
        for(auto it: adj[i]) {
            transpose[it].push_back(i);
        }
    }
    while(!st.empty()) {
        int node = st.top();
        st.pop();
        if(!vis[node]) {
            cout << "SCC: ";
            revDfs(node, vis, transpose);
            cout << endl;
        }
    }
    return 0;
}
```

BELLMAN FORD(DETECT NEGATIVE CYCLE)

```
struct node {
    int u;
    int v;
    int wt;
    node(int first, int second, int weight) {
        u = first;
        v = second;
        wt = weight;
    }
};

int main(){
    vector<node> edges;
    for(int i = 0;i<m;i++) {
        int u, v, wt;
        cin >> u >> v >> wt;
        edges.push_back(node(u, v, wt));
    }

    int src;
    cin >> src;
    vector<int> dist(N, inf);
    dist[src] = 0;
    for(int i = 1;i<=N-1;i++) {
        for(auto it: edges)
            if(dist[it.u] + it.wt < dist[it.v])
                dist[it.v] = dist[it.u] + it.wt;
        int fl = 0;
        for(auto it: edges) {
            if(dist[it.u] + it.wt < dist[it.v]) {
                cout << "Negative Cycle";
                fl = 1;
                break;
            }
        }
        if(!fl) {
            for(int i = 0;i<N;i++) {
                cout << i << " " << dist[i] << endl;
            }
        }
    }
    return 0;
}
```

TC: $O(V^*E)$

Check Bipartite:

```
bool bipartiteBfs(int src, vector<int> adj[], int color[]) {
    queue<int>q;
    q.push(src);
    color[src] = 1;
    while(!q.empty()) {
        int node = q.front();
        q.pop();

        for(auto it : adj[node]) {
            if(color[it] == -1) {
                color[it] = 1 - color[node];
                q.push(it);
            } else if(color[it] == color[node]) {
                return false;
            }
        }
    }
    return true;
}
bool checkBipartite(vector<int> adj[], int n) {
    int color[n];
    memset(color, -1, sizeof color);
    for(int i = 0;i<n;i++) {
        if(color[i] == -1) {
            if(!bipartiteBfs(i, adj, color)) {
                return false;
            }
        }
    }
    return true;
}
int main() {
    int n, m;
    cin >> n >> m;
    vector<int> adj[n];
    for(int i = 0;i<m;i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
}
```

Cheapest flights within K stops

```
int findcheapestPrice(int n, vector<vector<int>>& flights, int src, int dst, int k) {
    vector<pair<int,int>>adj[n];
    for(int i=0;i<flights.size();i++)
    {
        vector<int>v=flights[i];
        adj[v[0]].push_back({v[1],v[2]});
    }
    queue<pair<int,int>>q;
    q.push({src,0});
    int ans=INT_MAX;
    k++;
    while(!q.empty())
    {
        int size=q.size();
        if(k<0)
            break;
        while(size--)
        {
            pair<int,int>curr=q.front();
            q.pop();
            int new_node=curr.first;
            int dist=curr.second;
            if(new_node==dst&&k>=0)
            {
                ans=min(ans,dist);
            }
            for(auto it:adj[new_node])
            {
                if(it.second+dist>ans)
                    continue;
                q.push({it.first,it.second+dist});
            }
        }
        k--;
    }
    if(ans==INT_MAX)
        ans=-1;
    return ans;
}
```

Oliver and the Game

```
vector<int>in(MAXN,0);vector<int>out(MAXN,0);vector<int>visited(MAXN,0);vector<int>adj[MAXN];  
int timer=0;  
void dfs(int i)  
{  
    visited[i]=1;  
    in[i]=timer++;  
    for(int v:adj[i])  
    {  
        if(!visited[v])  
            dfs(v);  
    }  
    out[i]=timer++;  
    //left phle akar khtm ho gaya hga islie 0 6 5 bali query m false h kyuki 6 right m h aur 5 left m th phle right se dfs pura hga th uska in and out phle hi assign ho chuka hga  
}  
bool check(ll x,ll y)  
{  
    if(in[x]<in[y] and out[x]>out[y])//check wheter y lies in x subtree or not  
        return 1;  
    return 0;  
}  
signed main()  
{  
    ll n;  
    cin>>n;  
    ll m=n-1;  
    while(m--)  
    {ll u,v;cin>>u>>v;adj[u].pb(v);}  
    dfs(1);  
    ll q;cin>>q;  
    while(q--)  
    {  
        ll state,oliver,bob;  
        cin>>state>>oliver>>bob;  
        if(state==0)  
        {  
            if(check(oliver,bob))//oliver phle ho  
                yes;  
            else  
                no;  
        }  
        else if(state==1)  
        {  
            if(check(bob,oliver))//bob phle ho  
                yes;  
            else  
                no;  
        }  
    }  
}
```

