Optimal Task Assignment

In this lesson, you will learn how to assign tasks optimally to a set of workers in Python.

WE'LL COVER THE FOLLOWING

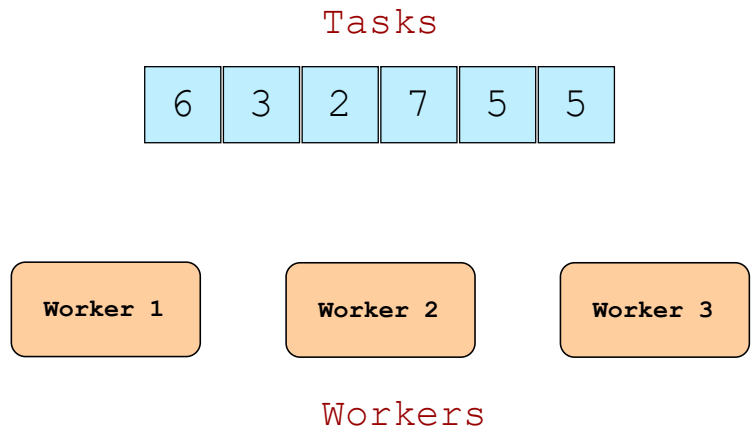
In this lesson, we will be solving the following problem:

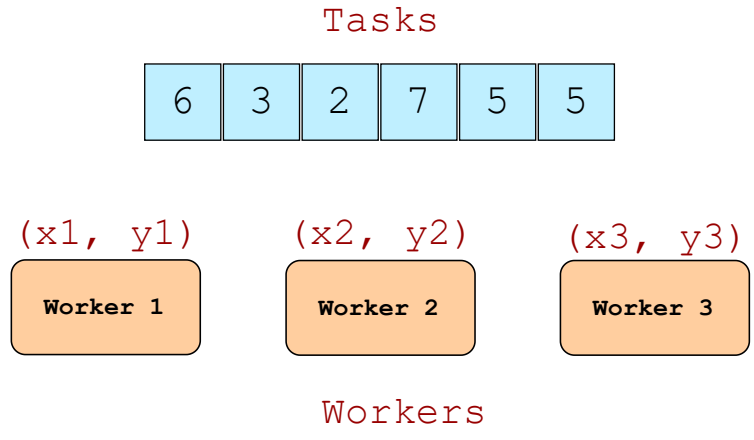
Assign tasks to workers so that the time it takes to complete all the tasks is minimized given a count of workers and an array where each element indicates the duration of a task.

We wish to determine the optimal way in which to assign tasks to some workers. Each worker must work on exactly *two* tasks. Tasks are independent of each other, and each task takes a certain amount of time.

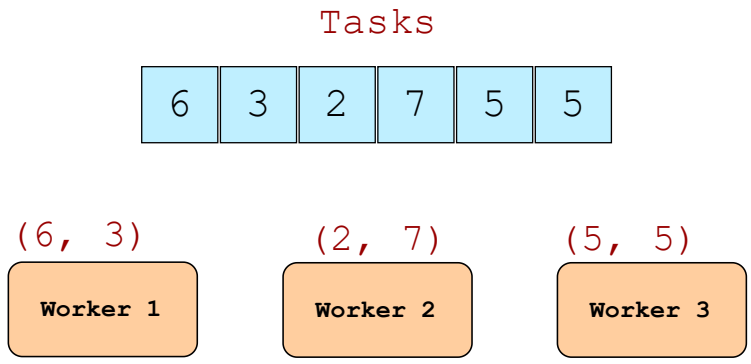
In the slides below, we have been given an array of tasks where the value of each element in the array corresponds to the number of hours required for each task.

