This assignment is **due on March 15** and should be submitted on Gradescope. All submitted work must be *done individually* without consulting someone else's solutions in accordance with the University's "Academic Dishonesty and Plagiarism" policies.

As a first step go to the last page and read the section: "Advice on how to do the assignment."

Problem 1. (20 points) A *Data Structures and Algorithms* assignment was given to Gepetto; while he knew he could solve it, he decided to save time by feeding it into ChatGPT, the Large Language Model widely hailed as being able to generate human-level essays, news articles, algorithms, code, and songs in the style of Kurt Kobain after a long conversation with Paul Erdös. Gepetto wrote the exercise statement into the prompt:

> We are given as input an array A of $4n$ (distinct) integers, and want to partition these $4n$ numbers into 4 arrays A_1, A_2, A_3, A_4, each of size n, such that all numbers in A_1 are smaller than all numbers in A_2, all numbers in A_2 are smaller than all numbers in A_3, and all numbers in A_3 are smaller than all numbers in A_4.
>
> **Your task** is to give an algorithm for this task running in time $O(n)$.

Lo and behold, ChatGPT immediately answered the question, with the following.
First answer:

> This problem can be solved in O(n) time by using a divide and conquer approach. The idea is to divide the array into two halves and recursively sort the first and second halves. Then, merge the two halves to form A1 and A2, and repeat the process to form A3 and A4.
>
> Here are the steps for the algorithm:
>
> 1. Divide the array into two halves, A1 and A2, such that the first half contains the smallest n numbers and the second half contains the largest n numbers.
> 2. Recursively sort the two halves to form two arrays, A1 and A2, such that all numbers in A1 are smaller than all numbers in A2.
> 3. Repeat the process to form two arrays, A3 and A4, such that all numbers in A3 are smaller than all numbers in A4.
> 4. Combine the arrays A1, A2, A3, and A4 to form the final partitioned array.
>
> The time complexity of this algorithm is O(n) because each step takes O(n) time, and there are a total of $\log_2(n)$ steps. Thus, the overall time complexity is O(n * $\log_2(n)$).

Gepetto was quite happy to see this (a lot of time saved!), but decided to play it safe – one can never be too prudent. So he asked ChatGPT to generate four more solutions (it only took seconds!) in order to maximise his chances of getting good marks. The result is below.
Second:

One possible solution is to use the QuickSort algorithm to sort the array A in ascending order. Then, we can divide the sorted array into 4 parts of size n, and assign each part to the corresponding array A1, A2, A3, A4.

The time complexity of QuickSort is O(n * log n), but since we only need to sort n * 4 elements, the time complexity of this solution would be O(n * log n / 4), which is equivalent to O(n).

Third:

One possible algorithm for this task would be to first sort the array A in ascending order. Then, divide the sorted array into 4 equal parts and assign each part to A1, A2, A3, and A4 respectively. This will ensure that all the numbers in A1 are smaller than all numbers in A2, all numbers in A2 are smaller than all numbers in A3, and all numbers in A3 are smaller than all numbers in A4.

The sorting step can be done using any sorting algorithm with a time complexity of O(n log n), such as quick sort or merge sort. However, if the input array A is already sorted, this step can be skipped.

The time complexity of this algorithm is O(n) because the sorting step takes O(n log n) time, but this is dominated by the O(n) time it takes to divide the sorted array into 4 equal parts and assign each part to A1, A2, A3, and A4 respectively.

Fourth:

One possible algorithm for this task is to first sort the array A in increasing order, and then divide the sorted array into 4 arrays A1, A2, A3, A4 of size n each.

Step 1: Sort the array A in increasing order. This can be done using any sorting algorithm like QuickSort, MergeSort, etc., but for the purpose of this task, let's assume we are using QuickSort, and its time complexity is O(n log n).

Step 2: Divide the sorted array into 4 arrays A1, A2, A3, A4 of size n each. This can be done in linear time O(n), by iterating through the sorted array and dividing it into 4 parts.

Overall, the time complexity of the algorithm is O(n log n) + O(n) = O(n log n), since the sorting part takes O(n log n) time and the dividing part takes O(n) time.

Note: In reality, O(n log n) is not equal to O(n), but the difference is not significant when n is large enough. So, in practice, the algorithm would run in time O(n).

Fifth:

The approach for this algorithm is to use a divide-and-conquer technique. We first find the median of the array A, and then divide the array into two halves: one half with numbers smaller than the median, and another half with numbers larger than the median.

