**Northeastern University**
CS5800 – Algorithms
Fall 2024, Prof. Ryan Rad

---

# Homework #2

Homework 2 is based on the topics of week 3 and 4 (and a bit of week 2).

**Important Note:** I would like to give you the luxury of designing your own challenges on Dynamic Programming (See Question 4). So again, there are FOUR questions only, where problems $1-3$ each worth *eight* marks while problem 4 worth *ten* marks. Therefore, HW2 is out of 34 marks.

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 #2 submission appropriately - e.g. RyanRad-HW2.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.

Please note that all problems are to be completed **individually**. While a solution must be perfect to receive full marks, we will be generous in awarding partial marks for incomplete solutions that demonstrate progress.

So that there is no ambiguity, there are two non-negotiable rules. A violation of either rule constitutes plagiarism and will result in you receiving an F for this course.

(a) If you meet with a classmate to discuss one of the Individual Problems, the articulation of your thought process (i.e., what you submit), must be an individual activity, done in your own words, away from others. Please remember that the solution-writing process is where so much of your learning will occur in this course: much more than anything we do in class, and even more than the time you spend on solving the problems. Do not be surprised if it takes you 3 to 5 times as long to write up a solution than it takes you to actually solve the problem.

(b) This Problem Set has been designed to be challenging, because struggling through problems is how we learn best. Your educational experience is cheapened by going online and finding the solution to a problem; even using the Internet to look for a "small hint" is unacceptable. In return, your wonderful TAs will be readily available during the office hours (drop-in or by appointment), and upon request, we will happily support you in the process.

## Problem #1 – A New Sorting Algorithm

Two Northeastern University students Elizabeth and James have proposed the following new sorting algorithm:

```
New-Sort (A, p, r)
1  if A[p] > A[r]
2      exchange A[p] with A[r]
3  if p + 1 < r
4      k = ⌊(r − p + 1)/3⌋          // round down
5          New-Sort (A, p, r − k)   // first two-thirds
6          New-Sort (A, p + k, r)   // last two-thirds
7          New-Sort (A, p, r − k)   // first two-thirds again
```

(a) Prove that the call NEW-SORT$(A, 1, n)$ correctly sorts the input array $A[1 : n]$, where $n$ represent length of A.

(b) Give a recurrence for the worst-case running time of NEW-SORT and a tight asymptotic $\Theta$-notation) bound on the worst-case running time.

(c) Compare the worst-case running time of NEW-SORT with that of insertion sort, merge sort, heapsort, and quicksort. Do the students deserve a straight A in the course?

## Problem #2 – Players′ Rating

In our Algorithms class, 8 students are avid chess players.

Each of these 8 students has a "rating" that measures their quality as a chess player. The higher the rating, the better they are.

For the purposes of this problem, assume that everyone keeps their chess rating a private secret; however, when two players have a chess match, the person with the higher rating wins 100% of the time.

These highly-competitive students have asked you to create various *comparison-based* algorithms to resolve some questions they have about their relative chess ratings.

For each of these questions, your algorithm will be a sequence of matches (e.g. A vs. B, C vs. D) that will solve the problem in the most efficient way.

(a) The students want you to sort them according to their chess rating, from best ($1^{\text{st}}$) to worst ($8^{\text{th}}$).

Prove that there does *not* exist an algorithm that is guaranteed to solve this problem in 15 or fewer matches.

(b) The students want you to determine which player has the highest chess rating and which player has the lowest chess rating.

Create an algorithm that is guaranteed to solve this problem in 10 matches, and clearly explain why there does *not* exist an algorithm that is guaranteed to solve this problem in 9 or fewer matches.

(c) The students want you to determine which player has the highest chess rating and which player has the second-highest chess rating.

Create an algorithm that is guaranteed to solve this problem in 9 matches, and clearly explain why there does *not* exist an algorithm that is guaranteed to solve this problem in 8 or fewer matches.

## Problem #3 – Counting Sort and Stability

Let $A$ be an array of $n$ non-negative integers, where the maximum element is $k$.

If $k = O(n)$, then Counting Sort is an $O(n)$ time algorithm to sort our array $A$. This is a significant improvement over comparison-based sorting algorithms which are at best $O(n \log n)$.

