**Reverse a Linked List in Groups of Size K**

You are given a singly linked list. The task is to reverse the linked list in groups of size   
k. If the number of nodes in the list is not a multiple of k, then the remaining nodes at   
the end should remain as they are. Implement the solution for this problem using a   
linked list.( linked list)

Input Format:   
• The first line contains two integers: n (the number of nodes in the linked list) and   
k (the size of the group to reverse).   
• The second line contains n space-separated integers, where each integer is the   
value of a node in the linked list.

 Output Format:   
• Output the linked list after reversing it in groups of size k.

**Constraints:**

NA

**Example:**

**Input:**  
5 3   
1 2 3 4 5   
**Output:**  
3 2 1 4 5

**Explanation:**

• The first group of 3 elements (1 2 3) is reversed to (3 2 1).   
• The remaining elements (4 5) are left as is, since there are fewer than k elements.

**Public Test Cases:**

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 5 3  1 2 3 4 5 | 3 2 1 4 5 |
| **2** | 6 2  1 2 3 4 5 6 | 2 1 4 3 6 5 |
| **3** | 7 3  1 2 3 4 5 6 7 | 3 2 1 6 5 4 7 |

#include<iostream>

using namespace std;

struct node {

int data;

struct node\* next;

};

void insert(struct node\*& head, int d) {

struct node\* newnode = new struct node;

newnode->data = d;

newnode->next = NULL;

if (head == NULL) {

head = newnode;

} else {

struct node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newnode;

}

}

struct node\* reverseKGroup(struct node\* head, int k) {

if (head == NULL || k == 1) return head;

struct node dummy;

dummy.next = head;

struct node\* prevGroupEnd = &dummy;

struct node\* cur = head;

while (true) {

struct node\* kthNode = prevGroupEnd;

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

kthNode = kthNode->next;

if (kthNode == NULL) return dummy.next;

}

struct node\* groupStart = prevGroupEnd->next;

struct node\* nextGroupStart = kthNode->next;

struct node\* prev = nextGroupStart;

struct node\* temp = groupStart;

while (temp != nextGroupStart) {

struct node\* next = temp->next;

temp->next = prev;

prev = temp;

temp = next;

}

prevGroupEnd->next = kthNode;

prevGroupEnd = groupStart;

}

}

void printList(struct node\* head) {

struct node\* temp = head;

while (temp != NULL) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

int main() {

int n, k;

struct node\* head = NULL;

cin >> n >> k;

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

int m;

cin >> m;

insert(head, m);

}

head = reverseKGroup(head, k);

printList(head);

return 0;

}

**Task Scheduler**

You are given **N** tasks, each with a unique task ID (1 to N). These tasks need to be processed in a specific order. However, some tasks have dependencies, meaning they must be completed before other tasks can be executed. You are also given **M** dependencies, where each dependency is represented as **(X, Y)**, meaning task **X**must be completed before task **Y.**

Your task is to determine the valid execution order of tasks using a queue-based approach. If multiple tasks are available to be executed at the same time, the task with the **lowest ID** should be executed first. If it is impossible to complete all tasks due to a circular dependency, return**-1**.(Queue)

**Constraints:**

* 1≤N≤1051 \leq N \leq 10^5
* 0≤M≤2×1050 \leq M \leq 2 \times 10^5
* 1≤X,Y≤N1 \leq X, Y \leq N
* No duplicate dependencies, i.e., (X, Y) pairs are unique.

**Example:**

NA

**Explanation:**

**Input Format**  
• The first line contains two integers N and M — the number of tasks and the number  
of dependencies.  
• The next M lines contain two integers X and Y, indicating that task X must be  
completed before task Y.

**Output Format**  
• Print N space-separated integers representing the valid execution order of tasks.  
• If it is impossible to complete all tasks, print -1.

**Public Test Cases:**

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 5 4  1 2  1 3  3 4  2 5 | 1 2 3 5 4 |
| **2** | 3 2  1 2  2 3 | 1 2 3 |
| **3** | 6 5  1 3  2 3  3 4  4 5  4 6 | 1 2 3 4 5 6 |

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

vector<int> taskScheduler(int n, int m, vector<pair<int, int>> &dependencies) {

vector<int> adj[n + 1], indegree(n + 1, 0), result;

queue<int> q;

// Build the adjacency list and indegree array

for (auto &dep : dependencies) {

int x = dep.first, y = dep.second;

adj[x].push\_back(y);

indegree[y]++;

}

// Add tasks with 0 indegree to the queue

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

if (indegree[i] == 0) {

q.push(i);

