

# CPSC 320 2018W2: Assignment 1

Please submit this assignment via GradeScope at <https://gradescope.com>, by **Monday January 21 at 10pm**. Assignments, submitted within 24 hours after the deadline will be accepted, but a penalty of 15% will be applied.

Be sure to identify everyone in your group if you're making a group submission. Reminder: groups can include a maximum of three students; we strongly encourage groups of two. Your group must make a **single** submission via one group member's account, marking all other group members in that submission **using GradeScope's interface**. Your group's submission **must**:

- Consist of a single, clearly legible file uploadable to GradeScope with clearly indicated solutions to the problems. (PDFs produced via L<sup>A</sup>T<sub>E</sub>X, Word, Google Docs, or other editing software work well. Scanned documents will likely work well. **High-quality** photographs are OK if we agree they're legible.)
- Include prominent numbering that corresponds to the numbering used in this assignment handout. Put these **in order** starting each problem on a new page, ideally. If not, **very clearly** and prominently indicate which problem is answered where.
- Include at the start of the document the **ugrad.cs.ubc.ca e-mail addresses** of each member of your team. (Please do **NOT** include your name on the assignment, however. If you don't mind private information being stored outside Canada and want an extra double-check on your identity, include your student number rather than your name.)
- Include at the start of the document the statement: "All group members have read and followed the guidelines for groupwork on assignments in CPSC320. As part of those rules, when collaborating with anyone outside my group, (1) I and my collaborators took no record but names (and GradeScope information) away, and (2) after a suitable break, my group created the assignment I am submitting without help from anyone other than the course staff." (Go read those guidelines! They are posted on the course website, along with the links to the assignments.)
- Include at the start of the document your outside-group collaborators' ugrad.cs.ubc.ca IDs, but **not** their names. (Be sure to get those IDs when you collaborate!)

Before we begin, a few notes on pseudocode throughout CPSC 320: Your pseudocode must communicate your algorithm clearly, concisely, correctly, and without irrelevant detail. Reasonable use of plain English is fine in such pseudocode. You should envision your audience as a capable CPSC 320 student unfamiliar with the problem you are solving. If you choose to use actual code, note that you may **neither** include what we consider to be irrelevant detail **nor** assume that we understand the particular language you chose. (So, for example, do not write `#include <iostream>` at the start of your pseudocode, and avoid idiosyncratic features of your language like Java's ternary (question-mark-colon) operator.)

Remember also to **justify your answers**. Justifications/explanations need not be long or formal, but should be clear and specific.

---

## 1 SMP and Gale-Shapley

1. Let  $\#E_k$  be the number of employers matched at the end of the  $k$ th iteration of the While loop on some execution of the G-S algorithm. If there are at least  $k + 1$  iterations of the While loop, must it be the case that  $\#E_k < \#E_{k+1}$ ?
2. Let  $\text{Set-}A_k$  be the set of applicants matched at the end of the  $k$ th iteration of the While loop on some execution of the G-S algorithm. If there are at least  $k + 1$  iterations of the While loop, must it be the case that  $\text{Set-}A_k \subseteq \text{Set-}A_{k+1}$ ?
3. Consider a variant of SMP in which each of  $n$  employers has exactly **two** positions, and the number of applicants is  $2n$ . As a function of  $n$ , how many valid solutions are there?

## 2 Progressing Towards Goodness in Gale-Shapley

Prove the following claim, from part 8.1 of the Stable Matching Problem Part II worksheet. You could use a proof by induction, with a structure that is modeled on the proof of problem 5.2 of the worksheet.

**Claim:** At the end of every iteration  $k$  of the While loop of Algorithm G-S, every employer  $e$  has only considered applicants that it ranks at least as high as  $\text{best}(e)$ . Moreover, if  $e$  has considered  $\text{best}(e)$  on or before iteration  $k$ , then  $e$  is matched with  $\text{best}(e)$  at the end of iteration  $k$ .

## 3 SMP with Identical Preference Lists

For positive integers  $n$ , let  $I_n$  be the instance of SMP with  $n$  employers and  $n$  applicants such that every employer has the same preference list, and also every applicant has the same preference list, namely:

$$e_i : a_1, a_2, \dots, a_n \qquad a_i : e_1, e_2, \dots, e_n.$$

Let  $S$  be the (infinite) set of instances  $\{I_n, n > 0\}$ .

1. Write down the instance  $I_2$ . (No justification needed.)
2. Show a good solution for the instance  $I_2$ . (No justification needed.)
3. Prove that for any  $n > 0$ , in the instance  $I_n$  we have  $\text{best}(e_i) = a_i$ , for  $1 \leq i \leq n$ .

## 4 Faster or Slower

Suppose that some algorithm  $A$  has running time  $f(n)$  and that algorithm  $B$  has running time  $g(n)$ , on all inputs of size  $n$ . Assume that  $f$  and  $g$  are functions  $\mathbb{N} \rightarrow \mathbb{N}^+$ , and that  $\lim_{n \rightarrow \infty} f(n)$  and  $\lim_{n \rightarrow \infty} g(n)$  are both infinity. Explain whether each statement in parts 2 and 3 below is true or false. Part 1 is already done for you.

