# Problem 1383: Maximum Performance of a Team

## Problem Information

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

## Problem Description

You are given two integers

n

and

k

and two integer arrays

speed

and

efficiency

both of length

n

. There are

n

engineers numbered from

1

to

n

.

speed[i]

and

efficiency[i]

represent the speed and efficiency of the

i

th

engineer respectively.

Choose

at most

k

different engineers out of the

n

engineers to form a team with the maximum

performance

.

The performance of a team is the sum of its engineers' speeds multiplied by the minimum efficiency among its engineers.

Return

the maximum performance of this team

. Since the answer can be a huge number, return it

modulo

10

9

+ 7

.

Example 1:

Input:

n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 2

Output:

60

Explanation:

We have the maximum performance of the team by selecting engineer 2 (with speed=10 and efficiency=4) and engineer 5 (with speed=5 and efficiency=7). That is, performance = (10 + 5) * min(4, 7) = 60.

Example 2:

Input:

n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 3

Output:

68

Explanation:

This is the same example as the first but k = 3. We can select engineer 1, engineer 2 and engineer 5 to get the maximum performance of the team. That is, performance = (2 + 10 + 5) * min(5, 4, 7) = 68.

Example 3:

Input:

n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 4

Output:

72

Constraints:

1 <= k <= n <= 10

5

speed.length == n

efficiency.length == n

1 <= speed[i] <= 10

5

1 <= efficiency[i] <= 10

8

## Code Snippets

**C++:**

```
class Solution {
public:
int maxPerformance(int n, vector<int>& speed, vector<int>& efficiency, int k)
{

}
};
```

**Java:**

```
class Solution {
public int maxPerformance(int n, int[] speed, int[] efficiency, int k) {

}
}
```

**Python3:**

```
class Solution:
def maxPerformance(self, n: int, speed: List[int], efficiency: List[int], k:
int) -> int:
```

**Python:**

```
class Solution(object):
def maxPerformance(self, n, speed, efficiency, k):
"""
:type n: int
:type speed: List[int]
:type efficiency: List[int]
:type k: int
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {number} n
* @param {number[]} speed
* @param {number[]} efficiency
* @param {number} k
* @return {number}
*/
var maxPerformance = function(n, speed, efficiency, k) {

};
```

**TypeScript:**

```
function maxPerformance(n: number, speed: number[], efficiency: number[], k:
number): number {

};
```

**C#:**

```
public class Solution {
public int MaxPerformance(int n, int[] speed, int[] efficiency, int k) {

}
}
```

**C:**

```
int maxPerformance(int n, int* speed, int speedSize, int* efficiency, int
efficiencySize, int k) {

}
```

**Go:**

```
func maxPerformance(n int, speed []int, efficiency []int, k int) int {

}
```

**Kotlin:**

```
class Solution {
fun maxPerformance(n: Int, speed: IntArray, efficiency: IntArray, k: Int):
```

```
Int {

}
}
```

**Swift:**

```
class Solution {
func maxPerformance(_ n: Int, _ speed: [Int], _ efficiency: [Int], _ k: Int)
-> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn max_performance(n: i32, speed: Vec<i32>, efficiency: Vec<i32>, k: i32)
-> i32 {

}
}
```

**Ruby:**

```
# @param {Integer} n
# @param {Integer[]} speed
# @param {Integer[]} efficiency
# @param {Integer} k
# @return {Integer}
def max_performance(n, speed, efficiency, k)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $speed
* @param Integer[] $efficiency
* @param Integer $k
```

```
* @return Integer
*/
function maxPerformance($n, $speed, $efficiency, $k) {

}
}
```

**Dart:**

```
class Solution {
int maxPerformance(int n, List<int> speed, List<int> efficiency, int k) {

}
}
```

**Scala:**

```
object Solution {
def maxPerformance(n: Int, speed: Array[Int], efficiency: Array[Int], k:
Int): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec max_performance(n :: integer, speed :: [integer], efficiency ::
[integer], k :: integer) :: integer
def max_performance(n, speed, efficiency, k) do

end
end
```