}

}

while (!q.empty()) {

int task = q.front();

q.pop();

result.push\_back(task);

// Reduce the indegree of neighboring tasks

for (int neighbor : adj[task]) {

indegree[neighbor]--;

if (indegree[neighbor] == 0) {

q.push(neighbor);

}

}

}

// If the result size is less than n, it means there was a cycle

if (result.size() < n) return {-1};

return result;

}

int main() {

int n, m;

cin >> n >> m;

vector<pair<int, int>> dependencies(m);

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

cin >> dependencies[i].first >> dependencies[i].second;

}

vector<int> result = taskScheduler(n, m, dependencies);

if (result[0] == -1) {

cout << -1 << endl;

} else {

for (int task : result) {

cout << task << " ";

}

cout << endl;

}

return 0;

}

**Merging Two Interleaved Sorted Linked Lists**

You are given two singly linked lists A and B, both sorted in non-decreasing order. Your   
task is to merge these two lists into a single linked list while preserving the relative order of   
elements.   
However, instead of merging them in the usual way, you must interleave the elements such   
that:   
• You take one element from A, then one from B, then the next from A, and so on.   
• If one list gets exhausted, append the remaining elements of the other list as they are.   
   
Input Format:   
• The first line contains two integers N (length of list A) and M (length of list B).   
• The second line contains N space-separated integers representing list A.   
• The third line contains M space-separated integers representing list B.   
   
Output Format:   
• Print the final merged linked list as space-separated integers.

**Constraints:**

• 1≤N,M≤10^5   
• −10^9≤ Listvalues <=10^9   
• Lists A and B are already sorted in non-decreasing order.

**Example:**

**Input:**  
4 3   
1 3 5 7   
2 4 6

**Output:**  
1 2 3 4 5 6 7

**Explanation:**

Input:   
4 3   
1 3 5 7   
2 4 6   
Processing:   
• Start interleaving:    
o Pick 1 from A   
o Pick 2 from B   
o Pick 3 from A   
o Pick 4 from B   
o Pick 5 from A   
o Pick 6 from B   
o Pick 7 from A (A is longer, so it continues)   
Output:   
1 2 3 4 5 6 7

**Public Test Cases:**

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 4 3  1 3 5 7  2 4 6 | 1 2 3 4 5 6 7 |
| **2** | 2 5  10 20  5 15 25 30 35 | 10 5 20 15 25 30 35 |
| **3** | 3 3  -10 -5 0  -8 -3 2 | -10 -8 -5 -3 0 2 |

#include <iostream>

#include <vector>

using namespace std;

struct ListNode {

int val;

ListNode\* next;

ListNode(int x) : val(x), next(NULL) {}

};

ListNode\* mergeInterleaved(ListNode\* A, ListNode\* B) {

ListNode\* dummy = new ListNode(0);

ListNode\* current = dummy;

while (A != NULL && B != NULL) {

current->next = A;

A = A->next;

current = current->next;

current->next = B;

B = B->next;

current = current->next;

}

if (A != NULL) current->next = A;

if (B != NULL) current->next = B;

return dummy->next;

}

ListNode\* createLinkedList(vector<int>& arr) {

ListNode\* dummy = new ListNode(0);

ListNode\* current = dummy;

for (int num : arr) {

current->next = new ListNode(num);

current = current->next;

}

return dummy->next;

}

void printLinkedList(ListNode\* head) {

while (head != NULL) {

cout << head->val << " ";

head = head->next;

}

cout << endl;

}

int main() {

int N, M;

cin >> N >> M;

vector<int> A(N), B(M);

for (int i = 0; i < N; i++) cin >> A[i];

for (int i = 0; i < M; i++) cin >> B[i];

ListNode\* listA = createLinkedList(A);

ListNode\* listB = createLinkedList(B);

ListNode\* mergedList = mergeInterleaved(listA, listB);

printLinkedList(mergedList);

return 0;

}

**A peak element**

You are given an array nums of n elements. A peak element is an element that is strictly   
greater than its neighbors. For the elements at the ends of the array, we only consider   
one neighbor. Write a function findPeakElement(nums) that returns the index of a peak   
element. .( Search)

Your solution should run in O(log n) time.   
Input:   
• An integer array nums of length n (1 ≤ n ≤ 10^5), where each element is between -10^4 and 10^4.   
Output:   
• Return the index of any peak element.

**Constraints:**

Your solution should run in O(log n) time.

