Best Codes (DECEMBER)

1. **LEETCODE 412**

```
vector<string> fizzBuzz(int n) {
    vector<string> res;
    if (n < 1)
      return res;
    for (int i = 1; i <= n; i++) {
      string tmp = "";
      if (i % 3 == 0)       tmp += "Fizz";
      if (i % 5 == 0)       tmp += "Buzz";
      if (tmp.length() == 0)  tmp = to_string(i);
      res.push_back(tmp);
    }
    return res;
  }
```

2. **HEAP(KTH SMALLEST ELEMENT)**

```
 int kthSmallest(int arr[], int l, int r, int k) {
  priority_queue<int>maxh;
  for(int i=l;i<r;i++){
    maxh.push(arr[i]);
    if(maxh.size()>k){
      maxh.pop();
    }
  }
  return maxh.top();
}
```

3. **Move zeros one side**

```
Input: [0,1,0,3,12]
Output: [1,3,12,0,0]
```

```
class Solution {
public:
  void moveZeroes(vector<int>& nums) {
    int n=nums.size();
   if(n==0|| n==1){
     return;
   }
    int left=0,right=0;
```

```
    int temp;
    while(right<n){
      if(nums[right]==0){
        right++;
      }
      else{
        temp=nums[left];
        nums[left]=nums[right];
        nums[right]=temp;

        left++;
        right++;
      }
    }
  }
};
```

4/* Function to reverse arr[] from start to end*/

```
void rvereseArray(int arr[], int start, int end)
{
  while (start < end)
  {
    int temp = arr[start];
    arr[start] = arr[end];
    arr[end] = temp;
    start++;
    end--;
  }
}
```

5. **Palindrome string**

```
int isPlaindrome(string S)
    {
      int n=S.size();
      int m=n-1;
      int i=0;
      while(i<m){
        if(S[i]!=S[m]){
          return 0;
          break;
        }
        i++;
```

```
            m--;
        }
        return 1;
    }
```

6.**Duplicate characters in string**

#Hashing

```
#include <iostream>
using namespace std;
const int CHARS =26;
int main()
{
 int count1[CHARS]={0};
 string str;
 getline(cin,str);

 for(int i=0;str[i]!='\0';i++){
    count1[str[i]-'a']++;
 }
for(int i=0;i<26;i++){
   if(count1[i]>1){
      cout<<count1[i]<<endl;
   }
   else continue;
}
   return 0;
}
```

7. **Reverse a linked list**

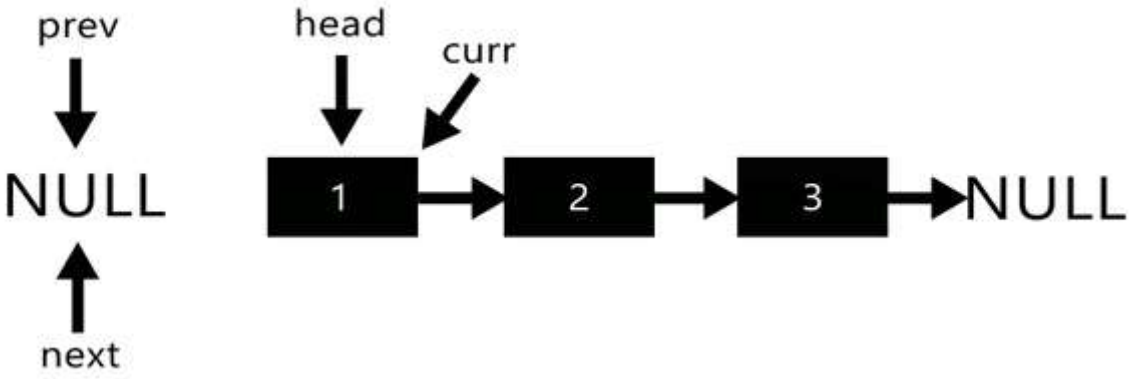
```
struct Node* reverseList(struct Node *head)
{
  Node* current=head;
  Node *prev=NULL,*next=NULL;
  while(current!=NULL){
     next=current->next;
     current->next=prev;
     prev=current;
     current=next;
  }
 head=prev;
}
```

