# Goal

To explore an advanced application of priority queues in order to gain a deeper understanding of the data structure.

# Background

You will be writing a basic application to help a user select a car to buy. You will write a menu-based user interface driver program (to be run in the terminal, no GUI), but most of the logic will be in implementing a priority queue-based data structure. You should write a PQ-based data structure that stores objects according to the relative priorities of two of their attributes, making it efficient to retrieve objects with the minimum value of either attribute. Your data structure should further be indexable to allow for efficient updates of entered items. You will want users to be able to enter details about cars that they are considering buying. The user should then be able to efficiently retrieve the car with the lowest mileage or lowest price. These retrievals should be possible on the set of all entered cars or on the set of all cars of a specific make and model (e.g., “lowest price Ford Fiesta”, “lowest mileage Cadillac Escalade”).

# Specifications

1. First, you must create a class to store data about cars to buy. Specifically, this class must contain the following information:
    - A unique VIN number: a 17-character string of numbers and capital letters, excluding I (i), O (o), or Q (q) to avoid confusion with numerals 1 and 0
    - The car’s make (e.g., Ford, Toyota, Honda)
    - The car’s model (e.g., Fiesta, Camry, Civic)
    - The price to purchase (in dollars)
    - The mileage of the car

- The color of the car

2. You must write a terminal menu-based driver program (again, no GUI). Specifically, your driver must present the user with the following options:

    1. Add a car
        - This will prompt the user for each of the above-listed attributes of a car to keep track of.

    2. Update a car
        - This option will prompt the user for the VIN number of a car to update, and then ask the user if they would like to update 1) the price of the car, 2) the mileage of the car, or 3) the color of the car

    3. Remove a specific car from consideration
        - This option will prompt the user for the VIN number of a car to remove from the data structure (e.g., if it is no longer for sale)
        - Note that this mean you will need to support removal of cars other than the minimum (price or mileage) car

    4. Retrieve the lowest price car
    5. Retrieve the lowest mileage car
    6. Retrieve the lowest price car by make and model
        - This option will prompt the user to enter a make followed by a model, and then return the car with the minimum price for that make and model.

    7. Retrieve the lowest mileage car by make and model
        - This option will prompt the user to enter a make followed by a model, and then return the car with the minimum mileage for that make and model.

    **Note:** Retrieval operations should *not* remove the car with minimum price or mileage from the data structure, just return information about that car. Cars should *only* be removed via the "remove a specific car from consideration" menu option.

3. To aid in the testing of your application, you will find example files with some test data stored in your folder (`cars.txt` and an alternative `cars2.txt`). Your program should read in the contents of `cars.txt` to initialize your data structure each time it is run (and you should rename the latter to the former to test the alternative example). You can assume that this file will already exist, and you do not need to write an updated version of the data structure back to the file.

4. To ensure efficiency of operations, you must base your data structure around the use of heaps with indirection (making them indexable). Note that retrieval operations on either attribute (e.g., retrieve minimum price, retrieve minimum mileage) **must have a $O(\log n)$ runtime** (both for all cars, and for a

specific make and model). Updates and removals must also have a $O(\log n)$ runtime. Take care in selecting your approach (especially the indirection data structure) to account for the types of keys you will need to store and the type and number operations that you will need to perform on them.

5. Because this project requires you to make a number of decisions about how to implement its requirements, write a documentation file explaining your implementation and **justifying your decisions**. Name this file `documentation.txt`. Document your approach carefully, to reduce the effort required to trace through your code during grading. State the runtime and memory requirements of your approach, and use this information as part of your explanation of why you chose your approach.

# Submission Guidelines

- Upload your submission to the provided Box folder named `cs1501-p3-abc123`, where `abc123` is your Pitt username.
- **DO NOT** upload any IDE package files.
- You must name the primary driver for your program `CarTracker.java`, and it must be outside of any package.
- You must be able to compile your program by running `javac CarTracker.java`.
- You must be able to start your program by running `java CarTracker`
- You must document and justify your approach in `documentation.txt`.
- You must fill out `info_sheet.txt`.
- The project is due at the precise date and time stated above. Upload your progress to Box frequently, even far in advance of this deadline. **No late assignments will be accepted.** At the deadline, your Box folder will automatically be changed to read-only, and no more changes will be accepted. Whatever is present in your Box folder at that time will be considered your submission for this assignment—no other submissions will be considered.

# Additional Notes and Hints

- You are allowed to use code provided by the book authors in implementing your solution. It is up to you to decide if it would be easier to modify the provided code to meet the requirements of this project or to start with a clean slate with entirely your own code.

- To make testing your program easier, you do not need to enforce that users enter properly-formatted VIN numbers. However, your data structure should be designed to operate efficiently on VIN numbers formatted as specified here.

- You may consider adding additional rows to `cars.txt` to expand your testing. During grading, we will test by placing other data in `cars.txt`, so you should be sure your code works for a wide variety of

scenarios and corner-cases.

# Grading Rubric

| Feature | Points |
|---|---|
| Adding a car works properly | 10 |
| Updating a car works properly | 10 |
| Removing a car works properly | 15 |
| Retrieval for all cars works properly | 10 |
| Retrieval for a given make/model works properly | 15 |
| Operations on either attribute are efficient due to heap-backed data structure | 15 |
| Validity of justifications | 15 |
| Menu-based driver program works properly and has appropriately labeled options | 5 |
| Assignment info sheet/submission | 5 |