ECE-250 – Algorithms and Data Structures (Winter 2012)

Tutorial 6 (2012-03-08)

1 – Insert the following values into an initially empty 4-way tree: 10, 17, 12, 5, 21, 43, 40, 35, 30

2 – Without attempting any actual balancing, is there a better strategy to place values in newly-created nodes to reduce the height of the tree, considering the average case, with random values evenly distributed? (retry the insertions, in the given order, with this new strategy)

3 – (a) Insert the following values into an initially empty Heap: 7, 15, 13, 25, 1, 5, 9, ensuring that we maintain a complete binary tree.

(b) Show the procedure to remove them in the appropriate order, without worrying about maintaining a complete tree.

(c) Repeat, maintaining a complete binary tree (for the first two removals, use the less efficient method, and for the rest use the preferred, more efficient method)

4 – Show the results for question 3 using an array representation for the heap (size and resizing policy is not really relevant for this exercise — you may assume a fixed-size array of size 8).

5 – (If enough time) A more or less optimized form of the bubble-sort makes the passes alternating direction — instead of doing n times the same loop, from 0 to $n-1$ (or rather, from 0 to $n-i-1$, where i is the outer loop's counter) always, we do it one time ascending, then one time descending, then ascending, and so on.

Another typical optimization uses the fact that we can stop after an outer loop's iteration where there were no swaps after completing the inner loop.

Write a C++ fragment of code to implement this optimized version of bubble-sort; assume a function prototype as follows:

```
template <typename Type>
void bubble_sort (Type * array, int n);
```