9/12

8 . **Detect Loop in the linked list**

#hashing #linked list

https://www.geeksforgeeks.org/reverse-a-linked-list/




```
while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
```

```
bool detectLoop(Node* head)
{
  unordered_set<Node*>m;
  while(head!=NULL){
    if(m.find(head)!=m.end()){
      return true;
    }
    m.insert(head);
    head=head->next;
  }
```

```
  return false;
}
```

9 Add 1 to a number represented as linked list

#Linked list

```
/* Function to create a new node with given data */
Node *newNode(int data)
{
  Node *new_node = new Node;
  new_node->data = data;
  new_node->next = NULL;
  return new_node;
}

/* Function to reverse the linked list */
Node *reverse(Node *head)
{
  Node * prev = NULL;
  Node * current = head;
  Node * next;
  while (current != NULL)
  {
    next = current->next;
    current->next = prev;
    prev = current;
    current = next;
  }
  return prev;
}

/* Adds one to a linked lists and return the head
node of resultant list */
Node *addOneUtil(Node *head)
{
  // res is head node of the resultant list
  Node* res = head;
  Node *temp, *prev = NULL;
```

```
    int carry = 1, sum;

    while (head != NULL) //while both lists exist
    {
        // Calculate value of next digit in resultant list.
        // The next digit is sum of following things
        // (i) Carry
        // (ii) Next digit of head list (if there is a
        // next digit)
        sum = carry + head->data;

        // update carry for next calulation
        carry = (sum >= 10)? 1 : 0;

        // update sum if it is greater than 10
        sum = sum % 10;

        // Create a new node with sum as data
        head->data = sum;

        // Move head and second pointers to next nodes
        temp = head;
        head = head->next;
    }

    // if some carry is still there, add a new node to
    // result list.
    if (carry > 0)
        temp->next = newNode(carry);

    // return head of the resultant list
    return res;
}

// This function mainly uses addOneUtil().
Node* addOne(Node *head)
{
    // Reverse linked list
    head = reverse(head);
```

```
    // Add one from left to right of reversed
    // list
    head = addOneUtil(head);

    // Reverse the modified list
    return reverse(head);
}
```

10 .Remove Duplicates from Sorted List

#linked list #leet code linked list writing ways #gfg

LEETCODE:

```
class Solution {
public:
ListNode* deleteDuplicates(ListNode* head) {
ListNode* current=head;
ListNode* nxt;
if(current==nullptr){
    return head;
}
while(current->next!=NULL){
    if(current->val==current->next->val){
        nxt=current->next->next;
        delete current->next;
        current->next=nxt;
}
else{
    current=current->next;
}
}
return head;
    }
};
```

GFG

```
Node *removeDuplicates(Node *root)
{
Node *current=root;
Node *nxt;
if(current->next==NULL){
    return current;
}
while(current->next!=NULL){
    if(current->data==current->next->data){
```

```
      nxt=current->next->next;
      free(current->next);
      current->next=nxt;
}
else{
   current=current->next;
}
}
return root;
}
```

11. Indorder/Post order/pre order tree traversal

```
void printPostorder(struct Node* node)
{
   if (node == NULL)
      return;

   // first recur on left subtree
   printPostorder(node->left);

   // then recur on right subtree
   printPostorder(node->right);

   // now deal with the node
   cout << node->data << " ";
}

/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct Node* node)
{
   if (node == NULL)
      return;

   /* first recur on left child */
   printInorder(node->left);

   /* then print the data of node */
   cout << node->data << " ";

   /* now recur on right child */
   printInorder(node->right);
```

```
}

/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct Node* node)
{
  if (node == NULL)
    return;

  /* first print data of node */
  cout << node->data << " ";

  /* then recur on left sutree */
  printPreorder(node->left);

  /* now recur on right subtree */
  printPreorder(node->right);
}
```

10/12

11 Middle of three

#heap #easy_solution

```
class Solution{
 public:
  int middle(int A, int B, int C){
    priority_queue<int>maxh;
    maxh.push(A);
    maxh.push(B);
    maxh.push(C);

    if(maxh.size()>2){
      maxh.pop();
    }
    return maxh.top();
  }
};
```

