QA:

#include <stdio.h>

int longestPalindromeSubstring(char \*s, int n) {

    int maxLength = 1; // All substrings of length 1 are palindromes

    int start = 0; // Initialize starting index of the palindrome substring

    // Loop to check for palindrome substrings of odd length

    for (int i = 0; i < n; i++) {

        int left = i - 1;

        int right = i + 1;

        while (left >= 0 && right < n && s[left] == s[right]) {

            if (right - left + 1 > maxLength) {

                start = left;

                maxLength = right - left + 1;

            }

            left--;

            right++;

        }

    }

    // Loop to check for palindrome substrings of even length

    for (int i = 0; i < n; i++) {

        int left = i;

        int right = i + 1;

        while (left >= 0 && right < n && s[left] == s[right]) {

            if (right - left + 1 > maxLength) {

                start = left;

                maxLength = right - left + 1;

            }

            left--;

            right++;

        }

    }

    return maxLength;

}

int main() {

    int N;    scanf("%d", &N);

char arr[N];

   scanf("%s",arr);

   printf("%d\n",longestPalindromeSubstring(arr,N));

    return 0;

}

QB:

#include <stdio.h>

#define MAX\_CHARS 26 // Number of lowercase English alphabets

int maxConsecutiveTroops(char \*troops, int n) {

    int maxCount = 0;

    int charIndex[MAX\_CHARS]; // Array to store the last index of each character

    for (int i = 0; i < MAX\_CHARS; i++) {

        charIndex[i] = -1; // Initialize all indices to -1

    }

    int start = 0;

    for (int end = 0; end < n; end++) {

        int index = troops[end] - 'a';

        if (charIndex[index] >= start) {

            start = charIndex[index] + 1; // Move the start index to the next position after the repeated character

        }

        charIndex[index] = end; // Update the last index of the character

        int currentCount = end - start + 1;

        if (currentCount > maxCount) {

            maxCount = currentCount; // Update maxCount if needed

        }

    }

    return maxCount;

}

int main() {

    int n;

    scanf("%d", &n);

    char troops[n];

    scanf("%s", troops);

    int result = maxConsecutiveTroops(troops, n);

    printf("%d\n", result);

    return 0;

}

QC:

#include <stdio.h>

void reverse(char\* str,int start,int end){

    while(start<end){

        int temp=str[start];

        str[start]=str[end];

        str[end]=temp;

        start++;

        end--;

    }

}

void reverseWords(char\* arr,int n){

    int start=0;

    for(int i=0;i<n;i++){

        if(arr[i]==' '){

            reverse(arr,start,i-1);

            start=i+1;

        }

    }

    reverse(arr,start,n-1);

}

int main() {

  int n; scanf("%d",&n);

  char arr[n+1];

getchar();

fgets(arr,sizeof(arr),stdin);

    reverseWords(arr,n);

    printf("%s\n",arr);

    return 0;

}

QD:

#include <stdio.h>

int main() {

  int n; scanf("%d",&n);

  char arr[n+1];

getchar();

fgets(arr,sizeof(arr),stdin);

char vow[n+1];

int k=0;

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

if(arr[i]=='a' || arr[i]=='e' || arr[i]=='i' || arr[i]=='o' || arr[i]=='u'){

vow[k++]=arr[i];

}

}

int j=k-1;

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

   if(arr[i]=='a' || arr[i]=='e' || arr[i]=='i' || arr[i]=='o' || arr[i]=='u'){

        arr[i]=vow[j--];

    }

}

    printf("%s\n",arr);

    return 0;

}

**BONUS**:

#include <stdio.h>

// Function to swap characters

void swap(char \*a, char \*b) {

    char temp = \*a;

    \*a = \*b;

    \*b = temp;

}

int isVowel(char ch) {

    return (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u');

}

int main() {

    int n;

    scanf("%d", &n);

    char arr[n + 1];

    getchar(); // consume newline character

    fgets(arr, sizeof(arr), stdin);

    // Two-pointer approach to reverse vowels

    int left = 0, right = n - 1;

    while (left < right) {

        // Find left vowel

        while (left < right && !isVowel(arr[left]))

            left++;

        // Find right vowel

        while (left < right && !isVowel(arr[right]))

            right--;

        // Swap vowels

        if (left < right) {

            swap(&arr[left], &arr[right]);

            left++;

            right--;

        }

    }

    printf("%s", arr);

    return 0;

}

QE:

#include <stdio.h>

void mergeArrays(int a[], int n, int b[], int m) {

    int merged[n + m]; // merged array to store the result

    int i = 0, j = 0, k = 0; // Pointers for arrays a, b, and merged

    // Merge arrays a and b into merged array

    while (i < n && j < m) {

        if (a[i] <= b[j]) {

            merged[k++] = a[i++];

        } else {

            merged[k++] = b[j++];

        }

    }

    // Copy remaining elements from array a

    while (i < n) {

        merged[k++] = a[i++];

    }

    // Copy remaining elements from array b

    while (j < m) {

        merged[k++] = b[j++];

    }

    // Print the merged sorted array

    for (int l = 0; l < n + m; l++) {

        printf("%d ", merged[l]);

    }

    printf("\n");

}

int main() {

    int n, m;

    scanf("%d %d", &n, &m);

    int a[n], b[m];

    for (int i = 0; i < n; i++) {

        scanf("%d", &a[i]);

