(sieve method) (O(n * log(log n)))

1. function to count number of prime numbers between 0 and number n

```
int countprime(int n){
 int cnt = 0
 vector<bool> prime(n+1, true);
 prime[0] = prime[1] = false;
 for(int i=2; i<n; i++){
  if(prime[i]){
    cnt++;
  for(int j=2*i; j<=n; j=j+i)
  {  prime[j] = 0
   }
  }
 }
 return cnt;  }
```

2. gcd(a, b) = gcd(a-b, b)
   cont till one is 0

```
int gcd(int a, int b){
  if(b==0)
  return a
  return gcd(b, a%b);
 }        // doesn't work if a=0
```

★ lcm(a,b) * gcd(a,b) = a * b

- left shift

$(x << y) == (x \times 2^{\wedge} y)$

$(x >> y) == (x \div 2^{\wedge} y)$

---

unordered_map<pair<int,int>, int> ok;

if(ok.find({d,K}) == ok.end())
    ok.insert({{d,K},1});
else
    ok[{d,K}]++;

Recursion

print → parameters

printfans → return true/false

count return 1/0
(add all 1's)

int **arr = new int*[n];
for(int i=0; i<n; i++){
  arr[i] = new int[n]
}
//dynamically allocate 2D arr

string decimalToBase(int n)
  string s = bitset<64>(n).to_string();
  const auto loc = s.find('1');
  if(loc != string::npos)
    return s.substr(loc);
  return "0";

mod props

- $(a+b)\%m = a\%m + b\%m$
- $(a-b)\%m = a\%m - b\%m$
- $(a*b)\%m = a\%m * b\%m$

→ In a 2D matrix to find a mid index to apply binary search

mid

→ int element = matrix[mid/col]
[mid%col];

- Queue using 2 stacks

deQueue

- if both stacks are empty print error
- if stack2 is empty
  push every thing from 1 to 2
  Pop the element from stack 2.

```
void push(int x)
{ S1.push(x); }
int pop() {
if (S1.empty() and S2.empty()
    Queue empty
if (S2.empty()) {
while (!S1.empty()) {
   S2.push(S1.top());
   S1.pop(); }
}
int top val = S2.top(); S2.pop()
return topval; }
```

Return all possible subsets

__/__/__

```
vector<vector<int>> ans;
vector<int> output;
int index = 0;
solve(nums, output, index, ans);
return ans;

void solve(vector<int> nums, vector<int>
output, int index, vector<vector<int>> &ans)
{ if(index >= nums.size()){
  ans.push_back(output);
  return;
}
//exclude
solve(nums, output, index+1, ans);
int element = nums[index];
output.push_back(element);
solve(nums, output, index+1, ans);
```

3 ({1,2,3}, {})

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

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

nCr

(2) Ya calculate ~~factorial~~ in O(1)

```
for(int i=0; i<k; i++)
  res *= (n-i);
  res /= (i+1);
}
```

$\frac{n!}{(n-r)!\,r!}$

$^5C_3$ → $\frac{1\times2\times3\times4\times5}{1\times2\times1\times2\times3}$

Striver

180

① Pascals Triangle

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

⑤ R ⟹ ③ R−1 C C−1 (if row/col & value asked)

or generally

→ vector<vector<int>> r(numRows); eg ${}^4C_2 = 6$

```
for(int i=0; i<numRows; i++){
  r[i].resize(i+1);
  r[i][0] = r[i][i] = 1
  for(int j=1; j<i; j++)
    r[i][j] = r[i-1][j-1] + r[i-1][j]
  }
return r;
}
};
```

② Kadane's maximum subarray (return sum of max subarr)

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

Pascal triangle to print n++ rows

```
ans = 1
cout<<ans
for(i=1; i<n; i++){
ans = ans * (n-i)
ans = ans / i;
cout<<ans; }
```

③ Sort array of (0, 1, 2)

→ Counting sort can be done
count number of zeros, 1's, 2's, place in vector

④ Best Time to Buy and sell stocks →

```
int fun (vector int &prices) {
int meh = prices.size()-1;
int maxi = prices[meh];
int mini = prices[0];
int dif = maxi - mini;
for( i → prices.size-1)
  if (mini > prices[i]) {
    mini = prices[i];
    maxi = mini;
  if (maxi < prices[i]) {
    maxi = prices[i];}
  if (dif < maxi - mini) {
    dif = maxi - mini;
  }
// check more for last element
return dif;
```

7 1 5 3 6 4

S
{1 → 6}

3 2 10 3 9 1
min 2 max 9

maxprice = max(Ai, maxprice)
or sum from right side
profit = max(profit, maxprice - A[i])

⑤ given a m x n matrix → if matrix's element is zero → set all row (colum which is come as zero

→ create a temp vector
for row, for column →

```
vector(int) one(matrix.size(), 1)
vector(int) two(matrix[0].size(), 1)
for(i=0 → row size)
  for(j → col size)
    if matrix[i][j] == 0 →
      one[i] = 0;
      two[j] = 0;
for(i → row) {
for(j → col) {
  if (one[i] == 0 || two[j] == 0)
    matrix[i][j] = 0
  }
}
```

[1 1 1]
[1 0 1]
[1 1 1]

[1 0 1]
[0 0 0]
[1 0 1]

⑥ next permutation $O(n \times n!)$

C++ inbuilt → next_permutation(arr, arr+3)

