Computation II
Algorithms & Data Structures

Project Guidelines

Augusto Santos and Diogo Rasteiro

April, 2025

1 Project Rules

This project aims to evaluate the knowledge and skills acquired throughout the semester, particularly your ability to apply algorithms, data structures and analyze the underlying overall running time complexity in the design and implementation of a simple deck of cards game.

- **Group composition**— Students must work in groups of up to 4 members. Group composition must be submitted on Moodle under the corresponding activity between May 1st and May 8th.
- **Deadline**— The final project must be submitted by **June 8th at 23:59**. Late submissions will incur a penalty of **2 points per day**. Projects submitted more than **3 days late** (i.e., after June 11th) will not be considered for evaluation.
- **Submission**— You must submit both your project code and a short accompanying report. Submission must be made through Moodle. A detailed description of the required deliverables is provided in Subsection 3.6 of the present document.
- **Defense**— All projects must be defended in person. Project defenses will take place on **June 11th and 12th**. Each group will be required

to answer questions about their design decisions, complexity analysis, and code implementation. Be prepared to answer *technical* questions. A link to schedule your defense time slot will be provided beforehand. **Failure to attend the defense will result in your project not being evaluated**.

2 Project Description

The main goal of this project is to develop a simple game involving a standard deck of cards, with functionality to detect specific *poker* hands. Students should implement the necessary classes and functions to ensure the game's functionality. Students should also analyze the overall complexity/performance of their implementation.

Your code should include classes to simulate a standard deck of 52 playing cards, storing the relevant information and methods required to execute the game's logic. The game must run in the Python console, allowing the player to draw a user-defined number of cards (between 3 and 15).

Once a hand is drawn, the goal is to detect special card combinations (or poker hands), display them, and inform the user. For this purpose, and as discussed in class, remember that a sorted list tends to decrease the complexity of many distinct downstream tasks. Therefore, the drawn cards must be sorted first by their values, either in ascending or descending order – this choice is up to you – and printed to the console. After this sorting step, the game must then evaluate the sorted hand and detect whether it contains any poker hands according to standard poker rules.

Before evaluating the hand, the player must choose one of the available sorting algorithms to apply. The drawn cards are then sorted and printed to the console, including information on their value and suit.

All-in-all, the identified poker hand(s) should be communicated to the player along with the complexity, and the program must also keep track of the total number of each type of combination encountered during the current run, displaying this tally, as well.

After each round, the player should be prompted to either play again or exit. If the player chooses to continue, a new round is started. If they choose to exit, the program should terminate *gracefully*–a courteous message and exit cleanly.

In summary, your code must perform the following steps:

1. Upon starting, create/instantiate a standard 52-card deck and shuffle it randomly.
2. Allow the user to draw a hand of cards, specifying the desired number of cards (between 3 and 15).
3. The user should choose the sorting algorithm for the cards.
4. Sort and display the drawn hand.
5. Detect and announce any poker hand(s) found in the hand.
6. Update and display a counter for each type of hand encountered during the session.
7. Prompt the player to either:
   (a) Play again: reshuffle the deck and repeat from step 2.
   (b) Exit: terminate the program.

Additionally, you are also invited to develop an extra functionality in your code beyond what is written here, in exchange for a bonus point. This can be anything, ranging from a visual upgrade to performing extra actions with the cards. The only limit is your creativity.

3 Project Requirements

3.1 Suggested Classes

Below is a list of classes you are encouraged to implement to ensure correct program functionality. These are suggestions, not hard constraints. What matters most is that all core requirements are fulfilled. If your approach requires more (or fewer) classes, dictionaries, or other data structures, you are free to use them-provided that the functionality remains sound and the implementation is well-structured.

That being said, the suggestions below **are simply a shallow base to help guide you** through the initial building of your code. You *will* need to build more than this for a successful project.

3.1.1 Card

`Card` objects must store information regarding their value and their suit. You do not need to consider Jokers or any other type of cards. In other words, the state of the objects instantiated by the `Card` class (defined within the method `__init__()`) should include

Values: 1 (Ace), 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen and King.

Suits: Spades, Clubs, Hearts and Diamonds.

3.1.2 Deck

A `Deck` must contain a full set of 52 unique cards, corresponding to every combination of value and suit. To store a sequence of `Card` objects, you must implement the necessary functionalities (or methods) for the deck, including:

- **Shuffling** — randomizing the order of the cards in the deck.
- **Drawing** — returning a specific number of cards.

3.1.3 Hand

`Hand` is an optional class that serves to encapsulate the logic of dealing with the hand of cards that a player possesses. As such, there is no need to make it a separate class, and everything required here can be achieved through independent functions. Drawn hands must be between 3 to 15 cards.

