Task Scheduling II

Description

Suppose you are given a set $S = \{a_1, a_2, \ldots, a_n\}$ of tasks, where task a_i requires p_i units of processing time to complete. You have one computer with a single CPU on which to run these tasks. Suppose also that the tasks are not all available at once. That is, each task has a ***release time*** r_i before which it is not available to be processed. In addition, ***preemption*** is allowed, so that a task can be suspended and restarted at a later time. For example, a task a_i with processing time $p_i = 6$ may start running at time 1 and be preempted at time 4. It can then resume at time 10 but be preempted at time 11 and finally resume at time 13 and complete at time 15. Task *ai* has run for a total of 6-time units, but its running time has been divided into three pieces. We say that the completion time of a_i is 15.

Given the ***N*** tasks, design an efficient solution to schedule them to **minimize** the average completion time of these processes.

Complexity

complexity of your algorithm should be **less than $O(N^2)$**

Function: **Implement it!**

```
public static double RequiredFunction(int[] r, int[] p)
```

Where r is release times array and p is processing times array.

`PROBLEM_CLASS.cs` includes this method.

Note: you must round your final answer value to the nearest hundredth (e.g., 6.47, 9.21, or 5.61).

Example

Example 1:

Consider three tasks:

- Task 1: Release time = 0, Processing time = 5.
- Task 2: Release time = 2, Processing time = 2.
- Task 3: Release time = 4, Processing time = 3.

Expected Inputs:

r = [0, 4, 2]

p = [5, 3, 2]

Expected Output:

7

Step-by-step explanation:

1) Time 0:

Only Task 1 is available.

Start processing Task 1.

2) Time 2:

Task 3 is released.

At this moment, Task 1 has already been processed for 2 units (3 remaining).

Task 1 is preempted and Task 3 is started.

3) Time 4:

Task 3 completes at time 4.

Task 2 is released.

Task 1 is resumed.

4) Time 7:

Task 1 completes at time 7.

Task 2 is started.

5) Time 10:

Task 2 completes at 10.

Completion times:

Task 3: 4.

Task 1: 7.

Task 2: 10.

The average completion time is (4 + 7 + 10) / 3 = 7.

Example 2: Multiple Preemptions

Tasks:

- Task 1: Release time = 0, Processing time = 10.
- Task 2: Release time = 2, Processing time = 2.
- Task 3: Release time = 3, Processing time = 1.
- Task 4: Release time = 5, Processing time = 3.

Expected Inputs:

r = [0, 2, 3, 5]

p = [10, 2, 1, 3]

Expected Output:

8.25

Step-by-step explanation:

1) Time 0:

Only Task 1 is available; start Task 1.

2) Time 2:

Task 2 is released. At this point, Task 1 has run for 2 units (remaining = 8).
Task 1 is preempted and Task 2 is started.

3) Time 3:

While Task 2 is running, Task 3 is released at time 3.
Task 2 will resume.

4) Time 4:

Task 2 completes at 4.
Task 1 (remaining = 8).
Task 3 (processing time = 1).
Task 3 is started.

5) Time 5:

Task 3 completes at 5.
Task 4 is released (processing time = 3) and started.

Time 5 to 8:
Run **Task 4** until completion (**finishes at time 8**).

6) Time 8:

Task 4 completes at 8.
Task 1 is resumed.

7) Time 16:

Task 4 completes at 16.

Completion Times:

Task 2: 4.
Task 3: 5.
Task 4: 8.
Task 1: 16.
Average Completion Time: (4 + 5 + 8 + 16) / 4 = 8.25.

Example 3: Tasks with Equal Processing Times

Tasks:

- Task A: Release time = 0, Processing time = 4
- Task B: Release time = 1, Processing time = 4
- Task C: Release time = 2, Processing time = 4
- Task D: Release time = 3, Processing time = 4

Expected Inputs:

r = [0, 1, 2, 3]

p = [4, 4, 4, 4]

Expected Output:

10

Completion Times:

Task A: 4.

Task B: 8.

Task C: 12.

Task D: 16.

Average Completion Time: (4 + 8 + 12 + 16) / 4 = 10.

C# Help

Getting the size of 1D array

```
int size = array1D.GetLength(0.
```

Getting the size of 2D array

```
int size1 = array2D.GetLength(0.

int size2 = array2D.GetLength(1.
```

Creating 1D array

```
int [] array1D = new int [size]
```

Creating 2D array

```
int [,] array2D = new int [size1, size2]
```

Sorting single array

Sort the given array "items" in ascending order

```
Array.Sort(items).
```

Sorting parallel arrays

Sort the first array "master" and re-order the 2nd array "slave" according to this sorting

```
Array.Sort(master, slave).
```