// for strings

```
string s = "glg";
bool val = next_permutation(s.begin(), s.end());
if (val == false)
    cout << "No. word Possible";
else
    cout << s << endl;
return 0;
```

→ general approach 1 3 5 4 2

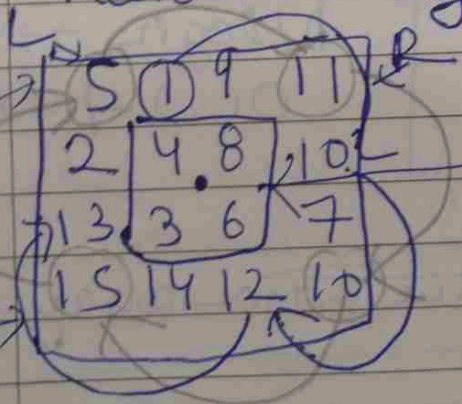
step ① traverse from R & check and find element smaller than previous element

step ② now traverse again and find the first element greater than that th now swap them

→ 1 4 5 3 2

---

⑦ Rotate Image $\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} \rightarrow \begin{bmatrix}7&4&1\\8&5&2\\9&6&3\end{bmatrix}$



L < R

→ Approach ① → Transpose entire matrix

② → (reverse each row)

```
void rotate(vector<vector<int>>& m) {
    int n = m.size();
    for(int i=0; i<n; i++)
        for(int j=0; j<i; j++)
            swap(m[i][j], m[j][i]);
    for(int i=0; i<n; i++)
        reverse(m[i].begin(), m[i].end());
}
```

(code to transpose a matrix)

8 merge two sorted arrays in O(1) space

```
1 4 7 8 10
2 3 9
```

```
void merge(a1[], m, a2[], n){
  for(i=n-1 to 0; i--)
  { int j;
  int last = a1[m-1];
  for(j=m-2; j>=0 && a1[j]>a2[i]; j--);
  a1[j+1] = a1[j]
  if(j!=m-2 || last>a2[i])
  { a1[j+1] = a2[i];
    a2[i] = last;
  }
}
```

gap algorithm

total size → 8/2 = 4

1 4 7 8 10 2 3 9 → second half pointer moves at second

swap swap

1 2 3 8 10 4 7 9

4/2 → 2

swap...

gap = 2/2 = 1

1 2 3 4 7 8 10 9

{ gap = ceiling(5/2) = 3

Nlogn → time

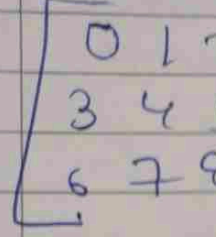
O(1) space

(or simply use extra array)

```
int p1 = m-1, p2 = n-1, i = m+n-1
while(p2>=0){ if(p1>=0 && nums1[p1]
nums1[i--] = nums1[p1--]  nums2[p2]
else { nums1[i--] = nums2[p2--];
}
```

start from here

9 search in a 2D matrix with each element sorted row wise




```
int i=0, j = matrix(0) size -1
while (i<matrix.size() && j>=0)
if (matrix[i][j] == target){
    return i;
}
else if (matrix[i][j] > target)
{ j--; }
else
{ i++; }
```

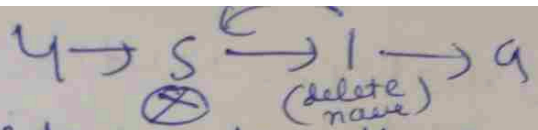
→ Or just use binary search with size → (m×n-1) and mid =
(i) = index/col
(j) = index % col

10 find majority element → n/3 size

```
, map<int, int> a;
→ { push elements in map }
→ traverse map an  for(auto i:m)
                    if i.second > n/3
                    ans.push_back(i.first);
```

(11) find middle of the linked list →

```
list node * temp = head; int count = 0
while (temp != NULL){
    count++; temp = temp->next;
List node * var = head
for(i=0; i < count/2; i++){
    var = var->next;
}
return var;
```



12 delete a node in the linked list; (head is not given)
(copy in node
delete next node)

13 two-sum
① brute force $O(n^2)$
② Hash table (on) (vector<int>& nums, target)
vector<int> ans;
unordered_map<int, int> mpp;
for(int i=0; i<nums.size(); i++){
if(mpp.find(target - nums[i]) != mpp.end()){
ans.push_back(mpp[target - nums[i]]);
ans.push_back(i);
return ans;
}
mpp[nums[i]] = i;
}return ans;
}

14 remove n^{th} node from end() →
count all nodes
ans = count - n;
if(count == n) // delete first node
move head to next;
else if ~~head~~(n == 1) // delete last node
ListNode *fast = head
ListNode *slow = head
while(fast->next != NULL)
slow = fast; fast = fast->next; }
slow->next = NULL
else
traverse ans times
temp->val = temp->next->val
temp->next = temp->next->next; }
return head

// optimized approach

→ create a slow, fast pointer →
move fast pointer n times,
→ now move both fast and slow pointer till fast reaches last node
→ delete slow pointer.
→ handle edge cases,

73 set matrix zero →

1	1	1		1	0 1
1	0	1	→	0	0 0
1	1	1		1	0 1

create two vectors
→ one of row size
other of column size
iterate through the array and if zero found → set index arrays' corresponding i, j to be zero →

2 → now iterate through i x j and set matrix[i][j] to be zero if (row[i]==0 || col[j]==0)

