1.Kth Smallest Element

Brute:

➔ Sort the array and return the (k-1)th element

Better:

➔ Create a Min heap and push all element and return the kth element from top

Best:

➔ Use Quick sort algorithm, to find to correct position .

➔ If pos > k recur for left subarray

➔ Else recur for right subarray (updated k=k-pos+l-1)

```
int quickSelect(vector<int> &arr,int l,int r,int k){

    if(k>0 and k<=r-l+1){
        int pos = partition(arr,l,r);

        if(pos==k-1) return arr[pos];

        if(pos-1 > k-1) // left subarray
            return quickSelect(arr,l,pos-1,k);
        else return quickSelect(arr,pos+1,r, k-pos+l-1);
    }
    return INT_MAX;
}
int printKthSmallestElement(vector<int> &arr,int n,int k){
    int i=0,j=n-1;
    return quickSelect(arr,i,j,k);
}
```

2.Find Transition Point

Brute:

➔ Linearly traverse and when a[i]==1 return i

Optimal:

➔ Use of binary search, if a[mid]==1

- Then reduce j=mid-1 and again iterate and store ans=mid

➔ Else i=mid+1

```
int findTransition(vector<int> arr,int n){

    // binary search
    int i=0,j=n-1;
    int ans=-1;
    while(i<=j){
```

```
        int mid = (i+j)/2;
        if(arr[mid]==1){
            j=mid-1;
            ans=mid;
        }
        else if(arr[mid]==0){
            i=mid+1;
        }
    }
    return ans;
}
```

3.Floor Square Root

Brute:

➔ Using the inbuilt function. (math.h)

Optimal:

➔ Use of binary search , Range -> 1 to n/2
➔ Calculate mid, id mid*mid==n return mid
➔ If mid*mid <=n , increment left part to mid+1
➔ Else decrement j to mid-1

```
int floorSqrt(int n){

    if(n==0 or n==1) return n;
    int i=1,j=n/2;
    int res=0;
    while(i<=j){
        int mid = (i+j)/2;
        if(mid*mid==n) return mid;
        res=mid;
        if(mid*mid <= n){
            i=mid+1;
        }
        else j=mid-1;
    }
    return res;
}
```

4.Element Appearing only Once (Sorted or Not)

Brute:

➔ If sorted, linearly traverse and keep a count

Better:

➔ Store the frequency of elements into map .

Best:

➔ Do a XOR of all values, return xor_values.
➔ b/c a ^ a ^ b = b

```
int elementAppearOnce(vector<int> arr,int n)

{
    int count=arr[0];
    for(int i=1;i<n;i++){
        count^=arr[i];
    }
    return count;
}
```

5. Index of extra element

Brute:

➔ Linearly traverse to array and find out the extra element's index.

Optimal:

➔ Apply binary search, from 1-> n-1
➔ Id a1[mid]==a2[mid] increment I -> mid+1
➔ Else store ans, and decrement j -> mid-1

```
int findExtra(int arr1[],int arr2[],int n)

{
    int i=0,j=n-1,ans=n;
    while(i<=j){
        int mid = (i+j)/2;
        if(arr1[mid]==arr2[mid]) i=mid+1;
        else{
            ans=mid;
            j=mid-1;
        }
    }
    return ans;
}
```

6.Merge Sort for LinkedList

Optimal:

➔ Find the middle (using slow and fast pointers)

