Take-Home Test 1 Sample Solutions

2. Let n and m be the number of vertices and edges, respectively, in a connected, undirected graph. Let $f(n) = n^2 + 2^{2+\log_2 n}$. Which of the following statements are ALWAYS true? Check all that apply.

- [x] $f(n) \in \Omega(n^2)$
- [x] $f(n) \in O(n^2)$
- [x] $f(n) \in \Theta(n^2)$
- [ ] $f(n) \in \Omega(m^2)$
- [x] $f(n) \in O(m^2)$
- [ ] $f(n) \in \Theta(m^2)$

While not required as part of your test, we have included here some justification of the selected answers. The second term of $f(n)$, $2^{2+\log_2 n}$, simplifies to $4n$. Thus, $f(n) \in \Theta(n^2)$, which accounts for the first three selections (because $f(n) \in \Theta(n^2)$ implies both $f(n) \in \Omega(n^2)$ and $f(n) \in O(n^2)$).

Because the graph is connected, the number of edges m is somewhere between $\Theta(n)$ (this occurs when, for example, the graph is a tree) and $\Theta(n^2)$ (which occurs for a fully connected graph). Thus, m^2 is somewhere between $\Theta(n^2)$ and $\Theta(n^4)$, which means that $f(n)$ is always in $O(m^2)$, but not always in $\Omega(m^2)$ or $\Theta(m^2)$.

3. An algorithm's running time

$$T(n)$$

is determined by the following recurrence relation, where n is the size of the input:

$$T(n) = \begin{cases} 8T\left(\frac{n}{2}\right) + f(n) & \text{if } n \geq 4 \\ \Theta(1) & \text{if } n \leq 3 \end{cases}$$

Select all function(s) $f(n)$ below for which the running time of the algorithm will be in $\Theta(n^3)$.

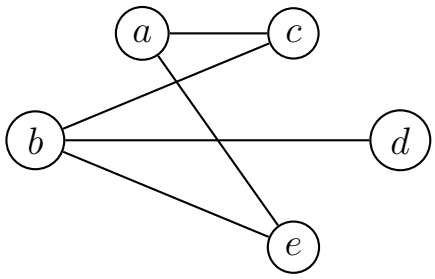
- [x] $f(n) = n\sqrt{n}$
- [x] $f(n) = n^2$
- [x] $f(n) = n^2 \log^2 n$
- [ ] $f(n) = n^3$
- [ ] $f(n) = n^3 \log^2 n$

Again, here we provide justification of the selected choices. We can bound the runtime of $T(n)$ using the Master Theorem, where $b = 2$, $a = 8$, and $\log_b a = 3$. To obtain $T(n) = \Theta(n)^3$, we need to achieve case 1 of the Master Theorem (which gives $T(n) \in \Theta(n^{\log_b a})$). This means we need $f(n) \in O(n^c)$, where $c < 3$. The choice $f(n) = n^2$ clearly fits the bill, as does $f(n) = n\sqrt{n} = n^{1.5}$. The $n^2 \log^2 n$ is a bit tricky, but we can use here the fact that $\log^j(n) \in O(n^k)$ for any positive j, k (to convince yourself of this: notice that any polynomial of n is exponential with respect to the logarithm of n). Thus, we can say that $n^2 \log^2 n \in O(n^{2+\epsilon})$, where ϵ is any number strictly between 0 and 1, which therefore falls into case 1 of the Master Theorem, as desired.

To confirm that $f(n) = n^3$ doesn't give us the desired bound: this is case 2 of the Master Theorem (because $f(n) \in \Theta(n^{\log_b a} \log^0 n)$), which gives a bound of $T(n) = \Theta(n^3 \log n)$. Similarly, the final

selection $f(n) = n^3 \log^2 n$ gives a bound of $T(n) = \Theta(n^3 \log^3 n)$ (we can also tell immediately that this recurrence cannot give us a final bound of $\Theta(n^3)$ because there is more than $\Theta(n^3)$ work happening outside the recursive call).

4. A graph $G = (V, E)$ is *bipartite* if we can partition the vertices V into two disjoint sets U and W such that no two vertices in U are connected, and no two vertices in W are connected. For instance, the graph below:



is bipartite if we define $U = \{a, b\}$ and $W = \{c, d, e\}$. In the *Bipartite Graph Problem* (BGP), we want to determine if a given input graph is bipartite. In this problem, you will reduce BGP to Boolean Satisfiability (SAT), which we saw in Assignment 1.

(a) Complete the following reduction. For each vertex v_i in V, define a variable $X_{i,U}$, which represents

whether v_i is in U

.

(b) In the space below, define the clauses of your reduced SAT instance (we do not recommend introducing additional variables here). You will not need to prove the correctness of your reduction, but you should clearly explain the clauses you are adding and why.