16 largest consecutive sequence
[10, 4, 100, 1, 101, 3, 2] → 1, 2, 3, 4
= 4

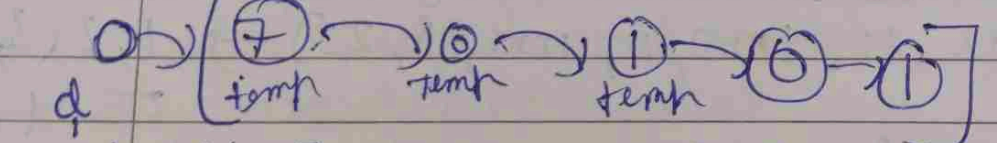
- Hash the elements
- check if previous exist
  if it does do nothing
  if it doesn't → count if next elements present

```
int longestConsecutive(vector<int> &nums){
unordered_set<int> hash;
for(int num : nums)
{ hash.insert(num);}
```

```
int longest streak = 0;
for( int num : nums ){
 if( !hash.count(num-1) )
    int currentNum = num
    int current.streak = 1;
    while( hash.count(currentnum+1) ){
          currentnum += 1;
          currentstreak += 1; }
    longeststreak = max(longest streak,
                        currentstreak);
      }
  }
  return longest streak;
  }
```

~~Longest substring without Repeating characters~~

Q Adding two numbers as linked list given in reverse order 2 → 4 → 3 → X
C=1 5 → 6 → 7 → 9



{342} + {5679}

```
temp ListNode* .add (ListNode* l1, ListNode* l2) {
   ListNode *dummy = new ListNode();
  ListNode *temp = dummy;
  int carry = 0;
  while (l1 != NULL || l2 != NULL || carry) {
  int sum = 0;
  if (l1 != NULL) {
      sum += l1->val;
      l1 = l1->next; }
```

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

18→ matrix multiplication

```
arr[a][b]   arr[c][d]  {b=c}
crr[a][d]
for(i→a)
  for(j→d)
    crr[i][j]=0
  for(k=0→b)
    crr[i][j] += arr[i][k] * arr[k][j]
}}}
```

19→ checking for loop in linked list → (cycle in linked list)

```
ListNode * fast = head
ListNode * slow = head;
while(fast != NULL){ if(fast == NULL) return 0;
else { if(fast->next == NULL) return 0;
else fast = fast->next; }
slow = slow->next;
if(fast == slow) return 1;
```

20 check if ll is a palindrome
① convert to string
check for palindrome.

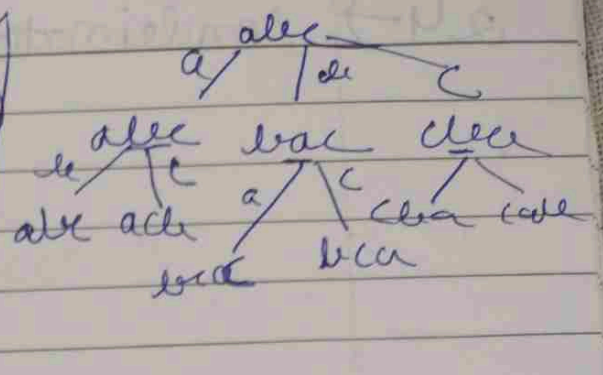
21 → Subsequence of a string →
→ eg "abc" → a, b, c, "", ab, bc, ca, abc,

```
→ vector<string> subsequence(string str){
  vector<string> ans;
  string output = "";
  int index = 0;
  solve(str, index, output, ans)
  return ans;
}
void solve(string, output, index,
  vector<string>&ans) {
  //base case
  if(index >= str.length()) {if(output.length()>0)
    ans.push_back(output);
  }
  //exclude
  solve(str, output, index+1, ans);
  //include
  char element = str[index];
  output.push_back(element);
  solve(str, output, index+1, ans);
}
```

eg abc → abc, bac, cab acb, bca, cba
→ total permutation = n!

22 Permutations of a string

```
vector<vector<int>> permute(vector<int>& nums)
{ vector<vector<int>> ans
int index = 0;
solve(nums, ans, index)
  return ans;
}
void solve(vector<int> nums, vector<vector<int>> &ans, int index){
    if (index >= nums.size()) {
      ans.push_back(nums)
      return;
    }
    for (int j = index; j < nums.size(); j++) {
      swap(nums[index], nums[j]);
      solve(nums, ans, index+1);
      swap(nums[index], nums[j]);
    }
}
```



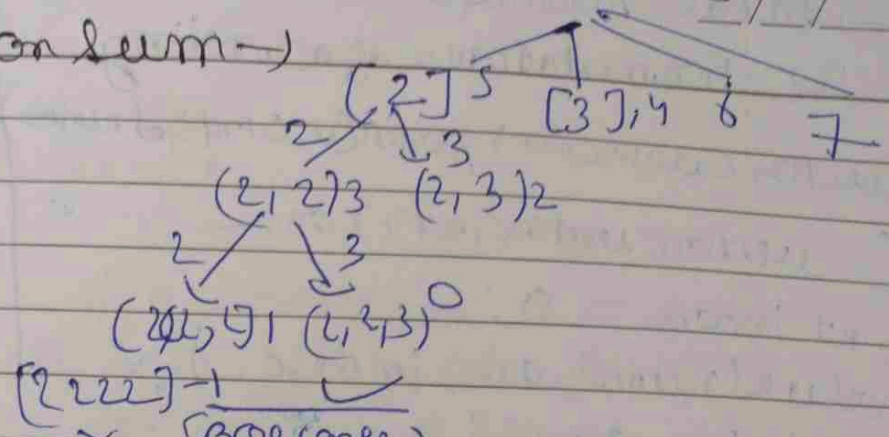
023 Max consecutive ones

(1 1 0 1 1 1 0) // 3

```
for(i → nn.size()) {
    if (i = 1)
      ans.push_back(1)
      if (ans.size()) > max
        max = ans.size
    else
      ans.clear
return max;
```

