~~Version: 3-23-2018 9:00 AM~~ Version 3 24 2018 11:45 PM

CS 2223 B15 Term. Homework 2

Homework Instructions

- This homework is to be completed individually. If you have any questions as to what constitutes improper behavior, review the examples as I have posted online http://web.cs.wpi.edu/~heineman/html/teaching_/cs2223/d18/#policies.
- Due Date for this assignment is 2PM March 30th. Homeworks received after 2PM receive a 25% late penalty. Homeworks received after 6PM will receive zero credit.
- Submit your assignments electronically using the canvas site for CS2223. Login to canvas.wpi.edu and locate HW2. You must submit a single ZIP file that contains all of your code as well as the written answers to the assignment.
- All of your Java classes must be defined in a packager USERID where USERID is your CCC user id.
- Submission information is found at the end of this document.

Questions:

1. Mathematical Analysis [10 points] (1 bonus point)
2. Sorting Experiment [30 points]
3. Data Type Exercise [40 points]
4. Heap Exercise [20 points]
5. Quincunx question [1 bonus point]

~~Version: 3-23-2018 9:00 AM~~ Version 3 24 2018 11:45 PM

Q1. Mathematical Analysis [10 points]

On the lecture for Day 07 (Mar 22 2018) I presented analysis to compute the closed formula for C(N). You will perform a similar analysis to compute the average number of array inspections for a binary array search of N sorted elements where $N=2^k$ for k>1 **and** you are searching only for elements that you know already exist in the array.

I recommend you create arrays of size $N=2^k$ for $0 < K < 7$ and fill these arrays with the values from 1 to N. Then simply search for each of the N values (1 .. N) and find a way to record the number of inspections it takes to locate each item.

You are to modify the Analysis program to output the following table

```
N       Avg.          Inspectk
1       1.00000       1
2       1.50000       1,2
4       2.00000       2,1,2,3
8       2.62500       3,2,3,1,3,2,3,4
16      3.37500       4,3,4,2,4,3,4,1,4,3,4,2,4,3,4,5
32      4.21875       5,4,5,3,5,4,5,2,5,4,5,3,5,4,5,1,5,4,5,3,5,4,5,2,5,4,5,3,5,4,5,6
64      5.12500       6,5,6,4,6,5,6,3,6,5,6,4,6,5,6,2,6, … ,6,5,6,3,6,5,6,4,6,5,6,7
```

In this table, each row has a value N, the computed average, and then N numbers separated by commas, which reflect the total number of array inspections needed to locate a_k in a binary array search for k=0 to N-1. For example, with N=4 elements, you start binary array search with lo=0 and hi=3. The first position searched is mid = (lo+hi)/2 = 1; note that integer division truncates the computation.

	1		

If the value being searched turns out to be smaller than a[1], then you would next inspect a[0], thus that takes 2 total comparisons to get to it

2	1		

If the value being searched is larger than a[1], then you would set lo=mid+1 = 2 and hi would be 3, thus mid = (2+3)/2 = 2. It therefore takes 2 comparisons to check a[2].

2	1	2	3

If the value being searched is larger than a[2], then you would set lo = mid+1 = 3 and hi would be 3, thus mid=(3+3)/2 = 3. It therefore takes 3 comparisons to check a[3]. Note that these values match the output of the row marked "4". Your program is done when it outputs the above table.

Bonus Question (1pt.)

Can you come up with a formula that computes the average exactly just using N? For convenience, you can also use k=log N. To gain this bonus point, simply include an extra column "Computed" in the above table which exactly matches the average empirical results.

Hint: I tried two separate approaches, one which almost worked (which was so frustrating; it worked for N up to 32 but was incorrect for N>= 64. This felt exactly like Euler's Prime-Generating Polynomial.) I ultimately solved the problem using maple on the CCC machines (you can run from command line as `/usr/local/maple16/bin/maple`). You can google lots of information about maple.

Q2. Sorting Experiments (30 pts)

The best way to evaluate comparison-based sorting algorithms is to empirically evaluate their performance on different kinds of input. To follow the scientific method, we will develop two hypotheses and seek to evaluate whether they are true or false. Your job is to provide empirical evidence.

> Hypothesis 1: When there are a large number of duplicate values, the overall sorting time should be reduced (and should be detectable).
>
> Hypothesis 2: When there are a large number of duplicate values, the number of exchanges is reduced (and should be detectable).