**Example:**

Input:   
nums = [1, 2, 3, 1]   
Output:   
2

**Explanation:**

3 is greater than it neighbour elements

**Public Test Cases:**

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 1 3 20 4 1 | 2 |
| **2** | 1 2 3 1 | 2 |
| **3** | 10 20 15 2 23 90 67 | 5 |

import java.util.\*;

public class peakElement{

public int findPeakElement(int[] nums){

int left = 0, right = nums.length-1;

while(left < right){

int mid = (left+right)/2;

if(nums[mid]>nums[mid+1]){

right = mid;

}else{

left = mid+1;

}

}

return left;

}

public static void main(String[] args){

Scanner read = new Scanner(System.in);

String[] input = read.nextLine().split(" ");

int[] nums = new int[input.length];

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

nums[i] = Integer.parseInt(input[i]);

}

peakElement sol = new peakElement();

int res = sol.findPeakElement(nums);

System.out.println(res);

}

#### } **2 D List Points**

You are given a 2D list points, where each element points[i] = [xi, yi] represents a point on a 2D plane. The points are sorted in strictly increasing order based on their x coordinates (i.e., xi < xj for all i < j). Additionally, you are given an integer k.

Your task is to find the maximum value of the equation:

`yi + yj + |xi - xj|`

subject to the constraint:

`|xi - xj| ≤ k`

where 1 ≤ i < j ≤ points.length.  
It is guaranteed that at least one valid pair (i, j) exists that satisfies the given constraint.

**Constraints:**

* 2 ≤ points.length ≤ 10^5
* points[i].length == 2
* -10^8 ≤ xi, yi ≤ 10^8
* 0 ≤ k ≤ 2 \* 10^8
* xi < xj for all 1 ≤ i < j ≤ points.length
* xi form a strictly increasing sequence.

**Example:**

**Example 1:**

**Input:**  
points = [[1,3],[2,0],[5,10],[6,-10]], k = 1

**Output:**  
4

**Explanation:**  
• The pair (1,2) satisfies the constraint: ∣1−2∣≤1|1 - 2| \leq 1, and the equation evaluates to 3+0+∣1−2∣=43 + 0 + |1 - 2| = 4.  
• The pair (3,4) also satisfies the constraint: ∣5−6∣≤1|5 - 6| \leq 1, and the equation evaluates to 10+(−10)+∣5−6∣=110 + (-10) + |5 - 6| = 1.  
• The maximum value obtained is 4.

**Example 2:**

**Input:**  
points = [[0,0],[3,0],[9,2]], k = 3

**Output:**  
3

**Explanation:**  
• Only the pair (1,2) satisfies the constraint ∣0−3∣≤3|0 - 3| \leq 3, and the equation evaluates to 0+0+∣0−3∣=30 + 0 + |0 - 3| = 3.  
• The maximum value is 3.

**Explanation:**

NA

**Public Test Cases:**

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 4 1  1 3  2 0  5 10  6 -10 | 4 |
| **2** | 3 2  1 2  2 3  3 4 | 8 |
| **3** | 5 3  1 10  2 5  4 7  5 12  6 4 | 20 |

#include <iostream>

#include <vector>

#include <deque>

#include <climits> // For INT\_MIN

using namespace std;

int findMaxValueOfEquation(vector<pair<int, int>>& points, int k) {

deque<pair<int, int>> dq; // Stores (yi - xi, xi)

int maxVal = INT\_MIN;

for (auto& point : points) {

int xj = point.first, yj = point.second;

// Remove points that do not satisfy |xi - xj| ≤ k

while (!dq.empty() && xj - dq.front().second > k) {

dq.pop\_front();

}

// If deque is not empty, update max equation value

if (!dq.empty()) {

maxVal = max(maxVal, dq.front().first + yj + xj);

}

// Maintain a decreasing order of (yi - xi)

int newVal = yj - xj;

while (!dq.empty() && dq.back().first <= newVal) {

dq.pop\_back();

}

// Add current point to deque

dq.emplace\_back(newVal, xj);

}

return maxVal;

}

int main() {

int n, k;

cin >> n >> k;

vector<pair<int, int>> points(n);

// Read points

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

cin >> points[i].first >> points[i].second;

}

// Compute and print the result

cout << findMaxValueOfEquation(points, k) << endl;

return 0;

}

#### Search elements in array

You are given a rotated sorted array nums containing n elements, which might contain   
duplicates. Your task is to implement a function search(nums, target) that finds the   
index of a given target in the array. If the target is not found, return -1.( Search)   
The array is rotated at an unknown pivot index, and the array can contain duplicates.   
Write an efficient solution with a time complexity of O(log n) or better.   
Input:   
• An integer array nums of length n (1 ≤ n ≤ 10^5), where each element is between -10^4 and 10^4.   
• An integer target (−10^4 ≤ target ≤ 10^4).   
Output:   
• Return the index of the target if it exists in the array, otherwise return -1.