```

bool canMeasureWater(int j1, int j2, int target) {
    if(j1+j2==target||j1==target||j2==target)
        return true;
    if(j1+j2<target)
        return false;
    queue<pair<int,int>>q;
    q.push({0,0});
    map<pair<int,int>,bool>m;
    while(!q.empty())
    {
        pii front=q.front();q.pop();int newx,newy;int x=front.first;int y=front.second;
        m[{x,y}]=true;
        if(x+y==target) return true;
        //fill x
        newx=j1;
        if(m[{newx,y}]==false)q.push({newx,y});
        //fill y
        newy=j2;
        if(m[{x,newy}]==false)q.push({x,newy});
        //empty x
        newx=0;
        if(m[{newx,y}]==false)q.push({newx,y});
        //empty y
        newy=0;
        if(m[{x,newy}]==false)q.push({x,newy});
        //transfer y->x;
        newx=x+min(j1-x,y);newy=y-min(j1-x,y);
        if(m[{newx,newy}]==false)q.push({newx,newy});
        //transfer x->y
        newy=y+min(j2-y,x);newx=x-min(j2-y,x);
        if(m[{newx,newy}]==false)q.push({newx,newy});
    }
    return false;
}

```

**Disjoint Set Union(TC of find par function is
4Alpha it is mathematically proven)**

```
void makeSet() {
    for(int i = 1;i<=n;i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}
// 7 -> 6 -> 4 -> 3
int findPar(int node) {
    if(node == parent[node]) {
        return node;
    }

    return parent[node] = findPar(parent[node]);
}

void union(int u, int v) {
    u = findPar(u);
    v = findPar(v);

    if(rank[u] < rank[v]) {
        parent[u] = v;
    }
    else if(rank[v] < rank[u]) {
        parent[v] = u;
    }
    else {
        parent[v] = u;
        rank[u]++;
    }
}
```

Kruskal's Algo (TC: ELOG(E))

```
struct node {
    int u;
    int v;
    int wt;
    node(int first, int second, int weight) {
        u = first;
        v = second;
        wt = weight;
    }
};

bool comp(node a, node b) {
    return a.wt < b.wt;
}

int findPar(int u, vector<int> &parent) {
    if(u == parent[u]) return u;
    return parent[u] = findPar(parent[u], parent);
}

void unionn(int u, int v, vector<int> &parent, vector<int> &rank) {
    u = findPar(u, parent);
    v = findPar(v, parent);
    if(rank[u] < rank[v]) {
        parent[u] = v;
    }
    else if(rank[v] < rank[u]) {
        parent[v] = u;
    }
    else {
        parent[v] = u;
        rank[u]++;
    }
}
```

```
int main(){
    int N,m;
    cin >> N >> m;
    vector<node> edges;
    for(int i = 0;i<m;i++) {
        int u, v, wt;
        cin >> u >> v >> wt;
        edges.push_back(node(u, v, wt));
    }
    sort(edges.begin(), edges.end(), comp);

    vector<int> parent(N);
    for(int i = 0;i<N;i++)
        parent[i] = i;
    vector<int> rank(N, 0);

    int cost = 0;
    vector<pair<int,int>> mst;
    for(auto it : edges) {
        if(findPar(it.v, parent) != findPar(it.u, parent)) {
            cost += it.wt;
            mst.push_back({it.u, it.v});
            unionn(it.u, it.v, parent, rank);
        }
    }
    cout << cost << endl;
    for(auto it : mst) cout << it.first << " - " << it.second << endl;
    return 0;
}.
```

Floyd Warshall Algo:

```
t{
    ll n;
    cin>>n;
    ll a[n][n];
    for(i,n)
    {for(j,n)cin>>a[i][j];}
    ll dist[n][n];
    for(i,n)
    {for(j,n)dist[i][j]=a[i][j];}
    ll i,j,k;
    for (k = 0; k < n; k++)
    {
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < n; j++)
            {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < n; j++)
        {
            if (dist[i][j] == INF)
                cout<<"INF"<<" ";
            else
                cout<<dist[i][j]<<" ";
        }
        cout<<endl;
    }
}
```

Minimum Cost Path

