**Assignment**

**Question 1):**

**Code;**

ListNode\* reverseList(ListNode\* head) {  
 ListNode\* prev = NULL;  
 ListNode\* curr = head;  
  
 while(curr != NULL)  
 {  
 ListNode\* forward = curr-> next;  
 curr-> next = prev;  
 prev = curr;  
 curr = forward;  
 }  
 return prev;  
 }

**Input:**



**Output:**



**Question 2);**

**Code;**

int LengthOfLongestRepeatingSubstring(string s) {

unordered\_map<char, long long> map;

int left = 0;

int ans = 0;

for (int r = 0; r < s.length(); r++) {

map[s[r]]++;

while (map[s[r]] > 1 && left <= r) {

map[s[left]]--;

left++;

}

ans = max(ans, r - left + 1);

}

return ans;

}

**Input:**



**Output:**



**Question 3):**

**Code;**

int calculateMaxPathSum(TreeNode\* root,int& maxSum) {

if(root == NULL)

return 0;

int leftSum = max(0,calculateMaxPathSum(root->left,maxSum));

int rightSum = max(0, calculateMaxPathSum(root->right,maxSum));

maxSum = max(maxSum,leftSum + rightSum + root->val);

return root->val + max(leftSum,rightSum);

}

int maxPathSum(TreeNode\* root) {

int maxSum = INT\_MIN;

calculateMaxPathSum(root,maxSum);

return maxSum;

}

**Input:**



**Output:**



**Question 4):**

**Code;**

string serialize(TreeNode\* root) {

if(!root) {

return "NULL,";

}

return to\_string(root->val)+","+serialize(root->left)+serialize(root->right);

}

TreeNode\* deserialize(string data) {

queue<string> q;

string s;

for(int i=0;i<data.size();i++)

{

if(data[i]==',')

{

q.push(s);

s="";

continue;

}

s+=data[i];

}

if(s.size()!=0)q.push(s);

return deserialize\_helper(q);

}

TreeNode\* deserialize\_helper(queue<string> &q) {

string s=q.front();

q.pop();

if(s=="NULL")return NULL;

TreeNode\* root=new TreeNode(stoi(s));

root->left=deserialize\_helper(q);

root->right=deserialize\_helper(q);

return root;

}

**Input:**



**Output:**



**Question 5):**

**Code;**

void Reverse(int arr[], int start, int end)  
{  
 while (start <= end)  
 {  
 int temp = arr[start];  
 arr[start] = arr[end];  
 arr[end] = temp;  
 start++;  
 end--;  
 }  
}  
  
int[] RotateElementRight(int a[], int n, int k)  
{  
   
 Reverse(a, 0, n - k - 1);  
 Reverse(a, n - k, n - 1);  
 Reverse(a, 0, n - 1);

return a;  
}

**Input:**





**Output:**



**Question 6):**

**Code;**

int Factorial(int n){

int ans=1;

for(int i=1;i<=n;i++){

ans\*=i;

}

return ans;

}

**Input:**

**5**

**Output:**



**Question 7):**

**Code;**

int SumOfDigits(int n){

int ans=0;

while(n>0){

ans+=n%10;

n/=10;

}

return ans;

}

**Input:**

**12345**

**Output:**



**Question 8):**

**Code;**

int GCD(int a,int b){

return b==0 ? a : GCD(b,a%b);

}

**Input:**

**GCD(15,20)**

**Output:**



**Question 9):**

**Code;**

#include <iostream>

#include <algorithm>

#include <vector>

int maximumDifference(std::vector<int> a) {

std::sort(a.begin(), a.end());

return a.back() - a.front();

}

**Input:**



**Output:**



**Question 10):**

**Code;**

#include <iostream>

#include <string>

#include <cctype>

using namespace std;

bool checkAlphabeticString(string s) {

for (char c: s) {

if (!isalpha(c)) {

return false;

}

}

return true;

}

int main() {

cout << boolalpha << checkAlphabeticString("agfhcujsAFEFSDC") << endl;

cout << boolalpha << checkAlphabeticString("agfh&cujsAFEFSDC") << endl;

return 0;

}

**OutPut:**

