Assignment 2

CS 577 *Asymptotic Analysis - Data Structures - Amortized Analysis* Fall 2024

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Related Readings: `http://pages.cs.wisc.edu/~hasti/cs240/readings/`

Name: ____________________ Wisc id: ____________________

- **Purpose of Homework:**
  - Algorithm design and analysis, like any skill, can only be developed through consistent practice and feedback. Whether it's cooking, playing basketball, integration, gardening, interviewing, or teaching, theoretical knowledge alone is not sufficient. The comfortable feeling of "Oh, sure, I get it" after following a well-presented lecture or hearing a TA explain a homework solution is a seductive, yet dangerous trap. True understanding comes from doing the thing—by actually solving the problems yourself.
  - The homework assignments are your opportunity to practice. Lectures, textbooks, office hours, labs, and guided problem sets are designed to build intuition and provide justification for the skills we want you to develop. However, the most effective way to develop those skills is by attempting to solve the problems on your own. The process is far more important than the final solution.
  - Expect to get stuck. It's normal to have no idea where to start on some problems. That's why you have access to a textbook, lecture slides, and discussions. The journey of wrestling with the problem is an essential part of the learning process.

Scoring Guidelines

1) 10 points						
2a) 1 point	2b) 4 points	2c) 5 points	2d) 5 points	2e) 5 points	2f) 2 points	2g) 8 points
3a) 5 points	3b) 7.5 points	3c) 7.5 points	3d) 5 points			
4a) 2 points	4b) 2 points	4c) 2 points	4d) 2 points	4e) 2 points		
5a) 5 points	5b) 10 points					

The first exercise is a bonus. The score is calculated as:

$$\text{Score} = 100 \times \left(\frac{\sum_{k=2}^{5} \text{Exercise}_k}{80} + \frac{\text{Exercise}_1}{20}\right)$$

Homework Guidelines

- **Collaboration and Academic Integrity:**
  - You are encouraged to work together on homework problems, but you must list everyone you worked with for each problem.
  - You must write everything in your own words and properly cite every external source you use, including ideas from other students. The only sources that you are not required to cite are the official course materials (lectures, notes, homework solutions).
  - Plagiarism is strictly prohibited. Using ideas from other sources or people without citation is considered plagiarism. Copying verbatim from any source, even with citation or permission, is also considered plagiarism. Don't cheat.
- **Submission Instructions:**
  - Submit your homework solutions as PDF files on Gradescope. Submit one PDF file per numbered homework problem.
  - Gradescope will not accept other text file formats such as plain text, HTML, LaTeX source, or Microsoft Word (.doc or .docx).
  - Homework submitted as images (.png or .jpg) will not be graded.
  - Each submitted PDF file should include the following information prominently at the top of the first page: [your full name]_[course title]_[homework assignment number].pdf
- **Solution Writing:**
  - When writing an algorithm, a clear description in English is sufficient. Pseudo-code is not required.
  - Ensure that your algorithm is correct by providing a justification, and analyze the asymptotic running time of your solution. Even if your algorithm does not meet the requested time bounds, you may receive partial credit for a correct, albeit inefficient, solution.
  - Pay close attention to the instructions for each problem. Partial credit may be awarded for incomplete or partially correct answers.

Asymptotic Notation

1. Sort the following functions in increasing order of their growth rates, i.e., find a sequence $g_1, g_2, g_3, \ldots$ such that $g_1 = O(g_2)$, $g_2 = O(g_3)$, etc. In this sequence, highlight the functions that have the same growth rate.

$$\begin{array}{cccc} n^2 & 2^{(\log_2 n)^3} & \frac{\log(n!)}{(\log n)^3} & n2^{2^{2^{100}}} \\ \log\left(\frac{n}{\log n}\right) & \sum_{k=1}^{n} k^3 & \sqrt{n}! & \log^4 n \\ \binom{n}{6} & \frac{n^3}{\log^{10} n} & (\log_2(n))^{\log_2(n)} & \log\left(\binom{2n}{n}\right) \\ n\sum_{k=0}^{n}\binom{n}{k} & \binom{2n}{n/4} & \sum_{k=1}^{n} k2^k & \sum_{k=1}^{n} k2^{-k} \end{array}$$

Stable Matching Problem & Gale-Shapley Algorithm

2. *Source: Kleinberg, Jon. Algorithm Design (Chapter 1).* In class, we discussed the Gale-Shapley algorithm. In this exercise, we will explore the properties of the algorithm under the assumption that we have n hospitals and n doctors.

