/\* learnings are-

1. use of bool for printing

2. for vector

{1,2,4,5}=>{1,2,3,4,5}

v.insert(v.begin()+2,3)  
V.count(v.begin()+I,v.end(),v[i])

v.erase(v.begin()+2)

for set

setname.contain(nnumber)

set.erase(num)

# Maths

## MOD

# For negative x🡪X% P=====(X%P+P)%P

# In [mathematics](https://en.wikipedia.org/wiki/Mathematics), Legendre's three-square theorem states that a [natural number](https://en.wikipedia.org/wiki/Natural_number) can be represented as the sum of three squares of integers�=�2+�2+�2if and only if *n* is not of the form �=4�(8�+7)N=4^a(8b+7) for nonnegative integers *a* and *b*.

\*/

\*\*For finding the maxfreq of any element in array 🡪

 cin>>n;

      vector<int>cnt(n+1);

      for (int i = 0; i <n; i++)

      {

        int x;

        cin>>x;

        ++cnt[x];

      }

      int mx=\*max\_element(cnt.begin(),cnt.end());

# 2 FOR STRINGS

1 ahhahhah 2 check substring

If we have to find the required string from jumbled string what we do is looping ->

Ex-1->SKDJHhENSALLFOG 2>FIND HELLO

hello=”hello”

cin >> word;

    int i,cnt=0,j=0;

    fo(i,word.length()){

        if(word[i]==hello[j]){

            cnt++,j++;

        }

    }

    if(cnt==5){

        cout<<"YES";

    }

    else{

        cout<<"NO";

    }

2.

* To lower case->transform(s.begin(),s.end(),s.begin(), ::tolower);
* String mai kuch add ya remove krne ke lie new string bhi use kr skte hai
* Remove krne ke lie continue kr skte hai

## CHECKING STRING PALINDROME OR NOT CAN REMOVE “K” ELEMENTS

   int cnt=0;

      for (int i = 0; i <n; i++)

      {

        mp[s[i]-'a']++;

      }

      for (auto c:mp)

      {

        if(c.second%2!=0)cnt++;

      }

      if(cnt>k+1)cout<<"NO"<<endl;

      else cout<<"YES"<<endl;

    }

* If you wants to find the count of character in a string like “8989889” want to find cnt of 8 or 9 you can do this🡪
* Vector of appropriate size vector<int>cnt(10)
* For(auto ch:s) ++cnt[ch-‘0’];
* string s;
* cin>>s;
* vector<int>cnt(10);
* for(auto ch:s)++cnt[ch-'0'];
* int mx=\*max\_element(cnt.begin(),cnt.end());

## IF YOU WANT TO FIND GOOD OR BAD STRINGS

BINARY WITH 0,1

CHECK CONDITIONS AND BOOLEAN

\*\* TO CHECK IF A string contain unique digit approach change no. to string using by to\_string(n) then put element of string to a set<int>s then compare length

\*\*agar koi number bhut bda ho to phele use string mai convert krke

String func(string s ) string he return krdo..(long long ka issue he nhi);

\*\*binary removal-🡪if 1100 then not possible to form a sorted string from 010111010100 in which can be formed by removing.

std::cout << "Before memset: " << str << std::endl; 🡪hello,world

// Set the first 5 characters of str to 'a'

memset(str, 'a', 5);

std::cout << "After memset: " << str << std::endl;s

## memset(str, 'a', 5);🡪aaaaa,world

## CREATING STRING –

STRING(SIZE,char);

### Creating substring

String original;

String temp=original.substr(starting index, length);

#### If we don’t know the starting index=

String temp=original.substr(original.find(“name”),len);

# KDANE ALGO

1. int best = 0, sum = 0; for (int k = 0; k < n; k++) {

sum = max(array[k],sum+array[k]);

best = max(best,sum); }

cout << best << "\n";

### variation of kdane in which sum cannot be less than 0

ll best = 0, sum = 0;

 for (int j = 0; j < n; j++) {

 sum +=v[j];

 best = max(best,sum);

 sum=max(sum,0ll);