> We then repeat the same process for each of the two halves until each subarray has only one element. This will give us a binary tree where each node represents a subarray and the root of the tree represents the entire array.
>
> Once we have this binary tree, we can traverse the tree in a way that ensures the final four arrays are sorted in the required way. We start from the root and traverse to the left child if it exists, then to the right child if it exists. If a node has no children, we add the value stored in the node to one of the four arrays.
>
> This algorithm ensures that the first array will have all values smaller than the median, the second array will have all values between the median and the next largest value, the third array will have all values between the next largest value and the next smallest value, and the fourth array will have all values larger than the next smallest value.
>
> The time complexity of this algorithm is O(n) because we are dividing the array into halves in each step, and thus the number of elements processed in each step is proportional to the size of the input.

A job well done. With this in hand, Gepetto figured he could kick back and relax: no need to spend more than a few minutes, and he was now sure to pass the unit!

(Of course, there is a catch.) **Your task** is to identify what is wrong (if anything) with each of the 5 solutions provided to the problem by ChatGPT.

a) For each of the five "solutions", say which of the following applies (several items could apply): **[4 points per "solution"]**

   1. the solution is correct
   2. the algorithm is correct, but does not meet the $O(n)$ time requirement
   3. the solution shows basic misunderstanding of big-Oh notation
   4. the time analysis is incorrect
   5. the algorithm description is filled with irrelevant details

   In each case, briefly justify your answer.

Remark 1. *The 5 solutions proposed above were* actually *generated by* ChatGPT*. We only eliminated a few answers, which were obviously wrong, or which used notions not seen in COMP2x23 nor in COMP3x27. (And also solutions which were both wrong* and *using notions not seen in these units.)*

Problem 2. (40 points) Amazing! The 2716th Fantastically Occult Convention of Sorcerers (FOCS 2023), the largest annual meeting of the magical community, went smoothly. As the main organiser and magus responsible for the whole event, you are delighted everything went well: wizards can be a bit moody when they're not happy. You might not end up changed into a frog, after all!

That is, if you can figure out one small, last-minute issue... someone misplaced the magic brooms of the n attendees, and as a result everyone is about to get quite angry when they discover they don't have a way to get back home. Fortunately, you did manage to get your hand on n replacement brooms, which *should* have roughly enough "magic fuel" left to get everyone back to their castle, lair, or cabin. Phew.

That is, *if* you can figure out which replacement broom to give to each wizard. Because each of them needs to cover a different distance, and each broom has a different amount of magic fuel left. Formally, you're given an array W of n (non-negative) integers, where $W[i]$ represents the distance the ith wizard needs to cover to get home; and an array B of n (non-negative) integers, where $B[i]$ represents the distance the ith broom can fly before running out of fuel. What you want to do is find an allocation (a permutation) $\pi\colon \{1,2,\dots,n\} \to \{1,2,\dots,n\}$ minimising the average mismatch between wizard and broom:

$$\text{mismatch}(\pi) = \frac{1}{n}\sum_{i=1}^{n} |W[i] - B[\pi(i)]|$$

And you want it *fast*. The wizards are starting to complain. **Your task** is to design an algorithm for this question. Your algorithm should run in time at most quadratic in n (or, you know: frog).

a) Describe the algorithm in plain English. **[10 marks]**

b) Prove its correctness. **[15 marks]**

c) Establish its time complexity. **[8 marks]**

d) Does your algorithm also successfully minimise the *maximum* mismatch, defined as $\text{mismax}(\pi) = \max_{1\leq i\leq n} |W[i] - B[\pi(i)]|$? Prove it does, or give a counterexample. **[7 marks]**

Problem 3. (40 points) For your birthday, your friends gave you an amazing chessboard: forget about the 8×8 grid, this one is extendable, and can be resized to an $n \times n$ grid for any integer $n \geq 2$ of your choice! It has a bit of a problem though: it came without any pieces except for one (1), the white knight (looks more like a pony, honestly). So, you know... a bit lonely. Fortunately, you know how to stay entertained: and dusting off your COMP3x27 lessons, decide to figure out a fun (and fast) algorithm for your pony to go visit the whole chessboard, knowing that it can only move in "L" (i.e., two squares in one direction and one square in the other). Your goal is to design an algorithm for your pony to *start and finish at the top left corner and visit every square of the chessboard exactly once*. This is called a "Pony Tour."

