CS 25100 (12283,LE2), Data Structures and Algorithms, Fall 2016

Project 2: Sorting with Magic Boxes

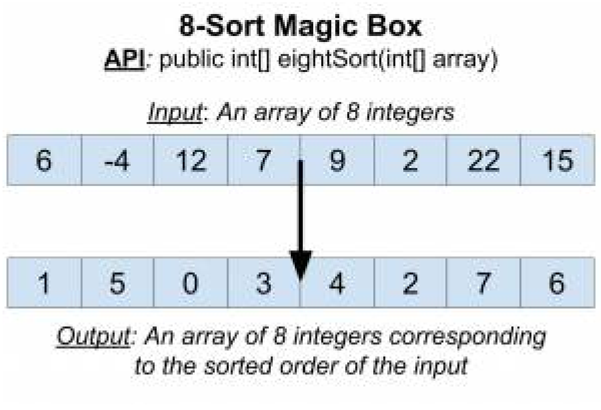
- **Due**: Friday, October 14, 11:59pm
- **Late Penalty**: 20% per day; a maximum of 3 late days (including slip days) are allowed.
- **Skeleton Code**: project2.zip

Introduction

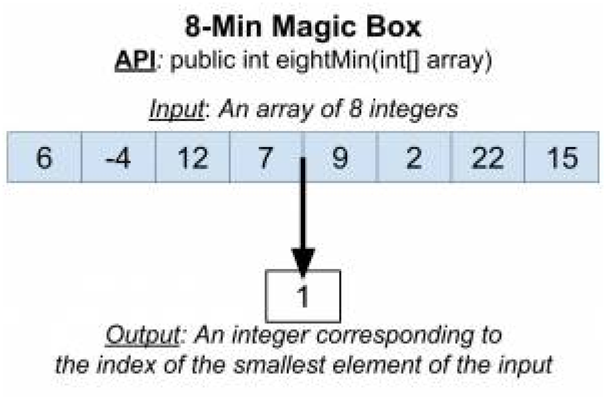
The ruler of Sortakastan has outlawed pairwise comparisons between entries in sorting algorithms. Instead, programs can only make comparisons indirectly by using magic boxes.

Two types of magic boxes are available:

- **8-Sort Box**: Given 8 elements in an array, the 8-Sort Box returns the permutation generating the sequence in increasing order. For example, the input 6, -4, 12, 7, 9, 2, 22, 15, returns the permutation 1, 5, 0, 3, 4, 2, 7, 6. This means the element placed in index 0 (in the sorted array) is at index 1 of the input array (element -4); the element placed in index 1 is at index 5 of the input array (element 2).


8-Sort Magic Box
API: public int[] eightSort(int[] array)
Input: An array of 8 integers
6 -4 12 7 9 2 22 15
1 5 0 3 4 2 7 6
Output: An array of 8 integers corresponding to the sorted order of the input

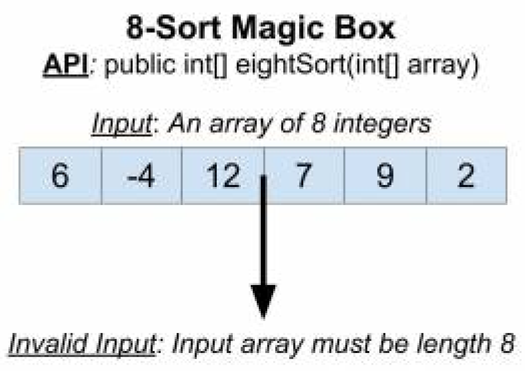
- **8-Min Box**: Given 8 elements in an array, the 8-Min Box returns the position of the minimum in the array. For example, the input 6, -4, 12, 7, 9, 2, 22, 15, returns index 1.


8-Min Magic Box
API: public int eightMin(int[] array)
Input: An array of 8 integers
6 -4 12 7 9 2 22 15
1
Output: An integer corresponding to the index of the smallest element of the input

Assume you are given an array A of size n. There exist trivial solutions for using magic boxes to sort the entries of array A, such as using the magic boxes to determine the minimum element in array A and then using known algorithms by simulating pair-wise comparisons and padding the eight elements with six elements smaller than the minimum. However, such an approach does not minimize the use of sorting boxes. The goals of this project are to adapt existing sorting algorithms to sort with magic boxes and minimize the number of times magic boxes are used. To simplify things, we will assume that array A only contains integers.

Rules and assumptions

- All sorting algorithms should sort elements in increasing order. For example, given the input array [1,2,5,3,4], the sorted output is [1,2,3,4,5].
- It costs “1 use” of a magic box when using the 8-sort or 8-min boxes. You will be comparing this total number of uses with the number of comparisons done in the standard sorting algorithms. Since the magic boxes work by magic, you can assume that the cost of using one magic box is equivalent to doing one standard comparison.
- The input to a magic box must be an array of size 8:


8-Sort Magic Box
API: public int[] eightSort(int[] array)
Input: An array of 8 integers
6 -4 12 7 9 2
Invalid Input: Input array must be length 8

- Directly comparing two integers from array A is not allowed:

```
    if(array[0] > array[1]) ...;  //This is against the rules!
```

- Moving data entries to other array locations and assigning am array location to a variable is allowed. However, making assignment statements and comparing two variables is not allowed.

```
    newArray[1] = array[0];
    temp = array[0];
```

- Comparing array indices is allowed:

```
   for(index = 0; index < array.length; index++);
```

Warm-up problems

Include all of your implementations in this section in the file `WarmUp.java`.

