# Problem 2212: Maximum Points in an Archery Competition

## Problem Information

**Difficulty:** Medium
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

Alice and Bob are opponents in an archery competition. The competition has set the following rules:

Alice first shoots

numArrows

arrows and then Bob shoots

numArrows

arrows.

The points are then calculated as follows:

The target has integer scoring sections ranging from

0

to

11

inclusive

.

For

each

section of the target with score

k

(in between

0

to

11

), say Alice and Bob have shot

a

k

and

b

k

arrows on that section respectively. If

a

k

>= b

k

, then Alice takes

k

points. If

a

k

< b

k

, then Bob takes

k

points.

However, if

a

k

== b

k

== 0

, then

nobody

takes

k

points.

For example, if Alice and Bob both shot

2

arrows on the section with score

11

, then Alice takes

11

points. On the other hand, if Alice shot

0

arrows on the section with score

11

and Bob shot

2

arrows on that same section, then Bob takes

11

points.

You are given the integer

numArrows

and an integer array

aliceArrows

of size

12

, which represents the number of arrows Alice shot on each scoring section from

0

to

11

. Now, Bob wants to

maximize

the total number of points he can obtain.

Return

the array

bobArrows

which represents the number of arrows Bob shot on

each

scoring section from

0

to

11

. The sum of the values in

bobArrows

should equal

numArrows

.

If there are multiple ways for Bob to earn the maximum total points, return

any

one of them.

Example 1:

| Scoring Section | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Alice's Arrows | 1 | 1 | - | 1 | - | - | 2 | 1 | - | 1 | 2 | - |
| Bob's Arrows | - | - | - | - | 1 | 1 | - | - | 1 | 2 | 3 | 1 |
| Points Obtained by Bob | - | - | - | - | 4 | 5 | - | - | 8 | 9 | 10 | 11 |

Input:

numArrows = 9, aliceArrows = [1,1,0,1,0,0,2,1,0,1,2,0]

Output:

[0,0,0,0,1,1,0,0,1,2,3,1]

Explanation:

The table above shows how the competition is scored. Bob earns a total point of 4 + 5 + 8 + 9 + 10 + 11 = 47. It can be shown that Bob cannot obtain a score higher than 47 points.

Example 2:

| Scoring Section | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Alice's Arrows | - | - | 1 | - | - | - | - | - | - | - | - | 2 |
| Bob's Arrows | - | - | - | - | - | - | - | - | 1 | 1 | 1 | - |
| Points Obtained by Bob | - | - | - | - | - | - | - | - | 8 | 9 | 10 | - |

Input:

numArrows = 3, aliceArrows = [0,0,1,0,0,0,0,0,0,0,0,2]

Output:

[0,0,0,0,0,0,0,0,1,1,1,0]

Explanation:

The table above shows how the competition is scored. Bob earns a total point of 8 + 9 + 10 = 27. It can be shown that Bob cannot obtain a score higher than 27 points.

Constraints:

1 <= numArrows <= 10

5

aliceArrows.length == bobArrows.length == 12

0 <= aliceArrows[i], bobArrows[i] <= numArrows

sum(aliceArrows[i]) == numArrows

## Code Snippets

**C++:**

```
class Solution {
public:
vector<int> maximumBobPoints(int numArrows, vector<int>& aliceArrows) {

}
```

```
};
```

**Java:**

```
class Solution {
public int[] maximumBobPoints(int numArrows, int[] aliceArrows) {

}
}
```

**Python3:**

```
class Solution:
def maximumBobPoints(self, numArrows: int, aliceArrows: List[int]) ->
List[int]:
```

**Python:**

```
class Solution(object):
def maximumBobPoints(self, numArrows, aliceArrows):
"""
:type numArrows: int
:type aliceArrows: List[int]
:rtype: List[int]
"""
```

**JavaScript:**

```
/**
* @param {number} numArrows
* @param {number[]} aliceArrows
* @return {number[]}
*/
var maximumBobPoints = function(numArrows, aliceArrows) {

};
```

**TypeScript:**

```
function maximumBobPoints(numArrows: number, aliceArrows: number[]): number[]
{
```

```
};
```

**C#:**

```
public class Solution {
public int[] MaximumBobPoints(int numArrows, int[] aliceArrows) {

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maximumBobPoints(int numArrows, int* aliceArrows, int aliceArrowsSize,
int* returnSize) {

}
```

