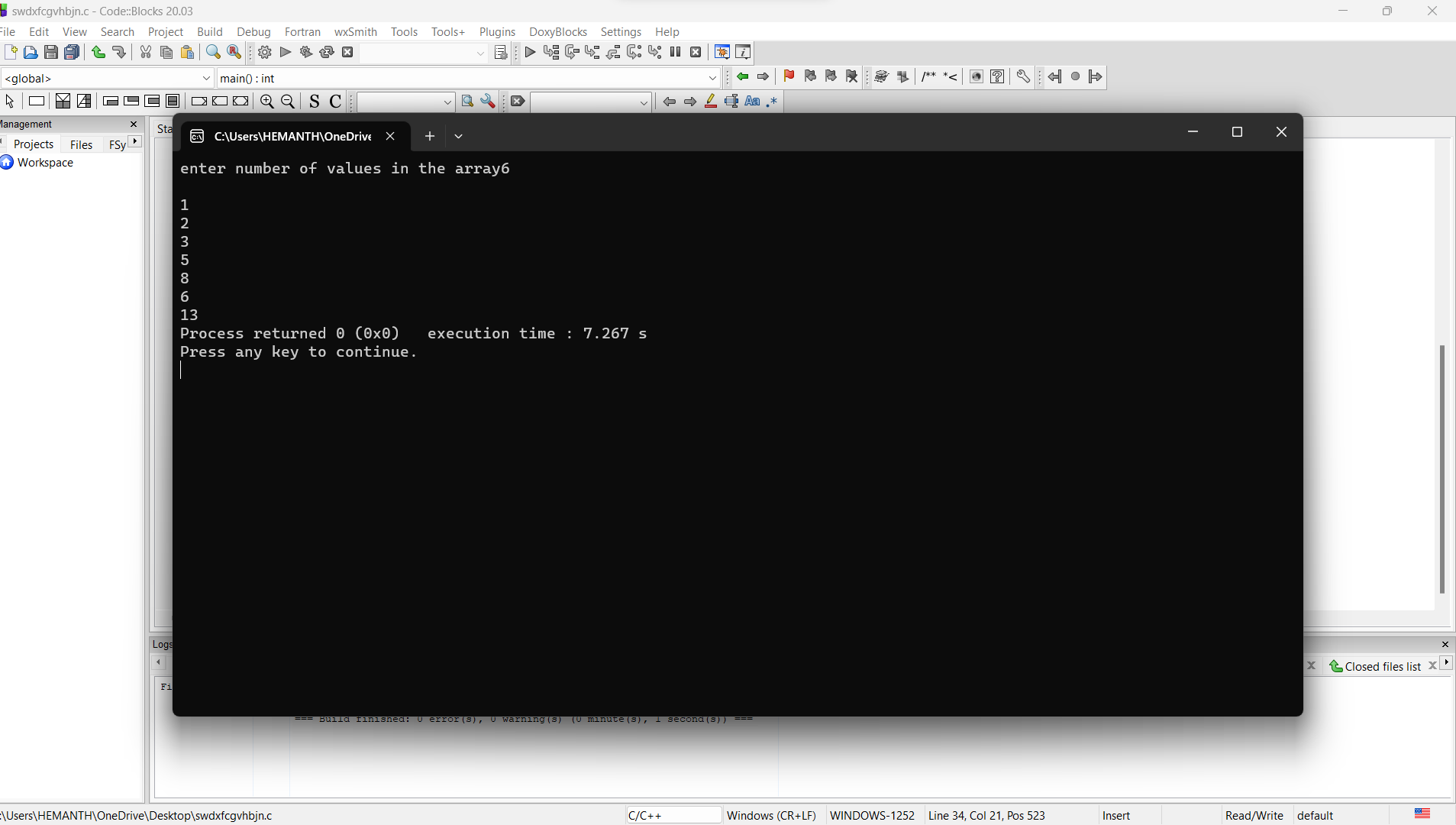
Humpy likes to jump from one building to another. But he only jumps to next higher building and stops when no higher building is available. Stamina required for a journey is **xor** of all the heights on which humpy jumps until he stops.

If heights are [1 2 4], and he starts from 1, goes to 2 stamina required is 1⊕2=3, then from 2 to

3. Stamina for the entire journey is 1⊕2⊕4=7. Find the maximum stamina required if can start his journey from any building.