```
#define fi first
#define se second
int dx[] = {-1, 0, 1, 0};
int dy[] = {0, 1, 0, -1};
class Solution
{
public:
    int minimumCostPath(vector<vector<int>>& grid)
    {
        int n=grid.size();
        int m=grid[0].size();
        vector<vector<int>> dist(n, vector<int>(m, INT_MAX));
        set<pair<int,pair<int,int>>>s;
        s.insert(make_pair(0,make_pair(0,0)));
        dist[0][0]=grid[0][0];
        while(!s.empty())
        {
            pair<int,pair<int,int>> front=*s.begin();
            s.erase(s.begin());
            int curr_dist=front.fi;
            int x=front.se.fi;
            int y=front.se.se;
            for(int i=0;i<=3;i++)
            {
                int newx=x+dx[i];
                int newy=y+dy[i];
                if(newx>=n||newy>=m||newx<0||newy<0)
                    continue;
                if(dist[newx][newy]>dist[x][y]+grid[newx][newy])
                {
                    if(dist[newx][newy]!=INT_MAX)
                        s.erase(s.find(make_pair(dist[newx][newy],make_pair(newx,newy))));
                    dist[newx][newy]=dist[x][y]+grid[newx][newy];
                    s.insert(make_pair(dist[newx][newy],make_pair(newx,newy)));
                }
            }
        }
        return dist[n-1][m-1];
    }
};
```

Total number of Spanning Tree Possible:

If a graph is a complete graph with n vertices, then total number of spanning trees is $n^*(n-2)$ where n is the number of nodes in the graph.

STEP 1: Create Adjacency Matrix for the given graph.

STEP 2: Replace all the diagonal elements with the degree of nodes. For eg. element at (1,1) position of adjacency matrix will be replaced by the degree of node 1, element at (2,2) position of adjacency matrix will be replaced by the degree of node 2, and so on.

STEP 3: Replace all non-diagonal 1's with -1.

STEP 4: Calculate co-factor for any element.

STEP 5: The cofactor that you get is the total number of spanning tree for that graph.

Number of Islands

```
int dx[4]={0,1,0,-1};
int dy[4]={1,0,-1,0};
bool isSafe(int r,int c,int n,int m,vector<vector<char>>& grid)
{
    if(r<0||r>=n||c<0||c>=m||grid[r][c]=='2'||grid[r][c]=='0')
        return false;
    return true;
}
void dfs(int r,int c,int n,int m,vector<vector<char>>& grid)
{
    grid[r][c]='2';
    for(int i=0;i<4;i++)
    {
        int newx=r+dx[i];
        int newy=c+dy[i];
        if(isSafe(newx,newy,n,m,grid))
            dfs(newx,newy,n,m,grid);
    }
}
int numIslands(vector<vector<char>>& grid) {
    int n=grid.size();
    int m=grid[0].size();
    int ans=0;
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            if(grid[i][j]=='1')
            {
                dfs(i,j,n,m,grid);
                ans++;
            }
        }
    }
    return ans;
}
```

TRIE:

MAXIMUM XOR BETWEEN ELEMENTS FROM TWO DIFF ARRAYS

```

1 class Trie {
2     public:
3     int data;
4     Trie* child[2];
5     Trie(){
6         data = 0;
7         for(int i=0; i<2; i++) {
8             child[i] = NULL;
9         }
10    }
11 };
12 void insert(Trie* root, int key) {
13     Trie*temp=root;
14     for(int i=31;i>=0;i--)
15     {
16         int x=(key>>i)&1;
17         if(temp->child[x]==NULL)
18             temp->child[x]=new Trie();
19             temp=temp->child[x];
20     }
21     temp->data=key;
22 }

```

```

int helper(Trie* root, int key) {
    Trie* temp = root;
    for(int i=31; i>=0; i--) {
        // find current bit in given prefix
        int x = (key >> i) & 1;

        // traverse trie, look for prefix that has opposite bit
        if(temp->child[1-x]) temp = temp->child[1-x];
        // if there is no opposite bit then looking for same bit
        // always true if above condition is false
        else if(temp->child[x]) temp = temp->child[x];
    }

    return key^(temp->data);
}

int Solution::solve(vector<int> &nums, vector<int> &B) {
    Trie* root = new Trie();
    for(int i=0; i<nums.size(); i++) {
        insert(root, nums[i]);
    }
    int ans = 0;
    for(int i=0; i<B.size(); i++) {
        //find max XOR value of current element with other elements
        ans = max(ans, helper(root, B[i]));
    }

    return ans;
}

```

Autocomplete Using Tries