**Go:**

```
func maximumBobPoints(numArrows int, aliceArrows []int) []int {

}
```

**Kotlin:**

```
class Solution {
fun maximumBobPoints(numArrows: Int, aliceArrows: IntArray): IntArray {

}
}
```

**Swift:**

```
class Solution {
func maximumBobPoints(_ numArrows: Int, _ aliceArrows: [Int]) -> [Int] {

}
}
```

**Rust:**

```
impl Solution {
pub fn maximum_bob_points(num_arrows: i32, alice_arrows: Vec<i32>) ->
Vec<i32> {

}
}
```

**Ruby:**

```
# @param {Integer} num_arrows
# @param {Integer[]} alice_arrows
# @return {Integer[]}
def maximum_bob_points(num_arrows, alice_arrows)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer $numArrows
* @param Integer[] $aliceArrows
* @return Integer[]
*/
function maximumBobPoints($numArrows, $aliceArrows) {

}
}
```

**Dart:**

```
class Solution {
List<int> maximumBobPoints(int numArrows, List<int> aliceArrows) {

}
}
```

**Scala:**

```
object Solution {
def maximumBobPoints(numArrows: Int, aliceArrows: Array[Int]): Array[Int] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec maximum_bob_points(num_arrows :: integer, alice_arrows :: [integer]) ::
[integer]
def maximum_bob_points(num_arrows, alice_arrows) do

end
end
```

**Erlang:**

```
-spec maximum_bob_points(NumArrows :: integer(), AliceArrows :: [integer()])
-> [integer()].
maximum_bob_points(NumArrows, AliceArrows) ->
.
```

**Racket:**

```
(define/contract (maximum-bob-points numArrows aliceArrows)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Maximum Points in an Archery Competition
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

class Solution {
public:
vector<int> maximumBobPoints(int numArrows, vector<int>& aliceArrows) {

}
};
```

### Java Solution:

```
/**
* Problem: Maximum Points in an Archery Competition
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[] maximumBobPoints(int numArrows, int[] aliceArrows) {

}
}
```

### Python3 Solution:

```
"""
Problem: Maximum Points in an Archery Competition
Difficulty: Medium
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maximumBobPoints(self, numArrows: int, aliceArrows: List[int]) ->
```

```
List[int]:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def maximumBobPoints(self, numArrows, aliceArrows):
"""
:type numArrows: int
:type aliceArrows: List[int]
:rtype: List[int]
"""
```

### JavaScript Solution:

```
/**
* Problem: Maximum Points in an Archery Competition
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} numArrows
* @param {number[]} aliceArrows
* @return {number[]}
*/
var maximumBobPoints = function(numArrows, aliceArrows) {

};
```

### TypeScript Solution:

```
/**
* Problem: Maximum Points in an Archery Competition
* Difficulty: Medium
* Tags: array
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function maximumBobPoints(numArrows: number, aliceArrows: number[]): number[]
{

};
```

### C# Solution:

```
/*
* Problem: Maximum Points in an Archery Competition
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[] MaximumBobPoints(int numArrows, int[] aliceArrows) {

}
}
```

### C Solution:

```
/*
* Problem: Maximum Points in an Archery Competition
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
```

```
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maximumBobPoints(int numArrows, int* aliceArrows, int aliceArrowsSize,
int* returnSize) {

}
```

**Go Solution:**

```
// Problem: Maximum Points in an Archery Competition
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maximumBobPoints(numArrows int, aliceArrows []int) []int {

}
```

**Kotlin Solution:**

```
class Solution {
fun maximumBobPoints(numArrows: Int, aliceArrows: IntArray): IntArray {

}
}
```

**Swift Solution:**

```
class Solution {
func maximumBobPoints(_ numArrows: Int, _ aliceArrows: [Int]) -> [Int] {

}
}
```

**Rust Solution:**

```
// Problem: Maximum Points in an Archery Competition
// Difficulty: Medium
```

```
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn maximum_bob_points(num_arrows: i32, alice_arrows: Vec<i32>) ->
Vec<i32> {

}
}
```

### Ruby Solution:

```
# @param {Integer} num_arrows
# @param {Integer[]} alice_arrows
# @return {Integer[]}
def maximum_bob_points(num_arrows, alice_arrows)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer $numArrows
* @param Integer[] $aliceArrows
* @return Integer[]
*/
function maximumBobPoints($numArrows, $aliceArrows) {

}
}
```

### Dart Solution:

```
class Solution {
List<int> maximumBobPoints(int numArrows, List<int> aliceArrows) {
```

```
}
}
```

### Scala Solution:

```
object Solution {
def maximumBobPoints(numArrows: Int, aliceArrows: Array[Int]): Array[Int] = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec maximum_bob_points(num_arrows :: integer, alice_arrows :: [integer]) ::
[integer]
def maximum_bob_points(num_arrows, alice_arrows) do

end
end
```

### Erlang Solution:

```
-spec maximum_bob_points(NumArrows :: integer(), AliceArrows :: [integer()])
-> [integer()].
maximum_bob_points(NumArrows, AliceArrows) ->
.
```

### Racket Solution:

```
(define/contract (maximum-bob-points numArrows aliceArrows)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?))
)
```