##### Constraints:

You must implement the solution with time complexity O(log n) or better.

##### Example:

Input:   
nums = [2, 5, 6, 0, 0, 1, 2]   
target = 0   
Output:   
3

##### Explanation:

NA

##### Public Test Cases:

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 2 5 6 0 0 1 2  0 | 3 |
| **2** | 6 7 8 2 3  3 | 4 |
| **3** | 11 22 33 44  33 | 2 |

#include <iostream>

#include <vector>

#include <sstream>

using namespace std;

int search(vector<int>& nums, int target) {

int left = 0, right = nums.size() - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (nums[mid] == target) {

return mid;

}

if (nums[left] == nums[mid] && nums[mid] == nums[right]) {

left++;

right--;

continue;

}

if (nums[left] <= nums[mid]) { // Left half is sorted

if (nums[left] <= target && target < nums[mid]) {

right = mid - 1;

} else {

left = mid + 1;

}

} else { // Right half is sorted

if (nums[mid] < target && target <= nums[right]) {

left = mid + 1;

} else {

right = mid - 1;

}

}

}

return -1;

}

int main() {

string input;

getline(cin, input); // Read the array input as a single line

stringstream ss(input);

vector<int> nums;

int num;

while (ss >> num) {

nums.push\_back(num);

}

int target;

cin >> target;

int result = search(nums, target);

cout << result << endl;

return 0;

}

#### Merge Sort

You are given an array of integers. Your task is to sort the array in non  
decreasing order using the Merge Sort algorithm. After sorting, you need to print   
the sorted array.( Sorting)

Input Format:   
• The first line contains an integer n, the size of the array.   
• The second line contains n space-separated integers representing the elements of   
the array.   
   
Output Format:   
• Print a single line containing n space-separated integers, the sorted array in non  
decreasing order.

##### Constraints:

NA

##### Example:

Test Case 1:   
Input:   
5   
3 6 1 8 2   
Output:   
1 2 3 6 8

Test Case 2:   
Input:   
6   
10 4 7 3 9 1   
Output:   
1 3 4 7 9 10

##### Explanation:

Test Case 1:   
   
Input Array: [3, 6, 1, 8, 2]   
   
After applying Merge Sort, the array is divided into smaller subarrays, sorted, and then   
merged back together.   
   
The sorted array is [1, 2, 3, 6, 8].   
   
Test Case 2:   
   
Input Array: [10, 4, 7, 3, 9, 1]   
   
Merge Sort divides the array into smaller subarrays, sorts them, and merges them back.   
   
The sorted array is [1, 3, 4, 7, 9, 10].

##### Public Test Cases:

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 5  3 6 1 8 2 | 1 2 3 6 8 |
| **2** | 6  10 4 7 3 9 1 | 1 3 4 7 9 10 |
| **3** | 4  5 5 5 5 | 5 5 5 5 |

#include <iostream>

using namespace std;

// Merge function to combine two sorted halves

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int leftArray[n1], rightArray[n2];

// Copy data to temporary arrays

for (int i = 0; i < n1; i++) leftArray[i] = arr[left + i];

for (int i = 0; i < n2; i++) rightArray[i] = arr[mid + 1 + i];

int i = 0, j = 0, k = left;

// Merge the temp arrays back into arr[]

while (i < n1 && j < n2) {

if (leftArray[i] <= rightArray[j]) arr[k++] = leftArray[i++];

else arr[k++] = rightArray[j++];

}

// Copy any remaining elements

while (i < n1) arr[k++] = leftArray[i++];

while (j < n2) arr[k++] = rightArray[j++];

}

// Merge Sort function

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

// Recursively sort both halves

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

// Merge the sorted halves

merge(arr, left, mid, right);

}

}

// Utility function to print the array

void printArray(int arr[], int n) {

for (int i = 0; i < n; i++) cout << arr[i] << " ";

cout << endl;

}

int main() {

int n;

cin >> n;

int arr[n];

for (int i = 0; i < n; i++) cin >> arr[i];

mergeSort(arr, 0, n - 1);

printArray(arr, n);

return 0;