    }

    for (int i = 0; i < m; i++) {

        scanf("%d", &b[i]);

    }

    // Merge arrays a and b

    mergeArrays(a, n, b, m);

    return 0;}

QF:

#include <stdio.h>

int main() {

    int n, k, x;

    scanf("%d %d %d", &n, &k, &x);

    long long int a[n]; // Using long long to avoid integer overflow

    // Input the cursed energies

    for (int i = 0; i < n; i++) {

        scanf("%lld", &a[i]);

    }

    long long int sum = 0;

    int count = 0;

    // Initialize the sliding window

    for (int i = 0; i < k; i++) {

        sum += a[i];

    }

    // Check the average energy of the initial window

    if (sum / k >= x) {

        count++;

    }

    // Move the window and update sum

    for (int i = k; i < n; i++) {

        sum = sum - a[i - k] + a[i];

        if (sum / k >= x) {

            count++;

        }

    }

    printf("%d\n", count);

    return 0;

}

QG:

#include <stdio.h>

long long maxPeanutButter(int a[], int n) {

    int left = 0, right = n - 1;

    long long nomWeight = 0, psyWeight = 0;

    long long maxWeight = 0;

    // Move pointers towards each other

    while (left <= right) {

        // If Nom's total weight is less than Psy's, he eats the jar on the left

        if (nomWeight <= psyWeight) {

            nomWeight += a[left];

            left++;

        }

        // Otherwise, Psy eats the jar on the right

        else {

            psyWeight += a[right];

            right--;

        }

        // Update the maximum weight eaten by Nom and Psy

        if (nomWeight == psyWeight) {

            maxWeight = nomWeight;

        }

    }

    return maxWeight;

}

int main() {

    int n;

    scanf("%d", &n);

    int a[n];

    for (int i = 0; i < n; i++) {

        scanf("%d", &a[i]);

    }

    printf("%lld\n", maxPeanutButter(a, n));

    return 0;

}

QH:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_N 100000

int main() {

    int n, k;

    unsigned long long a[MAX\_N];

    int satisfied[64] = {0};

    int num\_satisfied = 0;

    int max\_num\_satisfied = 0;

    int i, j;

    scanf("%d %d", &n, &k);

    for (i = 0; i < n; i++) {

        scanf("%llu", &a[i]);

    }

    for (i = 0; i < 60; i++) {

        num\_satisfied = 0;

        for (j = 0; j < k; j++) {

            if (a[j] & (1ULL << i)) {

                num\_satisfied++;

            }

        }

        satisfied[i] = num\_satisfied;

    }

    max\_num\_satisfied = num\_satisfied;

    for (i = k; i < n; i++) {

        for (j = 0; j < 60; j++) {

            if (a[i - k] & (1ULL << j)) {

                satisfied[j]--;

            }

            if (a[i] & (1ULL << j)) {

                satisfied[j]++;

            }

        }

        num\_satisfied = 0;

        for (j = 0; j < 60; j++) {

            if (satisfied[j] > 0) {

                num\_satisfied++;

            }

        }

        if (num\_satisfied > max\_num\_satisfied) {

            max\_num\_satisfied = num\_satisfied;

        }

    }

    printf("%d\n", max\_num\_satisfied);

  return 0;}

QI:

#include <stdio.h>

#include <math.h>

#define MOD 1000000007

int isPrime(long long int num) {

    if (num < 2) return 0;

    if (num % 2 == 0) return num == 2;

    if (num % 3 == 0) return num == 3;

    long long int sqrtNum = sqrt(num);

    for (long long int i = 5; i <= sqrtNum; i += 6) {

        if (num % i == 0 || num % (i + 2) == 0)

            return 0;

    }

    return 1;

}

long long int productOfNonPrimes(int n, long long int arr[]) {

    long long int product = 1;

    for (int i = 0; i < n; i++) {

        if (isPrime(arr[i]))

            continue;

        else

            product = (product \* arr[i]) % MOD;

    }

    return (product == 1) ? -1 : product;

}

int main() {

    int n;

  scanf("%d", &n);

    long long int arr[n];

    for (int i = 0; i < n; i++){

        scanf("%lld", &arr[i]);}

    long long int result = productOfNonPrimes(n, arr);

    printf("%lld\n", result);

    return 0;

}

QJ:

#include <stdio.h>

#include <stdint.h>

#define MOD 1000000007

// Function to calculate (base^exponent) % MOD

long long int power(long long int x, long long int y) {

    long long int result = 1;

    while (y > 0) {

        if (y % 2 == 1)

            result = (result \* x) % MOD;

        x = (x \* x) % MOD;

        y /= 2;

    }

    return result;

}

int main() {

    int n;

    scanf("%d", &n);

long long int a[n], b[n];

    // Input the elements of array a

    for (int i = 0; i < n; i++)

        scanf("%lld", &a[i]);

    // Input the elements of array b

    for (int i = 0; i < n; i++)

        scanf("%lld", &b[i]);

    // Calculate the product Qi=n i=1 (ai^bi) mod (10^9 + 7)

    long long int product = 1;

    for (int i = 0; i < n; i++) {

        // Calculate ai^bi modulo (10^9 + 7) using modulo exponentiation

        long long int term = power(a[i], b[i]);

        // Update the product

        product = (product \* term) % MOD;

    }

    printf("%lld\n", product);

    return 0;}