Implement a method `min8Min` that, given an array A, returns the value of the minimum element in A using the 8-Min box. As already stated, the number of times the 8-Min Box is used should be a minimized.

Implement two methods for checking whether a given array A is sorted in increasing order:

- method `isSorted8Sort` uses the 8-Sort Box and returns True if the array is sorted and False otherwise
- method `isSorted8min` uses the 8-Min Box and returns True if the array is sorted and False otherwise

The analysis section includes questions related to the performance of these three algorithms to be answered in your report.

Sorting with Magic Boxes

As already stated, one can pad an array of size 8 with entries (smaller than the min or larger than the max) and simulate comparing two elements. Do not use this approach unless you are sorting fewer than 8 elements.

In class you saw a number of comparison-based sorting algorithms including Selection Sort and Merge Sort. In this project you will adapt these two sorting algorithms to efficiently sort with Magic boxes. Both sorting algorithms should generate the elements in increasing order.

You will evaluate the efficiency of both algorithms experimentally and analytically. For the experimental part, you will compare your sorting algorithms to the standard Selection and Merge Sorts and determine the achieved speedups. The analytical questions will address asymptotic performance and stability.

Selection Sort

Selection Sort is an elementary sort described in section 2.1 of the text as well as in the course slides. Your implementations for selection sort should be located in the file `Selection.java`. You will need to have an implementation of the standard selection sort using pair-wise comparisons (method `sortSelection`). You can write your own or use the version from the Princeton library. You will need to manually count the number of pairwise comparisons used in the standard sort appropriately. Adapt the standard Selection Sort algorithm to design two Magic box selection sort algorithms:

- Method `sortSelection8Sort` uses the idea underlying Selection sort and uses the 8-Sort box to

compare elements.
- Method `sortSelection8Min` uses the idea underlying Selection sort and uses the 8-Min box to compare elements.

You will need to analyze your implementations as described in the analysis section below, which you will submit along with your source code.

Merge Sort

Merge Sort has an O(n log n) time worst-case asymptotic performance described in section 2.2 of the text as well as in the course slides. You will need to have an implementation of the standard recursive Merge sort making pair-wise comparisons. As with selection sort, you can write your own or use the version from the Princeton library. Your Merge sort implementation should use recursion and should be located in the method `sortMerge` in the file `Merge.java`.

Adapt the recursive Mergesort algorithm to use 8-Sort Magic boxes. More specifically, design an algorithm `sortMerge8Sort` that uses the idea underlying the recursive Merge sort and uses the 8-Sort box to compare elements during the merge step. Be sure to consider when to stop the recursion and to minimize the number of times the Magic box is used.

The analysis section will describe how to run and compare the performance of the two Merge Sort algorithms. For the analysis, state and explain the worst-case asymptotic performance of the Magic box based Merge sort algorithm. How does it compare to the standard Merge Sort with respect to the number of “comparisons?”

Data sets and code provided

Data Sets and Input Format

The skeleton code for this project includes multiple text files containing sample inputs. The text files first contain the number of elements in the array, followed by that number of integers (delineated by spaces). You can assume that the text files will be properly formatted, including containing the proper number of valid inputs (and in fact, getting the array out of the text string is done for you).

A random number generator is also provided in the `GenerateRandomNumber.java` file. To generate a file “test” containing an input of size n, run `java GenerateRandomNumber n > test`.

Code Skeleton and Methods Provided

You need to start your project using the provided skeleton code. You can download the skeleton code from here: project2.zip

Sample text files for testing are contained in the `txt` subdirectory. Your own implementation needs to be completed in the files `Merge.java`, `Selection.java`, and `WarmUp.java` as described above. You are free to add code into addition files if needed; such files will need to be included with your source code when you submit. Do not alter the contents of the files `GenerateRandomNumber.java` or `MagicBox.java` for your implementation, as these files will be replaced during grading. The skeleton code provides additional information about each file, including what you need to implement in each of them.

We have also provided a couple of files to help you test your code: `Test.java` and `WarmUpTest.java`. Note, for both test commands given, “file” should be the path to a file which is appropriately formatted. See “Data Sets and Input Format” above for more details about formatting.

- WarmUpTest.java:
  - This program should be run using the command `java WarmUpTest < file`
  - It will test all three methods that are to be implemented in WarmUp.java.
- Test.java
  - This program should be run using the command `java Test -a [sort type] < file`
  - It will run your sorting algorithms and print out the number of comparisons used as well as whether the sort was successful.
  - Run `java Test -h` to see the options for the sorting algorithm flag.

