Algorithm Design
Homework Assignment 4

Given: February 20, 2023 **Due:** March 3, 2023

---

1. Decide whether the following statements are true or false. If it is true, argue its correctness, otherwise, give a counterexample.

Define a couple (m, w) to be *nice* if w is ranked first on m's preference list and m is ranked first on w's preference list.

a. For every instance of the Stable Marriage Problem there is a stable matching containing some nice couple (m, w).

b. Consider an instance of the Stable Matching Problem in which there exists a nice couple (m, w). Then in every stable matching S for this instance, the pair (m, w) belongs to S.

c. Consider an instance of the Stable Marriage Problem in which there are n disjoint nice couples. Then there is a unique stable matching for this instance.

d. Consider an instance I of the Stable Marriage Problem for which there is a unique stable matching. Then I must consist of n disjoint nice couples.

2. The Hotel Partner Problem is as follows. There are $2n$ people and n hotel rooms. Each person maintains a preference list of the remaining $2n - 1$ people. The objective is to assign two people to each room such that the assignment is stable. An assignment is stable if there are no two people assigned to different rooms who prefer each other over their current partners. Give an instance of the Hotel Partner Problem for which there is no stable solution.

3. Hudson Shipping Lines, Inc., is a shipping company that owns n ships and provides service to n ports. Each of its ships has a *schedule* that says, for each day of the month, which of the ports it's currently visiting, or whether it's out at sea. (You can assume the "month" here has m days, for some $m > n$.) Each ship visits each port for exactly one day during the month. For safety reasons, HSL Inc. has the following strict requirement:

($*$) No two ships can be in the same port on the same day.

The company wants to perform maintenance on all the ships this month, via the following scheme. They want to *truncate* each ship's schedule: for each ship S_i, there will be some day when it arrives in its scheduled port and simply remains there for the rest of the month (for maintenance). This means that S_i will not visit the remaining ports on its schedule (if

any) that month, but this is okay. So the truncation of S_i's schedule will simply consist of its original schedule up to a certain specified day on which it is in a port P; the remainder of the truncated schedule simply has it remain in port P.

Now the company's question to you is the following: Given the schedule for each ship, find a truncation of each so that condition $(*)$ continues to hold: no two ships are ever in the same port on the same day.

Show that such a set of truncations can always be found, and give an algorithm to find them.

Example. Suppose we have two ships and two ports, and the "month" has four days. Suppose the first ship's schedule is

port P_1; *at sea*; *port* P_2; *at sea*

and the second ship's schedule is

at sea; port P_1*; at sea; port* P_2

Then the (only) way to choose truncations would be to have the first ship remain in port P_2 starting on day 3, and have the second ship remain in port P_1 starting on day 2.

4. Consider an instance of the stable matching problem with n men and n women. Let X and Y be some two stable matchings for this instance. We now construct a new pairing Z as follows. For each man m, if X pairs him with a woman w_x^m and Y pairs him with a woman w_y^m then in Z the man is paired with the woman he prefers most among w_x^m and w_y^m. Note that w_x^m and w_y^m could be the same woman. Prove or disprove that Z is a stable matching.

Now consider a pairing Z' in which a man m is paired with the woman he prefers the least among w_x^m and w_y^m. Prove or disprove that Z' is a stable matching.

5. For each of the parts below, list the functions in increasing asymptotic order. In some cases functions may be asymptotically equivalent (that is $f(n)$ is $\Theta(g(n))$). In such cases indicate this by writing $f(n) = g(n)$. When one is asymptotically strictly less than the other (that is, $f(n)$ is $O(g(n))$ but $f(n)$ is not $\Theta(g(n))$), express this as $f(n) \leq g(n)$. For example, given the set:

$$n^2 \qquad n \log n \qquad 3n + n \log n,$$

the first function is $\Theta(n^2)$ and the other two are $\Theta(n \log n)$, and therefore the answer would be

$$n \log n \;=\; 3n + n \log n \;\leq\; n^2.$$

You are *not* required to show your work; just writing your answer suffices.

(a)	$(3/2)^n$	$3^{(n/2)}$	$2^{(n/3)}$
(b)	$\lg n$	$\ln n$	$\lg(n^2)$
(c)	$n^{\lg 4}$	$2^{\lg n}$	$2^{(2 \lg n)}$
(d)	$\max(50n^2, n^3)$	$50n^2 + n^3$	$\min(50n^2, n^3)$
(e)	$\lceil n^2/20 \rceil$	$\lfloor n^2/20 \rfloor$	$n^2/20$

6. Consider the following basic problem. You're given an array A consisting of n integers $A[1], A[2], \ldots, A[n]$. You'd like to output a two-dimensional n-by-n array B in which $B[i, j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$ – that is, the sum $A[i] + A[i+1] + \cdots + A[j]$. (The value of the array entry $B[i, j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what is the output of these values.)

Here is a simple algorithm to solve this problem.

```
for (i=1; i <=  n; i++)
  for (j=i+1; j <= n; j++)
    Add up array entries A[i] through A[j]
    Store the result in B[i,j]
```

a. For the above code fragment give a bound of the form $O(f(n))$ on its running time on an input of size n. Justify your answer.

b. For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)

c. Although the algorithm you analyzed in parts (a) and (b) is the most natural way to solve the problem – after all, it just iterates through the relevant entries of the array B, filling in a value for each – it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$.