CS3230 Final Exam Solutions

1. (a) True

   Idea: $n^2 \leq n \lg n + n^2 \leq 2n^2$, so $n \lg n + n^2 = \Theta(n^2)$.

   (b) False

   Idea: $5^{\lg \lg n} = (2^{\log_2 5})^{\lg \lg n} = (2^{\lg \lg n})^{\log_2 5} = (\lg n)^{\log_2 5}$. Since $\log_2 5 > 2$, we have $5^{\lg \lg n} \neq O((\lg n)^2)$.

   (c) False

   Idea: $\frac{n^n}{(2n)!} < \frac{n^n}{(n+1)(n+2)\ldots(2n)} = \left(\frac{n}{n+1}\right)\left(\frac{n}{n+2}\right)\ldots\left(\frac{n}{2n}\right) \leq 1$, so $n^n \neq \omega((2n)!)$.

   (d) False

   Idea: P is a subset of NP.

   (e) True

   Idea: If there is a polynomial-time algorithm for one NP-complete problem, there is also one for every other NP-complete problem.

   (f) True

   Idea: KNAPSACK is NP-complete, so if there is a polynomial-time algorithm for it, then P = NP.

   Rubric: 2 points for each correct answer, 1 point for each "Don't know".

2. (a) $\Theta(n^2 \lg^2 n)$

   Idea: Case 2 of master method with $a = 4$, $b = 2$, $f(n) = n^2 \lg n$.

   (b) $\Theta(n)$

   Idea: Using the recursion tree, the work done is n at the top level and decreases by a multiplicative factor of $1/3+1/5+1/7 < 1$ for each subsequent level. Since the geometric sequence $1+c+c^2+\ldots$ converges for every $c < 1$, the total running time is $\Theta(n)$.

   (c) $\Theta(\lg \lg n)$

   Idea: Using the recursion tree, the work done at each level is $\Theta(1)$, so the running time is on the same order as the number of levels. To determine this number, let $n = 2^k$, and note that the next levels correspond to $2^{k/2}, 2^{k/4}$, and so on, so the number of levels is $\lg k = \Theta(\lg \lg n)$.

   (d) $\Theta(n^2)$

   Idea: The total amount of work is $3n+3(n-3)+3(n-6)+\cdots = 3(n+(n-1)+(n-2)+\ldots) = \Theta(n^2)$.

   (e) $\Theta(n)$

   Idea: Operation $i \in \{1, 2, 4, \ldots, 2^{\lfloor \lg n \rfloor}\}$ cost i^2, while all other $n - \lfloor \lg n \rfloor - 1$ operations cost 3 each. Hence, the total cost of the n operations is

   $$\begin{aligned}
   &(1 + 4 + 16 + \cdots + 4^{\lfloor \lg n \rfloor}) + 3(n - \lfloor \lg n \rfloor - 1) \\
   &= \frac{4^{\lfloor \lg n \rfloor + 1} - 1}{3} + 3(n - \lfloor \lg n \rfloor - 1) \\
   &\leq \frac{4n^2}{3} + 3n.
   \end{aligned}$$

   Moreover, the total cost is at least $4^{\lfloor \lg n \rfloor} \geq 4^{\lg n - 1} = \frac{n^2}{4}$. Dividing by n, we find that the amortized cost is $\Theta(n)$ per operation.

   Rubric: 3 points for each question. No partial credit. For part (a), writing as $\Theta(n^2(\lg n)^2)$ or $\Theta((n \lg n)^2)$ is fine.

3. Note that if you assign document i to friend k, you have to pay $\frac{w_i}{r_k}$ dollars to this friend. Hence, your goal is to find a permutation $\sigma : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ such that the sum

$$\frac{w_1}{r_{\sigma(1)}} + \frac{w_2}{r_{\sigma(2)}} + \cdots + \frac{w_n}{r_{\sigma(n)}}$$

is maximized; here, each document i is assigned to friend $\sigma(i)$.