• [2 3 6 7] target = 7 ⇒ [[2,2,3], [7]]

24 → combination sum →




```
void findcom(int ind, int target, vector<int>&arr, vector<vector<int>>&ans, vector<int>&ds){
    if(ind == arr.size()){
        if(target == 0){
            ans.push_back(ds);
        }
        return;
    }
    if(arr[ind] <= target){
        ds.push_back(arr[ind]);
        findcomb(ind, target - arr[ind], arr, ans, ds);
        ds.pop_back();
    }
    findcombination(ind+1, target, arr, ans, ds);
}
public:
vector<vector<int>> combsum(vector<int>&candidates, int target){
    vector<vector<int>> ans;
    vector<int> ds;
    findcom(0, target, candidates, ans, ds);
    return ans;
}
};
```

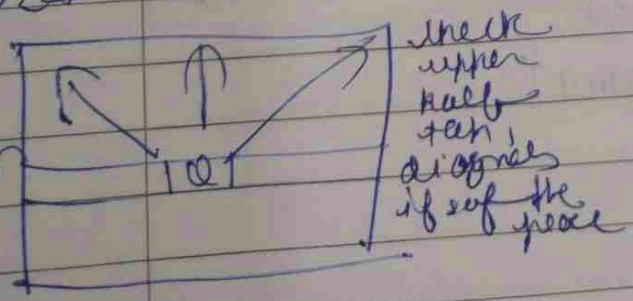
Q25 subarrays brute force

```
for( i(=0)<n )
  for( j = i → n )
    for k(=i → ≤ j ){
      cout<<arr[k]<<" ";
    }
  cout endl }
cout endl
```

Q26 subsets backtracking

```
char tem[100]; char leem[100]; cin>>tem
solve(tem, leem, 0, 0)
void solve( char* tem, char* leem, int i, int j)
{ if (tem[i]==='\0' {
  leem[j]==='\0'); if leem[0]==='\0'
                    cout<<NULL;
else  cout<<leem;
  return
  leem[j] = tem[i]
  solve(tem, leem, i+1, j+1)
  solve(tem, leem, i+1, j+1)
}
```

Q27 NQueens →



Q27 Best time to buy and sell stock - II

// allowed to buy/sell some days,
// only 1 stock at a time

```
ans = 0
for(i=1 -> i -> prices.size; i++)
    if(prices i > prices(i-1])
      ans += prices(i] - prices(i+1]
}} return ans
```

Q28 Minimum coins using dp.

(i) recursion

```
int solve(vector<int> coin, int n) {
if(n==0){
  return 0 }
if(n<0){
  return INT_MAX; }
int mini = INT_MAX;
for(i=0; i<coin.size(); i++)
  int temp = solve(coin, n-coin[i]);
if(temp != INT_MAX){
  mini = min(temp+1, mini);
}
return min;
```

29. Longest substring without repeating <

__/__/__

eg → pwwkew

P	0	i = 5
W	5	start = 2
K	3	
E	4	

```
string s; cin >> s
vector<int> dict(256, -1);
int maxlen = 0, start = -1;
for(int i = 0; i < s.size(); i++){
  if(dict[s[i]] > start)
    start = dict[s[i]];
    dict[s[i]] = i;
    maxlen = max(maxlen, i - start);
}
cout maxlen;
```

30 m coloring problem →

30 Binary tree symmetric →

```
bool temp(struct Node *l, struct Node *r){
  if(l==N && r==N)
    return true
  else(if(l==N || r==N || l->d != r->d)
    return false
  return(temp(l->r, r->l) && (l->l && r->r)
bool is symtric(A N* r)
  if(r==N)
    return true;
  return(temp(root->left, root->right));
}
```

// 2 unique numbers in a set of numbers

(31) → To find first bit which is non zero

```
num ←
int pos=0;
temp=num;
while((temp&1)==0){
pos++;
temp = temp>>1;
}
//pos is the required pos
```

31-1 filter out numbers which have set bit at 'pos'

```
int setA=0, setB=0;
int mask=(1<<pos);
for(i=0;i<arr.size();i++){
if((arr[i] & mask)>0){
  setA = setA ^ arr[i];
else {
   setB = setB ^ arr[i];
  }
cout<<setA<<"first unique";
cout<<setB<<"second unique";
```

32 Binary Tree with max path sum

```
int maxPathSum(TreeNode* root){
int maxi = INT_MIN
maxPathDown(root, maxi);
return maxi; }
```

__/__/__

int maxPathDown(TreeNode* node, int &max)
{ if (node == NULL) return 0;
int left = max(0, maxPathDown(node->left, maxi));
right = max(0, maxPathDown(node->right, maxi))
maxi = max(maxi, left+right+node->val);
return max(left, right)+node->val;
}

33 Check if 2 Binary Tree are same →
→ isvalidSame(TreeNode* p, TreeNode* q)
if (p==NULL || q==NULL)
return p==q
return (p->val==q->val) &&
isSameTree(p->left, q->left) &&
isSameTree(p->right, q->right);

// *min_element() // find min element
x = find(L.begin(), L.end(), m);
↑ find index of min element.