#### } The Temple of Sorting: The Puzzle of Magical Stones

In the vibrant land of Arithmoria, a group of adventurers has set off on a perilous   
journey to find the legendary "Puzzle Box" hidden deep within the ancient Temple of   
Sorting. However, the entrance to the temple can only be unlocked by solving a   
complex puzzle involving a sequence of magical stones.( Sorting)   
Each stone in the sequence has a number inscribed on it, and it is believed that the   
sequence holds the key to unlocking the temple door. However, a strange force causes   
the stones to appear jumbled, and the adventurers must reorder the sequence based on   
the following rules:   
1. Group 1: Stones with prime numbers must be placed at the beginning of the   
sequence in ascending order.   
2. Group 2: Stones with numbers that are perfect squares must be placed in the   
middle of the sequence in descending order.   
3. Group 3: All remaining stones must be placed at the end in ascending order.   
4. The relative order of numbers within each group must be preserved.   
The adventurers need your help to reorder the sequence of magical stones and unlock the entrance to the temple. Can you solve the puzzle and help them complete their   
quest?

Input Format:   
• The first line contains an integer n (1 ≤ n ≤ 1000), representing the number of   
magical stones in the sequence.   
• The second line contains n space-separated integers a[i] (-10^5 ≤ a[i] ≤ 10^5),   
representing the numbers on the stones.   
Output Format:   
• Output the reordered sequence of stones in a single line, with elements separated   
by spaces.

##### Constraints:

NA

##### Example:

Input: [3, 8, 5, 16, 9, 7]   
Output: [3, 5, 7, 16, 9, 8]

##### Explanation:

NA

##### Public Test Cases:

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 6  3 8 5 16 9 7 | [3, 5, 7, 16, 9, 8] |
| **2** | 7  7 25 36 9 49 12 8 | [7, 49, 36, 25, 9, 8, 12] |

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

using namespace std;

// Function to check if a number is prime

bool isPrime(int num) {

if (num <= 1) return false;

for (int i = 2; i <= sqrt(num); i++) {

if (num % i == 0) return false;

}

return true;

}

// Function to check if a number is a perfect square

bool isPerfectSquare(int num) {

if (num < 0) return false;

int root = sqrt(num);

return root \* root == num;

}

int main() {

int n;

cin >> n;

vector<int> primes, perfectSquares, others;

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

int num;

cin >> num;

if (isPrime(num)) {

primes.push\_back(num);

} else if (isPerfectSquare(num)) {

perfectSquares.push\_back(num);

} else {

others.push\_back(num);

}

}

// Sort primes in ascending order

sort(primes.begin(), primes.end());

// Sort perfect squares in descending order

sort(perfectSquares.begin(), perfectSquares.end(), greater<int>());

// Sort others in ascending order

sort(others.begin(), others.end());

// Output the reordered sequence

cout << "[";

bool first = true;

for (int num : primes) {

if (!first) cout << ", ";

cout << num;

first = false;

}

for (int num : perfectSquares) {

if (!first) cout << ", ";

cout << num;

first = false;

}

for (int num : others) {

if (!first) cout << ", ";

cout << num;

first = false;

}

cout << "]" << endl;

return 0;

#### } starting and ending position of the target  value

You are given a sorted array nums and a target value. You need to implement a function   
searchRange(nums, target) that returns the starting and ending position of the target   
value. If the target is not found, return [-1, -1]. .( Search)   
Your solution should be implemented in O(log n) time.

Input:   
• An integer array nums of length n (1 ≤ n ≤ 10^5), where each element is between -10^4 and 10^4.   
• An integer target (−10^4 ≤ target ≤ 10^4).   
Output:   
• Return the starting and ending index of the target in the array. If the target is not   
found, return [-1, -1].

##### Constraints:

Your solution should run in O(log n) time.

##### Example:

Input:   
nums = [5, 7, 7, 8, 8, 10]   
target = 8   
Output:   
[3, 4]

##### Explanation:

"Target not found" if the element is not found

##### Public Test Cases:

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 5 7 7 8 8 10  8 | [3, 4] |
| **2** | 1 2 3 3 3 4 5  3 | [2, 4] |
| **3** | 1 2 3 4 5  8 | Target not found |

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

class SearchRange {

public:

vector<int> searchRange(vector<int>& nums, int target) {

int first = findBound(nums, target, true);

if (first == -1) return {-1, -1}; // Target not found

int last = findBound(nums, target, false);

return {first, last};

}

private:

int findBound(vector<int>& nums, int target, bool isFirst) {

int left = 0, right = nums.size() - 1;

int bound = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (nums[mid] == target) {

bound = mid;

if (isFirst) {

right = mid - 1; // Search left for the first occurrence

} else {

left = mid + 1; // Search right for the last occurrence

}

} else if (nums[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return bound;

