CSCI 532 Homework 1

Brittany Boles, Redempta Manzi, Madie Munro

September 5th, 2024

Question 1 (2 marks): Suppose h ranks s as their top choice and s ranks h as their top choice. Is it true that (h, s) appears in every stable matching? Explain your answer.

Answer: It is true that (h, s) always appears in every stable matching if h ranks s as their top choice and s ranks h as their top choice because:

1. Hospitals propose to students in decreasing order of preference, meaning some hospital h prefers some student s over another student s' if h prefers s.

2. Students can only trade up hospitals, meaning some student s will trade up some other hospital h' for a hospital h if s prefers h.

h will only propose to s' if s rejects h. However, since s prefers h and vice versa, h will make an offer to s before s', which leads to s accepting h's proposal before s' has a chance to accept by statement 1.. Furthermore, s will only accept h' if h' proposed to s before h (i.e., s initially accepted h''s proposal because s was previously unmatched). By statement 2., and the fact that there are some pairs (h', s') still unmatched, s will eventually trade up h' for h when h proposes to s because s prefers h over h'.

$\therefore$ (h, s) always appears in every stable matching if h ranks s as their top choice and s ranks h as their top choice. $\square$

Question 2 (2 marks): Suppose h ranks s as their last choice and s ranks h as their last choice. Is it possible that (h, s) appears in a stable matching (assuming $n > 1$)? Explain your answer.

Answer: Yes it is possible, consider proof by contradiction. Lets assume (h, s) can't be a stable match. Consider the case where $s1$'s top choice is $h1$ and vice versa. Lets assume h has $s1$ as its top student as well and s has $h1$ as its top hospital. $h1$ will pick $s1$, and h can't "steal" $s1$, so h pairs with s. s can never upgrade as every hospital is paired. This is therefore a stable match, which is a contradiction to our claim above. Therefore, it is possible (h, s) will appear in a stable matching even if they are each other's last choice. □

h1	s1	s
h	s1	s

s1	h1	h
s	h1	h

Question 3 (2 marks): Suppose we have two television networks, whom we'll call A and B. There are n prime-time programming slots, and each network has n TV shows. Each network wants to devise a schedule—an assignment of each show to a distinct slot—so as to attract as much market share as possible. Here is the way we determine how well the two networks perform relative to each other, given their schedules. Each show has a fixed rating, which is based on the number of people who watched it last year; we'll assume that no two shows have exactly the same rating. A network wins a given time slot if the show that it schedules for the time slot has a larger rating than the show the other network schedules for that time slot. The goal of each network is to win as many time slots as possible. Suppose in the opening week of the fall season, Network A reveals a schedule S and Network B reveals a schedule T. On the basis of this pair of schedules, each network wins certain time slots, according to the rule above. We'll say that the pair of schedules (S, T) is stable if neither network can unilaterally change its own schedule and win more time slots. That is, there is no schedule S' such that Network A wins more slots with the pair (S', T) than it did with the pair (S, T); and symmetrically, there is no schedule T' such that Network B wins more slots with the pair (S, T') than it did with the pair (S, T). The analogue of Gale and Shapley's question for this kind of stability is the following: For every set of TV shows and ratings, is there always a stable pair of schedules? Resolve this question by doing one of the following two things:

A give an algorithm that, for any set of TV shows and associated ratings, produces a stable pair of schedules; or

B give an example of a set of TV shows and associated ratings for which there is no stable pair of schedules.

Answer:
Consider the potential schedules for A of tv shows $S = s1, s2, s3, s4, s5$ with rankings (1,4,5,8,9) and $S' = (4, 5, 8, 9, 1)$
Also consider the potential schedules for B of tv shows $T = t1, t2, t3, t4, t5$ with rankings (2,3,6,7,10) and $T' = (2, 6, 7, 10, 3)$

(S, T) => A wins 2 prime slots and B wins 3
(S, T') => A wins 4 prime slots and B wins 1
(S', T) => A wins 1 prime slots B wins 4
(S', T') => A wins 2 prime slots B wins 3

Consider the possible schedule S for network A and the possible schedule T for network B. Notice that no matter what schedule is picked by the networks (including schedules that shuffle the show ordering around), there will always be an unstable matching of schedules. It's easy to think of other possible schedules and see that they will always want to swap, because each network have a chance of upping their score. This stems from the fact that there is an odd number of tv shows and one network's shows are not always ranked higher than the other

(i.e., (0,1,2,3,4) and (5,6,7,8,9)), so there can't be a stable match. Someone will always gain by having a different tv show order. Other situations exist where we can get unstable matchings, however only this one is needed to counter the claim "for any set of TV shows and associated ratings, there exists a stable pair of schedules."

Another example:

Suppose network A has a schedule S with 2 shows with the ratings 70 and 90, and let's suppose network B has a schedule T also with 2 shows with the ratings 60 and 80. That is:

$$S = (70, 90)$$
$$T = (60, 80)$$

With this schedule pair, A wins all the time slots since A's first show a_1 is paired with B's first show b_1. Same with a_2 and b_2. Now, suppose B proposes a new schedule T' where its two shows were swapped. That is:

$$S = (70, 90)$$
$$T' = (80, 60)$$

Then B wins one slot over A. However, network A would want to switch their scheduling of shows to win all slots, so it proposes a new schedule S' such that

$$S' = (90, 70)$$
$$T' = (80, 60)$$

This process of show swapping will keep repeating since any new schedules proposed by both networks will produce pairs that are unstable (A has a schedule that will win more TV slots, so it'll try to take then all even if B proposes a schedule that takes one slot for itself). $\square$

