ALGORITHM 1

Binary Search

C++ • Autocomplete

```cpp
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int n = nums.size()-1;
        int low = 0, high = n;
        while( low <= high){
            int mid = low + (high-low)/2;
            if (nums[mid] == target) return mid;
            else if (nums[mid] > target) high = mid -1;
            else low = mid + 1;
        }
        return -1;
    }
};
```

Search insert position

C++ Autocomplete

```cpp
class Solution {
public:
    int binary(vector<int> nums, int start, int end, int target)
    {if(start>end)
        return -1;
        int mid=(start+end)/2;
        if(nums[mid]==target)
            return mid;
        else if(mid+1<=end &&nums[mid]<target && target<nums[mid+1])
            return mid+1;
        else if(mid-1>=0 && nums[mid-1]<target && target<nums[mid])
            return mid;
        else if(nums[mid]>target)
            return binary(nums, start, mid-1, target);
        else
            return binary(nums, mid+1, end, target);
    }
    int searchInsert(vector<int>& nums, int target) {
        if(target<nums[0])
            return 0;
        int start=0;
        int end=nums.size()-1;
      int ans=binary(nums, start, end, target);
        if(ans==-1)
            return nums.size();
        else
            return ans;
    }
};
```

Squares of a sorted array

C++ • Autocomplete

```cpp
class Solution {
public:
    vector<int> sortedSquares(vector<int>& A) {
        vector<int> res(A.size());
        int l = 0, r = A.size() - 1;
        for (int k = A.size() - 1; k >= 0; k--) {
            if (abs(A[r]) > abs(A[l])) res[k] = A[r] * A[r--];
            else res[k] = A[l] * A[l++];
        }
        return res;
    }
};
```

Move zeroes

C++ • Autocomplete

```cpp
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
          int j = 0;
        for (int i = 0; i < nums.size(); i++) {
            if (nums[i] != 0) {
                nums[j++] = nums[i];
            }
        }
        for (;j < nums.size(); j++) {
            nums[j] = 0;
        }
    }
};
```

Reverse string

C++ Autocomplete

```cpp
class Solution {
public:
    void reverseString(vector<char>& s) {
       if(s.size()==0)
           return;
        char temp=s[0];
        s.erase(s.begin());
        reverseString(s);
        s.push_back(temp);

    }
};
```

Middle of a linked list

C++ Autocomplete

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* middleNode(ListNode* head) {

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

Move nth node from end of list

C++ Autocomplete

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode *first=head,*second=head, *prev=NULL;
        int i=1;
        while(i!=n)
        {first=first->next;
            i++;}
        first=first->next;
        while(first!=NULL)
        {first=first->next;
            prev=second;
            second=second->next;}
        if(prev==NULL)return second->next;
        prev->next=second->next;
        return head;}
};
```

Longest substring without repeat character

C++ • Autocomplete

```cpp
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
         unordered_map<char,int> index;
        int start=0,res=0;
        for(int i=0;i<s.length();i++){

            if (index.find(s[i]) != index.end() && index[s[i]] >= start)
                 start = index[s[i]] + 1;

            index[s[i]] = i;
            res=max(res,i-start+1);
        }

        return res;
    }
};
```

Merge two binary trees

C++ • Autocomplete

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    TreeNode* mergeTrees(TreeNode* root1, TreeNode* root2) {
        if(root1==NULL)
            return root2;
        if(root2==NULL)
            return root1;

      root1->val=root1->val+root2->val;
        root1->left=mergeTrees(root1->left,root2->left);
        root1->right=mergeTrees(root1->right,root2->right);
        return root1;


```

Populating next right pointer in each node

```
17  */
18
19  class Solution {
20  public:
21      Node* connect(Node* root) {
22          if(root==NULL || root->left==NULL)
23              return root;
24          queue<Node*> q;
25          q.push(root);
26          root->left->next=root->right;
27          while(!q.empty())
28          {
29              int size=q.size();
30              for(int i=0;i<size;i++)
31              {
32              Node* x=q.front();
33                  q.pop();
34                  if(x->next==NULL && i<size-1)
35                  {
36                      x->next=q.front();
37                  }
38                  if(x->left)
39                  {
40                      x->left->next=x->right;
41                      q.push(x->left);
42                      q.push(x->right);
43                  }
44
45              }
46          }
47          return root;
48
49      }
50  };
```

