HOMEWORK 3

COMP3121 - ALGORITHM DESIGN

QUESTION 4

PUNEETH KAMBHAMPATI

z5164647

SOLUTION

```
# profits and deadlines are dictionaries mapping tasks to their
# respective profits and deadlines
def schedule(profits, deadlines):
    # we need an array to store the order of the tasks
    timeSlots = len(profits)

    sortedProfits = MergeSort(profits, descending=TRUE)

    # traverse the list of tasks in descending order of profits
    for task in sortedProfits:
        # check for latest available time slot before deadline
        for slot in range(deadlines[task]-1 , 0):
            if slot is empty:
                timeSlots[slot] = task

    return timeSlots
```

EXPLANATION

The primary goal for this problem would be to ensure the completion of the highest profit activities.

If two activities with the same deadline had different profits an optimal solution would be to ensure that the higher profit activity is given a time slot first and then to pick from the leftover slots for the second activity. This solution to the sub problem would yield an optimal solution when applied for the entire problem.

So the first priority for the solution will be to sort the tasks in descending order of priority. This part of the solution can be done in O(n*logn) using a sorting algorithm such as Merge Sort.

Technically, for a given set of N tasks, there would be N different possible time slots to fit all the tasks. As we go down the descending order of Profit list, the best time slot for a given task would be 1 unit before the deadline. As this will ensure that we are picking the highest possible profit task for that time slot and also not waste time slots that could be used for tasks with earlier deadlines. So we look for the first available time slot closest to the deadline.

In the worst case scenario, we would have to have O(n^2) comparisons to insert a given task in the timeline. This worst case could be if all tasks had the same deadline and each task would progressively require more comparisons and give us a sum of initial n integers.

Hence the overall complexity is,

$$O(n\log n) + O(n^2) = O(n^2)$$

This way, we can ensure that higher profit tasks take priority over lower tasks whilst not wasting any time slots.