Question 4 (5 marks): Implement a Divide-and-Conquer based polynomial multiplication algorithm (the $n^{1.59}$ algorithm discussed in class). Your program should work with polynomials of degree several thousand, at least. Also implement the basic $O(n^2)$ algorithm and use that to verify that your algorithm is getting the correct answers (at least for smaller values of n). Plot the average running time of the algorithm vs. n (test on random polynomials); most programming languages let you get the current system time, so have your program collect the running times. (Average over multiple trials to get a smooth plot.). Ensure your code is well-documented. Provide screenshots of running the program in your written submission.

```
PS C:\Users\madie\OneDrive\Desktop\CSCI 532\Algorithms> & C:/Users/madie/anaconda3/python.exe
s/HW1/HW_1_code.py"
Polynomial #1:
8 + 3x^1 + 1x^2 + 15x^3

Polynomial #2:
6 + 5x^1 + 2x^2 + 8x^3


New polynomial after multiplication (Naive approach):
48 + 58x^1 + 37x^2 + 165x^3 + 101x^4 + 38x^5 + 120x^6

New polynomial after multiplication (Simple Divide-and-Conquer approach):
48 + 58x^1 + 37x^2 + 165x^3 + 101x^4 + 38x^5 + 120x^6


Runtime of Naive Approach: 1.001037836074829

Runtime of Divide-and-Conquer Approach: 1.0004119873046875
```

Figure 1: Output comparing two approaches for polynomial multiplication (simple example)

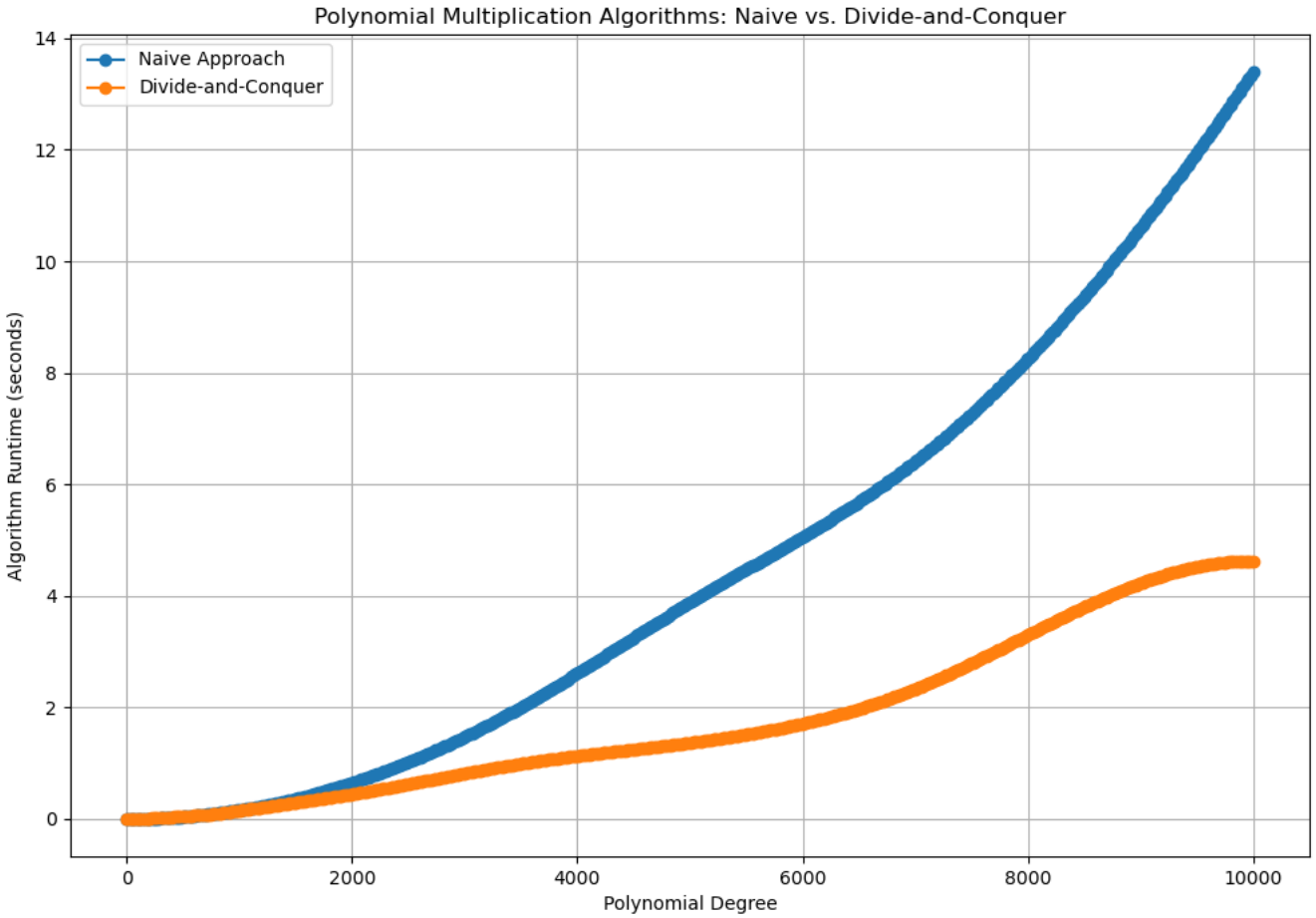

Figure 2: Simulated runtimes between the two approaches with random polynomials of different degrees