Check out the slides below where an example of assigning tasks optimally to workers has been illustrated:

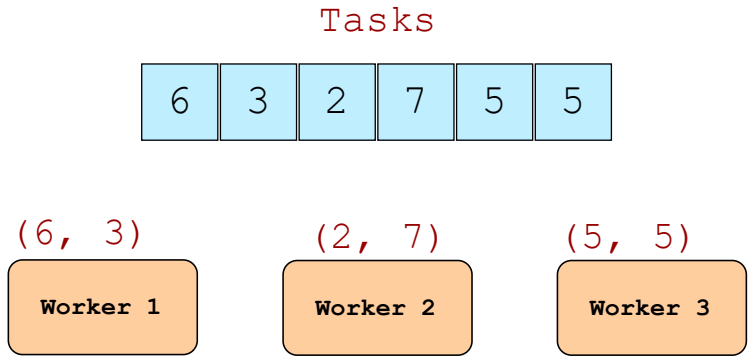




Each worker must be assigned two tasks.

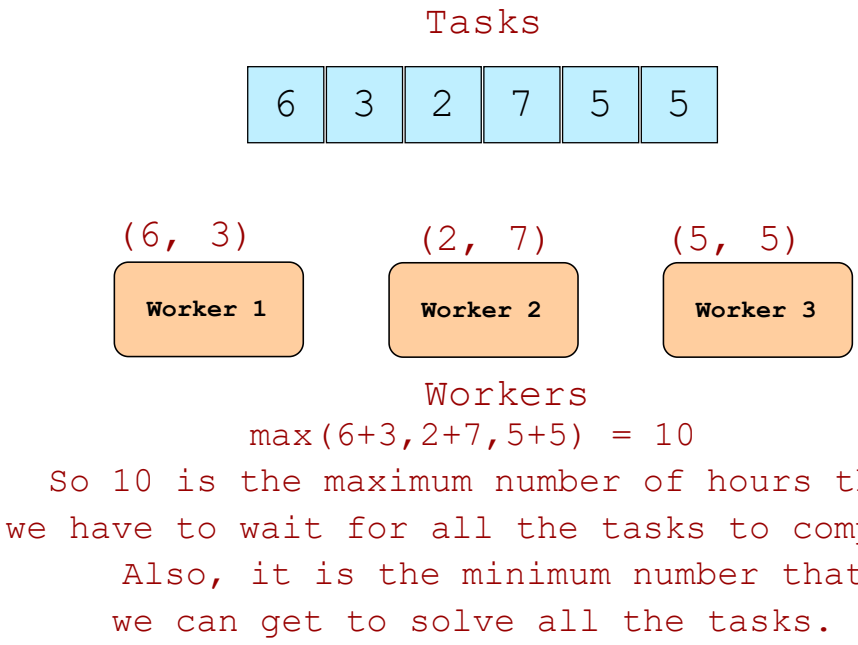


Tasks have been assigned to the workers.



Tasks have been assigned to the workers.

max(6+3,2+7,5+5) = 10



5 of 5

Now let's think of a general approach. From the above example, we know that we have to make pairs. If we generate all possible pairs, it would not be efficient as enumerating every possible pair would require generating $\frac{n(n-1)}{2}$ pairs where n is the number of tasks in the given array.

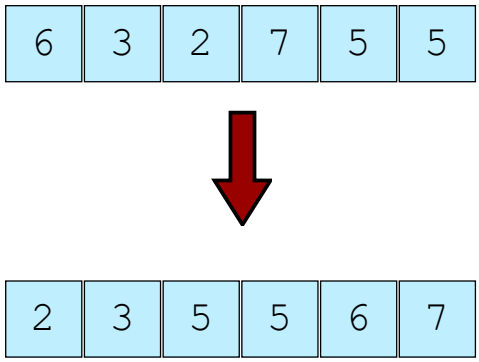
Therefore, we are going to make use of a different approach, i.e., the greedy approach.

In the greedy approach, we'll focus on the following rule:

- Pair the longest task with the shortest one.

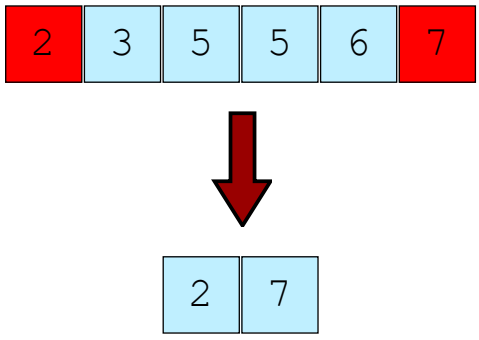
Let's find out how to do this in the slides below.