1. For some choice of  $g(n)$  with  $g(n) \in \Omega(f(n) \log n)$ :
  - (a)  $A$  is faster than  $B$  on all sufficiently large inputs.  
**SOLUTION** True. Choosing  $g(n) = f(n)(\lceil \log_2 n \rceil + 1)$  satisfies the condition that  $g(n) \in \Omega(f(n) \log n)$ . For this choice,  $g(n) > f(n)$  for all  $n$ , and so  $B$  is slower than  $A$  on all inputs.
  - (b)  $A$  is slower than  $B$  on all sufficiently large inputs.  
**SOLUTION** False. For all choices of  $g$  with  $g(n) \in \Omega(f(n) \log n)$ , we have that  $g(n) > f(n)$  for sufficiently large  $n$ . So  $B$  is slower than  $A$  on all sufficiently large inputs.

- (c)  $A$  is faster than  $B$  on some inputs, and slower than  $B$  on other inputs.

**SOLUTION** True. Let  $n_1 < n_2$  be such that  $f(n_1) > 1$  and  $f(n_2) > 1$ . Choose  $g(n) = 1$  for  $n \leq n_2$  and  $g(n) = f(n) \lceil \log_2 n \rceil$  for  $n > n_2$ . Then  $g(n) \in \Omega(f(n) \log n)$ ,  $g(n) > f(n)$  for all  $n > n_2$ , and  $g(n) < f(n)$  for  $n_1$  and  $n_2$ . So  $B$  is faster than  $A$  on inputs  $n_1$  and  $n_2$ , while being slower than  $A$  on all inputs of size greater than  $n_2$ .

2. For some choice of  $g$  such that  $g(n) \in \Theta(f(n))$ :
  - (a)  $A$  is faster than  $B$  on all sufficiently large inputs.
  - (b)  $A$  is slower than  $B$  on all sufficiently large inputs.
  - (c)  $A$  is faster than  $B$  on some inputs, and slower than  $B$  on other inputs.
3. For some choice of  $g$  such that  $g(n) \in o(f(n))$ :
  - (a)  $A$  is faster than  $B$  on all sufficiently large inputs.
  - (b)  $A$  is slower than  $B$  on all sufficiently large inputs.
  - (c)  $A$  is faster than  $B$  on some inputs, and slower than  $B$  on other inputs.

## 5 Comparing Substrings

Here you'll evaluate running times of algorithms whose input size is expressed using two parameters.

Let  $T[1..n]$  and  $T'[1..n]$  be strings of length  $n$ , over a finite alphabet (For example,  $T$  and  $T'$  might be over the alphabet  $\{A, C, G, T\}$ , and represent DNA strands.) A function `Match` indicates whether or not a letter of  $T$  matches a letter of  $T'$ . That is, for  $1 \leq i, j \leq n$ ,

$$\begin{aligned} \text{Match}(i, j) &= 1, & \text{if } T[i] = T'[j] \\ &= 0, & \text{otherwise.} \end{aligned}$$

Fix  $k, 1 \leq k \leq n$ . For  $1 \leq i, j \leq n - k + 1$ , the *score* of any two length- $k$  substrings  $T[i..i + k - 1]$  and  $T'[j..j + k - 1]$  of  $T$  and  $T'$  respectively is given by

$$\sum_{l=0}^{k-1} \text{Match}(i + l, j + l).$$

Algorithm *Compute-Scores* below computes all scores and stores them in a two-dimensional array called `Score`. Assume that calls to function `Match` take  $\Theta(1)$  time and that array `Score` has already been created.

**Algorithm** *Compute-Scores* ( $T[1..n], T'[1..n], k$ )

//  $T$  and  $T'$  are length- $n$  strings and  $1 \leq k \leq n$

For  $i$  from 1 to  $n - k + 1$

  For  $j$  from 1 to  $n - k + 1$

    // compute score of  $T[i..i + k - 1]$  and  $T'[j..j + k - 1]$  and store in `Score[i, j]`

`Score[i, j]` =  $\sum_{l=0}^{k-1} \text{Match}(i + l, j + l)$

1. What terms below describe the worst-case running time of this algorithm? Check all answers that apply. Here, a term  $\Theta(f(n, k))$  is correct if for any choice of  $k$  in the range  $[1..n]$ , the algorithm runs in  $O(f(n, k))$  time, and for some choice of  $k$  in the range  $[1..n]$  (where  $k$  may be expressed as a function of  $n$ ), the algorithm runs in  $\Omega(f(n, k))$  time.

- $\Theta(k^3)$ 
     
   $\Theta((n - k)^2 k)$ 
     
   $\Theta(n^2 k)$ 
     
   $\Theta(n^3)$

- 
2. Modify the algorithm of part 1, to improve the runtime by a factor of  $k$ .
  3. What is the worst-case running time of your algorithm of part 2? Try to find as simple an expression as possible.