}

};

int main() {

string input;

getline(cin, input); // Read entire line of input

vector<int> nums;

size\_t pos = 0;

while ((pos = input.find(' ')) != string::npos) {

nums.push\_back(stoi(input.substr(0, pos)));

input.erase(0, pos + 1);

}

nums.push\_back(stoi(input)); // Add the last element

int target;

cin >> target;

SearchRange solution;

vector<int> result = solution.searchRange(nums, target);

if (result[0] == -1) {

cout << "Target not found" << endl;

} else {

cout << "[" << result[0] << ", " << result[1] << "]" << endl;

}

return 0;

}

#### Set of Boxes

You are tasked with delivering a set of boxes from a storage location to various ports using a single ship. The ship has limits on the number of boxes it can carry and the total weight it can hold at once.  
You are given an array of boxes, where each box is represented by an array [port\_i, weight\_i], where port\_i indicates the port where the box needs to be delivered, and weight\_i represents the box's weight. Additionally, you are provided with the following information: (Queue)

* portsCount: The total number of ports.
* maxBoxes: The maximum number of boxes the ship can carry in one trip.
* maxWeight: The maximum weight the ship can carry in one trip.

The boxes must be delivered in the order they appear in the list. The ship will take the following steps:

1. Load a set of boxes onto the ship, ensuring it does not exceed the maxBoxes and maxWeight limits.
2. Deliver each box to its designated port. If the ship is already at the correct port, it can deliver the box directly. Otherwise, it will travel to the port and deliver the box.
3. After delivering the boxes, the ship will return to storage to pick up more boxes.

The goal is to determine the minimum number of trips the ship must make to deliver all the boxes.

##### Constraints:

* 1 <= boxes.length <= 105
* 1 <= portsCount, maxBoxes, maxWeight <= 105
* 1 <= port\_i <= portsCount
* 1 <= weight\_i <= maxWeight

##### Example:

**Example 1:**  
Input: boxes = [[1,1],[2,1],[1,1]], portsCount = 2, maxBoxes = 3, maxWeight = 3

Output: 4

**Example 2:**  
Input: boxes = [[1,2],[3,3],[3,1],[3,1],[2,4]], portsCount = 3, maxBoxes = 3, maxWeight = 6

Output: 6

**Example 3:**  
Input: boxes = [[1,4],[1,2],[2,1],[2,1],[3,2],[3,4]], portsCount = 3, maxBoxes = 6, maxWeight =7  
Output: 6

##### Explanation:

NA

##### Public Test Cases:

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 3 2 3 3  1 1  2 1  1 1 | 4 |
| **2** | 4 3 3 10  1 3  2 5  1 4  3 2 | 6 |
| **3** | 5 4 4 10  1 4  2 3  1 2  3 6  1 1 | 7 |

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

int minTrips(vector<pair<int, int>>& boxes, int portsCount, int maxBoxes, int maxWeight) {

queue<pair<int, int>> shipQueue; // Queue to store boxes on the ship (port, weight)

int trips = 0;

int totalWeight = 0;

int portVisits = 0;

int prevPort = -1;

int i = 0; // Index pointer for boxes

while (i < boxes.size() || !shipQueue.empty()) {

// Load boxes while within maxBoxes and maxWeight limits

while (i < boxes.size() && shipQueue.size() < maxBoxes && totalWeight + boxes[i].second <= maxWeight) {

shipQueue.push(boxes[i]);

totalWeight += boxes[i].second;

// If a new port is encountered, increase the port visit count

if (shipQueue.empty() || boxes[i].first != prevPort) {

portVisits++;

}

prevPort = boxes[i].first;

i++;

}

// One trip to deliver all boxes in the current queue

trips += portVisits + 1; // +1 to return to the storage

portVisits = 0;

totalWeight = 0;

prevPort = -1;

// Empty the queue (deliver all boxes)

while (!shipQueue.empty()) shipQueue.pop();

}

return trips;

}

int main() {

int n, portsCount, maxBoxes, maxWeight;

cin >> n >> portsCount >> maxBoxes >> maxWeight;

vector<pair<int, int>> boxes(n);

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

cin >> boxes[i].first >> boxes[i].second; // Port number and weight

}

cout << minTrips(boxes, portsCount, maxBoxes, maxWeight) << endl;

return 0;

}

#### Basic Calculator Using Stack to Evaluate Arithmetic Expressions

You need to implement a basic calculator that evaluates a mathematical expression   
represented as a string and returns the computed result. The expression consists of   
integers, addition (+), subtraction (-), and parentheses ((, )) to define order of operations.   
The input expression is always valid, meaning: (Stack)   
• Parentheses are properly closed.   
• No consecutive operators appear.   
• Numbers are always non-negative.   
• The unary minus (-) can be used, but unary plus (+) is not allowed.   
You cannot use built-in functions like eval() to directly evaluate the expression.   
Input Format   
• A single string s, representing a valid mathematical expression.   
• The string may contain spaces, which should be ignored during evaluation.   
• The length of s is between 1 and 300,000 characters.   
Output Format   
• A single integer representing the result of evaluating the expression.

