CS 401

Homework 1

1. a) False Consider the following preferences for males:

M_1	G_1	G_2	G_3
M_2	G_2	G_1	G_3
M_3	G_3	G_1	G_3

G_1	M_3	M_1	M_2
G_2	M_1	M_3	M_2
G_3	M_2	M_1	M_3

In the above example, $M_1 - M_3$ will have their first preference, but no G will yield their first preference and no re-matching occurs. Because of this, we prove that this is false.

b) True. Because we go down the list of M and go by first preferences, M will eventually propose to w. Even if w has already accepted another, because she favors M, she will choose M after.

c) False. Consider the following setup:

M_1	W_1	W_2	W_3
M_2	W_1	W_2	W_2
M_3	W_2	W_1	W_3

W_1	M_3
W_2	M_1
W_3	M_1

M_1 and M_2 will not pair with W_1 b/c she prefers M_3, and M_2 will not pair with W_2 b/c she prefers M_1, who will have asked her already.

2.

M	P			
A	~~E~~	~~G~~	H	F
B	E	F	G	H
C	G	E	F	H
D	F	E	H	G

F	P			
E	B	C	D	~~A~~
F	A	B	C	D
G	C	D	~~A~~	B
H	B	D	A	C

3. 2^n $\quad n^2 \log_2 n$ $\quad n^{4/3}$ $\quad 2^{\sqrt{\log_2 n}}$

$n \cdot (\log_2 n)^3$ $\quad 100n^2$ $\quad n^3/\log_3 n$ $\quad n!$

1. $2^{\sqrt{\log_2 n}}$
2. $n(\log_2 n)^3$
3. $n^{4/3}$
4. $100n^2$
5. $n^2 \log_2 n$
6. $n^3 / \log_3 n$
7. 2^n
8. $n!$

5. $n \cdot n$ matrix $\cdot$ $n \cdot n$ matrix results in an $n \cdot n$ matrix

$$\begin{array}{cc} & n \\ n & \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}_A \end{array} \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}_B = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}_C$$

a) for (each value from Ø to n of A) i

for (each value from Ø to n of B) j

for (each value from Ø to n) k

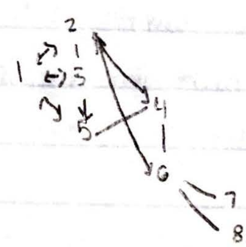
$C_{ij} \quad A_{ik} \cdot B_{kj}$

k enables us to go through A's row and B's column

b) The runtime is n^3 because we have an inner loop of n inside n, which is n^2 and we also do that n times, thus n^3

6. a)

	1	2	3	4	5	6	7	8
1	0	1	1	0	1	0	0	0
2	1	0	1	1	0	1	0	0
3	1	1	0	0	1	0	0	0
4	0	1	0	0	1	1	0	0
5	1	0	1	1	0	0	0	0
6	0	1	0	1	0	0	1	1
7	0	0	0	0	0	1	0	0
8	0	0	0	0	0	1	0	0



~~1~~ 2 3 8 4 6

b) ~~1~~, 2, 3, 5, 4, 6, 7, 8

c) 1, 2, 3, 5, 4, 6, 7 8

4. Create a set to hold all intersection values

Sort all items in set A from low to high

Repeat sort for set B

Repeat sort for set C

Create a pointer in each set's beginning and compare each of them

If any value in any set is < the other 2, move that pointer

Whenever you have all 3 with a match, add that to new set

Repeat this until one set is complete. Then, end.

Sorting will take O(nlogn) time each, and the comparisons will be no more than n each.

7. Because we know the graph is undirected, we can assume that cycles are whenever you can reach a node from another node from two different paths. This means an acyclic graph must represent similarly to a tree, as trees do not interconnect below.

We can look at each individual V that has no incoming E. These are our "roots." From here, we can apply DFS, indicating whether a V has already been "found" or not. If, while running DFS we find a V already found, we have a cycle. This ensures we check every root, and go to the ends of the graphs.