➔ mergeSort(first) // first = head

➔ mergeSort(second) // second = slow->next // next to middle

➔ merge(first,second)

```
void findMiddle(ListNode* src,ListNode** first,ListNode** second){

      ListNode* slow = src;
      ListNode* fast = src->next;

      while (fast != NULL) {
          fast = fast->next;
          if (fast != NULL) {
              slow = slow->next;
              fast = fast->next;
          }
      }
      *first = src;
      *second = slow->next;
      slow->next = NULL;
  }
  ListNode* merge(ListNode* first,ListNode* second){
      ListNode* result = NULL;

      if(first==nullptr) return second;
      if(second==nullptr) return first;

      if (first->val <= second->val) {
          result = first;
          result->next = merge(first->next, second);
      }
      else {
          result = second;
          result->next = merge(first, second->next);
      }
      return result;
  }
  ListNode* sortList1(ListNode** head) {
      ListNode* curr = *head;
      ListNode* first,*second;

      if(!curr or !curr->next) return *head;
      findMiddle(curr,&first,&second);
```

```
        sortList1(&first);
        sortList1(&second);

        *head = merge(first,second);
        return *head;}
```

7.Union of Linked List

In this method, algorithms for Union and Intersection are very similar. First, we sort the given lists, then we traverse the sorted lists to get union and intersection.
The following are the steps to be followed to get union and intersection lists.

1. *Sort the first Linked List using merge sort. This step takes O(mLogm) time. Refer this post for details of this step.*
2. *Sort the second Linked List using merge sort. This step takes O(nLogn) time. Refer this post for details of this step.*
3. *Linearly scan both sorted lists to get the union and intersection. This step takes O(m + n) time. This step can be implemented using the same algorithm as sorted arrays algorithm discussed here.*

The time complexity of this method is O(mLogm + nLogn) which is better than method 1's time complexity.

8.Sort a Stack

1. Pop all elements recursively and top element should combine
2. Fun combine():
    1. If empty, push element
    2. If element > top then, push element
    3. Else pop, combine(element) and push(popped element)

```
void combine(int v,stack<int> &s){

    if(s.empty()){
        s.push(v);
        return;
    }
    if(!s.empty() and s.top()<v){
        s.push(v);
    }
    else{
        int temp = s.top();
        s.pop();
        combine(v,s);
        s.push(temp);
    }
```

```
}
void solve(stack<int> &s){
    if(s.empty()) return;
    else{
        int temp = s.top();
        s.pop();
        solve(s);
        combine(temp,s);
    }
```

9. N Meetings in a Room

1. Sort according to end.
2. If anytime startof_next > end, end = endof_next and max++;

```
int maxMeetings(int start[], int end[], int n)

    {
        vector<vector<int>> ans;
        for(int i = 0; i < n; i++){
            ans.push_back({end[i],start[i]});
        }
        sort(ans.begin(),ans.end());
        int max_meeting = 1;
        int endm = ans[0][0];
        for(int i = 1; i < n; i++){
            if(ans[i][1] > endm){
                endm = ans[i][0];
                max_meeting++;
            }
        }
        return max_meeting;
     }
```

10. Max Chain Length

1. Sort according to end
2. If first_next > init_end
    a. Init_end = second_next, count++

```
vector<pair<int,int>> ans;

        for(int i=0;i<pairs.size();i++){
            ans.push_back({pairs[i][1],pairs[i][0]});
        }
        sort(ans.begin(),ans.end());
```

```
        int first = ans[0].first;
        int count=1;
        for(int i=1;i<pairs.size();i++){
            if(first<ans[i].second){
                count++;
                first = ans[i].first;
            }
        }
        return count;
```

11. Minimum Platforms

- ➔ Sort both arrival and departure
- ➔ For i->n,j=0
  - If dp[j] > ar[i] count++;
  - Else j++

```
int findPlatform(int arr[], int dep[], int n)

   {
        sort(arr,arr+n);
        sort(dep,dep+n);
        int i=1,j=0;
        int count=1;
        while(i<n ){
            if(arr[i]<=dep[j]){
                count++;

            }
            else{
                j++;
        }
        i++;
        }
        return count;
   }
```

