Northeastern University
CS5800 – Algorithms
Spring 2025, Prof. Ryan Rad

Homework #1

Homework 1 is based on the topics of week 1 and 2. There are **5** problems, problem 1 is *three* marks, and problems 2-5 each worth *ten* marks.
Note that there is a practice-only challenge problem at the end that is NOT required to be submitted.

Submit a single PDF file on Canvas, with the answer to all the problems you have tackled in this homework. Make sure you label your HomeWork #1 submission appropriately - e.g. RyanRad-HW1.pdf. Make sure you start your answer to each question on a brand new page! If you are using a pen and paper, please neatly handwrite your solutions on standard A4 paper, with your name(s) and question # at the top of each solution.

For Q2-a, the code can be submitted via a shareable URL to your Google Colab's link. Read more about Google Colab, it's very convenient and easy to work with.

Important Note:
All problems must be completed **individually**. While perfect solutions are needed for full marks, partial marks will be awarded for demonstrating meaningful progress.

To ensure fairness and integrity, two rules must be followed. Violating either rule constitutes plagiarism and will result in an **F** for the course:

1. **Write Your Own Solutions**:
   While you may have high-level discussions with classmates, the articulation of your thought process, implementation and writing must be completed independently, in your own words, and without collaboration.

2. **No Internet Assistance**:
   Struggling through challenging problems is essential to learning. Using the Internet, AI-assisted technology for solutions or hints undermines your growth and is prohibited. Use AI-assisted technology to understand the underlying concepts not solving the problems. If you need help, please seek support from our TAs during office hours or on Piazza.

Problem #1 – Happy Question!

Hello! I'm the Happy Question, a unique puzzle in your journey of complexity and code. You earn 8 marks just by reading me! Isn't that a delightful linear algorithmic problem to solve?

In the world of algorithms, where efficiency and speed often reigns supreme, it's easy to get caught up in the pursuit of optimal solutions. But what about the journey itself? Sometimes, the most rewarding experiences aren't about the final destination, but the process we embark upon. This concept, known as delayed gratification, is the idea of forgoing immediate pleasure for a larger, more satisfying reward in the future.

Invitation: Before we dive into our algorithmic puzzle, let's explore the idea of delayed gratification a bit further. Take a few minutes to read about the "Marshmallow Experiment" here: `https://jamesclear.com/delayed-gratification`

Puzzle: Now, let's consider the world of algorithms through the lens of delayed gratification. Can you identify and describe an algorithm that:

- Prioritizes immediate gratification: Perhaps an algorithm that seeks a quick solution, even if it's not the most efficient or accurate.
- Embraces delayed gratification: An algorithm that might take more time or computational resources upfront, but ultimately leads to a superior outcome.

Wondering what should you submit for this question? In a single paragraph (3-8 lines), explain how you've practiced immediate or delayed gratification in your life and how it influenced your decisions or outcomes.

Remember, in the midst of searching for optimal solutions, it's okay to pause and appreciate the journey. After all, in Algorithm Land, the Happy Question reminds us that sometimes, being present and engaged is a reward in itself!

Problem #2 – Hybrid Sort

Background In this assignment, we'll explore an advanced hybrid sorting algorithm that combines the strengths of multiple sorting techniques to achieve both efficiency and guaranteed worst-case performance. This algorithm is used in some standard library implementations of sorting functions.

Algorithm Description:

The algorithm works as follows:

– Start with a divide-and-conquer approach similar to Quicksort.
– Track the recursion depth during the sorting process.
– For small sub-arrays (typically less than 16 elements), use an insertion-based sorting method.
– If the recursion depth exceeds a certain threshold (typically $2\log_2(n)$, where n is the number of elements), switch to a heap-based sorting method.

Note: You don't need to implement Insertion sort, Quicksort or Heapsort from scratch.

(a) Implement the described hybrid sorting algorithm in a programming language of your choice (Preferably Python). The main idea is that you're creating a wrapper or manager algorithm that decides which sorting method to use based on specific conditions. You can use libraries for the underlying sorting implementations (Quicksort, Heapsort, Insertion sort) or implement them yourself. For Heapsort, you can refer to the heapq packages. Make use you understand who these sorting algorithms work.