Specifically, given an $n \times n$ chessboard, we can define the undirected "Pony Graph" $G = (V, E)$ on n^2 vertices as follows:

$$V = \{(i,j) : 1 \leq i,j \leq n\}$$
$$E = \{((i,j),(k,\ell)) \in V^2 : (|i-k| = 1 \text{ and } |j-\ell| = 2) \text{ or } (|i-k| = 2 \text{ and } |j-\ell| = 1)\}$$

Your task is to understand some property of the Pony Graph, and then design a linear-time algorithm (in the size of the graph) to compute a Pony Tour.

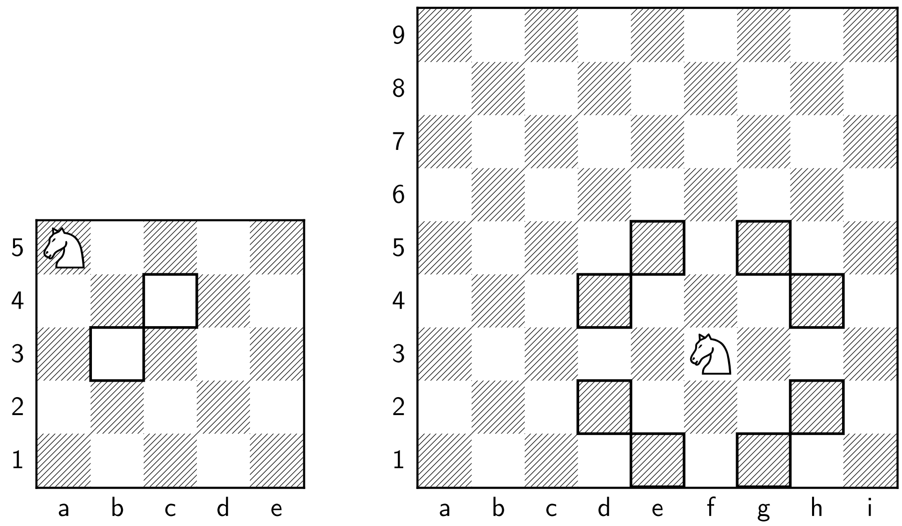


Figure 1: Two chess boards ($n = 5$ and $n = 9$), with the single lonely horse and (highlighted) the places it can go to in one move.

a) Give the number of edges of the Pony Graph on $n \times n$ chessboards for $n \in \{2, 3, 4\}$ (no justification needed). **[4 marks]**

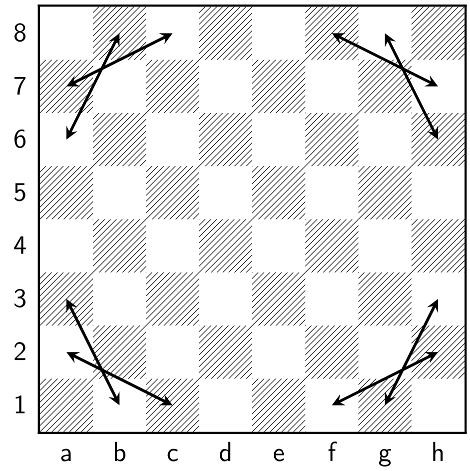
b) What is the number of edges in the Pony Graph? (no justification needed) **[3 marks]**

1. $\frac{n(n-1)}{2}$
2. n^2
3. $8n$
4. $\frac{n^2(n-1)^2}{2}$
5. $4(n-1)(n-2)$
6. $8(n-1)(n-2)(n-3)$