34 LCS (Length)
{ if (i<0 || j<0) return 0
if (t[i]==s[j])
return 1+f(t, s, i-1, j-1)
return max(f(t, s, i-1, j), f(t, s, i, j-1))
}

34 | 3
* Longest palindromic subsequence →
Reverse the string and find LCS of both strings.

35→ minimum insertions to make string palindrome

n → longest palindromic subsequence

wise of string

36 Shortest common supersequence

length = n + m − len(lcs)

```
i = n, j = m
while (i > 0 && j > 0)
{ if (s1[i-1] == s2[j-1])
  { ans += s1[i-1];
    i--, j--;
  else if (dp[i-1][j] > dp[i][j-1])
      ans += s1[i-1];
      i--;
  }
else
    ans += s2[j-1];
    j--;
}
while (i > 0) {
ans += s1[i-1];
i--;
}
while (j > 0)
  ans += s[j-1]; j--;
}
reverse( ans.begin( ), ans.end( ));
```

L

__/__/__

• (Bottom-up → Rules)

① copy the base case

② i =

j =

③ copy the Recurrence

34) Remove middle element from stack

```
void solve(stack<int> &inputstack, int
count, int size) {
  if (count == size/2) {
    inputstack.pop()
  solve(inputstack, count+1, size);
  inputstack.push(num); }
```

37) Buying and selling Stock II

```
f(ind, buy)
{ if(ind == n)
    return 0;

if(buy)
  profit = max{-prices[ind] + f(ind+1, 0) take
              { 0 + f(ind+1, 1) → not take

else
  profit = max{prices[ind] + f(ind+1, 1);
              { 0 + f(ind+1, 0);

}
```

(find min cost to cut the rope)

372 → minimum cost to cut the stick

```
{
  cost = cuts[j+1] - cuts[i-1]
       + f(i, ind-1)
       + f(i+1, j)
  mini = min(mini, cost);
}
return mini;
}
```

038 3sum ⟹ brute force $n^3 \log n$

[-1 0 1 2 -1 -4]

```
for(i=0→n-1)
  hash[a[i]]--
  for(j=i+1→n-1)
    hash[a[j]]--
    (if(-(a+b) present
        push to ans) in a(set).
    has(a[j])++.
  has(a[i])++;
```

-4, 1
2, 1
1, 1
0, 1
-1, 2

n^2.) 2 pointer → sort

a+b+c = 0
b+c = -a
fix a then do 2 sum for rest

sort

for i → 0 → size-2

if (i == 0 || (i > 0 && num[i] != num[i-1]))
lo = i+1, hi = n-1, sum = 0 - num[i]

__/__/__

```
while(lo<hi){
if(num[lo]+num[i]==sum){
  tem=i,lo,hi--
  ans=one+temp.
while(lo<hi && num[lo]==num[lo+1]) lo++
while(lo<hi && num[hi]==num[hi-1]) hi--;
 lo++
h--;
 }
else if(num[lo]+num[hi]<sum) lo++
else {h--;}}}
```

Q38 Longest Increasing subsequence

```
if(ind == n) return 0;
len = 0 + f(ind+1, prev_ind) // not take
if prev == -1 || arr[ind] > arr[prev_ind])
len = max(len, 1+f(ind+1, ind)) // take
return len
```

Q39 Minimum Insertions to make string palindrome →

(n − longest palindromic subsequence)

* keep palindromic subsequence intact → add rest of the elements in the mirror positions — eg

abcaa
aacba
ans = aaa

aaa = LPS

(a bc a cb a)

2 insertions

Q40 Longest arithmetic subsequence →

Q41 trapping rain water

Brute → for each add index

add(min of (left h, right h) − curr h)

(faster way → compute a prefix sum, suffix sum array which store running max from left, right

O(N) →

(41) Fast Exponentiation

```
int power(int a, int N){
if(N==0)
 return 1;
else if (N==1)
  return a
else{
int R = power(a, N/2);
if (N%2)==0
 return R*R
else
 return R*R*a;
}}
```

(42) Two roots of 1 / 2D vector into a 1D vector in O(N)

```
row * row_len + col
```

height of Binary

```
root = n  0
1 + max(left, right)
```

(42) Balanced binary tree

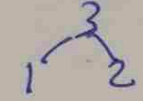
```
{{ if n=n
    return true }
  left = isbal(left)
  right = isbal(right)
  diff = (height(left) - height(right)) abs <= 1
  return (left & right & diff)
}
```

(43) diameter

```
if root = n return 0
lheight = h(root->left), di = (dia(r->l)
rheight = h(root->right), ri = (dia(r->r)
return max(di, ri, (lheig + rheight + 1))
```

44 check Identical tree →

```
if(r1 = null && r2 = null)
  return true
if(ri = null && r2 = null) return false
curr return (ri->val == r2->val)
left = (Ident((r1->l), (r2->l)))
right = (Ident((r1->r), (r2->r)))
return (curr & left & right);
```



true

__/__/____

45 Sum Tree

```
if(root = null)
  return true

left = sum(r→left)
right = sum(r→right)
curr = root→val == r→left + r→right
return (left && right && curr);
```

