CISC235
Winter 2007
Homework for weeks 10 and 11
in preparation for quiz 5
Solutions

Note: I received help for preparing these solutions from Yurai Nuñez and Henry Xiao.

Questions from section 7.5 of the text

2. As was discussed in class the number of keys stored in an m-way search tree of height h is given by the sum

$$(m-1)\sum_{i=0}^{h-1} m^i = m^h - 1$$

3. A function that prints out the contents of a B-tree in ascending order is shown below

```
BTreePrintAscend (BTreeNode *node ) {

// keyTally keeps track of the number of keys in this node

   for (int i = 0; i <= node->keyTally; i++) {
     if (i > 0 )print key[i-1];
     if (node->pointers[i] != 0) BTreePrintAscend( node->pointers[i] );
   }
}
```

5. The B-trees that result from the two insertion sequences are shown below. The trees are not identical. One can intuitively imagine that a "random" sequence of inserts may result in fewer split operations, however, the overall complexity of building a tree with n keys, or searching in the tree would only differ by a constant. Fewer nodes also represents less memory used, as was mentioned in class when I presented the solutions.

Figure 1: The tree on the left has one less node, and also incurred on less split than the tree on the tree on the right.

6. The asymptotic worst case of inserting into a 2-4 tree is characterized by an insertion at a leaf node and this is h the height of the tree. However, this can differ by a constant factor. If all of the nodes on the search path to the node where the insertion occurs are "full", then h split operations will also take place.

Solutions to week 11 problems.

1. The maximum number of keys that a B-Tree of order 6 (at most five keys per node) and of height 4 can hold is $6^4 - 1$. Four disk accesses are needed in the worst case. A B-Tree of order m and of height h can hold as many as $m^h - 1$ keys.

2. There are $2n(\lceil(\log_2 n)\rceil) = 2*32(5) = 320$ external memory accesses (reads and writes) are used to sort the data using a 2-way merge sort. Note that no pass 0 is used to obtain initial sorted runs.

3. There are $2n(\lceil(\log_4 n)\rceil) = 2*32(3) = 192$ external memory accesses (reads and writes) are used to sort the data using a 4-way merge sort. Note that no pass 0 is used to obtain initial sorted runs.

4. If we have enough internal memory to make initial runs of 8 sorted records then we end up with 4 runs. A binary merge uses $\log_2 4$ passes after the intial sorts for 2*32(2+1) = 192 external memory accesses. For the 4-way merge sort one pass is needed after the initial sort, that is $\log_4 4$ so we use 2*32(1+1) = 128 memory accesses.

5 The original list of unsorted numbers is:

41 40 35 6 74 8 87 89 99 24 2 8 44 19 93 23 63 91 4 5 70 57 38 59 3 4 83 83 46 18 53 17

Using 2-way merge sort

After pass 0:

6 8 35 40 41 74 87 89

2 8 19 23 24 44 93 99

4 5 38 57 59 63 70 91

3 4 17 18 46 53 83 83

pass1 yields:

2 6 8 8 19 23 24 35 40 41 44 74 87 89 93 99

3 4 4 5 17 18 38 46 53 57 59 63 70 83 83 91

And pass 2 yields the sorted list.

2 3 4 4 5 6 8 8 17 18 19 23 24 35 38 40 41 44 46 53 57 59 63 70 74 83 83 87 89 91 93 99

Using a 4-way merge sort

After pass 0:

6 8 35 40 41 74 87 89

2 8 19 23 24 44 93 99

4 5 38 57 59 63 70 91

3 4 17 18 46 53 83 83

And pass 1 yields the sorted list.

2 3 4 4 5 6 8 8 17 18 19 23 24 35 38 40 41 44 46 53 57 59 63 70 74 83 83 87 89 91 93 99

6. One possible insertion sequence leading to the given skip list is: (20,1), (42, 4), (12,2), (17,5), (39, 1), (48,3), (24,4), (31,3), (50,1), (44,2), where the pair (k,h) denotes a key with value k, and a tower of height h.

7. We average over all search paths. I will count the number of nodes in the search path rather than actual comparisons. The paths taken in sequence are (6 + 3 + 8 + 5 + 7 + 10 + 6 + 9 + 8 + 11) / 10 = 7.3.