Name: _____

Entry number: _____

- Always try to give algorithm with best possible running time. The points that you obtain will depend on the running time of your algorithm. For example, a student who gives an $O(n)$ algorithm will receive more points than a student who gives an $O(n^2)$ algorithm.
- You are required to give proofs of correctness whenever needed. For example, if you give an algorithm using network flow for some problem, then you should also give a proof why this algorithm outputs optimal solution.
- You may use any of the following known NP-complete problems to show that a given problem is NP-complete: 3-SAT, INDEPENDENT-SET, VERTEX-COVER, SUBSET-SUM, 3-COLORING, 3D-MATCHING, SET-COVER, CLIQUE.
- **Use of unfair means will be severely penalized.**

There are 8 questions for a total of 30 points.

---

(4) 1. Recall the deterministic algorithm for finding the k^{th} smallest number in an array containing distinct numbers:

> *The algorithm considers groups of 5 elements. It picks the middle element from each group and then finds the median element of these middle numbers. Let p denote the median of the middle numbers. It then partitions the array into two parts: left part consisting of numbers $< p$ and right part consisting of numbers $> p$. It then looks for the appropriate number in either the left or right part depending on the size of these parts.*

Suppose the algorithm considers groups of 3 elements instead of groups of 5 elements. Analyze the running time of this algorithm?

Solution: Analogous to the algorithm discussed in class, we get the following recurrence relation for the running time:

$$T(n) = T(\lceil n/3 \rceil) + T(\lceil 2n/3 + 2 \rceil) + O(n)$$

You may solve this using the recursion tree method. The recursion tree will be a full binary tree up to depth $\log_3(n)$. Each level contributes $O(n)$ to the running time. So, the total running time of the algorithm will be $O(n \log(n))$. This is same as the running time for sorting which is the naive way of finding the k^{th} smallest number. So, considering groups of 3 does not improve over the naive algorithm.

(2) 2. Consider the following problem:

> FACTOR: Given integers N, x, determine if N has a non-trivial factor less than x.

It is known that FACTOR $\in NP$ but it is not known if FACTOR is NP-complete. State whether the following statements are true or false or unknown with reasons:

(a) If P $\neq$ NP, then FACTOR cannot be solved in polynomial time.

> **Solution:** Unknown. Since, FACTOR is not known to be NP-complete, it may be possible to solve it in polynomial time even though $P \neq NP$.

(b) If P = NP, then any 1024 bit number can be factored within an hour.

> **Solution:** Unknown. Even if $P = NP$, the algorithms that solve problems in NP might run in time $O(n^{1000000000})$ time. Note that $O(n^{1000000000})$ is also polynomial time. The problem this is a very huge number and even if we employ all computing resources in the world, we will not be able to factor a number quickly.

(3) 3. Solve the following linear program using the simplex algorithm. Give the value of the variables that maximizes the objective function.

Maximize $3x_1 + 2x_2 + x_3$,
Subject to:

$$\begin{aligned} x_1 - x_2 + x_3 &\leq 4 \\ 2x_1 + x_2 + 3x_3 &\leq 6 \\ -x_1 \quad\quad + 2x_3 &\leq 3 \\ x_1 + x_2 + x_3 &\leq 8 \\ x_1, x_2, x_3 &\geq 0. \end{aligned}$$

> **Solution:** Pivot using $x_5 = 6 - 2x_1 - x_2 - 3x_3$. The maximum value of the objective function is 12 for the following assignments to variables: $x_1 = 0, x_2 = 6, x_3 = 0$.

(4) 4. Given n integers $x_1, ..., x_n$ and an integer P, a set $S = \{(i, j) : i < j \text{ and } x_i + x_j \geq P\}$ is said to be *valid pair set* if each i is present in at most one pair in S. Design an algorithm that outputs a valid pair set with largest cardinality.

