

COSC 222 – Lab 3: BucketSort

This lab illustrates an Object-Oriented design strategy towards implementing data structures. We will be using `Entry.java` objects which are key,value pairs, with `key` being an integer and `value` being a generic parameter. We will be sorting these Entries according to their key value. The file `Entry.java` is given to you and does not require any modification.

As you should know by now, bucket sort works by placing items into buckets and each bucket is sorted on its own. Once all items are placed in buckets, the bucket contents are extracted in sequence from smallest bucket to largest. In this lab, we will have a `SortedBucket` object which takes an item and stores it in sorted order upon adding it to the bucket. We will also be using a `BucketList` object which creates a list of `SortedBucket` objects, and decides which bucket each number (`Entry`) goes to.

Start with looking at the two Abstract Data Types:

SortedBucketADT.java: an interface that describes what functionality a bucket has. Aside from a constructor and a `toString()` method, these can `.add(Entry)` and will place the `Entry` in sorted order within the bucket, and they also have a `getBucketContents()` method which returns the contents of the bucket.

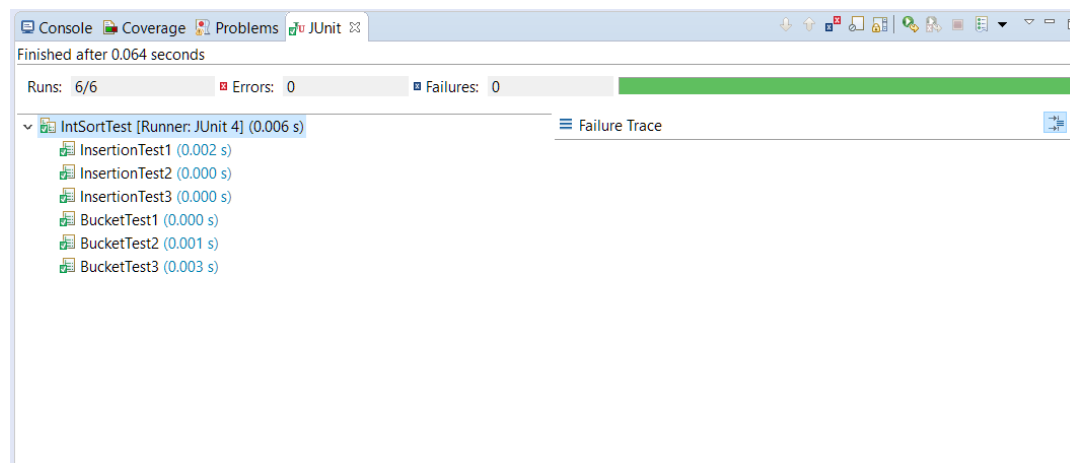
BucketListADT.java: an interface that describes the collection of buckets. The constructor takes 3 parameters: `int min`, `int max`, `int n`, which represent the expected minimum of `Entry` keys, expected maximum of `Entry` keys, and the number of buckets to create. It supports methods `.add(Entry)` and `.addAll(Entries)` to add one `Entry` or to add many `Entries`. The `.add()` method should decide which bucket the `Entry` gets placed in. It also has a way to extract the sorted order of `Entries` with a `.getSortedOrder()` method.


You do not need to modify these ADT files. Your task will be in implementing and testing them. Namely, `SortedBucket.java` will implement `SortedBucketADT.java` and `BucketList.java` will implement `BucketListADT.java`. These are partially implemented already, and contain `//TODO` items throughout the file to guide you in the implementation process.

Further, there is a `MainClass.java` present only to show you some sample intended usage of the data structure and sorting method. You can modify or delete this file as you please. Finally, there is a `Test.java` class file which illustrates how one can test the contents of each bucket after adding elements to the bucket list. Once again, there are `//TODO` items throughout this test file with more specific instructions.

Coverage Testing:

While unit testing ensures code quality (correctness of outputs given certain inputs), **coverage testing** ensures the **quality of the unit tests**. Coverage testing shows you how much of your code is being executed in a test class, a useful thing to know when trying to test your code.



Coverage testing works in conjunction with a regular test class. It can be accessed in Eclipse by pressing the run button with the green-red bar image:  You should also be able to run coverage testing by right clicking a test class and selecting **run as coverage test**. This will result in a summary of how much of your code was covered by your test!

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
Lab2_soln	97.6 %	449	11	460
src	97.6 %	449	11	460
(default package)	97.6 %	449	11	460
IntSort.java	95.1 %	176	9	185
IntSort	95.1 %	176	9	185
sort(int[])	0.0 %	0	4	4
BucketSort(int[], int)	98.5 %	133	2	135
InsertionSort(int[])	100.0 %	43	0	43
IntSortTest.java	99.3 %	273	2	275
IntSortTest	99.3 %	273	2	275
BucketTest3()	97.8 %	44	1	45
InsertionTest3()	97.7 %	43	1	44
BucketTest1()	100.0 %	9	0	9
BucketTest2()	100.0 %	45	0	45
InsertionTest1()	100.0 %	8	0	8
InsertionTest2()	100.0 %	44	0	44

Scoring:

- +1 for including your name/ID in each and every source code file you submit, including Test file(s)
- +1 for including a screenshot of your coverage test report
- +3 for completing the test file
- +5 for completing the implementation of a functioning bucketsort

SUBMIT: your edited .java files and a screenshot of your coverage report that shows that your tests provide complete coverage (should indicate over 90% coverage) of the project files. You may alternatively show your lab instructor (in lab time) your coverage report and your test cases. Submit the plain .java files – do not submit an archive/zip file or an exported project file.