The main constraint we need to encode in our solution is that if two nodes are connected by an edge, they must be in different subsets U or W. Thus, for each edge (v_i, v_j), we need at least one of the vertices v_i and v_j to be in U, and at least one of them to be in W. In terms of the reduction, the clauses to add are:

For each edge $(v_i, v_j) \in E$, add the clauses

$$(X_{i,U} \vee X_{j,U}) \wedge (\bar{X}_{i,U} \vee \bar{X}_{j,U}).$$

The first clause says that one of the vertices i and j must be in U, and the second says that one of them must be in W.

Last step of the reduction: G is bipartite if and only if the reduced SAT instance is satisfiable.

(c) Suppose the input graph to BGP has n vertices and m edges. What is the length of the SAT instance (i.e., the total number of literals) produced by your reduction, as a function of m and n?

For each edge in E, we introduce two clauses with total length 4. Thus, the length of the SAT instance is $4m$.

5. (a) Consider an SMP instance with $n \geq 2$. Is it possible for there to be a stable matching in which *nobody* is matched with their first choice of partner?

● Yes
○ No

(b) If you answered Yes to 5.1, give and briefly explain an instance of SMP in which there exists a stable matching where nobody is matched with their first choice of partner. If you answered No to 5.1, prove that no such matching can exist for any instance of SMP.

We can rule out an instance of size $n = 2$ because that would result in everyone getting their **last** choice of partner, and it's not hard to show that any solution like that must be unstable. So the next step is to try to construct an appropriate instance of size 3. Since nobody can have their first choice of partner, we'll try to match everyone with their **second** choice of partner (because that seems like the next easiest option). Suppose our matching is $\{(e_1, a_1), (e_2, a_2), (e_3, a_3)\}$ and the preference lists are:

```
e1:  X  a1  X    a1:  X  e1  X
e2:  X  a2  X    a2:  X  e2  X
e3:  X  a3  X    a3:  X  e3  X
```

We need to figure out how to set the X values so the matching is stable. We begin with a_1. a_1 must prefer one of e_2 or e_3. At this point, we haven't set e_2 or e_3's preferences for a_1, so it doesn't matter which employer a_1 prefers. Let's say she prefers e_2:

```
e1:  X  a1  X    a1:  e2  e1  e3
e2:  X  a2  X    a2:  X   e2  X
e3:  X  a3  X    a3:  X   e3  X
```

Now, e_2 **cannot** prefer a_1 to a_2, because otherwise a_1 and e_2 would form an instability. So e_2 must rank a_1 last:

```
e1:  X   a1  X     a1:  e2  e1  e3
e2:  a3  a2  a1    a2:  X   e2  X
e3:  X   a3  X     a3:  X   e3  X
```

Now e_2 prefers a_3 to their current partner, so we have to make sure that e_2 and a_3 don't form an instability, which means that a_3 needs to prefer e_3 (his current partner) to e_2:

```
e1:  X   a1  X     a1:  e2  e1  e3
e2:  a3  a2  a1    a2:  X   e2  X
e3:  X   a3  X     a3:  e1  e3  e2
```

Now e_1 has to prefer a_1 to a_3 to avoid an instability between e_1 and a_3:

```
e1:  a2  a1  a3    a1:  e2  e1  e3
e2:  a3  a2  a1    a2:  X   e2  X
e3:  X   a3  X     a3:  e1  e3  e2
```

And a_2 has to prefer e_2 to e_1:

```
e1:  a2  a1  a3    a1:  e2  e1  e3
e2:  a3  a2  a1    a2:  e3  e2  e1
e3:  X   a3  X     a3:  e1  e3  e2
```

And e_3 must prefer a_3 to a_2:

```
e1:  a2  a1  a3    a1:  e2  e1  e3
e2:  a3  a2  a1    a2:  e3  e2  e1
e3:  a1  a3  a2    a3:  e1  e3  e2
```

Ta-da! Now the matching is stable. Though, an important thing to note here is that this matching **cannot** be produced by Gale-Shapley: G-S with employers making the offers will yield the matching $\{(e_1, a_2), (e_2, a_3), (e_3, a_1)\}$, while G-S with applicants making the offers yields the matching $\{(e_1, a_3), (e_2, a_1), (e_3, a_2)\}$. That's why, if you're reasoning about stable matchings (in

the context of a proof, or a reduction to a black-box SMP solver), it's important to distinguish between "stable matching" and "what Gale-Shapley does," because they aren't necessarily the same.

You obviously didn't need to include this much explanation in your answer to the test, though for full credit you would need to provide a complete instance and state what was the stable matching with nobody getting their first choice of partner.