We claim that you should always assign documents with fewer words to friends who can type slower. Indeed, suppose that $w_i < w_j$ but $r_{\sigma(i)} > r_{\sigma(j)}$. Then the following calculations show that you would be better off switching the assignments of documents i and j:

$$\begin{aligned} \frac{w_i}{r_{\sigma(i)}} + \frac{w_j}{r_{\sigma(j)}} > \frac{w_i}{r_{\sigma(j)}} + \frac{w_j}{r_{\sigma(i)}} \quad &\iff \quad w_j \left(\frac{1}{r_{\sigma(j)}} - \frac{1}{r_{\sigma(i)}} \right) > w_i \left(\frac{1}{r_{\sigma(j)}} - \frac{1}{r_{\sigma(i)}} \right) \\ &\iff \quad (w_j - w_i) \left(\frac{1}{r_{\sigma(j)}} - \frac{1}{r_{\sigma(i)}} \right) > 0. \end{aligned}$$

Hence, we can sort the n documents from smallest to largest number of words, sort the n friends from slowest to fastest, and assign the smaller documents to slower friends. Since sorting can be done in time $O(n \lg n)$ (e.g., using Merge Sort), this algorithm runs in time $O(n \lg n)$.

Rubric: 10 points in total, broken down as follows:

- 1 point for writing down the amount $\frac{w_i}{r_k}$ that you have to pay if you assign document i to friend k
- 1 point for writing down the expression for the total payment $\frac{w_1}{r_{\sigma(1)}} + \frac{w_2}{r_{\sigma(2)}} + \cdots + \frac{w_n}{r_{\sigma(n)}}$
- 3 points for claiming that shorter documents should be assigned to slower friends
  - 2 points for claiming the opposite, i.e., shorter documents should be assigned to *faster* friends
- 3 points for proving the previous bullet point
  - No point for mistakenly proving the opposite claim by flipping the inequality
- 2 points for stating that the running time $O(n \lg n)$ follows from sorting (sorting should be mentioned, but no specific sorting algorithm like Merge Sort needs to be mentioned)

For a dynamic programming approach (similar to Assignment 8 Question 3), the maximum amount that can be awarded is 6 points:

- 1 point for writing down the amount $\frac{w_i}{r_k}$ that you have to pay if you assign document i to friend k
- 1 point for writing down the expression for the total payment $\frac{w_1}{r_{\sigma(1)}} + \frac{w_2}{r_{\sigma(2)}} + \cdots + \frac{w_n}{r_{\sigma(n)}}$
- 3 points for describing the dynamic programming. Partial credit can be awarded as appropriate.
- 1 point for stating the running time $O(n \cdot 2^n)$

For a brute-force approach, the maximum amount that can be awarded is 3 points:

- 1 point for writing down the amount $\frac{w_i}{r_k}$ that you have to pay if you assign document i to friend k
- 1 point for writing down the expression for the total payment $\frac{w_1}{r_{\sigma(1)}} + \frac{w_2}{r_{\sigma(2)}} + \cdots + \frac{w_n}{r_{\sigma(n)}}$
- 1 point for stating the running time $O(n \cdot n!)$
- No point for the algorithm itself

Comment: The majority of students correctly realized that documents with more words should be assigned to friends who can type faster. However, a significant subset of these students either did not provide a justification as to why this greedy strategy works, or made an argument about swapping but did not prove the swap inequality. Note that while the swap inequality is not hard to prove, it

is not obvious without doing the algebra and, moreover, some students either had difficulty proving it or proved it incorrectly, so the benefit of the doubt could not be given to those who did not prove it. Besides taking a global view as in the solution above, a common alternative approach is to argue a greedy choice property that the longest document should be assigned to the fastest friend, and then appealing to the optimal substructure within the remaining documents and friends. Some students made an incorrect greedy choice such as choosing a pair (i, k) that minimizes $\frac{w_i}{r_k}$; this results in the shortest document being assigned to the *fastest* friend, which is the opposite of the correct strategy.

