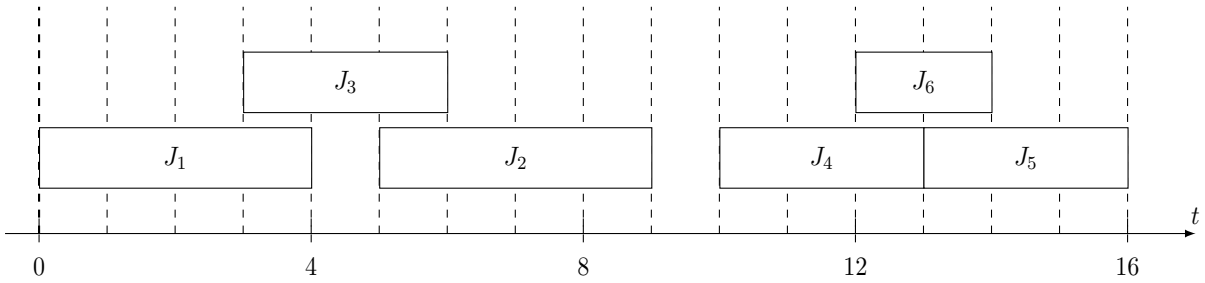
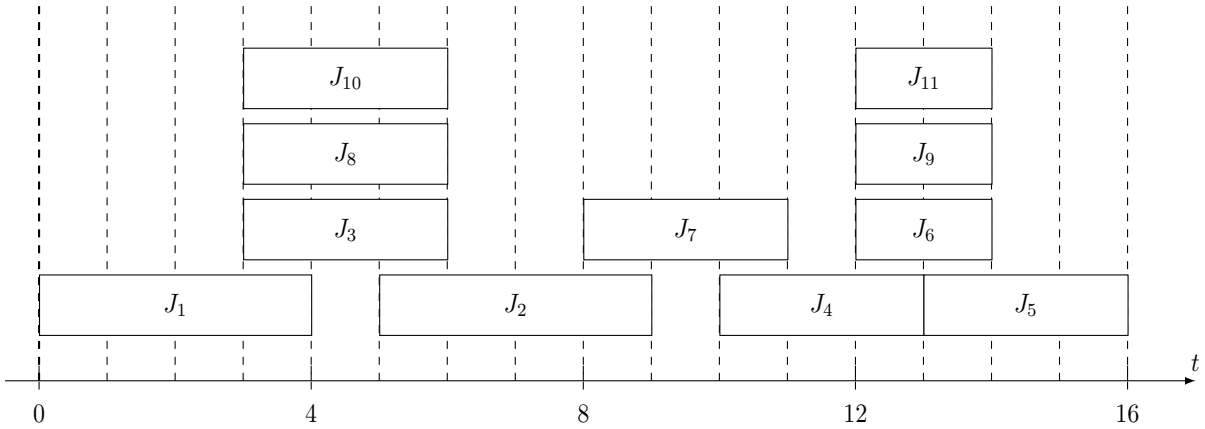
# HW7 Solutions

1. (a) In the following problem, the proposed greedy algorithm gives $\{J_3, J_6\}$ but the optimal scheduling should return $\{J_1, J_2, J_4, J_5\}$.




(b) In the following problem, the proposed algorithm will give $\{J_1, J_7, J_5\}$, but the optimal one should give $\{J_1, J_2, J_3, J_4\}$.




2. The following greedy algorithm solves the problem.

   (a) Start with zero processors.

   (b) Sort all jobs based on their start times.

   (c) Linearly scan the sorted jobs, and for each job:

      i. If there is no idle processor, assign a new one.

      ii. If a processor is idle because some previous job has finished, assign that one.

**Complexity:**

Sorting is $O(n \log n)$. For each job, we only need to see whether idle processors exist, which can be complete in $O(1)$ time if the id of idle processor is stored in a queue, and this is repeated for all jobs, therefore $O(n)$. Total complexity is therefore $O(n \log n + n) = O(n \log n)$.

**Correctness: Proof 1**

Let's suppose the above algorithm does not return an optimum solution. Then, we must have made a mistake at some step. Let's look at the first mistake we made in the algorithm, say the $i$-th step, and by changing the decision at the $i$-th step we are expecting to reduce the number of processors since our choice was a mistake.

(a) If there was no idle processor at the $i$-th step and we assigned the processor $j$: since assigning the processor $j$ was a mistake, we should have either assigned an existing processor or another processor appeared later, say processor $k$. It was impossible to not assign a new processor since all were occupied, so the only possibility is that we should have assigned the processor $k$. But then, since all the processors are identical, changing the assignment of processor from $j$ to $k$ does not substantially change what happens later.