/\*program of problem 3\*/  
#include<stdio.h>  
#define MAX 100  
  
int calmax(int arr[],int i,int n)  
{  
    int max=arr[i];  
    while(i<n)  
        {  
            if(arr[i+1]>arr[i]){  
                max=max^(arr[i+1]);  
                i+=2;  
            }  
        else  
            break;  
    return max;  
    }  
  
}  
int maximam(int maxima[],int n){  
int i=0;  
int fval=0;  
while(i<n){  
    if(maxima[i]>fval)  
        fval=maxima[i];  
 i++;  
}  
return fval;  
}  
  
int main()  
{  
    int g,n,i=0;  
    int maxima[MAX];  
    int arr[MAX];  
    printf("enter number of values in the array");  
    scanf("%d",&n);  
    while(i<n)  
        {  
            scanf("%d",&arr[i]);  
            i++;  
        }  
    i=0;  
    while(i<n){  
        g= calmax(arr,i,n);  
        maxima[i]=g;  
        i++;  
    }  
    int final=maximam(maxima,n);  
    printf("%d",final);  
}

**OUTPUT:**



You are given a stack of N integers such that the first element represents the top of the stack and the last element represents the bottom of the stack. You need to pop at least one element from the stack. At any one moment, you can convert stack into a queue. The bottom of the stack represents the front of the queue. You cannot convert the queue back into a stack. Your task is to remove exactly K elements such that the sum of the K removed elements is maximised in c

# PROGRAM:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_STACK\_SIZE 100

Struct Queue {

int front, rear, size;

unsigned capacity;

int\* array;

};

struct Queue\* createQueue(unsigned capacity) {

struct Queue\* queue = (struct Queue\*)malloc(sizeof(struct Queue));

queue->capacity = capacity;

queue->front = 0;

queue->size = 0;

queue->rear = capacity - 1;

queue->array = (int\*)malloc(queue->capacity \* sizeof(int));

return queue;

}

int isFull(struct Queue\* queue) {

return (queue->size == queue->capacity);

}

int isEmpty(struct Queue\* queue) {

return (queue->size == 0);

}

void enqueue(struct Queue\* queue, int item) {

if (isFull(queue))

return;

queue->rear = (queue->rear + 1) % queue->capacity;

queue->array[queue->rear] = item;

queue->size = queue->size + 1;

}

int dequeue(struct Queue\* queue) {

if (isEmpty(queue))

return -1;

int item = queue->array[queue->front];

queue->front = (queue->front + 1) % queue->capacity;

queue->size = queue->size - 1;

return item;

}

int max\_sum\_of\_removed\_elements(int stack[], int N, int K) {

if (K >= N) {

int sum = 0;

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

sum += stack[i];

}

return sum;

}

struct Queue\* queue = createQueue(N);

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

enqueue(queue, stack[N - 1 - i]);

while (queue->size > K) {

dequeue(queue);

}

int sum = 0;

for (int i = 0; i < queue->size; i++) {

sum += queue->array[i];

}

return sum;

}

}

int main() {

int stack[MAX\_STACK\_SIZE];

int N, K;

printf("Enter the number of elements in the stack: ");

scanf("%d", &N);

printf("Enter the elements of the stack:\n");

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

scanf("%d", &stack[i]);

