COL351 Holi2023: Tutorial Problem Set 5

1. Let $A[1, \ldots, n]$ and $B[1, \ldots, n]$ be increasing sequences of real numbers. We wish to find a permutation $g : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\}$ that maximizes $\sum_{i=1}^{n} A[i]B[g(i)]$. Using an exchange argument, prove that the identity permutation is the only maximizer. (This result is called the *rearrangement inequality*.)

2. Let $\mathcal{S}$ be a collection of sets in a common universe. A hitting set of $\mathcal{S}$ is a set H that intersects every set $S \in \mathcal{S}$. Given a finite collection of sets $\mathcal{S}$, a fundamental computational problem is to find a hitting set of $\mathcal{S}$ of minimum cardinality. While this problem is *hard* in general, a polynomial time algorithm could exist if $\mathcal{S}$ has an appropriate structure.

   Suppose $\mathcal{S}$ is a finite collection of closed intervals on the real number line. Design a polynomial time algorithm to compute the minimum cardinality hitting set of $\mathcal{S}$. Next, using appropriate data structures, show how you will implement the algorithm so that it runs in time $O(n \log n)$, where $n = |\mathcal{S}|$.

3. [Kleinberg-Tardos Chapter 4 Exercise 15, rephrased] Consider a set of intervals $S = \{[s_1, t_1], \ldots, [s_n, t_n]\}$, where $s_i \leq t_i$ for each i. A *dominating subset* of S is a set $D \subseteq S$ such that for each interval $I \in S$, there exists an interval $J \in D$ such that $I \cap J \neq \emptyset$. Design an algorithm that takes as input a set of intervals and outputs a dominating subset of minimum possible cardinality. For simplicity, you may assume that the s_i's and t_i's are all distinct.

4. Recall the assignment scheduling problem and its algorithm discussed in class. Observe that a straightforward implementation of the algorithm takes $\Theta(n^2)$ time in the worst case, where n is the number of assignments. Design an improved $O(n \log n)$ time algorithm for the assignment scheduling problem.

5. Consider the following variant of the assignment scheduling problem where the instructors are not as considerate – to pass, it is compulsory for you to submit all assignments. Moreover, associated with each assignment i is a deadline d_i and, instead of an incentive to finish the assignment on time, there is a penalty p_i per day of delay in submitting the assignment. As before, each assignment takes exactly one day. Design a polynomial time algorithm to compute an order for finishing assignments that minimizes the total penalty.

6. [Kleinberg-Tardos Chapter 4 Exercise 7, rephrased] Consider a set of n jobs, $J_1, \ldots, J_n$. Each job J_i consists of an activity S_i followed by another activity P_i. The activities S_i and P_i take time s_i and p_i respectively. All the activities S_i must be run sequentially on a machine M. On the other hand, each P_i is to be run on a separate machine M_i, and therefore, the P_i's can run in parallel (with one another and one of the S_j's). We want to decide the order in which the S_i's should be run on M to minimize the maximum of the completion times of the n jobs. The completion time of a job J_i is its waiting time before S_i starts executing on M, plus $s_i + p_i$. For example, if the S_i's are executed on M in the order $S_1, \ldots, S_n$, then the completion time of job J_i is $C_i = s_1 + \cdots + s_{i-1} + s_i + p_i$, where $s_1 + \cdots + s_{i-1}$ is the waiting time of J_i.

   Design an algorithm that takes as input $(s_1, p_1), \ldots, (s_n, p_n)$, and outputs an order in which the S_i's should be executed on M so as to minimize $\max_i C_i$.

7. Consider a movie hall having n seats, numbered $1, \ldots, n$. The seat booking website of this movie hall runs an algorithm which does the following. Each customer j (where $j = 1, \ldots, m$) submits a request for a subset S_j of seats to the algorithm. In response, the algorithm allocates to customer j all seats in S_j that haven't been assigned to any of the earlier customers $1, \ldots, j-1$.

   Unfortunately, the website goes down one day and, instead of running the above seat allocation algorithm, simply collects requests and stores them in an arbitrary order, say $R_1, \ldots, R_m$. You fix the

website, but you have no idea of the order in which the requests were submitted. To be fair to all the customers, you decide to feed the requests to the seat allocation algorithm in an order that maximises the minimum of the number of seats allocated to a customer. Note that the seat allocation algorithm cannot be changed; for each request, it will assign all the seats requested seats that were not assigned to previously submitted requests. Design a polynomial time algorithm that, given $R_1, \ldots, R_m$, orders the requests optimally.