This is an optional, extra credit assignment worth up to one full homework assignment (40 points). You must do this assignment individually. You must select one of the three options below (which are order from least difficult to most difficult). For each, you will need to include the resources (e.g., articles, sites, videos, etc.) you used in researching the approaches.

**Option 1: Sorting**

Implement three different comparison-based sorting algorithms for ArraySeq and compare their performance to merge sort implemented in class (by modifying and extending the corresponding perf test). You must choose three of the following sorting algorithms implement:

- Randomized Quick Sort (where the pivot is selected randomly)
- Array-Based Heap Sort
- Array-Based Tree Sort
- Intro Sort (a hybrid sorting algorithm)
- Library Sort
- Shell Sort
- Patience Sorting
- Tim Sort (a hybrid sorting algorithm)
- Cocktail Shaker Sort

Each of the three algorithms must be implemented as part of your ArraySeq class. To test the algorithms you must modify and extend the `hw4_perf.cpp` performance test harness. As part of this, you should: (1) remove all but the array-based merge sort tests; and (2) add tests for the three new sorting algorithms you implement. You must use the modified performance test to generate test data as well as modify the gnuplot script to graph your results. Your writeup must include a brief high-level description (the “basic idea”) of each of your three sorting algorithms, the performance graphs, a discussion comparing the results, and a discussion of any issues and challenges you had.

**Grading.** For this option, you must submit all of your code including your performance test driver, your performance test graphs, your input data sets, your gnuplot script, and a short write up. Your grade will depend on your implementation, your performance tests and graphs, and your write up.

**Option 2: Dynamic Binary MinHeap**

Implement a dynamic, pointer-based (i.e., not an array-based) version of the binary min-heap data structure discussed in class called `MinHeap`. Your implementation should store key values similar
to our seq ADT, but with different functions. Thus, you won’t be inheriting from Map or Sequence. Your implementation should include public functions for:

- inserting a key into the heap (insertion based on the key value)
  ```cpp
  void insert(const K& key);
  ```
- checking if the heap is empty
  ```cpp
  bool empty() const;
  ```
- obtaining the min key from the heap, which should throw an out_of_range exception if the heap is empty
  ```cpp
  K find_min() const;
  ```
- erasing the min key from the heap, which should throw an out_of_range exception if the heap is empty
  ```cpp
  void remove_min();
  ```
- providing the keys of the heap in sorted order (as an ArraySeq) via heapsort
  ```cpp
  ArraySeq<K> heap_sort() const;
  ```
- the size of the min heap in terms of the number of key stored
  ```cpp
  int size() const;
  ```
- constructor, copy constructor, move constructor, copy assignment operator, move assignment operator, and destructor

Your sort function must create a copy of the current heap structure, then repeatedly call your remove-min function to perform heap sort. You must create a set of (google unit) tests to demonstrate that your heap implementation is working correctly along with a writeup describing your design and any issues in your implementation.

**Grading.** For this option, you must submit all of your code including your unit tests along with a short write up explaining your tests and any challenges/issues you had in creating your min-heap implementation. Your grade will depend in part on the tests you develop as well as your write up.

**Option 3: SkipList Based Map**

A Skip List is a (probabilistic) linked-list based data structure that stores data in sorted order and provides direct access to certain nodes in the list for (binary-search like) node access. For this option, you must implement a SkipListMap that supports our Map ADT functions. You also will need to (1) develop a suite of (google unit) tests for your implementation (similar to our homework assignments), (2) compare your SkipListMap to BinSearchMap, HashMap, and BSTMap by modifying the hw7_perf.cpp, (3) generate performance data, (4) modify the gnuplot script from
HW-7 to display corresponding performance graph results; and (5) provide a writeup like that of HW-7. This option is by far the most difficult of the three. However, there are a number of resources online that describe skip lists and implementation suggestions. You might also find the following paper on skip lists beneficial (copies of which are available online):


**Grading.** This option will follow the same grading approach as for class homeworks 1–8.

## 1 Submission

Submit all of your code, test cases, and writeup to GitHub classroom (a link will be provided in piazza). You must provide all code necessary for me to compile and run your implementations. You must also include all resources you used as part of your writeup. Finally, include any instructions necessary for compiling and running your programs if different from the approach we’ve used for assignments in class.