##### Constraints:

• 1 ≤ s.length ≤ 3\*10^5   
• s consists of digits (0-9), '+', '-', '(', ')', and spaces (' ').   
• No two operators appear consecutively.   
• + cannot be used as a unary operator.   
• - can be used as a unary operator (e.g., "-1" or "-(2 + 3)").   
• The result fits within a signed 32-bit integer (-2³¹ ≤ result ≤ 2³¹ - 1).   
• The expression is always valid.

##### Example:

Input: "1 + 1"   
Output: 2

##### Explanation:

The expression evaluates to 1 + 1 = 2.

##### Public Test Cases:

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | "1 + 1" | Result: 2 |
| **2** | " 2-1 + 2 " | Result: 3 |
| **3** | "(1+(4+5+2)-3)+(6+8)" | Result: 23 |

#include <iostream>

#include <stack>

#include <cctype> // For isdigit()

using namespace std;

int calculate(string s) {

stack<int> stack;

int num = 0, result = 0, sign = 1;

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

char c = s[i];

if (isdigit(c)) {

num = num \* 10 + (c - '0'); // Build the number

}

else if (c == '+') {

result += sign \* num;

num = 0;

sign = 1; // Reset sign to positive

}

else if (c == '-') {

result += sign \* num;

num = 0;

sign = -1; // Change sign to negative

}

else if (c == '(') {

stack.push(result);

stack.push(sign);

result = 0;

sign = 1; // Reset for the new expression inside ()

}

else if (c == ')') {

result += sign \* num; // Apply last computed number

num = 0;

result \*= stack.top(); // Multiply with stored sign

stack.pop();

result += stack.top(); // Add stored result before '('

stack.pop();

}

}

result += sign \* num; // Add the last number

return result;

}

int main() {

string expression;

cout << "Enter the expression: ";

getline(cin, expression); // Read the entire line as input

cout << "Result: " << calculate(expression) << endl;

return 0;

}

#### "The Enchanted Sorting Spell of Sorteria"

In the mystical land of Sorteria, the great King Algoran has a magical scroll containing   
a list of enchanted numbers. These numbers hold the secrets to unlocking the kingdom’s   
hidden treasures. However, due to an ancient curse, the numbers have been jumbled,   
making it impossible to decipher the scroll.   
The kingdom’s most brilliant wizard, Sortimus, has discovered that the numbers must   
be rearranged in a special way:   
1. All prime numbers must appear first in descending order.   
2. The remaining non-prime numbers must appear afterward in ascending order.   
3. The relative order of prime and non-prime numbers in their respective groups   
must be preserved from the original scroll.   
King Algoran has summoned you, the most skilled programmer in Sorteria, to help   
break the curse and restore order to the scroll. Can you write a spell (a program) that   
will correctly sort the enchanted numbers?   
   
Input Format:   
• The first line contains an integer n (1 ≤ n ≤ 1000), representing the number of   
enchanted numbers on the scroll.   
• The second line contains n space-separated integers a[i] (-10^5 ≤ a[i] ≤ 10^5),   
representing the numbers inscribed on the scroll.   
Output Format:   
• Output the restored sequence of numbers in a single line, with elements separated   
by spaces.

##### Constraints:

NA

##### Example:

**Input:**  
6   
3 8 5 4 2 9   
   
**Output:**  
5 3 2 4 8 9

##### Explanation:

NA

##### Public Test Cases:

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 6  3 8 5 4 2 9 | 5 3 2 4 8 9 |
| **2** | 5  7 6 11 15 19 | 19 11 7 6 15 |
| **3** | 4  16 7 8 9 | 7 8 9 16 |

#### The Enchanted Serpent of LinkedRealm

In the mystical kingdom of LinkedRealm, there exists an Enchanted Serpent, a magical   
creature that takes the form of a circular linked list. This serpent continuously consumes   
magical orbs that appear in the realm. Each orb carries an integer value, and the serpent   
stores these values in its body as nodes of a circular linked list.   
However, the serpent has a peculiar rule: Every Kth orb is cursed, and upon consuming it,   
the serpent removes it from its body. This continues until only one orb remains, which is   
declared the Orb of Destiny.   
As the royal sorcerer, your task is to determine the final remaining orb after following the   
serpent's elimination process.   
   