12. A Program to check if strings are rotations of each other or not

```
1. Create a temp string and store concatenation of str1 to
   str1 in temp.
                    temp = str1.str1
2. If str2 is a substring of temp then str1 and str2 are
```

```
        rotations of each other.

     Example:
                      str1 = "ABACD"
                      str2 = "CDABA"

      temp = str1.str1 = "ABACDABACD"
      Since str2 is a substring of temp, str1 and str2 are
      rotations of each other.
```

bool areRotations(string str1, string str2)
{
/* Check if sizes of two strings are same */
if (str1.length() != str2.length())
return false;

string temp = str1 + str1;
if(temp.find(str2)){
return true;
}
else
return false;
}
11/12
13.Move last element to front of a given Linked List
#L.L # Double pointer

In the first function header, *head is a pointer to a node object which is allocated somewhere in memory:

```
 void middle(struct Node *head);

              _____________________
             |                     |
    *head --> |     Node object     |
             | [val=1][*next=NULL] |
             |_____________________|
```

while in the second function header, **head_ref is a pointer to a pointer to a node object somewhere in memory:

```
 void push(struct Node** head_ref, int new_data);

              _____________________
             |                     |
    *head --> |     Node object     |
      ^       | [val=1][*next=NULL] |
      |       |_____________________|
      |
  **head_ref
```

/* We are using a double pointer

```
head_ref here because we change
head of the linked list inside
this function.*/
void moveToFront(Node **head_ref)
{
  /* If linked list is empty, or
  it contains only one node,
  then nothing needs to be done,
  simply return */
  if (*head_ref == NULL || (*head_ref)->next == NULL)
    return;

  /* Initialize second last
  and last pointers */
  Node *secLast = NULL;
  Node *last = *head_ref;

  /*After this loop secLast contains
  address of second last node and
  last contains address of last node in Linked List */
  while (last->next != NULL)
  {
    secLast = last;
    last = last->next;
  }

  /* Set the next of second last as NULL */
  secLast->next = NULL;

  /* Set next of last as head node */
  last->next = *head_ref;

  /* Change the head pointer
  to point to last node now */
  *head_ref = last;
}
```

14 Add two numbers represented by linked lists

```
345+45=>  ->5->4->3->NULL
          ->5->4
          =093
```

```
=>390
/* Adds contents of two linked lists and
return the head node of resultant list */
Node* addTwoLists(Node* first, Node* second)
{

  // res is head node of the resultant list
  Node* res = NULL;
  Node *temp, *prev = NULL;
  int carry = 0, sum;

  // while both lists exist
  while (first != NULL
      || second != NULL) {
    // Calculate value of next
    // digit in resultant list.
    // The next digit is sum of
    // following things
    // (i) Carry
    // (ii) Next digit of first
    // list (if there is a next digit)
    // (ii) Next digit of second
    // list (if there is a next digit)
    sum = carry + (first ? first->data : 0)
        + (second ? second->data : 0);

    // update carry for next calulation
    carry = (sum >= 10) ? 1 : 0;

    // update sum if it is greater than 10
    sum = sum % 10;

    // Create a new node with sum as data
    temp = newNode(sum);

    // if this is the first node then
    // set it as head of the resultant list
    if (res == NULL)
      res = temp;

    // If this is not the first
```

```
    // node then connect it to the rest.
    else
      prev->next = temp;

    // Set prev for next insertion
    prev = temp;

    // Move first and second
    // pointers to next nodes
    if (first)
      first = first->next;
    if (second)
      second = second->next;
  }

  if (carry > 0)
    temp->next = newNode(carry);

  // return head of the resultant list
  return res;
}
```

15. Binary Tree Level Order Traversal

how to use queue and return vector when the method is given in vector return type

```
vector<vector<int>> levelOrder(TreeNode* root) {
     vector<vector<int>> res;
     if(!root) return res;
      queue<TreeNode*> q;
      q.push(root);
      while(!q.empty()){
          int size = q.size();
          vector<int> v;
          for(int i = 0; i < size; i++){
              TreeNode* temp = q.front();
              q.pop();
              v.push_back(temp->val);
              if(temp->left) q.push(temp->left);
              if(temp->right) q.push(temp->right);
          }
          res.push_back(v);
      }
      return res;
  }
```

AVL TREE

Why AVL Trees?

