Problem 4.

General Idea: Sort the tasks according to their required time.
And the order should be the sorted order (increasingly).
Considering the time complexity, apply quick-sort to sort.

Algorithm:

Input: tasks with require time of each task

Output: the order that minimize the total time until each task finished.

return quick-sort (tasks, required time).

Algorithm analysis: For it is a quick-sort, time complexity is $O(n \log n)$.

Let ~~us~~ then prove. our order $w_1 w_2 \cdots w_n$ is the optimal order.

Assume w_i, w_j are two random tasks in our order, where $1 \le i < j < n$.

Because we have sorted tasks in increasing order, so $t_i \le t_j$.

Let swap w_j, w_j to make a new order.

The total time $T_{original} = t_1 + (t_1+t_2) + \cdots + (t_1+t_2+\cdots t_i) + \cdots + (t_1+t_2+\cdots+t_j) + (t_1+t_2+\cdots t_n)$

The new time $T_{new} = t_1 + (t_1+t_2) + \cdots + (t_1+t_2+\cdots+t_j) + \cdots + (t_1+t_2+\cdots+t_i) + (t_1+t_2+\cdots+t_n)$

$T_{new} - T_{original} = (j-i)(t_j - t_i) \ge 0.$

Thus. $T_{new} \ge T_{original}$. We prove that one change of two tasks will always make the $T_{new} \ge T_{original}$. Thus no matter how many tasks will change. T_{new} will always greater than or equal to $T_{original}$. So the original order is the opitimal one.