}

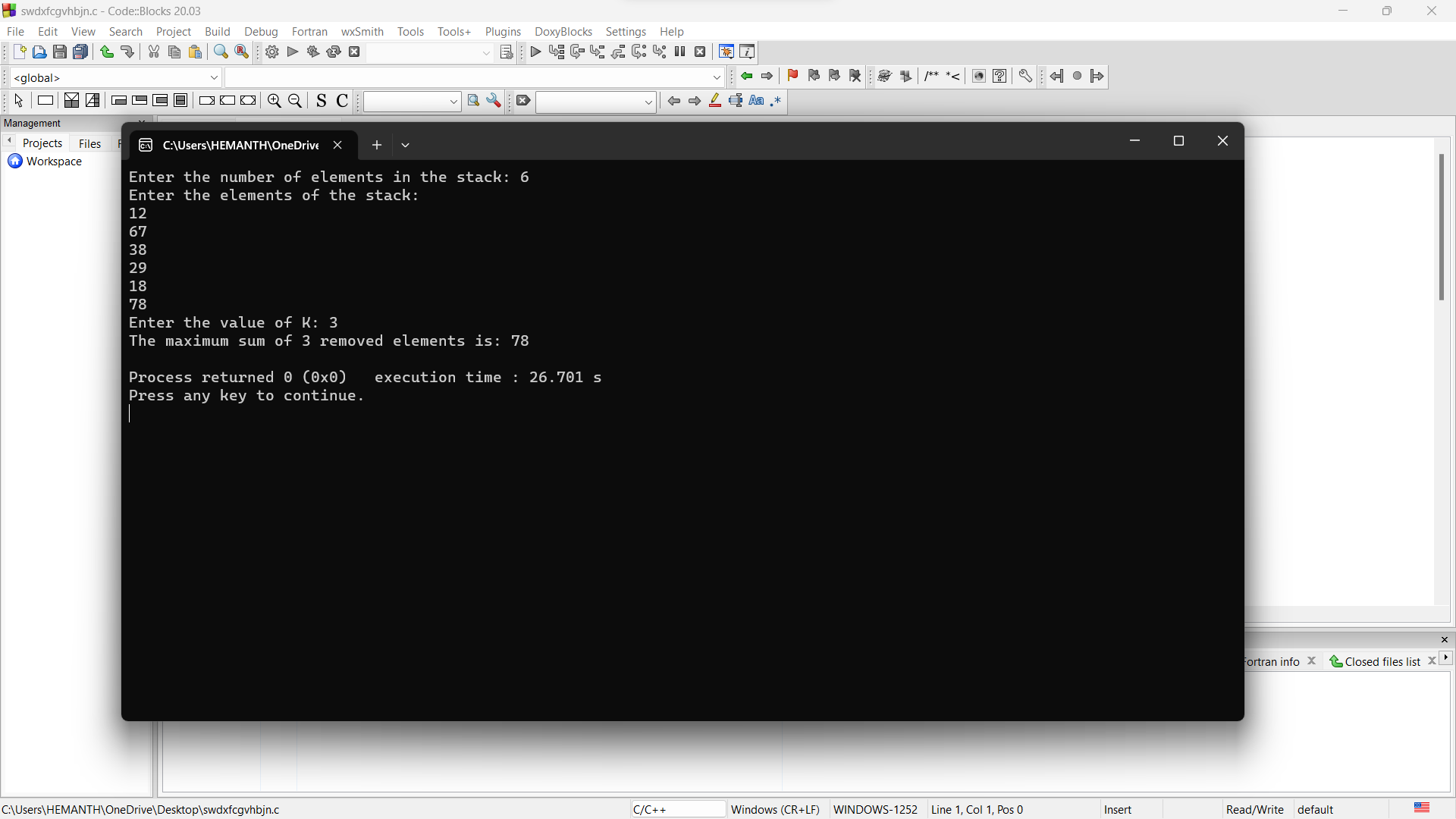
printf("Enter the value of K: ");

scanf("%d", &K);

int max\_sum = max\_sum\_of\_removed\_elements(stack, N, K);

printf("The maximum sum of %d removed elements is: %d\n", K, max\_sum);

}



Your task is to construct a tower in N days by following these conditions: Every day you are provided with one disk of distinct size. The disk with larger sizes should be placed at the bottom of the tower. The disk with smaller sizes should be placed at the top of the tower. The order in which tower must be constructed is as follows: You cannot put a new disk on the top of the tower until all the larger disks that are given to you get placed. Print N lines denoting the disk sizes that can be put on the tower on the i^th day**.**

#include <stdio.h>

int main() {

int numDays;

printf("Enter the number of days (N): ");

scanf("%d", &numDays);

int tower[numDays];

int currentTop = 0;

printf("Disk sizes that can be added to the tower on each day:\n");

for (int day = 1; day <= numDays; day++) {

tower[currentTop] = day;

for (int level = currentTop; level >= 0; level--) {

printf("%d ", tower[level]);