The sorting algorithms you will evaluate include:

- MergeSort
- QuickSort as presented in the book using the **`partition`** function shown on p. 291
- QuickAlternate as presented in the book using alternate partition as included in repository. Find in **`algs.hw2.QuickAlternate`**

For each algorithm, you are to execute T=3 trials and report the lowest reported value for the following statistics for input size ranging from 4096 to 131072 at multiples of 2. The relevant statistics are:

- Number of **`exch`** invocations
- Number of **`less`** comparisons
- Stopwatch performance for time of execution

You will perform the experiment on two data sets that you will have to create (see code):

- `generateUniqueData` – which creates $N=2^k$ unique integers in random order
- `generateHighDuplicateData` – which creates $N=2^k$ integers with specific duplication rates in random order

To get started on this question, copy the relevant files from the **`algs.days.dayNN`** packages that already implement the various sorting algorithms. Then you should modify these local copies to record the number of **`exch`** invocations as well as the number of **`less`** operations. Be sure to use **long** data type to store these counts, rather than 32 bit integers (hint: see the Insertion example for **day06**).

Hint: You need to determine the proper place in your code where you reset your counters for these operations.

~~Version: 3-23-2018 9:00 AM~~ Version 3 24 2018 11:45 PM

(15 pts) I have provided a template class, **`SortComparison`**, in **`algs.hw2`** which you should copy into your own project and modify accordingly. This class will properly produce the output that is expected.

Your goal is to produce empirical results for the desired range (low=2^{13}=8192, to high=2^{19}=524288). The format of the output is shown in the sample code.

(15 pts) For each algorithm, compare the empirical results and determine whether you have been able to provide support for each hypothesis – or provided evidence to refute the hypotheses. The results may be inconclusive. The results may support one (or both) of the hypotheses for all three algorithms, or just for one or two of them.

~~Version: 3-23-2018 9:00 AM~~ Version 3 24 2018 11:45 PM

Q3. Data Type Exercise (40 pts)

This assignment gives you a chance to demonstrate your ability to program with Linked Lists. You are to implement the following data type which maintains a bag of items by keeping each elements just once in the linked list with a count for the number of times that element exists in the bag. This implementation is appropriate when there is a high probability of data duplication.

This assignment will prove to be a challenging programming assignment.

```
package USERID.hw2;

public class MultiSet<Item extends Comparable<Item>> {

  class Node {
    private Item       item;
    private int        count;
    private Node       next;
  }

  // operations that are independent of the size of the MultiSet
  public MultiSet() { ... }
  public int size() { ... }

  // operations that are dependent (in some way) on U and N. See documentation
  public MultiSet(Item[] initial) {... }
  public int multiplicity(Item it) { ... }
  public boolean identical (MultiSet other) { ... }
  public Item[] toArray() { ... }
  public boolean add(Item it) { ... }
  public boolean remove(Item it) { ... }
  ~~public boolean contains(Item it) { ... }~~
  public int multiplicity(Item it) { ... }
  public MultiSet<Item> intersects(MultiSet<Item> other) { ... }
  public MultiSet<Item> union(MultiSet<Item> other) { ... }
```

The implementation must conform to performance specifications that are included in the sample template found in **algs.hw2** in the Git repository. More documentation is found in the sample file. In all of the performance specifications, N refers to the total number of items in the **MultiSet** while U refers to the total number of unique items.

We will validate the output against a set of test cases that we develop for the grading. Individual breakdown of points is found on the rubric. We will release the test utility that will validate the correctness of your implementation as well as the execution performance.

This programming assignment is challenging if you have not programmed extensively with linked lists. Do not wait. Get started on this assignment as soon as you can.

Bonus Points

1. (1 pt) Write an Iterator for MultiSet that outputs each of the elements in the MultiSet type. If a particular element has multiplicity K, then that element is output K times.

~~Version: 3-23-2018 9:00 AM~~ Version 3 24 2018 11:45 PM

Q4. Heap Exercise (15 pts)

The heap data type can be used to implement a Max Priority Queue (p. 318). Copy the MaxPQ code that you will find in **`algs.days.day09`**.

From this base code, add the resizing logic that we used earlier for allowing a queue to support arbitrary-sized data. As a Maximum Priority Queue, it is not expected that you should also support the delete minimum operation. However, this can be done, and this is the challenge posed by this question.

You must implement a **`delMin`** function that removes the smallest priority item from the priority queue.

Modify the **`HeapExercise`** class. You will process randomized collections of uniformly distributed floating point values, as constructed using the **`StdRandom.uniform`**`()` method call (as you see on p. 256 of the book).

