Homework 3

Math 182, Winter 2025

When designing an algorithm, you must include the following:

1. A short (1-2 sentence) sketch of the main idea of the algorithm.
2. The algorithm itself, written in pseudo-code.
3. The runtime of the algorithm.
4. A proof that the algorithm is correct.
5. A proof that the runtime is correct.

If you wish to sort a list, assume that you can do so in $O(n \log n)$ time; you do not need to write out the details of the sorting algorithm.

Question 1:

Suppose that you are in some country where there are n types of coins, which have values $v_1, v_2, ..., v_n$ and that you have access to an unlimited supply of each type of coin. One day, you decide to buy some item with cost v and you want to do so using as few coins as possible (assume that there is some choice of coins whose values add up to v). However, it takes you a very long time to figure out the minimum number of coins required, so you try to come up with an efficient algorithm to solve the problem.

You come up with the following algorithm: repeatedly find the coin with the highest value less than or equal to the currently remaining amount you need to pay and subtract its value from the amount left to pay until you hit 0. Pseudocode for this algorithm is given below:

```
ChooseCoins(A,v):
  result = new array of same size as A
  while v $\neq$ 0:
    i = index which maximizes A[i], subject to the constraint that A[i]$\leq$ v
    v = v - A[i]
    result[i] = result[i]+1
  return result
```

Array A contains the values of all of the coins; `result[i]` is the number of the i-th coin used. Note that we are assuming that a solution is possible, i.e. that there is some way of obtaining exactly v by adding up values from A.

Is the algorithm described above correct? Prove that it is, or give an input on which it is not correct.

Example: Suppose the coins have values 1, 2, 7, and 20. If the cost of the item is 31, then the smallest number of coins is 4: $20 + 7 + 2 + 2$.

Question 2:

Design an efficient algorithm to solve the following problem. Given two sequences of numbers $a_1, a_2, \ldots, a_m$ and $b_1, b_2, \ldots, b_n$ (with duplicates allowed), determine whether $a_1, \ldots, a_m$ occurs as a subsequence of $b_1, \ldots, b_n$.

Recall that a subsequence of a sequence $b_1, b_2, \ldots, b_n$ is a sequence of the form $b_{i_1}, b_{i_2}, \ldots, b_{i_j}$, where $1 \leq i_1 < i_2 < \cdots < i_j \leq n$. You may assume that the two sequences are given as arrays A and B.

For full credit, your algorithm should run in $O(n + m)$ time.

Example: If $A = [4, 5, 2]$ and $B = [4, 3, 2, 4, 5, 3, 2]$, then the answer is yes: we can take the subsequence of B consisting of the fourth, fifth, and seventh elements. If $A = [4, 5, 2]$ and $B = [4, 3, 2, 4, 5, 3, 4]$, then the answer is no.

Question 3:

Design an efficient algorithm to solve the following problem. Given some collection of intervals $[s_1, t_1]$, $[s_2, t_2], \ldots, [s_n, t_n]$, find the smallest number of intervals whose union is equal to the union of the entire collection. (This is known as an *optimal covering.*) You may assume that the left endpoints of the intervals are all distinct (i.e. that for all $i \neq j$, $s_i \neq s_j$). For full credit, your algorithm should run in $O(n \log n)$ time.

Example: If the intervals are $[3, 6], [5, 10], [1, 4], [6, 10], [7, 12], [10, 12]$ then the smallest number of intervals is four: the union of $[1, 4], [3, 6], [6, 10], [10, 12]$ will cover every other interval.

Question 4:

Design an efficient algorithm to solve the following problem. Suppose we are given n jobs, each from a different customer. As in the minimizing lateness problem from class, only one job can be completed at a time, and they cannot be split into pieces or abandoned once started. Each job i will take t_i time to complete. Given a schedule, let c_i denote the finishing time of job i. That is, if the first job is job i, $c_i = t_i$. If job j is done immediately afterwards, c_j will be $c_i + t_j$. Each job is given a weight w_i based on the importance of the customer.

The goal is to build a schedule that will minimize $\displaystyle\sum_{i=1}^{n} w_i c_i$.

For full credit, your algorithm should run in $O(n \log n)$ time.

Example: Suppose there are two jobs: the first takes time $t_1 = 1$ and has a weight $w_1 = 10$, while the second job takes time $t_2 = 3$ and has weight $w_2 = 2$. Then doing job 1 first would give a weighted completion time of $10 \cdot 1 + 2c(1 + 3) = 18$, while doing the second job first would give a weighted completion time of $2 \cdot 3 + 10 \cdot (3 + 1) = 46$.