Input Format:   
• The first line contains two integers N (number of magical orbs) and K (elimination   
interval).   
• The second line contains N space-separated integers representing the values of orbs   
in the initial circular linked list (in insertion order).   
   
Output Format:   
• Print a single integer: the value of the last remaining orb in the linked list.

##### Constraints:

• 1≤N≤10^5   
• 1≤K≤10^5   
• 1≤listvalues<=10^9

##### Example:

**Input:**  
5 2   
10 20 30 40 50

**Output:**  
30

##### Explanation:

Input:   
5 2   
10 20 30 40 50   
Processing:   
• Initial circular list: 10 → 20 → 30 → 40 → 50 → (loops back to 10)   
• Remove 2nd node → 20 is removed → [10, 30, 40, 50]   
• Remove 2nd node → 40 is removed → [10, 30, 50]   
• Remove 2nd node → 10 is removed → [30, 50]   
• Remove 2nd node → 50 is removed → [30]   
• Last remaining orb = 30   
Output:   
30

##### Public Test Cases:

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 5 2  10 20 30 40 50 | 30 |
| **2** | 6 3  5 15 25 35 45 55 | 5 |
| **3** | 10 4  1 2 3 4 5 6 7 8 9 10 | 5 |

#include <iostream>

using namespace std;

struct Node {

int value;

Node\* next;

Node(int val) : value(val), next(nullptr) {}

};

int findLastOrb(int N, int K, int values[]) {

// Create circular linked list

Node\* head = new Node(values[0]);

Node\* prev = head;

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

Node\* node = new Node(values[i]);

prev->next = node;

prev = node;

}

prev->next = head; // Make it circular

// Start elimination process

Node\* current = head;

while (current->next != current) { // Only one node should remain

for (int i = 1; i < K - 1; i++) {

current = current->next;

}

// Remove the K-th node

Node\* toDelete = current->next;

current->next = current->next->next;

delete toDelete; // Free memory

current = current->next; // Move to the next node

}

int lastOrb = current->value;

delete current; // Free the last remaining node

return lastOrb;

}

int main() {

int N, K;

cin >> N >> K;

int values[N];

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

cin >> values[i];

}

cout << findLastOrb(N, K, values) << endl;

return 0;

#### }

#### Integer Threshold

You are given an integer array nums and an integer threshold.

Find any subarray of nums with length k such that every element in the subarray is  
greater than threshold / k. (Stack)

Return the size of any such subarray. If no valid subarray exists, return -1. A subarray is a contiguous sequence of elements within an array.

**Input Format**  
• An integer array nums, where 1 ≤ nums.length ≤ 100,000.  
• An integer threshold, where 1 ≤ threshold ≤ 109.

**Output Format**  
• A single integer representing the size of any valid subarray.  
• If no valid subarray exists, return -1.

##### Constraints:

* 1 ≤ nums.length ≤ 100,000
* 1 ≤ nums[i], threshold ≤ 1,000,000,000
* The solution should be optimized for large inputs.

##### Example:

**Input**: nums = [1,3,4,3,1], threshold = 6

**Output:** 3

##### Explanation:

NA

##### Public Test Cases:

| **#** | **Input** | **Expected Output** |
| --- | --- | --- |
| **1** | 1 3 4 3 1  6 | 3 |
| **2** | 6 5 6 5 8  7 | 1 |

#include <iostream>

#include <vector>

#include <sstream>

using namespace std;

class IntegerThreshold {

public:

int findValidSubarraySize(vector<int>& nums, int threshold) {

int n = nums.size();

for (int k = 1; k <= n; k++) { // Try different subarray lengths

int required = threshold / k;

int count = 0;

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

if (nums[i] > required) {

count++;

} else {

count = 0; // Reset if condition breaks

}

if (count >= k) return k; // Valid subarray found

}

}

return -1; // No valid subarray found

}

};

int main() {

string input;

getline(cin, input);

stringstream ss(input);

vector<int> nums;

int num;

while (ss >> num) {

nums.push\_back(num);

}

int threshold;

cin >> threshold;

IntegerThreshold solution;

int result = solution.findValidSubarraySize(nums, threshold);

cout << result << endl;

return 0;

}