CURTIN UNIVERSITY (CRICOS number: 00301J)

Department of Computing, Faculty of Engineering and Science

Data Structures and Algorithms (COMP1002)

PRACTICAL 8 - HEAPS

AIMS

- To implement a Heap
- Implement a HeapSort using your heap

BEFORE THE PRACTICAL:

- Read this practical sheet fully before starting.

ACTIVITY 1: CREATING A HEAP

We are going to write a heap and use it to do HeapSort, an O(N log N) sorting algorithm that needs a heap to work. Create a new Java class called DSAHeap. It should have *at minimum* the following:

```
public class DSAHeap {
        private DSAHeapEntry[] m_heap;
        private int m_count;

        public DSAHeap(int maxSize) { ... }

        public void add(int priority, Object value) { ... }
        public Object remove() { ... }
        public void heapSort(DSAHeapEntry[] list) { ... }
   //trickleUp and trickleDown MUST be recursive
        private void trickleUp(int index) { ... }
        private void trickleDown(int index) { ... }
}

public class DSAHeapEntry {
  public int priority;
  public Object value;

  public DSAHeapEntry(int inPriority, Object inValue) { ... };
   }
```

Following are a few notes on the implementation details of the heap.

- Note that there are no generics in this definition,
- Implement it as a max heap, so that larger values indicate higher priority.
- Since we are using an array to represent the heap, remember that we use arithmetic to determine the array indexes of the children given the parent's index (and similarly to go from children to the parent's index). So, if we are at currIdx, the following arithmetic will get use the left child, right child and parent of the currIdx node:
  - leftChildIdx = (currIdx * 2) + 1
  - rightChildIdx = (currIdx * 2) + 2
  - parentIdx = (currIdx – 1) / 2

- add() must add the priority-value pair at the correct place in the heap tree. To do this, remember we start it off at the end of the array, then trickle it up (by using arithmetic to get its parent) until the parent is of equal or higher priority.

- remove() must return the highest-priority element and remove it from the m_heap array. This will involve removing the 0-th (root) element, placing the last (m_count-1) element at the root and then trickling it down.

Use the pseudo-code in the lecture notes as a guide. The imports for the methods will be different for this worksheet as we are using class fields here.

ACTIVITY 2: HEAPSORT

- To implement heapSort() properly, we should be using a ***callback interface*** to get the key from the Object in the array to sort. This overly complicates things here, so we will have to make our heapSort non-generic (tight coupling).
- **This is not good programming practice!!** however we want you to learn the heap sort algorithms involved. You will learn callback interfaces in a future unit.
- The idea is that the imported array-to-sort will have to be an array of DSAHeapEntry objects, heapSort() will need to replace m_heap with this array.
- Then you can use the pseudo code in the lecture notes.

Write a good test harness using either a random list of varying size such as found in the SortsTest, or read in the numbers from the 7000 names file.

SUBMISSION DELIVERABLE:

Your classes (all that are required for this program) are due at the beginning of your next tutorial. Also include any other relevant classes that you may have created. Please only submit the *.java files, although if you have a Makefile, including that would be very welcome.

You do need to submit your test harness in order to obtain full marks. See the Marking Guide below.

SUBMIT ELECTRONICALLY VIA BLACKBOARD, under the *Assessments* section.

MARKING GUIDE

Your submission will be marked as follows:

- [2] UML diagram for all classes and any class variables and interesting methods.
- [4] Your DSAHeap is implemented properly, with driver code to test functionality.
- [2] Your heap sort algorithm is properly implemented, with driver code to test functionality.
- [2] Your test harness works and allows testing on either an array of arbitrary length random numbers or loading in a large amount of numbers from a file (such as the 7000 names one).