Most of the BST operations (e.g., search, max, min, insert, delete, etc) take O(h) time where h is the height of

Most of the BST operations (e.g., search, max, min, insert, delete.. etc) take O(h) time where h is the height of the BST. The cost of these operations may become O(n) for a skewed Binary tree. If we make sure that height of the tree remains O(Logn) after every insertion and deletion, then we can guarantee an upper bound of O(Logn) for all these operations. The height of an AVL tree is always O(Logn) where n is the number of nodes in the tree (See this video lecture for proof).

18/12

16.Kth Largest Element in an Array

#heap

```
class Solution {
public:
  int findKthLargest(vector<int>& nums, int k) {
      priority_queue<int,vector<int>,greater<int>> minH;
    int n=nums.size();
    for(int i=0;i<n;i++){
    minH.push(nums[i]);
    if(minH.size()>k){
      minH.pop();
    }
  }
    return minH.top();
  }
};
```

17. **Sort an array of 0s, 1s and 20s**

if we simple sort the array the time complexity will be O(nlog n)

Dutch Flag national Algorithm. 0(N)

```
class Solution {
public:
  void sortColors(vector<int>& nums) {
    int lo=0;
    int mid=0;
    int hi=nums.size()-1;

    while(mid<=hi){

      switch(nums[mid]){

          case 0:
```

```
            swap(nums[lo++],nums[mid++]);
            break;

            case 1:
            mid++;
            break;

            case 2:
            swap(nums[mid],nums[hi--]);
            break;
        }
      }
    }
};
```

23/12

17.Intersection of Two Arrays

```
     vector<int> x;

     unordered_set<int> s;

     for(int i=0;i<nums1.size();i++)
     {
         s.insert(nums1[i]);
     }
    for(int i=0;i<nums2.size();i++)
    {
        if(s.find(nums2[i])!=s.end())
        {

            x.push_back(nums2[i]);
            s.erase(nums2[i]); //to handle the case when same element occurs again
        }
    }

     return x;
 }
```

24/12

18.Frequency of each character in a String using unordered_map in C++

```
void printFrequency(string str)
{
   // Define an unordered_map
   unordered_map<char, int> M;

   // Traverse string str check if
```

```
    // Traverse string str check if
    // current character is present
    // or not
    for (int i = 0; str[i]; i++)
    {
        // If the current characters
        // is not found then insert
        // current characters with
        // frequency 1
        if (M.find(str[i]) == M.end())
        {
            M.insert(make_pair(str[i], 1));
        }

        // Else update the frequency
        else
        {
            M[str[i]]++;
        }
    }

    // Traverse the map to print the
    // frequency
    for (auto& it : M) {
        cout << it.first << ' ' << it.second << '\n';
    }
}
```

25/12

19.Count occurrences of a string that can be constructed from another given string

#hashing

```
// Function to find the count
int findCount(string str1, string str2)
{
    int len = str1.size();
    int len2 = str2.size();
    int ans = INT_MAX;

    // Initialize hash for both strings
    int hash1[26] = { 0 }, hash2[26] = { 0 };

    // hash the frequency of letters of str1
```

```
    // hash the frequency of letters of str1
    for (int i = 0; i < len; i++)
        hash1[str1[i] - 'a']++;

    // hash the frequency of letters of str2
    for (int i = 0; i < len2; i++)
        hash2[str2[i] - 'a']++;

    // Find the count of str2 constructed from str1
    for (int i = 0; i < 26; i++)
        if (hash2[i])
            ans = min(ans, hash1[i] / hash2[i]);

    // Return answer
    return ans;
}
```

```
Input: str1 = "geeksforgeeks", str2 = "geeks"
Output: 2

Input: str1 = "geekgoinggeeky", str2 = "geeks"
Output: 0
```

26/12

20.Sort Array By Parity

```
class Solution {
public:
    vector<int> sortArrayByParity(vector<int>& A) {
        int start = 0;
        int end =A.size()-1;
        while(start<end){
            while(start< end and A[start]%2==0){
                start++;
            }
            while(start< end and A[end]%2==1){
                end--;
            }
            swap(A[start], A[end]);
            start++;
            end--;
        }
        return A;
    }
};
```

```
Input: [3,1,2,4]
Output: [2,4,3,1]
The outputs [4,2,3,1], [2,4,1,3], and [4,2,1,3] would also be accepted.
```

27/12

21. https://leetcode.com/contest/biweekly-contest-42/problems/number-of-students-unable-to-eat-lunch/

