1.204 Computer Algorithms in Systems Engineering

Spring 2010

Problem Set 4: Satellite data sets

Due: 12 noon, Monday March 29, 2010

Please see the code provided for the implementation. There are three separate solutions in the Java zip file. All use identical versions of Merge, SatelliteData and MinHeap.
a. Solution with an explicit tree uses MergeMain and TreeNode
b. Solution without an explicit tree uses MergeMainNoTree and Node
c. Solution used to answer question 3 below uses MergeMainNoTreeAnalysis and Node

Analysis questions:

1. In a full binary tree, each node has either zero or two children. In each step of the algorithm, we add two children (which are either nodes or trees) to a node or a tree. In the initial step, we add two nodes to a single node and thus creating a full binary tree. Adding two nodes to a full binary tree also creates another full binary tree. Joining two full binary trees also creates another full binary tree. Thus, at the end of the optimal merge algorithm, we have a full binary tree.

2. For a binary heap, the insert operation takes O(lg n) steps, and the deletion operation also takes O(lg n) steps. Creating the initial heap of all the nodes takes O(n) steps, since it uses heapify. Thus the algorithm takes:

   a. O(n) steps to create the heap initially
   b. Then in each step of the algorithm, we have two delete operations and one insert operation on the heap. After each step, we reduce the number of elements in the heap by 1.
   c. Thus the total run time is:

   $$O(n) + \Sigma[O(\lg(n-1)) + O(\lg(n-2)) + O(\lg(n-1))]$$

   The second term in the above expression is O(n lg n), since

   $$\lg(n) + \lg(n-1) + \ldots = \lg(n!),$$

   and by Stirling's formula,

   $$\lg(n!) \sim n \lg(n) + cn, \text{ where } c \text{ is a constant.}$$

   Thus the run time for this algorithm is O(n lg n).

3. See code provided. An example run is shown below. Note that the number of steps per data element increases essentially exactly as lg n, which means the algorithm performance is indeed almost exactly O(n lg n), where the number of data elements is n. In this example run, each dataset has exactly 200 randomly generated elements.

```
ArraysSteps Data  Ratio
1     0     200   0.0
2     400   400   1.0
3     1000  600   1.6666666
4     1600  800   2.0
5     2400  1000  2.4
6     3200  1200  2.6666667
7     4000  1400  2.857143
8     4800  1600  3.0
9     5800  1800  3.2222223
10    6800  2000  3.4
11    7800  2200  3.5454545
12    8800  2400  3.6666667
13    9800  2600  3.7692308
14    10800 2800  3.857143
15    11800 3000  3.9333334
16    12800 3200  4.0
17    14000 3400  4.117647
18    15200 3600  4.2222223
19    16400 3800  4.3157897
20    17600 4000  4.4
```