 }

## 🡪MAXI VALA TRIKA

Comparator is used for swapping of pairs-

Bool comp(vector<int>&ele1,vector<int>&ele2){

Return ele1[1]<ele2[1];

}

# Prefix and suffix

vector<int>prefix(n,0);

        vector<int>suffix(n,0);

        vector<int>ans(n,0);

        prefix[0]=nums[0];

        for(int i=1;i<n;i++){

          prefix[i]=prefix[i-1]\*nums[i];

        }

        suffix[n-1]=nums[n-1];

        for(int i=n-2;i>=0;i--){

          suffix[i]=suffix[i+1]\*nums[i];

        }

        ans[0]=suffix[1];

        ans[n-1]=prefix[n-2];

USE OF PREFIX AND SUFFIX SUM TO FIND UNIQUE ELEMENT

vi prefix(n,0),suff(n,0);

 fo(i,n){

    se.insert(s[i]);

    prefix[i]=se.size();

 }

 se.clear();

 for(int i=n-1;i>=0;i--){

    se.insert(s[i]);

    suff[i]=se.size();

 }

se.clear();

one of the imp concept---

## prefix sum+hashing to find a subarray with 0 sum;

bool subarray(vector<int>&v,int n){

  unordered\_set<int>us;

  int sum=0;

  for (int i = 0; i <n;i++)

  {

   //v[i]\*=i%2==0?-1:1;

    sum+=v[i];

   if(sum==0||us.find(sum)!=us.end()){

    return true;

   }

   us.insert(sum);

 }

 return false;

  }

## Prefix sum

prefixs[n+1]{};

prefixs[0]=a[0];

for (int  i = 1; i <=n; i++)

{

  prefixs[i]+=prefixs[i-1]+a[i];

}

//return prefixs[r]-prefixs[l-1];

# Structure binding

For(auto[a,b]:mp(anydatastructure name){  
cout<<a<<b<<endl;

}

1. 1 ques reference for using 2 array as one-
2. **vector**<pair<**int**,**int**>> sorted(n);
3. **for**(**int** i = 0; i < n; i++){
4. cin >> sorted[i].second;
5. }
6. **for**(**int** i = 0; i < n; i++){
7. cin >> sorted[i].first;
8. sorted[i].first = abs(sorted[i].first);
9. }

## maxoccurence

// max occurence

map<char,int>mp;

for(auto c:s){

  maxi=max(maxi,++mp[c]);

}

for(auto c:s){

  if(mp[c]==maxi)

    cout<<c<<" ";

  cout<<endl;

}

# sliding window

Subarray/subtring+largest or max+k(size of window)

sliding window

int findSubstring(string s){

vector<int> map(128,0);

int counter; // check whether the substring is valid

int i=0,j=0,cnt=0; //two pointers, one point to tail and one head

int n; //the length of substring

for(int i=0;i<n;i++) { /\* initialize the hash map here \*/

}

while(j<n){

if(map[s[end++]]-- ?){ /\* modify counter here \*/ }

while(/\* counter condition \*/){

/\* update d here if finding minimum\*/

//increase begin to make it invalid/valid again

}

/\* update cnt here if finding maximum\*/

}

return cnt;

}

# 6 For map or pairs

Give pairs of map

1. Mp.find()🡪iterator
2. Mp.contains()

# 7 set

Set can use pairs but uset can not

Subset(not ordr)=powerset!=subsequence(not cont but order) substring(continuos) \*\*subarray or substring=continuous

## Iterating a set and printing its element

void display(set<string>&s)

{

  set<string>::iterator itr;

  // Displaying set elements

  for (itr = s.begin();

       itr != s.end(); itr++)

  {

    cout << \*itr << " ";

  }

}

# 8 iterators

Iterator.begin()+x=interator to index

## Index of max element

### Max\_element(v.begin(),v.end())-v.begin() gives the index of max element

# 9 comaparator

bool comparatar(datatype a ,datatype b){

    if(a<b)return true ;

    else return false;