Analysis Questions

Your **typed** report must answer each question briefly and clearly. Data presented in any experimental analysis should be presented in tables or charts. All material must be readable and clear and prepared in a professional manner. For charts and tables that can easily be produced with Excel (or equivalent software), handwritten material **will not** be accepted.

Experimental Analysis

The experimental part of the project asks you to compare the number of *comparisons* made. In the two standard sorting algorithms, this refers to the number of times two array elements are compared. In the Magic box algorithms, this refers to the number of times the Magic box is used.

(1) Consider the following three pairs of sorting algorithms

1. Standard Selection Sort and `sortSelection8Sort`
2. Standard Selection Sort and `sortSelection8Min`
3. Standard Merge Sort and `sortMerge8Sort`

For each pair of algorithms, consider inputs of size 500, 1000, 5000, 10000, 50000 and 100000.

- For each input size, generate 10 random inputs and determine the average number of comparisons done by each algorithm.
- Plot the performance (time vs input size) of each member of a pair. Also plot the speedup for the pairs. The speedup is defined as number of comparisons done in the magic box sort/number of comparisons done in the standard sort.
- Discuss and explain the rate of speedup. What is the observed speedup for each pair? Is the increase constant or does it change with respect to n? Explain the behavior.

(2) Consider the performance of all three Magic Box algorithms. Is one of the Magic Box sorts consistently faster for all sizes of n? Explain the behavior you observe.

Analytical Questions

In all analysis questions, count one use of a magic box as one operation. When comparing a sorting algorithm using Magic boxes with a comparison-based sorting algorithm, one unit of operation is thus either one Magic box use or one pair-wise comparison.

Answer the following questions in a clear and precise manner:

1. For an array of size n, n ≥ 8, how many times is the 8-Min box used in method `min8Min`? State and explain an exact bound.
2. For an array of size n, n ≥ 8, how many times is the 8-Min box used in method `isSorted8min`? How many times is the 8-Sort box used in method `isSorted8Sort`. State and explain exact bounds.
3. For an array of size n, n ≥ 8, state and explain the worst case asymptotic performance of the algorithms `sortSelect8Sort` and `sortSelect8Min`
4. For an array of size n, n ≥ 8, state and explain the worst case asymptotic performance of the algorithm `sortMerge8Sort`. How does it compare to the standard Merge Sort with respect to the number of "comparisons"?
5. Assume the 8-Sort box is a stable sorting box. Under this assumption, is your algorithm `sortMerge8Sort` stable? What about `sortSelect8Sort`? Explain your answers.
6. Instead of an 8-Sort box, the ruler of Sortakastan grants you access to a more powerful Magic box. For an array of size n, you have access to a √n-Sort box. Describe how you would use such a box (you do not need to implement the resulting algorithm). How many times is the Magic boxed used when sorting an array of size n (express in terms of n and in the asymptotic sense)?

Grading

- **60** Warm-up algorithms. Standard Selection and Merge sort. Using Magic boxes: warm-up algorithms, two selection sorts, one merge sort.
- **30** Report: responses analytical question; presentation of experimental results and their discussion and explanations; overall quality of the report.
- **10** Code Quality

Submission Instructions

You are responsible for ensuring your submission meets the requirements. If not, points will be deducted from code quality allocation. You project will not be graded before the deadline. Your score on Vocareum will remain 0 until we grade your project.

Code

- Submit only the source files of your implementation. This means .java files only.
  - No .jar files or .class files should be submitted.
  - If a precompiled binary file is submitted without the .java file, you will receive a 0.
- In each of the `.java` files, you must correctly fill your name, login ID, your project completion date, and your PSO section number.
- Do **not** submit MagicBox.java, GenerateRandomNumber.java, or any .class files.
- Your code must compile to be graded by the autograder. Be sure to test your code before

submitting.
- Your should submit a single folder named as “project2” containing the followings
  - WarmUp.java
  - Selection.java
  - Merge.java
  - Any extra .java files you created or used (such as the Princeton libraries)
  - Your report in one .pdf file. Maximum size allowed is 10MB.

Report

- File name: `<username>.pdf`
- Your report must be typed.
  - Use LaTeX, or OpenOffice Write, or Microsoft Word.
  - State your name on top of the report.
  - Handwritten reports, including digitized versions, will not be accepted.
  - Figures and diagrams that **cannot** be easily generated may be hand drawn and digitized for inclusion into the .pdf file representing your report. Note that tables and graphs that can easily be genrated by Excel or similar **cannot** be hand drawn!
- Your report must be submitted as a PDF.
  - Other file formats will not be accepted.
- Your report must be submitted using Vocareum.
  - Submissions via any other means, to any other venue, will not be accepted.

All work must be submitted in Vocareum following the submission instructions at Vocareum Submission Guidelines. Please be aware of:

- Only your final submission will be graded.
- A submission **after** the deadline is considered late. DON'T wait until the last second to submit.
- If you are submitting late and want to use your slip days, you need to select “use slip days” when submitting on Vocareum.
- For project 2, submissions will not be accepted more than three days after the due date, even if slip days are used.