(a) Let $A = [4, 5, 0, 1, 3, 4, 3, 4, 3, 0, 3]$. Walk through each step of the Counting Sort algorithm on this input array, to produce the correct output of $[0, 0, 1, 3, 3, 3, 3, 4, 4, 4, 5]$.

(b) We say that a sorting algorithm is *stable* provided that whenever there are two equal elements ($x$ and $y$), where $x$ appears before $y$ in the input, then $x$ must appear before $y$ in the sorted output.

For example, in the above input array $A$, if we highlight the three numbers marked 4 using the colors red, white, and blue (in that order), then in the sorted output array, our 4's will be listed red, white, and blue (in that order).

Clearly explain why the Counting Sort algorithm is stable.

(c) Consider Quicksort and Heapsort. For each of these two sorting algorithms, explain whether the algorithm is stable. If the answer is YES, briefly explain why. If the answer is NO, provide a simple counterexample to show that stability is not guaranteed.

## Problem #4 – Build a Portfolio on Dynamic Programming

LeetCode (www.leetcode.com) is a popular website for Northeastern MSCS students, especially when preparing for job interviews.

`https://leetcode.com/problemset/algorithms/`

There are over a thousand "coding challenges" from which students can practice and improve their skills in Algorithm Analysis and Design, and the website supports numerous programming languages, including C, Java, and Python.

In this Programming Project, you will create a **portfolio** consisting of TWO LeetCode problems on Dynamic Programming you will solve over the next two weeks. You can choose ANY set of TWO problems from the following site (click on "Dynamic Programming" to filter the results), but see the following note first.

**Important Note:** You can pick anything excluding the DP problems **except** the problems we will discuss in class during Week 5 – 6:

- Fibonacci or Climbing Stairs
  - Leetcode 70. Climbing Stairs
  - Leetcode 509. Fibonacci Number
- Rod Cutting
  - Leetcode 1547. Minimum Cost to Cut a Stick (similar problem)
- Longest Common (or Increasing) Sub-sequence
  - Leetcode 1143. Longest Common Subsequence
  - Leetcode 300. Longest Increasing Subsequence
- Activity-Selection
  - Leetcode 435. Non-overlapping Intervals (closest equivalent)
- Knapsack or Coin Change
  - Leetcode 322. Coin Change
  - Leetcode 416. Partition Equal Subset Sum (0/1 Knapsack variant)
- House Robber
  - Leetcode 198. House Robber

**NOTE:** To maximize your learning and problem-solving skills, we encourage you to explore these concepts through fresh challenges. The idea is for you to get more exposure to problem-solving, so for your own benefit, please stay away from these problems!

[**Read Carefully**] This HW offers a lot of flexibility but there are four important rules:

(I) If the problem took you less than $30-40$ minutes to solve, you may not include it in your portfolio. (Since then the question was an *exercise* for you, rather than a *problem* or a *challenge*.)

(II) You may only include problems you have solved after **Sep 22, 2024**.

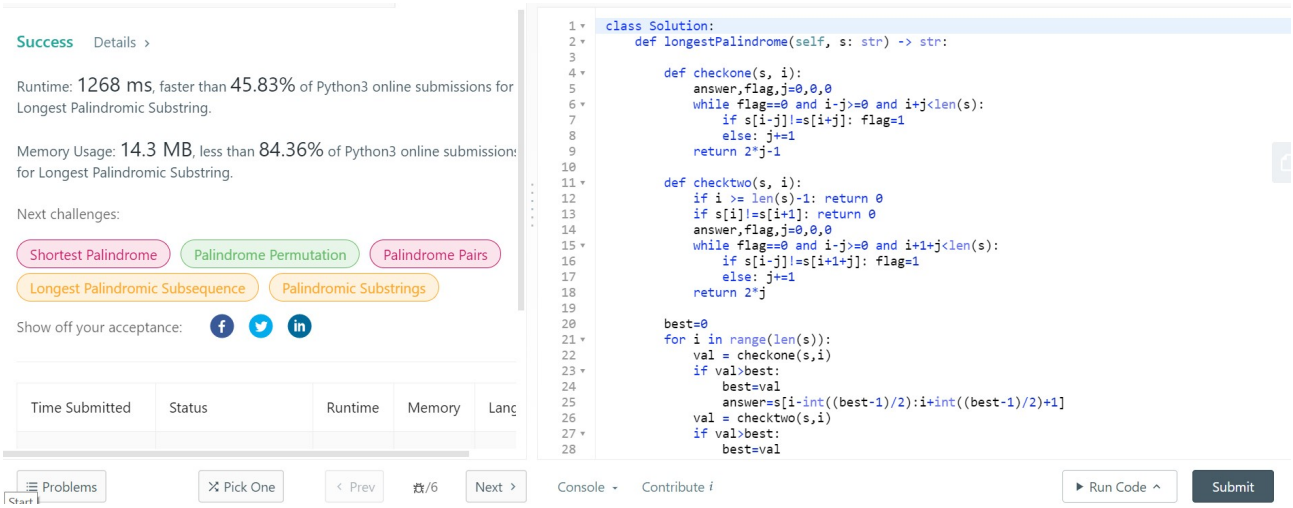
(III If you click on the LeetCode "Solution" or "Discuss" button *before* you solve the problem or look at any online resources for hints to solve a LeetCode problem, especially sites such as Chegg, GitHub, Stack Overflow, and Quora, you cannot include it in your portfolio. (You may look at these sites *after* you solve the problem.)