0 1 matrix

C++ • Autocomplete

```cpp
class Solution {
public:
    vector<vector<int>> updateMatrix(vector<vector<int>>& mat) {

        queue<pair<int, int>> q;
        int m=mat.size();
        int n=mat[0].size();
         vector<vector<int>> visited(m,vector<int> (n,0));
        for(int i=0;i<m;i++)
        {for(int j=0;j<n;j++)
            { if(mat[i][j]==0)
                {q.push({i,j});
                 visited[i][j]=1;}}}
vector<int> dx={-1,0,1,0};
        vector<int> dy={0,-1,0,1};
        while(!q.empty())
        {auto t=q.front();
            int curr=mat[t.first][t.second];
            q.pop();
            for(int i=0;i<dx.size();i++)
            {int u=t.first+dx[i];
                int v=t.second+dy[i];
                if(u>=0 && v>=0 && u<m && v<n &&  visited[u][v]==0)
                {  mat[u][v]=curr+1;
                 q.push({u,v});
                  visited[u][v]=1;}}

        }
        return mat;

    }
};
```

Rotting oranges

C++ Autocomplete

```cpp
class Solution {
public:
    int orangesRotting(vector<vector<int>>& grid) {
        if(grid.size()==0)return 0;
        int row=grid.size();
        int col=grid[0].size();
        int total=0;
        queue<pair<int,int>> q;
        vector<vector<int>> visited(row, vector<int> (col,0));
        for(int i=0;i<row;i++)
        {for(int j=0;j<col;j++)
            {if(grid[i][j]==1 || grid[i][j]==2)total++;
                if(grid[i][j]==2)q.push({i,j}); }}
        int dx[4]={0,0,1,-1};
        int dy[4]={1,-1,0,0};
        int count=0;
        int ans=0;
        int size=0;
        int u,v;
        int t=0;
        while(!q.empty())
              {size=q.size();
               t+=size;
               while(size--)
               {auto node=q.front();
               int f=node.first;
               int s=node.second;
               q.pop();
               for(int i=0;i<4;i++)
               { u=f+dx[i];
                   v=s+dy[i];
                   if(u<0 || v<0 || u>=row || v>=col)
                   continue;
                 if(grid[u][v]==1)
```

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

C++ • Autocomplete

```
        int count=0;
        int ans=0;
        int size=0;
        int u,v;
        int t=0;
        while(!q.empty())
              {size=q.size();
               t+=size;
               while(size--)
               {auto node=q.front();
               int f=node.first;
               int s=node.second;
               q.pop();
               for(int i=0;i<4;i++)
               { u=f+dx[i];
                   v=s+dy[i];
                   if(u<0 || v<0 || u>=row || v>=col)
                   continue;
                 if(grid[u][v]==1)
                 {grid[u][v]=2;
                  q.push({u,v});
                 }
               }

               }
               if(!q.empty())ans++;
              }

        if(t==total)
            return ans;
        else
            return -1;
    }
};
```

Merge two sorted list

```
 *     ListNode(int x, ListNode *next) : val(x), next(next
 * };
 */
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        ListNode *temp,*temp2;
        if(l1==NULL)
            return l2;

        if(l2==NULL)
            return l1;
        if(l1->val>l2->val)
        {temp=l2;
        // temp->next=NULL;
         l2=l2->next;
         temp2=mergeTwoLists(l1,l2);

        }
        else
        {temp=l1;
        // temp->next=NULL;
         l1=l1->next;
         temp2=mergeTwoLists(l1,l2);

        }
        temp->next=temp2;
        return temp;


    }
};
```

Reverse linked list

C++ • Autocomplete

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
      ListNode *temp,*newhead,*revlist;
        if(!head || !head->next)
            return head;
        newhead=head->next;
        head->next=NULL;
        revlist=reverseList(newhead);
        newhead->next=head;
        return revlist;