PF Hard 46 Edit Distance [Insert, Delete, Replace a character]
(convert word1 to word2)

```
if(i==a.length()){
return b.length()-j;
}
if(j==b.length()){
return a.length()-i;
}
else{
insert = 1+solve(a,b,i,j+1)
del = 1+solve(a,b,i+1,j)
rep = 1+solve(a,b,i+1,j+1)
}
return min(i,d,r);
```

if(char match) { call for remaining
else insert one, delete one, replace one
final = min(i,d,r)

// dp[i][j] replace (a.len)x[b.len]-ki

failed at 3

```
table(string a, b)
it=a.len, h=b.len - dp(i+1)(j+1)(p+1,q))
for(j=0; j<b.length; j++){
    dp[a.length][j] = b.len - j)
for(i→a.len;
  dp[i][b.len] = a.len - i
for(i=a.len-1→0) for(j=b.len-1→0)
if(a[i]=b[j]) dp[i][j] = dp[i+1][j+1];
else{ i = 1+dp[i][j+1]
d = 1+dp[i+1][j]
r = 1+dp[i+1][j+1]
dp[i][j] = min(i,d,r)
return dp[0][0];
```

47 Josephus problem (4,2) ⇒ ans=1
(2,1) ⇒ ans=2

```
f(n,k)
if(n==1)
{return 1;}
return (f(n-1,K)+k-1)%n+1; }
```

// Iterative →

```
int safe = 1
for(i=1 → i<=n)
  safe = (safe+k-1)%i+1;
  return safe position
```

48 ~~Largest Rectangular Area in Histogram~~

① Brute force →

```
for(i=0; i<n; i++)
  while(left)
  {
  while(right){
  }
  area = max(area, );
}
```

approach-2

dfs →

```
void dfs(int i, V visit,
         vi adj[]) {
  visit[i]=1
  for(auto it: adj[i]){
    if(!visit[it]){
      dfs(it, visit, adj);}}}
```

49 number of provinces →

```
for(i=1; i<=V; i++)
  { if(visit[i]==0) count++
    {dfs(i)} }
```

// count gives num of provinces

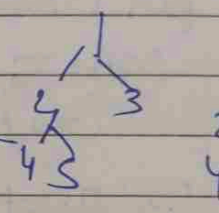
50 Convert adj matrix to adj list.

```
vector<int> adj[V];
for(i=0; i<V; i++){
for(j=0; j<V; j++){
  if(adj[i][j]==1 && i!=j){
    adj[i].push(j), adj[j].push(i);
}}
```

51 → group Anagrams →

```
vector<vector<string>> f(vector<string>&s){
int n = size();
unordered_map<string, vector<string>> mp
for(i:s)
{ for(it:s[i])
  string temp(26,'0').
  for(it:s[i])
   temp[it-97]++
  }
mp[temp].push_back(s[i])
}
for(auto it:mp){
  ans.push_back(it.second);
}
return ans;
```

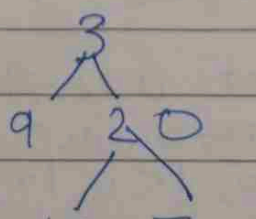
52 level order traversal line by line




```
if(root==NULL)return;
queue<node*> q;
q.push(root)
while(1){ int nodeCount = q.size();
if(nodeCount==0)break;
while(nodeCount>0){
   node *node = q.front();
   cout<<node->data<<" ";
   q.pop();
   if(node->left!=NULL)
    q.push(node->left);
   if(node->right!=NULL)
    q.push(node->right);
   nodeCount--;
}
cout<<endl
}
}
```

```
Insert into BST →
if root NULL
return new TreeNode(val)
}
if(root,val)
return root;
}
f(TreeNode*root, val){
if(r->data ≥ val)
{ if r->left=null
  r->left = new TreeNode(val)
  return
  if(root->left, val)
}else{
}
```

53 Zig-Zag traversal tree



(3 20 9 15 7)

```
vector<int> z(root)
vector<int> ans;
queue<TreeNode*> q;
q.push(root)
bool dir = true
while(!q.empty)
```

{
int size = q.size();
vector<int> temp(size);
for(int i=0; i<size; i++){
TreeNode *node = q.front()
if(node->left) q.push(node->left
if(node->right) q.push(node->r)
q.pop();
int index = (ltor)?i : (size-i-1);
temp[index] = node->val;
ans.push_back(temp);}}

(54) tree from Inorder + Preorder

Inorder left Ino right

[ ] 0 [ ]

[12]

Preorder | Pre(left) Pre(right)

```
In* buildtree(V1 preorder, V1 inorder){
map<int,int> inMap;
for(i+inorder.size()){
  inMap[inorder[i]]=i;
TreeNode* root = f(preorder, 0, preorder.size()-1, inorder, 0,
                   inorder.size()-1, inMap);
return root; }
```

```
TreeNode* f(V1 preorder, int prestart, preend, V1 inorder,
            in start, inend, inmap)
{ if(prestart > preend || instart > inend) return NULL
TreeNode* root = preorder[prestart]
int inroot = inmap[root->val]
int numsleft = inRoot - Instart;

root->left = f( preorder, prestart+1, prestart+numsleft,
                inorder, instart, inroot-1, inmap)
root->right = f( preorder, prestart+numsleft+1,
                 preend, inorder, inroot+1, inend, inmap)

return root }
```

(55) ~~[illegible]~~ (O(n)?, O(1) space)