(IV) Under no circumstance may you use Co-Pilot, LLMs or any other AI-assisted programming tools.

Here is how your portfolio will be assessed.

(a) **(8 marks)** For each of the problems you are including in your portfolio (two problems), provide the problem number, problem title, difficulty level, and the screenshot (showing all the details, submission time, etc) of you getting your solution accepted by LeetCode. You will receive up to 4 marks for each problem you submit.

Here is an example from a sample portfolio: Longest Palindromic Substring (medium)




You will get full credit for *any* correct solution accepted by LeetCode, regardless of the difficulty of the problem, and regardless of how well your runtime and memory usage compares with other LeetCode participants.

(b) **(2 marks)** For **one** of the two problems you solved, explain the various ways you tried to solve this problem, telling us what worked and what did not work. Describe what insights you had as you eventually found a correct solution. Reflect on what you learned from struggling on this problem, and describe how the struggle itself was valuable for you. Reflect on what might be causing the obstacle (e.g. lack of familiarity with a particular data structure, lack of knowledge about a fast and efficient algorithm, etc.)

**Note:** For your reflections, write a *minimum of 250 words.*

## Extra Challenge Question (*practice ONLY*)

### Alice in a Coding Interview

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

Alice is a second-year M.S. of Computer Science student in our beautiful Vancouver Campus. She has recently participated in a coding interview with a tech company in Vancouver.

During the interview, she faces the following question which also happens to be one of the main topics of Week 3 of our Algorithm class.

- Let's say you are running an e-commerce platform, and you want to identify the top k most popular products in your store based on customer purchase history. Given an integer array *nums* and an integer $k$, return the $k$ most (frequently sold) popular product IDs. You may return the answer in any order.

**Example 1:**

```
Input: nums = [1,1,1,2,2,3], k = 2
Output: [1,2]
```

**Example 2:**

```
Input: nums = [1], k = 1
Output: [1]
```

**Constraints:**

- `1 <= nums.length <= 10^5`
- `-10^4 <= nums[i] <= 10^4`
- `k` is in the range `[1, the number of unique elements in the array]`.
- It is **guaranteed** that the answer is **unique**.

If you were Alice, how would you approach this question?

(a) First, try to quickly come up with a brute-force solution. This should not take more than 10 minutes. Your only concern should be the correctness of your algorithm, not its efficiency.
- Write a simple pseudocode for your algorithm, briefly describe the time complexity of your algorithm, and code it up in Python.

(b) Now, try to see if you can optimize your algorithm in (a), and come up with a more efficient solution. There is no requirement on how fast your solution needs to be, it just needs to be better than the one given in (a).
- Write a simple pseudocode for your algorithm, briefly describe the time complexity of your algorithm, and code it up in Python.

(c) Finally, with the learning from the previous two steps, I want you to come up with a decent solution for this problem. Note that your algorithm's time complexity must be better than $O(nlogn)$, where $n$ is the array's size.
- Write a simple pseudocode for your algorithm, briefly describe the time complexity of your algorithm, and code it up in Python.