Gale–Shapley (preference lists for hospitals and doctors)

```
Initialize $M$ to an empty matching.
While (some hospital $h$ is unmatched and hasn't proposed to every doctor)
    $d \leftarrow$ first doctor on $h$'s list to whom $h$ has not yet proposed.
    If ($d$ is unmatched)
        Add $h$–$d$ to matching $M$.
    Else If ($d$ prefers $h$ to current partner $h'$)
        Replace $h'$–$d$ with $h$–$d$ in matching $M$.
    Else
        $d$ rejects $h$.
Return stable matching $M$.
```

(a) Execute the Gale-Shapley algorithm (resolve any ties alphabetically).

Hospitals' Preferences

Hospital A:	Doctor X > Doctor Y > Doctor Z
Hospital B:	Doctor Y > Doctor Z > Doctor X
Hospital C:	Doctor Z > Doctor X > Doctor Y

Doctors' Preferences

Doctor X:	Hospital C > Hospital A > Hospital B
Doctor Y:	Hospital A > Hospital C > Hospital B
Doctor Z:	Hospital B > Hospital C > Hospital A

(b) Swap one preference and observe how the algorithm behaves differently.

Hospitals' Preferences

Hospital A:	Doctor X > Doctor Y > Doctor Z
Hospital B:	Doctor X > Doctor Z > Doctor Y
Hospital C:	Doctor Z > Doctor X > Doctor Y

Doctors' Preferences

Doctor X:	Hospital C > Hospital A > Hospital B
Doctor Y:	Hospital A > Hospital C > Hospital B
Doctor Z:	Hospital B > Hospital C > Hospital A

(c) Prove the correctness of the following invariant: if a doctor is matched, they remain matched and can only switch to a hospital that they prefer more than their current match. (Complete the proof.)

(d) Show that the algorithm produces a valid matching, i.e., no hospital is matched to more than one doctor, and no doctor is matched to more than one hospital.

(e) Prove by contradiction that all hospitals are matched in the final matching.

(f) Prove that all doctors are matched in the final matching.

Definition: Given a perfect matching M, a hospital h and a doctor d form an unstable pair if both h prefers d to its current doctor and d prefers h to its current hospital.

(g) Consider a pair (h, d) that is in the final matching M^*. Prove by contradiction that (h, d) cannot be an unstable pair.

Hint: Consider the case where hospital h has proposed to d and the case where h has never proposed to d.

Hogwarts: Amortized Analysis & Data Structures

3. *At Hogwarts, students use magical belts to carry and organize their enchanted items. These belts have different functionalities depending on the house the student belongs to. One day, Professor Snape discovered that Harry Potter and Hermione Granger were using their beloved Gryffindor magical belts to solve a magical challenge. As punishment, he forced them to use only the Slytherin magical belts, which had completely different functionality. At the same time, Professor McGonagall, wanting to teach Draco Malfoy a lesson, made him use the Gryffindor magical belts instead of the familiar Slytherin belts. Now, Harry and Hermione had to find a way to use the Slytherin belts to simulate the functionality of the Gryffindor belts, while Malfoy had to do the opposite.*

Gryffindor Magical Belt

- `G-InitBelt`: Creates a new belt, ready to hold items.
- `G-BeltIsEmpty`: Checks if the belt has no items.
- `G-RemoveItem`: Takes the last item placed on the belt.
- `G-AddItem`: Places an item on the belt, which goes at the start of the belt.
- `G-BeltSize`: Tells the wizard how many items are currently on the belt.

Slytherin Magical Belt

- `S-InitBelt`: Creates a new belt, ready to hold items.
- `S-BeltIsEmpty`: Checks if the belt has no items.
- `S-RemoveItem`: Takes the first item placed on the belt.
- `S-AddItem`: Places an item on the belt, which goes at the end of the belt.
- `S-BeltSize`: Tells the wizard how many items are currently on the belt.

Your mission is to help Harry, Hermione, and Malfoy adapt to their new circumstances by designing the correct moves and spells.

