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

Assignment 4 – Suggested solutions

1 – (a) Insert the following values (in the given order) into an initially empty min heap: 46, 34, 83, 75, 25, 57, 93, 27, 17.

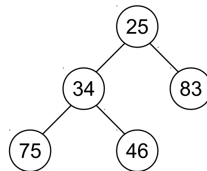
You need to justify the steps, but a single sentence describing the steps suffices — provided that all the key details are indeed described (you still need to show the step-by-step contents of the heap, after each insertion).

(b) Dequeue the elements, showing the contents of the heap after each removal.

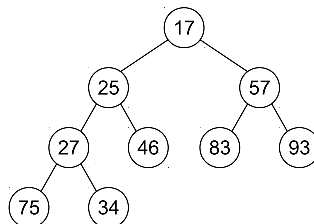
Solution:

(a) The first value is trivially inserted. Then, for the others, we place the element as the last element in a complete binary tree (i.e., following a breadth-first traversal pattern), and adjust (percolating up).

34 is inserted as the left child, then it is swapped with the root. 83 is then inserted as the right child, and since $83 > 34$, no adjustment is needed. 75 is then inserted as the left of 46, and no adjustment is needed. 25 is inserted as the right child of 46, requiring a swap with 46, then requiring a swap with the root. So far, the heap would be as shown below:



57 is then inserted as the left child of 83, requiring a swap with 83; then, 93 is inserted as the right child of 57, requiring no swaps. 27 is inserted as the left child of 75, requiring a swap with 75, then a swap with 34. Finally, 17 is inserted as the right child of 34, requiring swaps all the way to the root. The final contents is:



(b) Since the removal strategy is not explicitly specified, there are several possible approaches that could be used here. We could go for the simple one, that does not take care of maintaining a complete binary tree. This is the easiest way: we simply remove the root, and promote the lower of its children, repeating then for that removed node, promoting the lower of its children, all the way to a leaf node.

The other approach would require taking the "last" element (going with the breadth-first traversal pattern) and placing it at the root, then percolating down.

I'll just show the first two removals using each of these strategies:

- Simple approach:

  Removing 17 requires promoting 25 (the lower of 25 and 57); then, removing 25 requires promoting 27, and then, promoting 34. By pure coincidence, this is the same outcome as if we had used the other strategy.

  Then, removing 25 requires promoting 27, which requires promoting 34 (the lower of 34 and 46), which then requires promoting 75 (the only child at this point)

- Maintaining a complete binary tree:

  Removing 17 is done by taking the last element (34) and placing it at the root, then percolating down — swapped with 25, then swapped with 27.

  Removing 25 then places 75 at the root, to be percolated down — again by coincidence it ends up producing the exact same outcome (emphasis on the fact that it is a coincidence; there's no guarantee of which path the swaps are going to take, since the lower of the children can be either the left child or the right child).

2 – (a) Dequeue each of the elements from the following min heap represented as an array (you must show the procedure with the processing as an array, including obtaining the subscripts for parents and children, and not drawing the "tree view" of the heap).

	2	5	7	6	12	8	18	11	31	13	21	29	9	19

(b) Convert the following array of unsorted values into a min heap (in-place, with a linear time procedure): $\{48, 23, 17, 26, 14, 87, 53, 89, 32, 49, 95, 12\}$

Solution:

Since we are representing the heap as an array, we must always maintain a complete binary tree, which means that we must always move the last element to the root position, then percolate it down (as opposed to simply remove the root, and then promote the lower of its children, and so on until promoting some leaf node — this does not maintain a complete binary tree)

Dequeueing 2 moves 19 to the root, so we percolate it down; position 1, means that its children are at position 2 and 3, so we swap it with 5; then, from position 2, its children are at position 4 and 5; so we swap with 6. Then, from position 4, its children are at positions 8 and 9; so we swap with 11, and this one is already a leaf node (size is now 13 elements, and from position 8, the children would be at positions 16 and 17). After removing 2, the heap's contents is as follows:

	5	6	7	11	12	8	18	19	31	13	21	29	9	

Then, dequeueing 5 moves 9 to the root; swap it with 6, then swap it with 11, and that's already its final position, since its children are 19 and 31 (both greater than 9). The contents of the heap is as follows:

	6	11	7	9	12	8	18	19	31	13	21	29		

Dequeueing 6 moves 29 to the root; then we swap it with 7; from position 3, children are at positions 6 and 7 (values 8 and 18), so we swap with 8, and that one is now a leaf (its child was 29, but at this point, 29 was removed from the back, leaving 8 as a leaf, since there are 11 elements).

	7	11	8	9	12	29	18	19	31	13	21			

And so on ...

(b) The *heapification* procedure does this. We start at the back and move to its parent. From there, moving backward, we percolate down each element as needed.

From position 12, its parent is at position $12 \div 2 = 6$, so we swap 87 with 12 (its only child, so we move the larger value down):

$$\{48, 23, 17, 26, 14, 12, 53, 89, 32, 49, 95, 87\}$$

Then, 14 is swapped with 49, the lower of its children (49 and 95):

$$\{48, 23, 17, 26, 14, 12, 53, 89, 32, 49, 95, 87\}$$

26 does not need to be swapped (its children are 89 and 32), then 17 is swapped with 12, and it doesn't need to be swapped any further (its only child is 87):

$$\{48, 23, 12, 26, 14, 17, 53, 89, 32, 49, 95, 87\}$$

23 is swapped with 14, and it is at the correct position (children are 49 and 95), and 48 is swapped with 12, then with 17, then it is at the correct position, since its child is 87. The contents after heapification is:

$$\{12, 14, 17, 26, 23, 48, 53, 89, 32, 49, 95, 87\}$$

3 – Describe a traversal strategy for a min-heap that visits every element that is less than a given value. The run time must be $O(m)$, where m is the number of elements that satisfy the condition.