> **Solution:** Here is a greedy algorithm that solves the problem:
>
> 1. Sort the x_i's in decreasing order and consider them in this order.
> 2. $S \leftarrow \phi$; $i = 1$; $j = n$
> 3. While$(j > i)$
> 4. if $(x_i + x_j \geq P)$
> 5. $S \leftarrow S \cup (i, j)$.
> 6. $i \leftarrow i + 1$; $j \leftarrow j - 1$
> 6. else $j \leftarrow j - 1$
>
> We can show that the greedy algorithm returns the optimal solution using a simple exchange argument. Start from an optimal solution S_{OPT}. Suppose $x_i + x_j < P$, then j cannot appear in any pair in S_{OPT}. If this happens then our greedy algorithm is consistent with the optimal solution. Suppose $x_i + x_j \geq P$. The greedy solution picks the pair (i, j). Suppose S_{OPT} does not contain (i, j). The following possible cases can arise:
>
> (a) i and j are not in any pair in S_{OPT}: In this case adding (i, j) will increase the size of S_{OPT}.

(b) i is not present in any pair in S_{OPT} but $(k, j) \in S_{OPT}$: $S_{OPT} = S_{OPT} - (k, j) \cup (i, j)$ is another optimal valid pair set.

(c) j is not present in any pair in S_{OPT} but $(i, k) \in S_{OPT}$: $S_{OPT} = S_{OPT} - (i, k) \cup (i, j)$ is another optimal valid pair set.

(d) $(i, k_1), (k_2, j) \in S_{OPT}$: If $x_{k_1} \leq x_{k_2}$, then we know that $x_{k_2} \geq P/2$ and $x_{k_1} + x_{k_2} \geq P$. So, $S_{OPT} = S_{OPT} - (i, k_1) - (k_2, j) \cup (i, j) \cup (k_1, k_2)$ is another optimal solution. If $x_{k_1} > x_{k_2}$, then again $S_{OPT} = S_{OPT} - (i, k_1) - (k_2, j) \cup (i, j) \cup (k_1, k_2)$ is another optimal solution.

Using the above exchange repeatedly we get that the greedy algorithm is the optimal solution.

The Makespan problem Recall the minimum makespan problem that we discussed in class. Consider the following variant of the problem:

> k-MAKESPAN: Given n jobs with integer durations $d_1, ..., d_n$ and an integer D, determine if these jobs can be scheduled on k machines such that the maximum finishing time of any job is $\leq D$.

Let us start analyzing the above problem for different values of k. We note that the 1-MAKESPAN and n-MAKESPAN problems are very easy. The next question asks you to show that the 3-MAKESPAN problem in NP-complete.

(5) 5. Show that 3-MAKESPAN is NP-complete.

Solution: 3-MAKESPAN is in NP since there is an efficient certifier that takes as input the problem and a schedule and checks if the finishing time of all jobs as per the schedule is $\leq D$. The schedule acts as a short certificate for this problem.

To show that 3-MAKESPAN is NP-hard we give a reduction from the SUBSET-SUM problem.

Claim 1: SUBSET-SUM $\leq_p$ 3-MAKESPAN.

Proof: Given an instance of the SUBSET-SUM problem $(\{x_1, ..., x_n\}, W)$. Let $S = \sum_{i=1}^{n} x_i$. We construct the following instance of the 3-MAKESPAN problem:

$$(\{x_1, ..., x_n, W + 1, S - W + 1, S + 1\}, S + 1)$$

We will show the following claim:

Claim 1.1: There is a subset $S \subseteq \{1, ..., n\}$ such that $\sum_{i \in S} x_i = W$ if and only if the above $n + 3$ jobs can be scheduled on 3 machines with max. finishing time $\leq S + 1$.

Proof: (only if) We schedule S on machine 1, $\{1, ..., n\} - S$ on machine 2. Furthermore, we schedule $(n + 1)^{th}$ job on machine 2, $(n + 2)^{th}$ machine on machine 1 and $(n + 3)^{th}$ job on machine 3. We see that all machines finish in time $\leq S + 1$.