Tasks



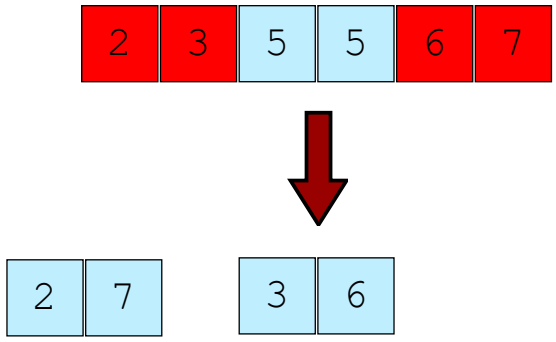
Sorted Tasks

Sorted Tasks



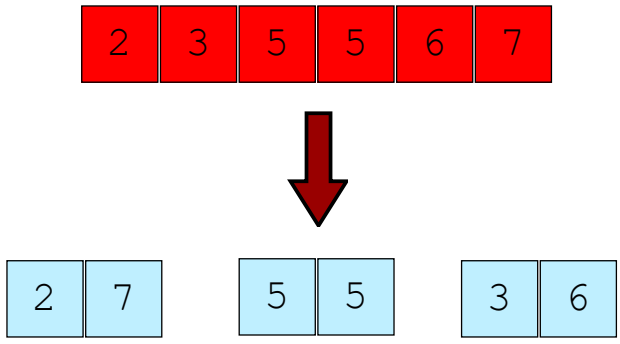
Pick the shortest and the longest one.

Sorted Tasks

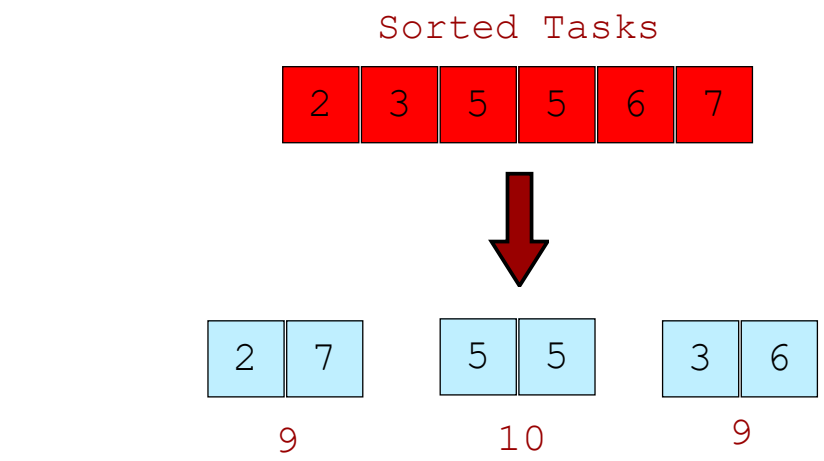


Pick the shortest and the longest one.

Sorted Tasks



Pick the shortest and the longest one.



5 of 5

The time complexity for the algorithm depicted in the slides above will $O(nlogn)$ due to sorting.

Implementation

Now that you are familiar with the algorithm, let's jump to the code in Python:

```
A = [6, 3, 2, 7, 5, 5]

A = sorted(A)

for i in range(len(A)//2):
    print(A[i], A[~i])
```

Explanation

Yes, the implementation was that simple! After sorting the array using `sorted` on **line 3**, the `for` loop on **line 5** iterates for half the length of the array and

prints out the pairs using indexing. So `~i` on **line 6** is the bitwise complement operator which puts a negative sign in front of `i`. Thus, the negative numbers as indexes mean that you count from the right of the array instead of the left. So, `A[-1]` refers to the last element, `A[-2]` is the second-last, and so on. In this way, we are able to pair the numbers from the beginning of the array to the end of the array.

That pretty much does it for this lesson! I hope everything's clear so far. In the next lesson, we'll have a look at a problem involving sorted arrays. See you there!