→ (Catalan numbers) 1 1 2 5 14 42

$C_2 = C_0C_1 + C_1C_0$ | $C_3 = C_0C_2 + C_1C_1 + C_2C_0 = 5$

$C_4 = C_0C_3 + C_1C_2 + C_2C_1 + C_3C_0$ ie $C(n) = C_0C_{n-1} + C_1C_{n-2} \cdots C_{n-1}C_0$

Recursive Sol

```
int result = 0
for(i=0; i<n; i++)
{result += catalan(i) * catalan(n-1-i)} // dp solution
return result }
```

56 subarray sums divisible by K {4 5 0 -2 -3 1} K=5 → (7)

```
int count=0; unordered_map<int,int> mp; mp[0]=1; int sum=0
for(int i=0; i<nums.size(); i++){
sum += nums[i];
int rem = sum % k if(rem<0) rem += k; if(mp.find(rem) != mp.end()) { count += mp[rem]; }
mp[rem]++; } return count; }
```

(57) detect a loop in a graph (directed) using DFS

first do normal dfs and

have a additional visited array

which stores the elements visited in current path →

if both visited and path visited = 1 then return true

(58)

```
TreeNode* lca(TreeNode* node, TreeNode* u, TreeNode* v)
{ if(node==NULL) return NULL;
if(node->val == u->val || node->val == v->val)
  return node;
Node *l = lca(node->left, u, v);
Node *r = lca(node->right, u, v);
if(l!=N && r!=N) return node
if(l!=NULL) return l;
if(r!=NULL) return r;
return NULL; }
```

(1) detect a cycle in undirected graph using BFS

(59) merge Interval → [[1,3],[2,6],[8,10],[15,18]]
output → [[1,6],[8,10],[15,18]]

→ sort(intervals.begin(), intervals.end());
ans.pb(interval[0]); int curr = intervals[0][1];
j=0; for(i=1; i++) {
if intervals[i][0] <= curr) {
curr = ans[j][1] = max(ans[j][1], intervals[i][1];
} else { ans.pb(intervals[i]) curr = intervals[i][1];
j++; } } return ans; }
{O(nlogn) space(O(1)}

60 missing and Repeating numbers →
→ ~~int~~ sum, sum of squares
curr missing = (extra + diff)
diff = Σn − sum
diSumSq = Σn² − sumSq
→ Σn² − (extra+diff)² + extra² = sumSq
Σn² − sumSq = diff² + 2 * diff * extra
so extra = (diSumSq − diff² / 2 * diff)

61 majority voting algorithm →
int count = 0;
int element = nums[0];
for(int i=0; i<nums.size(); i++) {
if(count == 0) {
count++;
element = nums[i]; }
else {
if(element == nums[i]) {
count++; } else { count--; }
}
return element;

* int to string → to_string
string s1 = to_string(7)(n).to_string();

62 K max sum combinations →
sort(a.begin(), a.end()); sort(b.begin, b.end);
priority_queue<pair<int, pair<int, int>>> q;
set<pair<int, int>> st;
st.insert({n-1, n-1});
q.push({a[n-1]+b[n-1], {n-1, n-1}});
while(k--){
sum = q.top().first;
ans.pb(sum);
q.pop();
l = q.top().second.first
r = q.top().second.second
if(l>0 && r>0 && st.find({l-1, r}) == st.end()){
q.push({a[l-1]+b[r], {l-1, r}});
st.insert({l-1, r}); }
if // same for (l, r-1). 3
return ans;

1 2 3 4
5 6 7 8

(63) next = smaller element →
→ stack use karke last se start karo,
useless element POP karte jao +
new elements add karte jao

(64) → (check if a BST is BST or not) { bool isBST(int min, int max, root)
if (root == NULL) {
return true; }
if (root->data >= r || root->data <= l) {
return false; }
return isBST(root->data, r, root->right) &&
isBST(l, root->data, root->left); }

(65) Balanced Binary Tree O(n) →
int lh = height(root->left);
int rh = height(root->right); if(lh == -1 || rh == -1)
return -1;
if(abs(lh-rh) >= 2){ return -1;
return max(lh, rh) + 1; }

(66) Cycle in directed graph →
check visited and notvisited; if
both true, cycle is there

(67) search 2D matrix
row = mid/m;
col = mid % m;
}

(68) find Intersection point of LL →
count length → increase head of
bigger one by difference

(69) remove duplicate in sorted →
curr = 0;
for(i=1; i<n; i++){
if(arr[i] != arr[curr]){
curr++;
arr[curr] = arr[i]; } }
return curr+1;

(70) detect Cycle in linked list →
Node *p = head;
Node *q = head if(head->next = head) T
while(p!=NULL && p->next!=NULL && q!=NULL){
p = p->next->next;
q = q->next;
if(p==q && p!=NULL && q!=NULL)
T
}
F

(71) symmetric tree →
root1->data == root2->data && f(1->l, 2->r)
&& f(1->r, 2->l)

(72) Number of Islands → _/_/_

[DFS/BFS] → jao every 1 pe dfs chalao + counter badhao for every dfs call

(73) → Rabin Karp → Generate all permutation

```
void f( vi &ds, vi &nums, vvi &ans, int freq[]){
  if(ds.size == nums.size()){
   ans.push back ds  }
  for( i=0; i < n ; i++ ){
  if (!freq [i]){
     ds.push back(nums[i]);
     freq[i]=1;
     recurPermute (ds, nums, ans, freq))
     freq [i]=0;
     ds.pop_back();
```