12. Count of strings that can be formed using a, b and c under given constraints

- ➔ A can be N times, B can be 1 time and c can be 2 times;
- ➔ F(n-1,b,c) + f(n-1,b-1,c) + f(n-1,b,c-1)
- ➔ => n*(n-1)*(n-2)/2

```
int countStr(int n, int bCount, int cCount)
{
    // Base cases
    if (bCount < 0 || cCount < 0) return 0;
    if (n == 0) return 1;
    if (bCount == 0 && cCount == 0) return 1;

    // Three cases, we choose, a or b or c
    // In all three cases n decreases by 1.
    int res = countStr(n-1, bCount, cCount);
    res += countStr(n-1, bCount-1, cCount);
    res += countStr(n-1, bCount, cCount-1);

    return res;
    //or
    return (n*n*n+3*n*n+2*n)/2;
}
```

13. Number of Hops

➔ F(n-3) + f(n-2) + f(n-1) // 1 , 2, 3 steps

```
int findStep(int n){

        if (n == 0)
            return 1;
        else if (n < 0)
            return 0;

        else
            return findStep(n - 3) + findStep(n - 2)
                   + findStep(n - 1);
    }
```

14. Points to Win Game

➔ Pick and non pick approach

➔ Push 3,5,10 to vector and then apply subset sum with unlimited supply

```
long long int solve(int i,long long int n,vector<int> ans){
```

```
    if(n==0){
        return 1;
    }
    int pick=0;
    if(n>=ans[i])  pick = solve(i,n-ans[i],ans);
    int npick = solve(i+1,n,ans);

    return pick+npick;
}
```

15. Diagonal Sum in Binary Tree

➔ Go to right of tree and if left is present, push in queue

➔ Add all rights to sum, at any moment, it reaches null then, push sum to vector. And iterate for next.

```
vector <int> diagonalSum(Node* root) {

    while(!q.empty()){
        for(int i=0;i<n;i++){
            Node* temp  = q.front();
            q.pop();
            while(temp){
                if(temp->left) q.push(temp->left);
                sum+=temp->data;temp=temp->right;
            }
        } ans.push_back(sum);
```

16. Different Occurrences

➔ Kind off lcs with if matches, we still have option to pick or not.

```
const int mod = 1e9+7;

    int solve(string s,string t,int i,int j,vector<vector<int>> &dp){
        if(j<0) return 1;
        if(i<0) return 0;
        if(dp[i][j]!=-1) return dp[i][j];
        int pick=0,npick=0;
        if(s[i]==t[j]){
            pick+= solve(s,t,i-1,j-1,dp);
            npick+=solve(s,t,i-1,j,dp);
            return dp[i][j] = (pick+npick)%mod;
        }
        return dp[i][j] = solve(s,t,i-1,j,dp)%mod;
```

```
    }
    int subsequenceCount(string s, string t)
    {
        int m = s.size();
        int n = t.size();
        vector<vector<int>> dp(m+1,vector<int>(n+1,0));
        for(int i=0;i<m+1;i++) {
            for(int j=0;j<n+1;j++){
                int pick=0,npick=0;
                if(j==0) dp[i][j]=1;
                else if(i==0) dp[i][j]=0;
                else if(s[i-1]==t[j-1]){
                    pick+= dp[i-1][j-1];
                    npick+=dp[i-1][j];
                    dp[i][j] = (pick+npick)%mod;
                }
                else dp[i][j]=dp[i-1][j];
            }
        }
        return dp[m][n]%mod;
    }
```

17. Minimum Spanning Tree

- ➔ Sort according to weights, then iterate for nodes.
- ➔ If both nodes do not have same parents, add weights to it and them combine .

```
vector<int> parent;

    int find(int v){
        if(parent[v]==-1) return v;
        return parent[v]=find(parent[v]);
    }
    void combine(int u,int v){
        v=find(v);
        u=find(u);
        parent[u]=v;
    }
    int spanningTree(int V, vector<vector<int>> adj[])
    {
        parent.assign(V+1,-1);
        vector<array<int,3>> edges;
        for(int i=0;i<V;i++){
            for(auto it : adj[i]){
```

```
                edges.push_back({it[1],i,it[0]});
            }
        }
        sort(edges.begin(),edges.end());
        int mst=0;
        for(auto it : edges){
            if(find(it[1])!=find(it[2])){ // both of them are not of same parents
                mst+=it[0]; // add weight
                combine(it[1],it[2]); // combine them
            }
        }
        return mst;
    }
```