Most remaining students used a dynamic programming approach similar to Assignment 8 Question 3, which works but leads to an exponential running time.

4. (a) First, we observe that FabulousHalf is in NP: Given a set of sheets $S \subseteq \{1, 2, \ldots, n\}$, a verifier can check in time $O(n)$ whether the set has size $n/2$ and whether $\sum_{i \in S} a_i \geq A/2$ and $\sum_{i \in S} b_i \geq B/2$.

Next, we show that FabulousHalf is NP-hard by reducing from PartitionEqual. Consider an instance $I = (x_1, \ldots, x_n)$ of PartitionEqual, and let $X = \sum_{i=1}^{n} x_i$. Create an instance I' of FabulousHalf by letting $a_i = x_i$ and $b_i = X - x_i$ for every i. Clearly, this reduction can be done in polynomial time, and a_i, b_i are nonnegative integers for all i.

($\Rightarrow$) If I is a YES-instance of PartitionEqual, there exists a set $S \subseteq \{1, 2, \ldots, n\}$ of size $n/2$ such that $\sum_{i \in S} x_i = \sum_{i \notin S} x_i$. Since $a_i = x_i$, this means $\sum_{i \in S} a_i = \sum_{i \notin S} a_i = A/2$. Moreover, since $b_i = X - x_i$ and $|S| = n/2$, we have $\sum_{i \in S} b_i = nX/2 - \sum_{i \in S} x_i = nX/2 - \sum_{i \notin S} x_i = \sum_{i \notin S} b_i$, so $\sum_{i \in S} b_i = B/2$. Hence, the same set S shows that I' is a YES-instance of FabulousHalf.

($\Leftarrow$) For the opposite direction, if I' is a YES-instance of FabulousHalf, there exists a set $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} a_i \geq A/2$ and $\sum_{i \in S} b_i \geq B/2$. Since $a_i = x_i$ and $b_i = X - x_i$, this means that $\sum_{i \in S} x_i \geq X/2$ and $nX/2 - \sum_{i \in S} x_i \geq nX/2 - X/2$—the latter inequality can be rewritten as $\sum_{i \in S} x_i \leq X/2$. It follows that $\sum_{i \in S} x_i = X/2$. Hence, the same set S shows that I is a YES-instance of PartitionEqual.

Rubric: 10 points in total, broken down as follows:

- 2 points for showing membership in NP
- 3 points for describing the reduction
  - 1 point for letting $a_i = x_i$
  - 2 points for letting $b_i = X - x_i$
- 5 points for proving the correctness of the reduction
  - 1 point for stating that the reduction can be done in polynomial time
  - 2 points for showing that if I is a YES-instance of PartitionEqual, then I' is a YES-instance of FabulousHalf
  - 2 points for showing that if I' is a YES-instance of FabulousHalf, then I is a YES-instance of PartitionEqual

Most students duly showed membership in NP. However, not many came up with a correct reduction. While a natural starting point is to set $a_i = b_i = x_i$, this does not immediately work—for example, if $(x_1, x_2, x_3, x_4) = (1, 2, 3, 5)$, then I is a NO-instance of PartitionEqual but I' is a YES-instance of FabulousHalf (by taking the first and last sheets). Reversing the order of the numbers on the back side (i.e., taking $b_i = x_{n+1-i}$) does not fix this problem, as can be seen in the same example.

Some students took $b_i = -x_i$, $b_i = 2X/n - x_i$ (which ensures $\sum_{i=1}^{n} b_i = \sum_{i=1}^{n} a_i$), or $b_i = X/2 - x_i$. While these choices lead to a correct mapping between YES-instances of the two problems, b_i can potentially be negative, which is not allowed in the definition of FabulousHalf. Besides $X = \sum_{i=1}^{n} x_i$, another correct choice used by some students is $X = \max_{i=1}^{n} x_i$.

