CS:3330 Exam 1 Solution, Spring 2017

1(a) (i) $2^{2.5\log_2 n} = \Theta(n^{2.5})$

(ii) The running time of the MergeSort algorithm $= \Theta(n \log n)$.

(iii) $100n^2 + 1000000 = \Theta(n^2)$

(iv) $(\log_2 n)^2 \cdot \sum_{i=1}^{n} \Theta(1/2^i) = \Theta(\log^2 n)$ because $\sum_{i=1}^{n} 1/2^i = O(1)$.

(v) $n^{1.5}/(\log_2 n)^4$

(vi) $2^{7\sqrt{\log_2 n}}$

The ordering is: (iv), (vi), (ii), (v), (iii), (i)

1(b) (i) $100n^3 + 10n^2 + 15 = \Theta(n^2)$.
False. $100n^3 + 10n^2 + 15$ grows asymtotically faster than n^2.

(ii) I prefer an algorithm running in $\Theta(\sqrt{2^n})$ time relative to an algorithm running in $\Theta\left(3^{\log_2 n}\right)$.
False. $3^{\log_2 n}$ can be rewritten as $n^{\log_2 3}$, which is a polynomial function, whereas $\sqrt{2^n}$ is an exponential function. So the first algorithm is not more efficient than the second.

(iii) There is an algorithm that solves MVC (i.e., produces an optimal solution for every input) in $O((m+n)\log n)$ time.
False. No one knows how to solve MVC in polynomial time and the consensus among computer scientists is that no one ever will. To actually prove this is the famous (open) P vs NP problem.

(iv) $n^{\log_2 n} = \Theta(2^{(\log_2 n)^3})$.
False. Simplifying the function on the right hand side as:

$$2^{(\log_2 n)^3} = \left(2^{(\log_2 n)}\right)^{(\log_2 n)^2} = n^{(\log_2 n)^2}.$$

we see that the function on the right hand side grows asymptotically faster than the function on the left hand side.

2(a) Since B starts at 1, doubles in each iteration (of the inner loop) and needs to reach n, we see that the inner loop runs in $\Theta(\log n)$ rounds. This is true for every execution of the outer loop, which executes n times. Therefore, the function executes in $\Theta(n \log n)$ time.

2(b) The variable j starts at n and decrease by $2 \cdot \varepsilon$ in each iteration (of the inner loop) and so the inner loop runs in $\Theta(n/\varepsilon)$ time. This is true for every execution of the outer loop, which executes n times. Therefore, the function executes in $\Theta(n^2/\varepsilon)$ time.

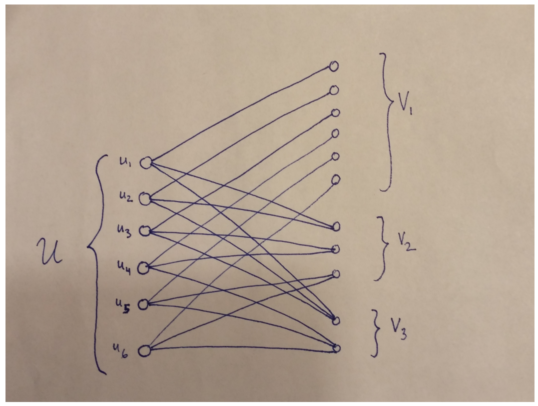
2(c) There is an initialization loop that runs in $\Theta(W)$ time. Following this, there is a nested loop in which the inner loop runs in $\Theta(W)$ time. This inner loop is enclosed in an outer loop that always runs in $\Theta(n)$ time. Thus the nested loop runs in $\Theta(nW)$ time. Thus the total running time is $\Theta(W + nW) = \Theta(nW)$.

3(a) Yes, there does always exist a perfect matching with no strong instability. Such a perfect matching can be found by using a (slightly) modified version of the Gale-Shapley algorithm. Since there are ties in preferences, each man considers women in decreasing order of preference with *ties broken arbitrarily*. An engaged woman w ignores a proposal from a man m', unless she strictly prefers m' to her current partner m. In other words, if she is indifferent between m and m', then she'll stay with m. (One can show that this version of the Gale-Shapley algorithm find a perfect matching with no strong instabilities on every input.)

3(b) No. Consider an input with two men m_1 and m_2 and two women w_1 and w_2. Suppose these individuals have the following preferences: m_1 strictly prefers w_1 over w_2 and similarly m_2 strictly prefers w_1 over w_2. The women w_1 and w_2 are indifferent between the two men.

Now consider the matching $\{(m_1, w_1), (m_2, w_2)\}$ and note that this contains a weak instability because m_2 strictly prefers w_1 over his current partner and w_1 is indifferent between m_2 and her current partner. Similarly, the other possible matching, $\{(m_1, w_2), (m_2, w_1)\}$ also has a weak instability.

4(a) Here is a drawing that shows G_3.



4(b) The vertex cover produced by the GreedyDegreeBased algorithm on G_3 is $V_1 \cup V_2 \cup V_3$, which is a set with 11 vertices. An optimal vertex cover for G_3 is $\{u_1, u_2, \ldots, u_6\}$.

4(c) On G_k, the GreedyDegreeBased algorithm produces the vertex cover $V_1 \cup V_2 \cup \ldots \cup V_k$, which has size $n(1 + 1/2 + 1/3 + \cdots + 1/k)$. The set $\{u_1, u_2, \ldots, u_n\}$ is an optimal vertex cover on G_k. Thus the ratio of the size of vertex cover produced by GreedyDegreeBased to the size of an optimal vertex cover is

$$\frac{n(1 + 1/2 + 1/3 + \cdots + 1/k)}{n} = (1 + 1/2 + 1/3 + \cdots + 1/k).$$

According to the note given in the problem, for $k = 10$, this quantity is less than 3, but for $k = 11$, this quantity is greater than 3. Thus $G_1 1$ is the smallest example of a graph in the given family of graphs, for which the GreedyDegreeBased algorithm is not a 3-approximation.

5(a) The running time of this algorithm is $O(1)$.

5(b) The function returns 0 if the variable *count* has value 0 at the end of the **for**-loop. For *count* to have value 0, it must be the case that in all 100 random attempts, the algorithm picks an index i such that $L[i] = 1$. Since a quarter of the elements are 0, the probability of picking an index i, uniformly at random, such that $L[i] = 1$ is 3/4. Since the 100 attemps are indpendent, the probability that this happens all 100 times is $(3/4)^{100}$. Thus the probability that the CountZeroes function returns 0 is $(3/4)^{100}$.

(c) CountZeroes will return the correct answer, namely $10^6/4$, if exactly 25 (out of 100) of the indices i it chooses uniformly at random point to 0. Here we are performing $m = 100$ independent random trials and asking for the probability that we see 0 exactly 25 times. According to the expression for the binomial distribution, this is

$$\Pr[X = 25] = \binom{100}{25} \cdot (1/4)^{25} \cdot (3/4)^{75}.$$