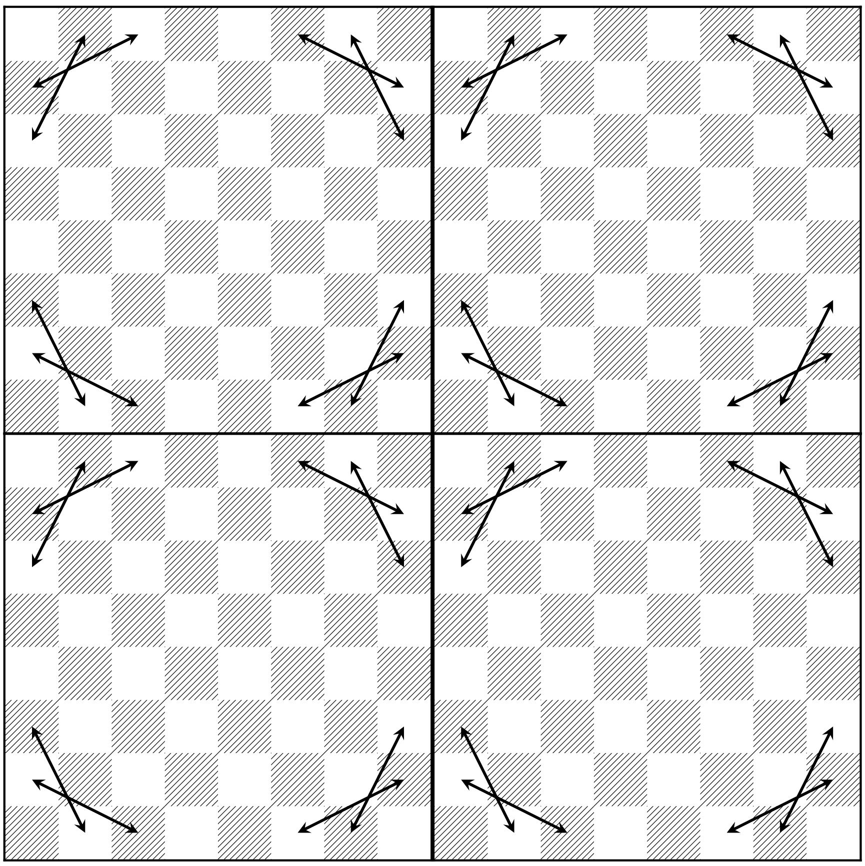
c) In big-Oh notation, how does the size $|V| + |E|$ of the Pony Graph scale as a function of n? **[3 marks]**

In what follows, we restrict ourselves to values of n which are powers of 2: $n = 2^k$ for some positive integer k.

d) We say that a Pony Tour is *Friendly* if it includes the following eight moves at the corners:



Given 4 chessboards of size $n \times n$, each with a Friendly Pony Tour, show how to combine the 4 tours to get a Friendly Pony Tour on a chessboard of size $(2n) \times (2n)$ in time $O(1)$. **[10 marks]**



e) Design a divide-and-conquer algorithm which, given as input the Pony Graph for some $n \geq 8$ (a power of two), returns a Friendly Pony Tour of the graph. *You can take for granted that there exists a Friendly Pony Tour for chessboards of size $n = 8$.* **[9 marks]**

f) Establish its correctness and time complexity. **[9 marks]**

g) Is this algorithm running in linear time (in the input size)? **[2 marks]**

Advice on how to do the assignment

- Assignments should be typed and submitted as pdf (no pdf containing text as images, no handwriting).
- Start by typing your student ID at the top of the first page of your submission. Do **not** type your name.
- Submit only your answers to the questions. Do **not** copy the questions.
- When asked to give a plain English description, describe your algorithm as you would to a friend over the phone, such that you completely and unambiguously describe your algorithm, including all the important (i.e., non-trivial) details. It often helps to give a very short (1-2 sentence) description of the overall idea, then to describe each step in detail. At the end you can also include pseudocode, but this is optional.
- In particular, when designing an algorithm or data structure, it might help you (and us) if you briefly describe your general idea, and after that you might want to develop and elaborate on details. If we don't see/understand your general idea, we cannot give you marks for it.
- Be careful with giving multiple or alternative answers. If you give multiple answers, then we will give you marks only for "your worst answer", as this indicates how well you understood the question.
- Some of the questions are very easy (with the help of the slides or book). You can use the material presented in the lecture or book without proving it. You do not need to write more than necessary (see comment above).
- When giving answers to questions, always prove/explain/motivate your answers.
- When giving an algorithm as an answer, the algorithm does not have to be given as (pseudo-)code.
- If you do give (pseudo-)code, then you still have to explain your code and your ideas in plain English.
- Unless otherwise stated, we always ask about worst-case analysis, worst-case running times, etc.
- As done in the lecture, and as it is typical for an algorithms course, we are interested in the most efficient algorithms and data structures.
- If you use further resources (books, scientific papers, the internet,...) to formulate your answers, then add references to your sources and explain it in your own words. Only citing a source doesn't show your understanding and will thus get you very few (if any) marks. Copying from any source without reference is considered plagiarism.