(b) For integers $p \in \{0, 1, \ldots, A\}$, $q \in \{0, 1, \ldots, B\}$, and $k \in \{0, 1, \ldots, n\}$, let $m[p, q, k]$ be the minimum size of a set $S \subseteq \{1, 2, \ldots, k\}$ such that $\sum_{i \in S} a_i = p$ and $\sum_{i \in S} b_i = q$. We have the

following recurrence:

$$m[p,q,k] = \begin{cases} 0 & \text{if } k = 0 \text{ and } (p,q) = (0,0) \\ \infty & \text{if } k = 0 \text{ and } (p,q) \neq (0,0) \\ \min\{m[p,q,k-1],\ m[p-a_k, q-b_k, k-1]+1\} & \text{if } k > 0, p \geq a_k, q \geq b_k \\ m[p,q,k-1] & \text{otherwise} \end{cases}$$

Here, if $k = 0$ then $(p,q) = (0,0)$ is clearly the only achievable pair. Else, if $k > 0$, we try checking whether using the k-th sheet of paper leads to a smaller set than using only the first $k-1$ sheets. To get the final answer, we take the minimum among the entries $m[p,q,n]$ such that $p \geq A/2$ and $q \geq B/2$.

The running time is $O(nAB)$. To achieve this running time, we fill in the table in the order $m[*,*,0], m[*,*,1], m[*,*,2], \ldots, m[*,*,n]$. Since the value of $m[p,q,k]$ depends only on $m[*,*,k-1]$, we have all the necessary values to compute $m[p,q,k]$. There are $A \cdot B \cdot n$ entries, and computing each entry takes time $O(1)$. Computing the final answer takes time $O(AB)$.

Rubric: 10 points in total, broken down as follows:

- 2 points for mentioning 'dynamic programming' or using a dynamic programming approach
- 2 points for indicating a feasible meaning of $m[p,q,k]$
- 3 points for stating the correct recurrence based on the meaning of $m[p,q,k]$
- 2 points for describing how to obtain the final answer
- 1 point for analyzing the running time

Partial credit can be awarded as appropriate.

A fair number of students recognized that a dynamic programming approach is appropriate for this problem. While the flavor is very similar to knapsack, getting all cases of the recurrence right requires nontrivial care. There are alternative ways to define the table entries, for example by changing the conditions in the definition of $m[p,q,k]$ to $\sum_{i \in S} a_i \geq p$ and $\sum_{i \in S} b_i \geq q$ (instead of $=$). The recurrence would have to be modified accordingly for such definitions.

(c) Even though the algorithm in part (b) runs in time polynomial in n, A, and B, the input of this problem has size polynomial in $n, \lg A$, and $\lg B$. Hence, the algorithm does not solve FabulousHalf in time polynomial in the size of the input.

Rubric: 3 points for the correct explanation. No partial credit.

A fair number of students correctly identified the flaw in Bob's argument. There was another (unintended) flaw in the sentence "The answer to FabulousHalf is 'Yes' if and only if this minimum size is at most $n/2$". While this sentence is true when n is even (because if there exists a fabulous set of size at most $n/2$, then there also exists one of size $n/2$ by adding arbitrary sheets until the set is of size $n/2$), it is not true if n is odd (because $n/2$ is not an integer, so the answer to FabulousHalf is always 'No').[1] An answer that shows an awareness of this odd/even issue (either by explaining it or by giving a counterexample) was also accepted. Simply saying that "at most $n/2$ is different from exactly $n/2$" (or similar statements) was not considered sufficient.

Several students tried to identify a flaw in the direction of the reduction, in the relationship between decision and optimization problems, or in the relationship between P and NP. There is no flaw in any of those places. In particular, giving a polynomial-time algorithm for an NP-complete problem would indeed imply that P = NP.

[1] Bob could have fixed this part of his argument by saying "The answer to FabulousHalf is always 'No' for odd n, whereas for even n it is 'Yes' if and only if this minimum size is at most $n/2$.".