}

# 10 binarysearch STL

BINARY\_SEARCH(V,V+7,VALUE )🡪TRUE OR FALSE

LOWER BOUND->FIRST VAL>=T

UPPER BOUND ->FIRST VAL >T

LB-1<T

UB-1<=T

WATCH EDGE CASES SUCH AS WHEN (IT==A)BEGINNING INDEX THEN WE CAN NOT IT--;

## \*\*Floor and ceil should lean toward the changing side. var!=mean

### = on the side which we have to get rid of.(last or first index)

bs wiill move l🡪mid when pred true(left udao)

bs will move r->mid when pred false(right udao)

bt find tf junction ans is last true or first false(l,r)

## ALTERNATE BINARY SEARCH

l=minspace-1,r=maxspace+1

   long firstBadVersion(long n) {

     long l = 0;

    long r = n+1;

    while (r-l>1) {

        long m = l + (r - l) / 2;

        if (isBadVersion(m)==0) {

            l = m;

        } else {

            r = m;

        }

    }

        return r;

    }

in real valyue preferred log2max/e

recurssion

int count = 0; // Global parameters that are useful for results.

int minEnd = INT\_MAX; // Key parameters characterizing the "active set" for overlapping intervals, e.g. the minimum ending point among all overlapping intervals.

sort(points.begin(), points.end()); // Sorting the intervals/pairs in ascending order of its starting point

for each interval {

      if(interval.start > minEnd) { // If the

     // changing some states, record some information, and start a new active set.

    count++;

    minEnd = p.second;

      }

     else {

    // renew key parameters of the active set

    minEnd = min(minEnd, p.second);

      }

 }

return the result recorded in or calculated from the global information;

int findMinArrowShots(vector<pair<int, int>>& points) {

int count = 0, minEnd = INT\_MAX;

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

for(auto& p: points) {

if(p.first > minEnd) {count++; minEnd = p.second;}

else minEnd = min(minEnd, p.second);

}

return count + !points.empty();

}

# LINKEDLIST

REVERSE A LINKED LIST

ListNode\* fn(ListNode\* head) {

ListNode\* curr = head;

ListNode\* prev = nullptr;

while (curr != nullptr) {

ListNode\* nextNode = curr->next;

curr->next = prev;

prev = curr;

curr = nextNode;

}

return prev;

}

# BACKTRACKING

Steps can be think of as subset because result is a 2d vector and steps is 1d vector of subset

## TEMPLATE

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

vector<int> step;

vector<vector<int>> result;

helper(nums, 0, step, result);

return result;

}

void helper(const vector<int>& nums, int start, vector<int>& step, vector<vector<int>>& result) {

///CONDITION TO PUSH IN RESULT

result.push\_back(step);

for (int i = start; i < nums.size(); ++i) {

//condition to push in step

step.push\_back(nums[i]);

helper(nums, i + 1, step, result);

step.pop\_back();

}

}

## GENERATE ALL STIRNG PERMUTATION

class Solution {

private:

void recurPermute(int index, vector<int> &nums, vector<vector<int>> &ans) {

if(index == nums.size()) {

ans.push\_back(nums);

return;

}

for(int i = index;i<nums.size();i++) {

swap(nums[index], nums[i]);

recurPermute(index+1, nums, ans);

swap(nums[index], nums[i]);

}

}

public:

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

vector<vector<int>> ans;

recurPermute(0, nums, ans);

return ans;

}

};

# DP

## LCS-longest common subsequence

  int dp[1001][1001];

    int LCS(string text1, string text2) {

        int n=text1.size();

        int m=text2.size();

         for(int i=0;i<n+1;i++){

        for(int j=0;j<m+1;j++){

             if(i==0||j==0)dp[i][j]=0;

            }

        }

         for(int i=1;i<n+1;i++){

        for(int j=1;j<m+1;j++){

             if(text1[i-1]==text2[j-1]){

                dp[i][j]=1+dp[i-1][j-1];

             }

             else {

               dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

             }

            }

        }

        return dp[n][m];

    }