**Day-16**

**Problem1**- [**152. Maximum Product Subarray**](https://leetcode.com/problems/maximum-product-subarray/)

Solution-

class Solution {

public:

int maxProduct(vector<int>& nums) {

int n= nums.size();

int pre=1;

int suff=1;

int m=0;

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

{

if(pre==0)pre=1;

if(suff==0)suff=1;

pre =pre\*nums[i];

suff=suff\*nums[n-i-1];

m=max(m,max(pre,suff));

}

if(n==1 && nums[0]<0)

return nums[0];

return m;

}

};}

**Problem2**- **GROUP ANAGRAM**

Solution-

class Solution {

public:

vector<vector<string>> groupAnagrams(vector<string>& strs) {

unordered\_map<string,vector<string>>m;

int n=strs.size();

string temp;

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

{

temp=strs[i];

sort(strs[i].begin(),strs[i].end());

m[strs[i]].push\_back(temp);

}

vector<vector<string>>v;

for(auto i:m)

{

v.push\_back(i.second);

}

return v;

}

};**Problem 3**-WORD WRAP

class Solution {

public:

int helper(int curr, int spaces, vector<int>& nums, int k) {

if (curr == nums.size()) // Base case: end of the words

return 0;

int a = INT\_MAX; // Initialize a with the maximum integer value

int new\_spaces = spaces + 1 + nums[curr]; // Calculate new space required if we add the current word to the same line

if (new\_spaces <= k) { // Check if the current word can be added to the current line

a = helper(curr + 1, new\_spaces, nums, k);

}

// Calculate cost if the current word starts a new line

int b = helper(curr + 1, nums[curr], nums, k) + (k - spaces) \* (k - spaces);

return min(a, b); // Return the minimum cost

}

int solveWordWrap(vector<int> nums, int k) {

return helper(1, nums[0], nums, k); // Start recursion with the second word

}

};