18. Josephus Problem

```
int solve(int i,vector<int> &s,int n,int k){

        if(s.size()==1) return s.back();
        i = (i + k-1)%n;
        vector<int>::iterator it;
        it = s.begin()+i;
        s.erase(it);
        return solve(i,s,n-1,k);
    }
    int findTheWinner(int n, int k) {
        // vector<int> num(n);
        // for(int i=0;i<n;i++){
        //     num[i] = i+1;
        // }
        // return solve(0,num,n,k);
        int ans=0;
        for(int i=1;i<=n;i++){
            ans = (ans+k)%i;
        }return ans+1;
```

19. Max Profit

We will have a flag (buy / not buy) and K trans.
If buy is set, we have two choices:

1. Either buy or not buy

If sell is set, we have two choices,

1. Sell (if sell, then decrement K by 1) or not sell

```
if(trans==0) return 0;

    if(i==n) return 0;
    int pro=0;
    if(dp[i][buy][trans]!=-1) return dp[i][buy][trans];
    if(buy){
        return dp[i][buy][trans] =  max(-
prices[i]+f(i+1,0,trans,n,prices,dp),f(i+1,1,trans,n,prices,dp));
    }
    else{
        return dp[i][buy][trans] max(prices[i]+f(i+1,1,trans-
1,n,prices,dp),f(i+1,0,trans,n,prices,dp));
    }
```

20. Unique BST's

Catalan number, summation(Ci * Cn-i-1)

```
vector<long long> dp(N+1);

        dp[0]=dp[1]=1;
        for(int i=2;i<=N;i++){
            for(int j=1;j<=i;j++){
                dp[i] = (dp[i]%mod + (dp[j-1]%mod*dp[i-j]%mod)%mod)%mod;
            }
        }
        return dp[N]%mod;
```

21. Count subsequences of type a^i b^j c^k

We traverse given string. For every character encounter, we do the following:

1. *Initialize counts of different subsequences caused by different combination of 'a'. Let this count be aCount.*
2. *Initialize counts of different subsequences caused by different combination of 'b'. Let this count be bCount.*
3. *Initialize counts of different subsequences caused by different combination of 'c'. Let this count be cCount.*
4. *Traverse all characters of given string. Do following for current character* ***s[i]***
    - ***If current character is 'a',*** *then there are following possibilities :O(1).*
        - *Current character begins a new subsequence.*

- *Current character is part of aCount subsequences.*
- *Current character is not part of aCount subsequences.*

- *Therefore we do aCount = (1 + 2 * aCount);*
- ***If current character is 'b'**, then there are following possibilities :*
  - *Current character begins a new subsequence of b's with aCount subsequences.*
  - *Current character is part of bCount subsequences.*
  - *Current character is not part of bCount subsequences.*
- *Therefore we do bCount = (aCount + 2 * bCount);*
- ***If current character is 'c'**, then there are following possibilities :*
  - *Current character begins a new subsequence of c's with bCount subsequences.*
  - *Current character is part of cCount subsequences.*
  - *Current character is not part of cCount subsequences.*
- *Therefore we do cCount = (bCount + 2 * cCount);*

5. *Finally we return cCount;*

```
int countSubsequences(string s){

    int a=0,b=0,c=0;
    for(int i=0;i<s.size();i++){
        if(s[i]=='a'){
            a = 1+2*a;
        }
        else if(s[i]=='b'){
            b = a+2*b;
        }
        else {
            c = b+2*c;
        } return c;
```