2% Bonus Marks: Write a C++ function that implements that traversal to print the values (you may assume a `Binary_node` class as described in class — of course, with the assumption of the min heap constraints)

Solution:

The idea is similar to a depth-first traversal (notice that we sort of discard the possibility of breadth-first based on the intuition that for a heap, there is absolutely no relationship between nodes "sideways").

If we are traversing the entire tree in a depth-first manner, we know that the defining property of a min heap tells us that if at some node we see a value that is not less than the given value, we have the guarantee that everything below that node will not be less than the given value either (since everything below that node is necessarily greater than the node).

So, this condition (finding a node that is greater than the given value) is the "stop" condition for the recursion — in any case, it is the condition that determines that we do not continue descending to any children (of course, that, combined with the possibility of reaching a leaf node).

As for showing that this is $O(m)$, we observe that we only look at nodes that don't match the condition when they're the children of a node that matches the condition. Thus, in addition to the m nodes matching the condition, we will visit at most $2\,m_L$, where m_L is the number of nodes that match the condition and for which some of its children do not match the condition. Since $m_L \leqslant m$, then $m + 2\,m_L \leqslant 3\,m = O(m)$.

A C++ implementation would be as follows (as an example, the procedure outputs the values being visited):

```
template <typename Type>
void heap_trav_lt (const Binary_node<Type> * tree, const Type & value)
{
    if (tree == NULL || tree->retrieve() >= value)
    {
        return;
    }

    cout << "Visited " << tree->retrieve() << endl;

    heap_trav_lt (tree->left(), value);
    heap_trav_lt (tree->right(), value);
}
```

4 – For the following array of values: $\{48, 23, 17, 26, 14, 87, 53, 89\}$:

(a) Sort the values (showing the procedure, step-by-step), using merge sort. For the recursion's base case, you may use the point where the array size reaches two, at which point sorting boils down to a conditional swap of the values.

(b) Run the first iteration of quick sort, using median of three as the approximation for the median (that is, the step that ends with every value in the first chunk being less than any of the values in the second chunk).

Show the step-by-step procedure.

Solution:

(a) We split into sub-sequences $\{48, 23, 17, 26\}$ and $\{14, 87, 53, 89\}$ then, into $\{48, 23\}$, $\{17, 26\}$, $\{14, 87\}$, and $\{53, 89\}$. Handling the base case for each of these 2-element sub-sequences, we get sorted arrays $\{23, 48\}$, $\{17, 26\}$, $\{14, 87\}$, and $\{53, 89\}$. We merge the first two; starting at the

beginning, we pick 17 (the lower of 23 and 17) and advance; then 23 (the lower of 23 and 26), then 26 (the lower of 26 and 48). For the other branch of the recursion, we start with 14 (the lower of 14 and 53) and advance; then 53 (the lower of 53 and 87), then 87 (the lower of 87 and 89), and finally 89; the merged sub-sequences are now $\{17, 23, 26, 48\}$ and $\{14, 53, 87, 89\}$. To merge these two, we start with 14 (the lower of 17 and 14), then 17 (17 / 53), then 23 (23 / 53), then 26 (26 / 53), then 48 (48 / 53), and then the rest from the second sequence.

(b) For quick sort, we choose the pivot as the median of the first, middle, and last elements — 48, 26, and 89; median 48. We put 48 temporarily aside, move 26 to the first position, and 89 to the middle:

$$\{26, 23, 17, 89, 14, 87, 53, \ \}$$

Start scanning forward from second position looking for a value greater than 48; find 89. Scan backward from second-to-last position looking for a value less than 48; find 14. Swap these two values; repeating the procedure we'll find the same two values again, except that now the "pointers" are reversed, so we finish the iterations; we take the element at the larger pointer and move it to the end, and place the pivot there:

$$\{26, 23, 17, 14, 48, 87, 53, 89\}$$

The next recursive steps would be: sort the sequences $\{26, 23, 17, 14\}$ and $\{87, 53, 89\}$.

5% Bonus Marks – Write a C++ function that determines whether a given binary tree is a min heap.

Solution:

The function is almost a direct transcription of our recursive definition of a heap: a non-empty binary tree is a min heap if the root is lower than either one of its children, and both subtrees are themselves min heaps:

```
template <typename Type>
bool is_min_heap (const Binary_node<Type> * tree)
{
    if (tree == NULL)
    {
        return true;
    }

    if (tree->left() != NULL && tree->left()->retrieve() < tree->retrieve()
        ||
        tree->right() != NULL && tree->right()->retrieve() < tree->retrieve())
    {
        return false;
    }

    return is_min_heap (tree->left()) && is_min_heap (tree->right());
}
```

5% Bonus Marks – Prove that removal (of the root node) from a heap has worst-case run time $\Omega(\log n)$; that is, prove that removal from a heap can not have a worst-case run time faster than $\Theta(\log n)$.

Solution:

Two-word Proof: Heap sort :-)

(that should, in principle, be proof enough — but since this is an assignment and we want to be formal about things, then we would expand)

The idea is that we prove the statement by reduction from sort — the reduction being essentially the heap sort procedure; if we could remove from a heap faster than $\Theta(\log n)$, then we would execute heap sort faster than $\Theta(n \log n)$, since we convert to a heap in linear time, then require n removals from the heap.