**Erlang:**

```
-spec max_performance(N :: integer(), Speed :: [integer()], Efficiency ::
[integer()], K :: integer()) -> integer().
max_performance(N, Speed, Efficiency, K) ->
.
```

**Racket:**

```
(define/contract (max-performance n speed efficiency k)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?)
exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Maximum Performance of a Team
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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 maxPerformance(int n, vector<int>& speed, vector<int>& efficiency, int k)
{

}
};
```

### Java Solution:

```
/**
* Problem: Maximum Performance of a Team
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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 maxPerformance(int n, int[] speed, int[] efficiency, int k) {
```

```
}
}
```

### Python3 Solution:

```
"""
Problem: Maximum Performance of a Team
Difficulty: Hard
Tags: array, greedy, sort, queue, heap

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 maxPerformance(self, n: int, speed: List[int], efficiency: List[int], k:
int) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def maxPerformance(self, n, speed, efficiency, k):
"""
:type n: int
:type speed: List[int]
:type efficiency: List[int]
:type k: int
:rtype: int
"""
```

### JavaScript Solution:

```
/**
* Problem: Maximum Performance of a Team
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
```

```
* 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} n
* @param {number[]} speed
* @param {number[]} efficiency
* @param {number} k
* @return {number}
*/
var maxPerformance = function(n, speed, efficiency, k) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Maximum Performance of a Team
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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 maxPerformance(n: number, speed: number[], efficiency: number[], k:
number): number {

};
```

**C# Solution:**

```
/*
* Problem: Maximum Performance of a Team
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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 MaxPerformance(int n, int[] speed, int[] efficiency, int k) {

}
}
```

### C Solution:

```
/*
* Problem: Maximum Performance of a Team
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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
*/

int maxPerformance(int n, int* speed, int speedSize, int* efficiency, int
efficiencySize, int k) {

}
```

### Go Solution:

```
// Problem: Maximum Performance of a Team
// Difficulty: Hard
// Tags: array, greedy, sort, queue, heap
//
// 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 maxPerformance(n int, speed []int, efficiency []int, k int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun maxPerformance(n: Int, speed: IntArray, efficiency: IntArray, k: Int):
Int {

}
}
```

### Swift Solution:

```
class Solution {
func maxPerformance(_ n: Int, _ speed: [Int], _ efficiency: [Int], _ k: Int)
-> Int {

}
}
```

### Rust Solution:

```
// Problem: Maximum Performance of a Team
// Difficulty: Hard
// Tags: array, greedy, sort, queue, heap
//
// 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 max_performance(n: i32, speed: Vec<i32>, efficiency: Vec<i32>, k: i32)
-> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} speed
# @param {Integer[]} efficiency
# @param {Integer} k
# @return {Integer}
```

```
def max_performance(n, speed, efficiency, k)

end
```

**PHP Solution:**

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $speed
* @param Integer[] $efficiency
* @param Integer $k
* @return Integer
*/
function maxPerformance($n, $speed, $efficiency, $k) {

}
}
```

**Dart Solution:**

```
class Solution {
int maxPerformance(int n, List<int> speed, List<int> efficiency, int k) {

}
}
```

**Scala Solution:**

```
object Solution {
def maxPerformance(n: Int, speed: Array[Int], efficiency: Array[Int], k:
Int): Int = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec max_performance(n :: integer, speed :: [integer], efficiency ::
```

```
[integer], k :: integer) :: integer
def max_performance(n, speed, efficiency, k) do

end
end
```

**Erlang Solution:**

```
-spec max_performance(N :: integer(), Speed :: [integer()], Efficiency ::
[integer()], K :: integer()) -> integer().
max_performance(N, Speed, Efficiency, K) ->
.
```

**Racket Solution:**

```
(define/contract (max-performance n speed efficiency k)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?)
exact-integer? exact-integer?)
)
```