    }
};
```

Combinations

C++ • Autocomplete

```cpp
class Solution {
public:
    vector<vector<int>> ans;
    void def(int idx, vector<int> box, int k, int li, vector<int> temp)
    {     if(idx==k)
    {
        ans.push_back(temp);
        return;
    }

        for(int i=li+1;i<box.size();i++)
        {   temp.push_back(box[i]);
            def(idx+1, box,k,i, temp );
          temp.pop_back();

        }
    }
    vector<vector<int>> combine(int n, int k) {
        vector<int> box(n);
        for(int i=0;i<box.size();i++)
            box[i]=i+1;
        vector<int> temp;

        def(0, box,k,-1, temp);
        return ans;
    }
};
```

Permutations

C++ • Autocomplete

```cpp
class Solution {
public:
    void rev(int c, vector<int> nums, vector<int> box,vector<vector<int>>& ans)
    {
        if(c==nums.size())
        {
            ans.push_back(box);
            return;
        }
        for(int i=0;i<nums.size();i++)
        { if(box[i]==-11)
        {
            box[i]=nums[c];
            rev(c+1, nums, box, ans);
            box[i]=-11;
        }

        }
    }
    vector<vector<int>> permute(vector<int>& nums) {
        int n=nums.size();
        vector<int> box(n,-11);
        vector<vector<int>> ans;
        rev(0, nums, box, ans);
        return ans;
    }
};
```

Letter case permutation

C++ • Autocomplete

```cpp
class Solution {
public:
    void rev(int i,string& s,vector<string>& ans, string temp )
    {
        if(i==s.size())
        {  ans.push_back(temp);
         return;
        }

        if(isdigit(s[i]))
        {   temp=temp+s[i];
            rev(i+1, s, ans, temp);
        }
        else
        {
            string u=temp;
          u.push_back(tolower(s[i]));
            rev(i+1, s, ans, u);
            temp.push_back(toupper(s[i]));
            rev(i+1, s, ans, temp);

        }


    }
    vector<string> letterCasePermutation(string s) {
        vector<string> ans;
        rev(0, s,ans, "" );
        return ans;
    }
};
```

Climbing stairs

C++ Autocomplete

```cpp
class Solution {
public:
    int f(int n , vector<int>& t)
    {
         t[0]=0;
        t[1]=1;
            if(n==0)
                return 1;
        if(n==1)
            return 1;
        if(t[n]!=-1)
            return t[n];
        int left=f(n-1,t);
        int right=f(n-2,t);
        return t[n]=left+right;

    }
    int climbStairs(int n) {
        vector<int> t(46,-1);

        return f(n,t);
    }
};
```

House robber

C++ • Autocomplete

```cpp
class Solution {
public:
    int rob(vector<int>& nums) {
        int include=0;
        int exclude=0;
        for(int i=0;i<nums.size();i++)
        {int u=include;
         int v=exclude;
         int y=nums[i]+exclude;
         include=y;
         exclude=max(u,v);
    }
        return max(include, exclude);
    }
};
```

Power of 2

C++ • Autocomplete

```cpp
class Solution {
public:
    bool isPowerOfTwo(int n) {
        if(n==0) return false;
        while(n%2==0) n/=2;
        return n==1;
    }
};
```

Number of one bit

C++ Autocomplete

```cpp
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int count=0;
        while(n)
        {
            count++;
            n=n&(n-1);
        }
        return count;

    }
};
```

Reverse bits

C++ Autocomplete

```cpp
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t num=0;
        int i=0;
        while(i<32)
        {int x=n&1;
         num=num*2+x;
         n=n>>1;
            i++;
        }
        return num;

    }
};
```

Single number

C++ Autocomplete

```cpp
class Solution {
public:
    int singleNumber(vector<int>& nums) {

        int ans=nums[0];
        for(int i=1;i<nums.size();i++)
        {
            ans=ans^nums[i];
        }
        return ans;
    }
};
```