You're expected to work on the discussion problems before coming to the lab. Discussion session is not meant to be a lecture. TA will guide the discussion and correct your solutions if needed. We will not release 'official' solutions. If you're better prepared for discussion, you will learn more. TAs will record names of the students who actively engage in discussion and report them to the instructor; they are also allowed to give some extra points to those students at their discretion. The instructor will factor in participation in final grade.

1. (Intermediate) In the Interval Selection (Activity Selection) problem, we're given $n$ interval $I_i = (s_i, f_i)$, $1 \leq i \leq n$ and would like to choose the maximum number of mutually disjoint/non-overlapping intervals. Explain why the greedy algorithm that recursively chooses the earliest ending interval (and discards overlapping intervals) is optimal.

   (a) Why is the following claim true? Claim: There is an optimal solution that includes $I_1$ which ends the earliest.

   (b) Explain why the claim implies the earliest finishing algorithm is optimal.

2. (Basic) What is fixed-length code? What is prefix (or prefix-free) code? Is a fixed-length code always prefix-free? Why do you need prefix-freeness?

3. (Basic) Assume the following when executing the Huffman algorithm: When combining two trees, the tree with lowest root frequency becomes the left child and the tree with the second-lowest root frequency becomes the right child. Left children are associated with the bit 0, right children with the bit 1.Run the Huffman's algorithm on the following input: $a : 3$, $b : 2$, $c : 4$, $d : 8$, $e : 7$, $f : 14$. What is the codeword for each character? Give a tree representation.

4. (Basic) Repeat the previous question on the input, $a : 1$, $b : 2$, $c : 4$, $d : 5$, $e : 6$, $f : 9$.

5. (Basic) Explain why a prefix-free code is not optimal if some node has exactly one child in the tree representation.

6. (Advanced) Single Machine Scheduling + Completion Time Minimization. We are given $n$ jobs, $1, 2, ..., n$ with sizes $p_1, p_2, ...p_n$. To complete job $i$ we need to process job $i$ for $p_i$ units of time. We have a single processor and can start processing jobs from time 0. At each time, we can process at most one job. Our goal is to find a schedule that minimizes the total completion time. Show an optimal algorithm and prove the optimality. For simplicity, you may assume that jobs must be scheduled non-preemptively, meaning that once you start processing a job, you have to finish it before you process other jobs.

   Example. Say that we have two jobs with sizes $p_1 = 1$ and $p_2 = 2$. If you process job 1 first and then job 2, you can complete jobs 1 and 2 at times 1 and 3, resp., thus having total completion time 4. On the other hand, if you finish job 2 and then job 1, then you'll complete jobs 1 and 2 at times, 3 and 2, resp., thus having total completion time 5. So the former schedule is better.

7. (Advanced) Single Machine Scheduling Meeting Jobs Deadlines. As before, we have $n$ jobs, $1, 2, ..., n$ with sizes $p_1, p_2, ...p_n$. Here each job $j$ also has a deadline $d_j$, meaning that we are required to complete the job by time $d_j$. We want to check if there is a schedule that completes all jobs by their respective deadlines. Show that scheduling the job with the earliest deadline (a.k.a. Earliest Deadline First (EDF)) finds such a schedule if it exists. As before, the scheduler can start processing any job from time 0.

   * Note: This holds true even if jobs arrive over time. But for simplicity, let's assume that all jobs are available for schedule from the beginning, time 0.