(b) Analyze the time complexity of this hybrid algorithm. Discuss best-case, average-case, and worst-case scenarios.

(c) Conduct an empirical analysis:
– Generate arrays of various sizes (e.g., $1,000$, $10,000$, $100,000$, $500,000$ elements) with random, nearly sorted, and reverse sorted data.
– Compare the performance of this hybrid algorithm with standard Quicksort, Heapsort, and Insertion Sort). Present your results in tables or graphs.

(d) Briefly discuss general trends and observation from (c) and suggest two practical real-world applications where this hybrid algorithm might be particularly useful.

Problem #3 – Guess My Number

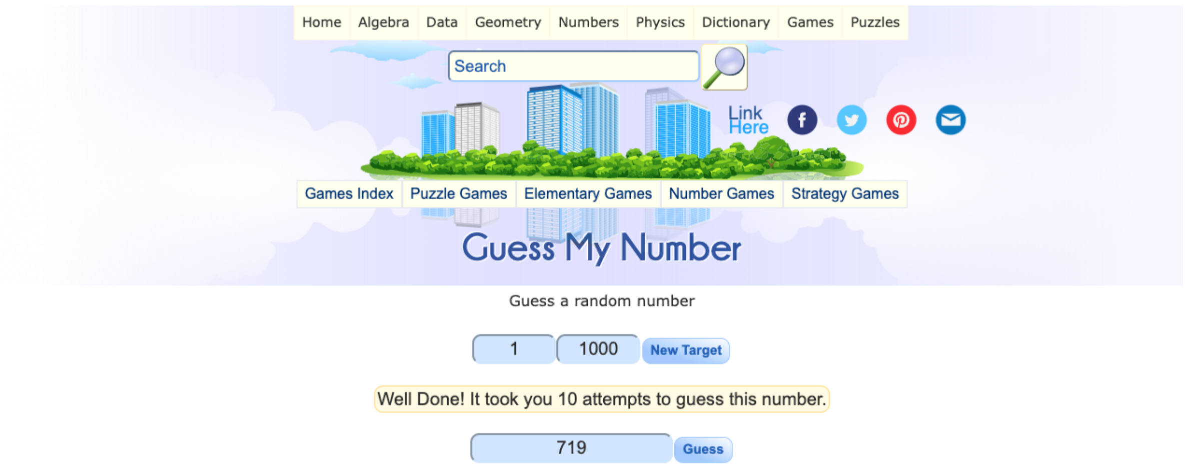
Play one or more games of “Guess My Number”, which you can do so on this website:

`https://www.mathsisfun.com/games/guess_number.html`

What you will do is change the Range of numbers from 1 to 1000, and then click on Start. The computer will pick a random integer between 1 and 1000, with each number equally likely to be chosen.

After each guess, the computer will tell you whether your guess was correct, whether it was too high, or whether it was too low. The game stops when you have correctly guessed the computer’s number.

For example, when I did this game, I was able to get the computer’s number in 10 guesses.



Well Done! It took you 10 attempts to guess this number.

(a) Attach a screenshot of you winning this game in at most 10 guesses. (No proof or explanation is necessary – all you need to do is insert a .jpg image, just as I did above.)

(b) Suppose the range of numbers is between 1 and n, where n is a positive integer. If $n = 15$, prove that the game can always be won in at most 4 guesses, no matter what the computer’s number is. Clearly explain how your “guessing algorithm” works.

(c) Let $T(n)$ be the maximum number of guesses required to correctly identify an integer between 1 and n. For example, you have shown above that $T(15) = 4$. Determine a recurrence relation for $T(n)$, and apply the Master Theorem to show that $T(n) = \Theta(\log n)$.

(d) For each integer n, let $A(n)$ be the *average* number of guesses required to correctly guess the computer’s number, where each of the target numbers from 1 to n is equally likely to be chosen. For example, you can show that $A(15) = \frac{49}{15}$ using your algorithm from part (b).