#stack #vector #2D #Contest

```
class Solution {
public:
   int countStudents(vector<int>& students, vector<int>& sandwiches) {
      reverse(sandwiches.begin(),sandwiches.end());
      queue<int>q;
      for(int x:students) q.push(x);

      int it=0;
      while(not q.empty() and not sandwiches.empty()){

         if(q.front()==sandwiches.back()){
            sandwiches.pop_back();
            q.pop();
            it=0;
         }
         else{
            q.push(q.front());
            q.pop();
            it++;
         }
         if(it==q.size()) return q.size();
      }
      return q.size();
   }
};
```

22. https://leetcode.com/problems/average-waiting-time/

#2D vector #accumulate

```
 double averageWaitingTime(vector<vector<int>>& customers) {
         vector<int> waiting {};
         int nextStart = 0;
         for (int i = 0; i < customers.size(); i++) {
                 if (customers[i][0] >= nextStart)
                         nextStart = customers[i][0];

                 int curWait = nextStart + customers[i][1] - customers[i][0];
                 nextStart += customers[i][1];
                 waiting.push_back(curWait);
         }
```

```
        return accumulate(waiting.begin(), waiting.end(), 0.0) / waiting.size();
}
```

28/12

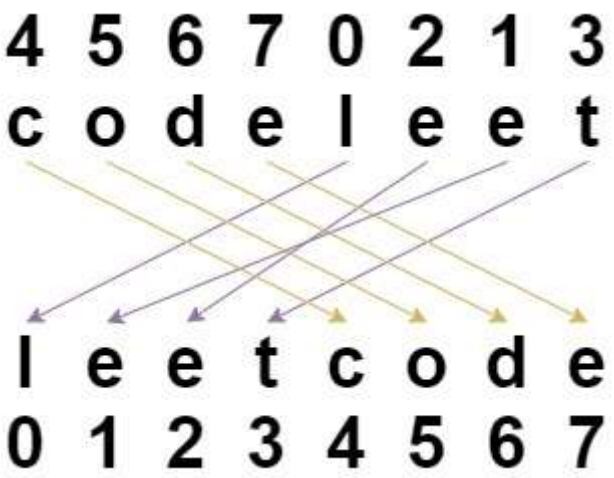
23. https://leetcode.com/problems/find-the-difference/
    #string

```
Input: s = "abcd", t = "abcde"
Output: "e"
Explanation: 'e' is the letter that was added.
```

```
class Solution {
public:
    char findTheDifference(string s, string t) {
   int p=t.size();
   //since it's given "random shuffling string s and then add one more letter at a random position.",we sort
it first
   sort(s.begin(), s.end());
   sort(t.begin(), t.end());


        for(int i=0;i<p;i++){
          if(t[i]!=s[i]){
              return t[i];
          }
        }
        return t[t.size()-1];
    }
};
```

24. https://leetcode.com/problems/shuffle-string/
    #string



```
Input: s = "codeleet", indices = [4,5,6,7,0,2,1,3]
Output: "leetcode"
Explanation: As shown, "codeleet" becomes "leetcode" after shuffling.
```

```
class Solution {
public:
    string restoreString(string s, vector<int>& indices) {
        int m=indices.size();
        string res=s;
        vector<char>r;

        for(int i=0;i<m;i++){
            for(int j=0;j<m;j++){
                if(indices[j]==i){
       r.push_back(res[j]);
        }
        }
        }

        for(int j=0;j<r.size();j++){
            res[j]=r[j];

        }

        return res;
    }
};
```

25. https://leetcode.com/problems/rotate-string/

#string #queue

```
Example 1:
Input: A = 'abcde', B = 'cdeab'
Output: true

Example 2:
Input: A = 'abcde', B = 'abced'
Output: false
```

```
class Solution {
public:
    bool rotateString(string A, string B) {
   queue<int>q,q1;
   int n=A.size();
    for(int i=0;A[i]!='\0';i++){
        q.push(A[i]);
                 q1.push(B[i]);
       }

    //if the string A and B are equal
        if(A==B){
```

```
            return true;
        }
       int i=0;
      while(i<n){
         q.push(q.front());
            q.pop();
            if(q==q1)
                return true;
            i++;
        }
       return false;
    }
};
```