}

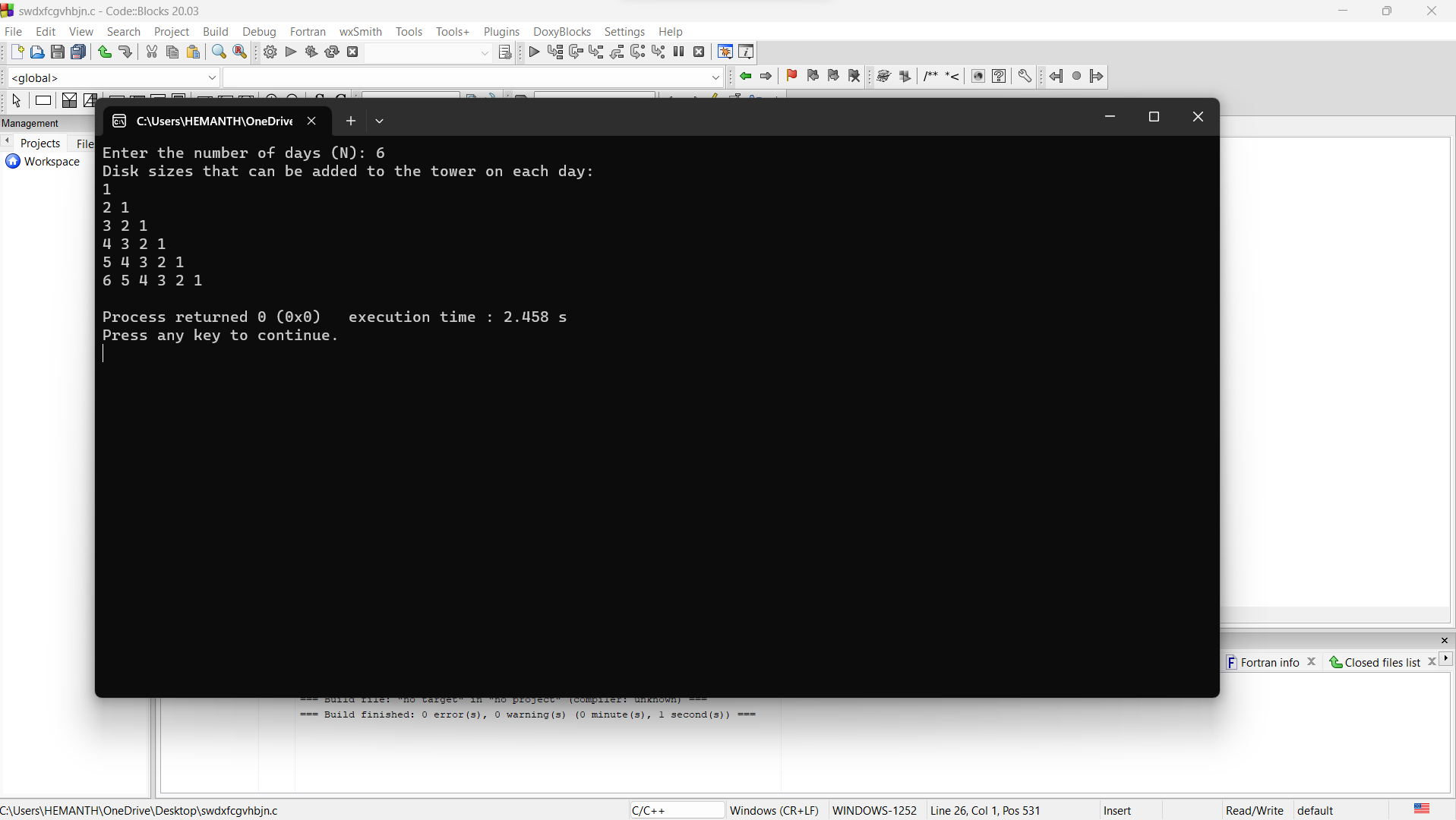
printf("\n");

currentTop++;

}

return 0;

}

****

You are given two arrays each of size n , a and b consisting of the first n positive integers each exactly once, that is, they are permutations. Your task is to find the minimum time required to make both the arrays empty. The following two types of operations can be performed any number of times each taking 1 second: In the first operation, you are allowed to rotate the first array clockwise. In the second operation, when the first element of both the arrays is the same, they are removed from both the arrays and the process continues.

# #include <stdio.h>

# int minTimeToEmptyArrays(int arrayA[], int arrayB[], int n) {

# int indexA = 0, indexB = 0, rotations = 0;

# 

# while (indexA < n && indexB < n) {

# if (arrayA[indexA] == arrayB[indexB]) {

# indexA++;

# indexB++;

# } else {

# indexA++;

# rotations++;

# }

# }

# while (indexA < n) {

# indexA++;

# rotations++;

# }

# return rotations;

# }

# int main() {

# int n;

# printf("Enter the value of n: ");

# scanf("%d", &n);

# int arrayA[n], arrayB[n];

# 

# printf("Enter the elements of array A: ");

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

# scanf("%d", &arrayA[i]);

# }

# 

# printf("Enter the elements of array B: ");

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

# scanf("%d", &arrayB[i]);

# }

# int result = minTimeToEmptyArrays(arrayA, arrayB, n);

# printf("The minimum time required to make both arrays empty is: %d seconds.\n", result);

# return 0;

# }

# 