(74) Longest Palindromic Substring

→ check maximum size Palindrome for each with i as centre 2 case → odd, even

```
for(i=0 i<n i++)
 int l=i, r=i;
 while(l>=0 && r<n && s[l]==s[r]){
     if(r-l+1 > maxi) maxi = r-l+1
   l--; r++;
l=i, r=i+1
while( l>=0 && r<n && s[l]==s[r]){
      if(r-l+1 > maxi ) maxi=r-l+1
   l--; r++
```

Bipartite Graph DFS → visited → 1 / 2

```
else if(color[it]==color[i]){
 return true; }
if(!vis[it]){
 if dfs(vis, adj, it, !vis[i]))
```

(76) longest common prefix

(1) way → find smallest element and use binary search to find the solution.

(77) BT (Lowest common ancestor)

// if both f(l), f(r) return
return root,
if only 1 returns, return that val.

{ if (root = NULL || P || Q)
return root }
if (f(root→l) == NULL
return f(root)→right
if (f(root→r) == NULL
return f(root→left)
else { return root }

(78)

diameter BT

→ (node* root, int dia)
if (root = null)
return 0;
lh = f(root→l, d)
rh = f(root→r, d)
dia = max(dia, lh+rh);
return 1+ max(lh, rh)
}

(79) Kahn's algo Topo sort

→ { int u = edges[i][0];
int v = edges[i][1];
indegree[v]++
adj[u].pb(u) }

while (!q.empty()) {
int k = q.front();
ans.push back(k)
for (auto it : adj[k]) {
ind[it]--;
if (ind[it] == 0) {
q.push(it); } } }
return ans;

Q (Maximum Product Subarray)

$O(n^3)$ BF, $O(n^2)$ better

$O(n)$ → int pre = 1, suf = 1;

for (i = 0; i < n; i++) { // re Handle zeroes
if (pre = 0) pre = 1; pre *= nums[i]
if (suf = 0) Suf = 1; suf *= nums[n-i-1]
if (pre > maxi) maxi = pre
if (suf > maxi) maxi = suf }
return maxi

if even zeroes → just product
→ we add zeroes, we will anyway either prefix or suffix

316 J
417 T
405 T

81 min sum path

min (f(grid, i+1, j, n, m, dp), f(grid, i, j+1, n, m));

82 → 4 sum | sort

for(i)

for(j = i+1)

2 pointer from { (j+1 → n-1) }

$O(n^3)$

(83) n meetings in a room →
create a vector of pair and
sort it based on end times
→ now increment pointer.

(84) starting point of a loop in linked list
→ find first meting meeting
point of slow and fast.
→ then restart slow at head.
→ now the point they meet again
will be the new head.

(85) Next Greater element
→ use stack push(-1)
iterate in reverse
push in stack if top > arr[i] // now push top to ans
Else pop while this happens

86 Stock span problem → find number of consecutive days in left where price[i] <= price(x)

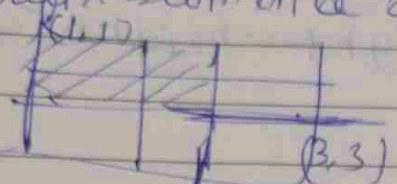
push(0)
for(i = 1; i < n; i++)
while(st.empt and
st.top() <= arr[i]) { st.pop(); }
if(st empty) push i+1;
else { i - st.top; }
st.push(i);
}

86 left view → level order traversal → first element of each level

87 nth root of M

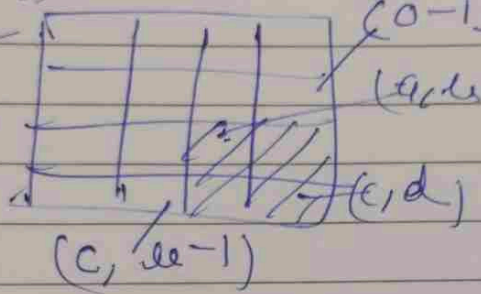
→ use binary search for 1 → M
predicate func being 1 * mid till n times

Prefix sum in a 2D array



prefixsum(3,3) =
pf(3,2) + pf(2,3) =
pf(2,2) + arr(3)(3)

A ∪ B = (A + B − A∩B)



sum =
pf(c,d) −
pf(c,b-1) −
pf(a-1,d)
+ pf(a-1,b-1)

88) merge sorted list →

```
f(l1, l2) {
if (l1 == N)
    return l2 3
if (l2 == N)
    return l1
if (l1->val <= l2->val)
  l1->next = f(l1->next, l2)
  return l1
3 else {
  l2->next = f(l1, l2, next)
  return l2
3 3
```

1 → 2 → 4
1 → 3 → 4
1 → 1 → 2 → 3 → 4 → 4

89 Trapping Rainwater

```
int n = heights.size
left = 0;
right = n-1;
res = 0;
maxleft = 0;
maxright = 0;
while(left <= right){
if(height[left] <= height[right]){
  if(height[left] >= maxleft) maxleft =
    height[left];
else res += maxleft - height[left];
  left++; 3 else { if (height[right] >=
maxright = height[right]; else {
```

__/__/____

```
if (height[right] >= maxright)
maxright = height[right];
else if (height[right] >= maxright)
maxright = height[right];
else res += maxright - height[right];
right++;
}
return res;
}
```