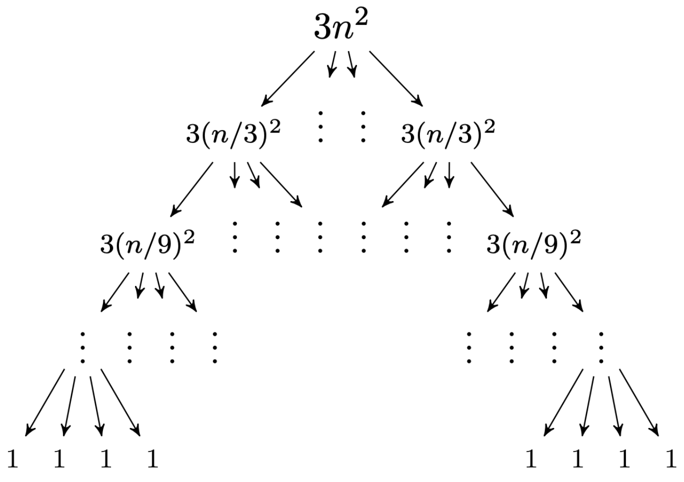
Determine the exact value of $A(1023)$, carefully justifying all of the steps in your calculation.

Problem #4 – Tree recurrence

(a) Draw the recurrence tree for the following recurrence formula (only the first three levels):

$$T(x) = \begin{cases} 1 & \text{if } x = 1 \\ T(n/2) + T(n/3) + n & \text{otherwise} \end{cases}$$

(b) Write the recurrence for the following recurrence tree:



(c) Let $f(n)$ and $g(n)$ be two functions, defined for each positive integer n. By definition, $f(n) = O(g(n))$ if there exist positive constants c and n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. Determine if the following statements are correct or incorrect, and briefly justify your answer in each case.

- $4^n = O(2^n)$
- $n^{1000} = O(2^n)$

Problem #5 – Divide and Conquer: Optimal Timing in Investment

An investment company has an AI-powered model that predicts stock prices for n consecutive days. Given these predictions, determine the maximum possible profit that can be made by buying and selling a single stock. **Note**: Due to potential tax implications associated with day trading, we assume that only a single purchase and sale are allowed. Your solution should just return the maximum possible profit.

Sample Input: [100, 115, 90, 120, 85, 130]
Sample Output: 45

Explanation: The maximum profit can be achieved by buying at 85 and selling at 130, resulting in a profit of 130 - 85 = 45.

(a) **Naive Approach:** Describe (single paragraph) the idea and logic behind your brute force solution and its time complexity.

(b) **Pseudocode:** Now, design a DAC algorithm and provide detailed pseudocode for your algorithm, ensure that the pseudocode is comprehensive, and add comments when necessary to enhance its readability. Design a $O(n \log n)$ DAC algorithm for this problem to qualify for full credit.

(c) **Algorithm Analysis:** Analyze the time complexity of your proposed DAC algorithm from part (b). Write the recurrence relation for your algorithm and apply the Master Theorem to determine its complexity.

Extra Challenge Question (*practice ONLY*)

Closest Pair Problem in Supply Chain Management

Attention: This question is crafted for students seeking an extra challenge.
You're welcome to **skip** this and submission is **NOT** required.

In our class, we explored the standard closest pair problem in a 2D plane. Now, let's consider a practical application in the field of supply chain management. Imagine a scenario with numerous suppliers and vendors dispersed across a geographical region. These suppliers might be factories or farms producing goods, while vendors could be markets or retail stores requiring these goods. A key goal in this context is to enhance supply chain efficiency by reducing transportation costs and time. Minimizing travel distance is crucial not only for cost reduction but also for lowering environmental impact.

(a) **Modified Algorithm:** Discuss how you would modify the standard closest pair algorithm to address the needs of the stakeholders in the scenario described above. Focus on the aspects of the algorithm that would be adapted or enhanced to optimize matching between suppliers and vendors, taking into account factors like distribution of locations and varying supply and demand.

(b) **Algorithm Analysis:** Analyze the time complexity of your proposed algorithm from part a. Write the recurrence relation for your algorithm and apply the Master Theorem to determine its complexity. Compare this with the complexity of the standard closest pair algorithm, and explain any differences or similarities.

(c) **Pseudocode:** Provide a detailed pseudocode for your modified algorithm, similar to the one discussed in class. Ensure that the pseudocode is comprehensive and reflects all the modifications you proposed in part a. Highlight any steps that specifically address the supply chain scenario.