Regardless of your choice, the following functionalities are required:

- Creating and storing a hand drawn from the deck.
- Sorting the cards present.
- Detecting poker hands.
- Discarding a hand and drawing a new one.

3.1.4 Game

Much like the previous section, `Game` is an optional class meant to symbolize the game itself and the overarching functions that run it. These include:

- Controlling `Deck` and `Hand` objects and functions.
- Display all necessary information to the player.
- Register the player's choices via console input and execute the necessary commands.
- Ending the game.

3.2 Sorting Algorithms

When displaying the cards present in a hand to the player, they *must* be sorted according to their values. This sorting can be done either smallest-to-largest or largest-to-smallest - we leave this decision up to you. However, the player must be allowed to choose the sorting algorithm to use during each round.

You must implement the following four sorting algorithms from scratch (i.e., without using Python's built-in `sorted()` or `sort()` functions):

- Heap Sort
- Binary Insertion Sort
- Merge Sort
- A sorting algorithm of your choice *not taught in class*, with average time complexity of at most $O(n \log n)$

Your code should include a prompt allowing the player to select which sorting algorithm to use before displaying the hand.

Hand Name	Description
One Pair	Two cards of the same rank.
Two Pair	Two distinct pairs of cards.
Three of a Kind	Three cards of the same rank.
Straight	Five consecutive cards of mixed suits (Aces can be high or low).
Flush	Five cards of the same suit, not necessarily in sequence.
Full House	Three cards of one rank combined with a pair of a different rank.
Four of a Kind	Four cards of the same rank.
Straight Flush	Five consecutive cards all of the same suit.
Royal Flush	A straight flush consisting of 10, J, Q, K, A all in the same suit.

Table 1: Valid Poker Hands

3.3 Poker Hand detection

When a hand is drawn, your code must be able to detect any present poker hands in it, and print the information on the console. The list of hands to be detected is in table 1.

Note:

- In Straights, Aces can be high (meaning they come after K) or low (meaning they come before 2). However, they cannot be both (so Q, K, A, 2, 3 is not a valid Straight, for example).
- If there are no poker hands detected, that information should be reported.
- There is no need to detect high card as a hand type.

3.4 Bonus Functionality

As mentioned previously, you are also tasked with adding a certain bonus functionality to your program. This is meant to be a creative step, and you are free to add anything you want, provided it does not interfere with the previously stated requirements of the game.

Some ideas include:

- A "prettier" display of the cards beyond simply printing the value and suit on the console.
- Doing something extra with the cards at the end of each round.
- Keeping track of all seen hands even across different runs of the program (i.e. implementing a save system).
- Etc...

It just needs to be *something* additional to the program. Successfully doing so will increase your project's grade by one value.

3.5 Documentation

One of the most important skills a programmer can have is the ability to properly document their code. As such, one of the evaluation components will be how easy it is to interpret and read your code. This includes:

- How descriptive and accurate the names of your variables and functions are.
- The quality and quantity of the comments in your code.
- The presence of docstrings in your code, and their quality.

It is not required to use a specific style. Just ensure that your code is readable to an outsider and easy to understand.

3.6 Report

Alongside your code, you should submit a brief report consisting of at most 5 pages of content (not including the cover, if you wish to make one). This report should contain at least the following:

- A brief overview of the organization of your code, including the classes you created and the major functions.
- An explanation of important decisions you had to make during the implementation, such as how to store the cards and deck, how you implemented sorting alongside them, and other aspects you find important.

- An analysis of how the implemented sorting algorithms are expected to behave within this context, which do you consider more efficient and why.
- A description of your algorithm/strategy to find the different poker hands along with an **estimation of the running time complexity** of your approach.
- Brief instructions on how to run your project (which file to run).
- Any additional information you may find relevant.

You do not need to explain us any Python concepts or lines of code. We are interested in *why* you did things a certain way, not the *how*.

Finally, you do not need to pad out your text to fill 5 pages. We are mostly interested in the key components and interpretations of your work, there is no need to detail every little thing.

4 Delivery

As mentioned previously, the project must be delivered by the **8th of June**, by **23h59**. Any delays impose a penalty of 2 points per day, with a maximum tolerated delay of 3 days.

Delivery should consist of a zip file submitted through Moodle containing the following:

- All files required to run your project.
- Your report in a PDF format.

Important Note: We should be able to run your code through executing the Python file. It is your responsibility to ensure the code you submit works, and if we are required to fix it in order to run it, your grade will be penalized. Remember to always test your submission.