\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Find Word in String\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int strStr(string haystack, string needle) { if(needle.length()==0){

return 0; }

if( haystack.length() == needle.length() ){ if(haystack==needle){return 0;}

else{return -1;}

}

string slide="";

for(int i=0;i<needle.length();i++)

slide=slide+haystack[i];

if(slide==needle){return 0;}

for(int i= needle.length() ; i<haystack.length() ; i++){ slide.erase(slide.begin());

slide= slide + haystack[i];

if(slide==needle){ return i-( needle.length() -1; }

}

return -1;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*remove duplicate from array\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int removeDuplicates(vector<int>& nums) {

int count = 0;

for(int i=1; i<nums.size(); i++){

If(nums[i] == nums[i-1]) count++;

else nums[i-count] = nums[i];

}

return (nums.size()-count);

}

\*\*\*\*\*\*\*\*\*\*\*\* Recursive call for tree formation and condition checking\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void find(int indx, int target ,vector<int> &arr, vector<vector<int>> &ans, vector<int>&ds){

if(indx == arr.size()){

if(target == 0)

ans.push\_back(ds);

return; }

if(arr[indx]<=target){

ds.push\_back(arr[indx]);

find(indx+1, target-arr[indx] ,arr,ans,ds);

ds.pop\_back();

}

while(indx + 1 < arr.size() && arr[indx] == arr[indx+1])

indx++;

find(indx+1,target,arr,ans,ds);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Add one at last pos (all element+ve and less than 9 i.e.[9,9]=[1,0,0])\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

vector<int> plusOne(vector<int>& digits) {

int n=digits.size();

for(int i=n-1 ; i>=0;i-- ){

if (digits[i] == 9) digits[i] = 0;

else {

digits[i]++;

return digits; }

}

digits[0] =1;

digits.push\_back(0);

return digits;

}

\*\*\*\*\*\*\*\*\*\*\*First Unique Char In String\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int firstUniqChar(string s) {

unordered\_map<char,int> m;

for(auto &c:s)

m[c]++;

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

if( m[s[i]]==1 )

return i;

}

return -1;

}

\*\*\*\*\*\*\*\*\*\*\*\*check for every element if its double or half exist in the array\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool checkIfExist(vector<int>& arr) {

set<int> m;

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

if( m.find(2\*arr[i])!=m.end() || (m.find(arr[i]/2)!=m.end() && arr[i]%2==0)) return 1;

else m.insert(arr[i]);

}

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Count no. of prime less than N (n not included)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int countPrimes(int n) {

vector<bool> prime(n, true);

prime[0] = false, prime[1] = false;

for (int i = 0; i < sqrt(n); ++i) {

if (prime[i]) {

for (int j = i\*i; j < n; j += i) prime[j] = false;

}

}

return count(prime.begin(), prime.end(), true);}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*palindrome Checker for all characters and number\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool isPalindrome(string s) {

for(int i=0,j=s.size()-1; i<j; i++,j--) {//Move 2 pointers from each end until they collide

while(isalnum(s[i])==false && i<j) i++; //Increment left pointer if not alphanumeric

while(isalnum(s[j])==false && i<j)j--; //Decrement right pointer if no alphanumeric

if(toupper(s[i])!=toupper(s[j])) return false; //Exit and return error if not match

}

return true;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*max sum of k elements in array[n] with negative no.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double findMaxAverage(vector<int>& nums, int k) {

double sum=0,res=INT\_MIN;

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

if(i<k) sum+=nums[i];

else{

res=max(res,sum);

sum+=nums[i]-nums[i-k];

}

}

res = max(res,sum);//for single elment sum gets updated not res

return res/k;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Traiiling Zeros in factorial (i.e. max factor of 5)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int trailingZeroes(int n) {

if(n<5) return 0;

int c = 0;

for(int i=5; n/i >=1; i\*=5)

c+=n/i;

return c;

}