(b) If there was an idle processor at the $i$-th step and we assigned that processor, say $j$: since assigning the processor $j$ was a mistake, we should have either assigned a new processor or another idle processor, say $k$. Assigning another idle processor $k$ does not make any difference as the previous case, then the only possibility is that we should have assigned a new processor at this step. However, if we assign a new processor, this cannot reduce the number of processors.

In conclusion, it is not possible to reduce the number of processor by changing the decision at the 'first mistake', which means our initial assumption was wrong, and there is no 'first mistake' in our algorithm, *i.e.,* our algorithm is correct.

**Correctness: Proof 2**

There is another more succinct and cleaner proof. It is easy to show that the lower bound of the required processors is the 'maximum overlap size', where the overlap size stands for the number of jobs that are overlapped. Then, the only thing we need to show is that the number of processors used in our algorithm is same as the maximum overlap size.

It is easy to show that (I'm omitting it, but you need to show this) in any time of the algorithm, the number of used processors is exactly equal to the number of currently overlapped jobs. Then, throughout the algorithm, we only need as many processors as the maximum overlap size, which is already shown to be the lower bound of the processor numbers.

Since the number of processors we are using is the lower bound already, this must be an optimum solution.

3. We choose our first box to be placed at location $l_1 + d$. This will cover all houses in the range $l_1 \cdots l_1 + 2 * d$. For every subsequent house, at location $l_i$, we check if $l_i \leq l_1 + 2d$ and mark the house as covered. If the house is not covered, (say) house at $l_j$, we place the next service box at distance $l_j + d$, and thus, end up covering all houses within distance $l_j + 2d$, and proceed to mark all houses within range $l_j \cdots l_j + 2d$ as marked. When all houses, have been marked, we have found a positioning for all the distribution boxes.
Correctness: Induction on the number of boxes: Consider the positioning for the first box. The best way to position it must be at $l_1 + d$ (and not less), otherwise, the range of houses that can be covered from this box will be less than $l_1 + 2d$, and we can extend the range by moving the box to position $l_1 + 2d$. Therefore, the best way of positioning box 1 corresponds to finding the best distribution of boxes for the remaining set obtained by removing the houses covered by box 1. Similarly, for the 2nd box, we consider the first house $l_j$ which is not covered by the first box, and should position it at $l_j + d$, or by the same reasoning as above, argue that any other positioning can be improved by moving the position of the second box to $l_j + d$. Therefore, placing box 1 and box 2 in the best way corresponds to optimally distributing boxes in the set of houses that are not yet covered. Hence , we can state that we must always place a box at a distance d further than the position of the uncovered house. This proves, that the above method of distribution is optimal.

4. Consider a tree representation $T_b$ of all the 8-bit characters such that we have a complete binary tree of height $2^8$, and the tree $T_h$ which is obtained by Huffman encoding of the characters in terms of probability of occurrence. The cost of a tree is given by $\sum_{i \in character-set} L_i * P_i$ *where* $L_i$ *and* $P_i$

is the length of the path from root to character leaf and the probability of the character occurring respectively. In $T_h$, consider the characters with the lowest probabilities $P_1$ and $P_2$ and the character with the highest probability $P_n$.
Suppose the leaves of $T_h$ are not at the same height, then we know that $P_n$ is at depth less than $P_1$ and $P_2$ (Huffman Coding Definition). In this case, modify $T_h$ by replacing $P_1$ and $P_2$ and their parent with $P_1$ and by replacing $P_n$ with a subtree with 2 child leaves $P_2$ and $P_n$. In short, we have moved $P_2$ from the lowest level, and make it the sibling of $P_n$, let's call this modified tree $T_{prime}$.
We now analyze the cost difference between the two trees $T_h$ and $T_{prime}$.:

$$cost(T_h) - cost(T_{prime}) = -(P_1 + P_2) + P_n$$

as it is the change (decrease) in cost by removing $P_2$ from the lowest level and reducing height of $P_1$ by 1 + increase in cost by increasing level of $P_n$ by 1
But we know that $P_n < 2 * P_1$ *and* $P_2 \geq P_1$, $\therefore P_1 + P_2 \geq 2P_1 \geq P_n$
Hence, the cost difference between $T_h$ and $T_{prime}$ is non-positive, implying that by reducing the height of the tree by moving characters, we do not decrease the efficiency of character representation.