You are to execute T=10 trials and report the highest range of the **number of comparisons** for input size ranging from N=4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192. Complete this evaluation as follows:

1. Randomly construct a Heap priority queue containing N values using **`StdRandom`**.
2. Once constructed, perform 1000 iterations of the following sequence
    a. Remove minimum priority value in priority queue (during which you should count the number of comparisons)
    b. Insert random value

You are to report on the highest range of the **comparison** statistics for each of the Heap sizes.

Q4.1 Worst Case Analysis (5 pts)

Your task is to determine a formula that characterizes (in the worst case) the number of comparisons that are performed. This formula must be based on N, the number of elements in the heap.

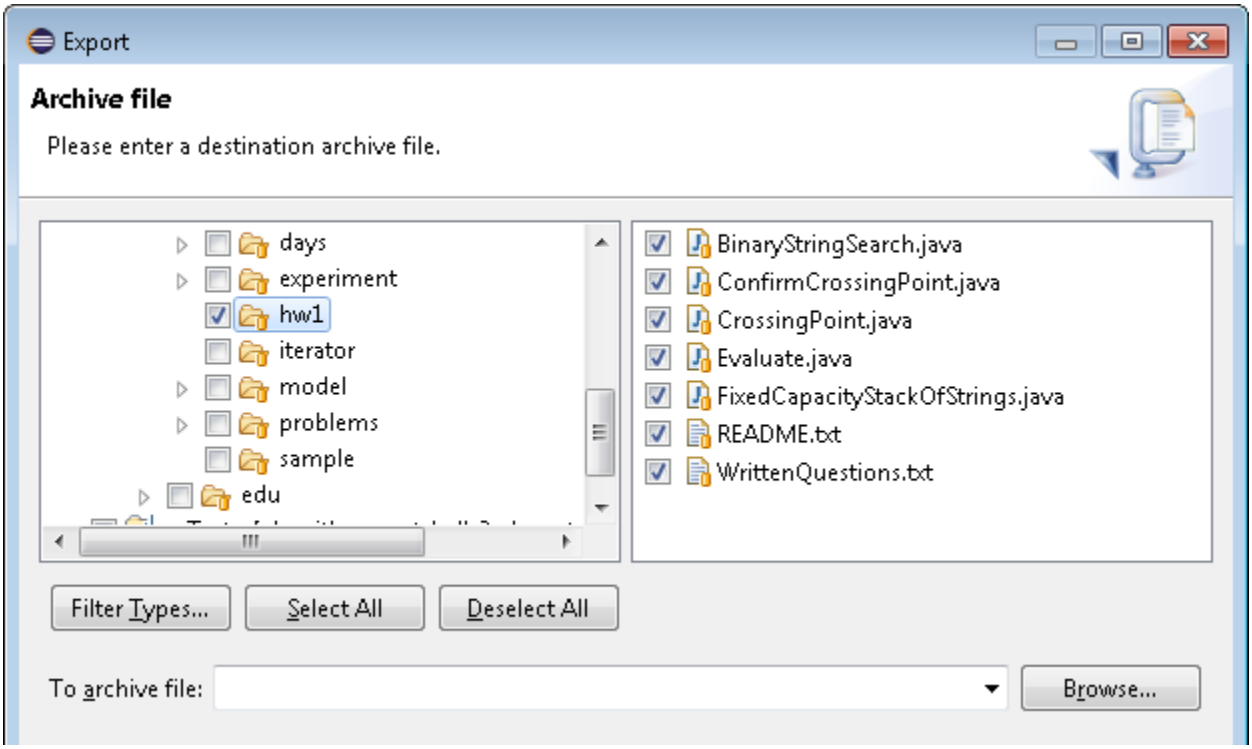
Do your experimental results validate your mathematical analysis?

~~Version: 3-23-2018 9:00 AM~~ Version 3 24 2018 11:45 PM

Submission Details

Each student is to submit a single ZIP file that will contain the implementations. In addition, there is a file "WrittenQuestions.txt" in which you are to complete the short answer problems on the homework.

The best way to prepare your ZIP file is to export your entire **`USERID.hw2`** package to a ZIP file using Eclipse. Select your package and then choose menu item "**Export…**" which will bring up the Export wizard. Expand the **General** folder and select **Archive File** then click **Next**.



You will see something like the above. Make sure that the entire "hw2" package is selected and all of the files within it will also be selected. Then click on **Browse…** to place the exported file on disk and call it USERID-HW2.zip or something like that. Then you will submit this single zip file in canvas.wpi.edu as your homework2 submission.

Addendum

If you discover anything materially wrong with these questions, be sure to contact the professor or TA/SAs posting to the discussion forum for HW2 on piazza; you may also email cs2223h-staff@cs.wpi.edu.

When I make changes to the questions, I enter my changes in red colored text as shown here.