- Describe how to implement a Slytherin belt using two Gryffindor belts, so that each Slytherin operation requires $O(1)$ amortized belt operations.
- Describe how to implement a Gryffindor belt using two Slytherin belts, so that each stack operation requires $O(\sqrt{n})$ at the worst case.

(a) Implementation of **SlytherinUsingGryffindorBelts**:

(b) Analysis of Runtime Complexity using "Gringotts" Banking/Accounting Method:

(c) Implementation of **GryffindorUsingBeltsSlytherin**:

(d) Analysis of Runtime Complexity:

4. *At Hogwarts, magical devices are often used by students to assist with their studies and daily tasks. One such device is Hermione Granger's enchanted hourglass, a unique magical artifact she uses to measure time and perform a variety of spells. This hourglass isn't just for keeping track of time—it's enchanted to trigger specific magical charms as the sand shifts from one compartment to another. The hourglass has many compartments, each representing a different spell or charm. As the sand shifts into a new compartment, it activates a charm. The complexity of the charm depends on the compartment number, with more advanced spells being triggered as the sand moves to higher compartments. Here's how Hermione's enchanted hourglass works:*

SandShift(Hourglass[0..∞]) :

```
Step 1: $i \leftarrow 0$
Step 2: While Hourglass[$i$] = 1 (meaning all the sand has shifted from this compartment)
    Start –WhileBlock–
    Set Hourglass[$i$] ← 0 (reset the current compartment)
    $i \leftarrow i + 1$ (move to the next compartment)
    End –WhileBlock–
Step 3: Set Hourglass[$i$] ← 1 (start filling the next compartment with sand)
Step 4: TriggerCharm($i$) (trigger the magical charm associated with this compartment)
```

The time required to cast each charm increases with the compartment number. Let $T(i)$ represent the time it takes for the charm in compartment i to complete. Now, imagine Hermione uses the enchanted hourglass n times, starting with all the sand at the bottom. The time it takes for each **SandShift** depends on the complexity of the **TriggerCharm**. **The Challenge:** Help Hermione determine the average (amortized) time it takes to complete each **SandShift** for the following scenarios:

(a) What is the amortized time per SandShift if $T(i) = 42$? **Spell:** Lumos - A basic lighting spell. Every time the sand shifts, the room lights up with the same bright intensity.

(b) What is the amortized time per SandShift if $T(i) = 2^i$? **Spell:** Expelliarmus - A disarming charm. As the sand shifts into higher compartments, the spell's power doubles, disarming increasingly powerful magical objects.

(c) What is the amortized time per SandShift if $T(i) = 4^i$? **Spell:** Expecto Patronum - A protective charm that conjures a Patronus. As the sand moves to higher compartments, the Patronus becomes stronger, requiring more magic and focus.

(d) What is the amortized time per SandShift if $T(i) = \sqrt{2^i}$? **Spell:** Riddikulus - A spell that turns something scary into something silly. The complexity increases, but at a slower rate than exponential, making each higher compartment a bit more challenging.

(e) What is the amortized time per SandShift if $T(i) = \frac{2^i}{i+1}$? **Spell:** Protego - A shield charm. The spell's power increases exponentially with each compartment, but the increase is tempered by the precision needed to direct it, represented by the $i + 1$ in the denominator.

Renting a house with Ideal Lake Monona View

5. Professor Manolis has just arrived in Madison, Wisconsin, and is looking for renting a house with a beautiful view of Lake Monona. Madison is known for its stunning lakes, particularly Lake Monona, and the Monona Terrace offers one of the best views in the city. However, not every building near the Monona Terrace has an unobstructed view of the lake. Professor Manolis wants to make sure that his

new house will have a perfect view of the lake. To help him out, suppose you are given an array `A[1..n]` that stores the height of n buildings on a city block, indexed from west to east. Building i has a good view of Lake Monona if and only if every building to the east of i is shorter than i.

Below is an algorithm that computes which buildings have a good view of Lake Monona. What is the running time of this algorithm?

Listing 1: GoodCandidate Algorithm

```
GoodCandidate(A[1..n]):
    initialize a stack S
    for i ← 1 to n
        while (S not empty and A[i] > A[Top(S)])
            Pop(S)
        Push(S, i)
    return S
```

(a) Give a quick explanation of a quadratic bound

(b) Don't turn in: Your professor likes only linear time algorithms