(if) If there is a schedule such that all machines finish in $\leq S + 1$ time, then this implies that the $(n + 3)^{th}$ job is scheduled on one machine. WLOG, let this machine be machine 3. Now we note that the $(n + 1)^{th}$ job and $(n + 2)^{th}$ job is scheduled on separate machines. Let them be scheduled on machine 1 and 2 respectively. Since the sum of duration of the remaining n jobs is S. There is a subset of jobs that sum to exactly W (which is scheduled on machine 2 and the remaining jobs are scheduled on machine 1).

Now that we have proved that 3-MAKESPAN is NP-complete, we know that a polynomial time algorithm is unlikely. However, we know that all instances of the problem might not be hard. For example, what about when D is small? The next question asks to give a polynomial time algorithm for problem instances where D is small.

(4) 6. Suppose you are given problem instances of 3-MAKESPAN where $D = O(n)$. Give a polynomial time algorithm for solving the problem. Discuss the running time of your algorithm.

Solution: This is solved using Dynamic Programming. Let $M(i, D_1, D_2, D_3)$ be 1 if jobs 1 through i can be scheduled such that the machine 1 finishes in time $\leq D_1$, machine 2 finishes in time $\leq D_2$, and machine 1 finishes in time $\leq D_3$ and 0 otherwise. We get the following recursive formulation:

$$M(i, D_1, D_2, D_3) = Max\,(M(i-1, D_1 - x_i, D_2, D_3), M(i-1, D_1, D_2 - x_i, D_3), M(i-1, D_1, D_2, D_3 - x_i))$$

The base case is that for all i, if x_i is strictly larger than D_1, D_2, and D_3, then $M(i, D_1, D_2, D_3) = 0$. The final answer can be obtained by reading $M(n, D, D, D)$. The running time for filling the entire table is $O(n \cdot D^3) = O(n^4)$.

Now consider the optimization version of the problem.

> MIN-3-MAKESPAN: Given n jobs with duration $d_1, ..., d_n$, determine a schedule of these n jobs on 3 machines that minimizes the maximum finishing time of any job.

The optimization version of a problem is usually harder than the decision version. The next question asks you to show this formally.

(2) 7. Show that MIN-3-MAKESPAN is NP-hard.

Solution: Given an algorithm B for the MIN-3-MAKESPAN problem, we can design the following algorithm A for solving the 3-MAKESPAN problem. A first uses B to compute the minimum value of maximum finishing time. Let m be this value. If $D \geq m$, then A outputs 1 else A outputs 0.

The reason why A works correctly is simple. If $m \leq D$, then there is a schedule such that the max. finishing time of any job is $\leq D$ since the schedule returned by B is such a schedule. If $m > D$, then there is no schedule possible, since otherwise the minimum value of maximum finishing time is $< m$.

We have seen that one way to deal with NP-hard problems is to give efficient approximation algorithms. In the lectures we looked at a greedy algorithm that gives 2-approximation. The next question asks you to analyze the approximation factor of a variant of this algorithm.

(6) 8. Consider the following greedy algorithm for the MIN-3-MAKESPAN problem: *Sort the jobs in decreasing order of their duration. Consider jobs in this order and schedule a job on a machine with the smallest current load.* Show that this algorithm gives a (3/2) approximation factor.

Solution: Let $x_1, ..., x_n$ be the durations in decreasing order. Let G be the greedy solution and OPT denote the optimal solution. We have:

$$\frac{\sum_{i=1}^{n} d_i}{3} \leq OPT \quad (1)$$

Let j be the last job on the machine that finishes last. If $j \leq 3$, then $G = OPT$. If $j > 3$, then we have the following argument:

Let s be the finishing time of the previous job on this machine. Then we have:

$$s \leq \sum_{i=1}^{n} d_i/3 \leq OPT \tag{2}$$

Since $j > 3$, there is at least one more job on this machine. The duration of this job is larger than d_j. This gives us that $2d_j \leq OPT$. This is because even in the optimal solution, there will be two jobs from among jobs $1, ..., j$ scheduled on a single machine by pegionhole principle. This means that $OPT \geq 2d_j$.

We have $G = s + d_j \leq OPT + OPT/2 = (3/2)OPT$.