Priority Queues, Heaps and Heapsort
Computer Science 1321
Spring Semester 2012

0. Introduction:

Early in the semester we studied linear data structures, stacks, queues and deques (double ended queues). The queues we studied were simple first in first out data structures. The entity that entered the queue first was the entity that was "served" first. Simple, straightforward.

On the other hand, you would bet that if you were in a waiting line at the hot dog stand, and the President of the United States came by, he would get his dog before you do! He has somewhat higher "priority" than you do. Not that he's any more important than you, he just has certain privileges you don't.

In this unit, we shall investigate how we can create a priority queue, or a heap. A priority queue is like a queue in the sense that the item at the front of the queue is served first. But inside the queue, items are arranged according to their priority. The items with the highest priority are at the front of the queue.

Theoretically there could be (and are) several different ways of implementing a priority queue. However, for this discussion we will use what is called a binary heap to implement our priority queue.

The ADT for a priority queue still has the same operations we would expect: enqueue, dequeue, empty, peek, print, size.

1. Binary Heap:

The data structure "binary heap" is a binary tree with certain properties. We give some definitions:

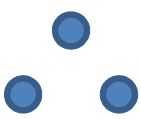
Def: binary tree: a tree in which each node has at most two children. This means that a node could have 0, 1, or 2 children.

Def: Strictly binary tree: A binary tree in which each node of the tree has either two childred or none.

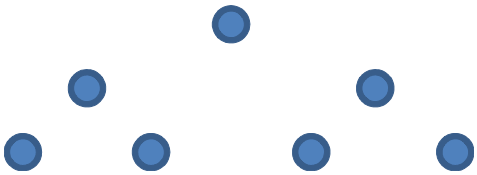
Def: Complete binary tree: A binary tree in which every level is full. There are binary trees of size 1, of size 3 (a node and its two children), of size 7, of size 15, etc.

Complete Binary tree of size 1:

Complete Binary tree of size 3:



Complete Binary tree of size 7:



What are the sizes of complete binary trees? What is the general formula? Can you prove it? How would you prove it?

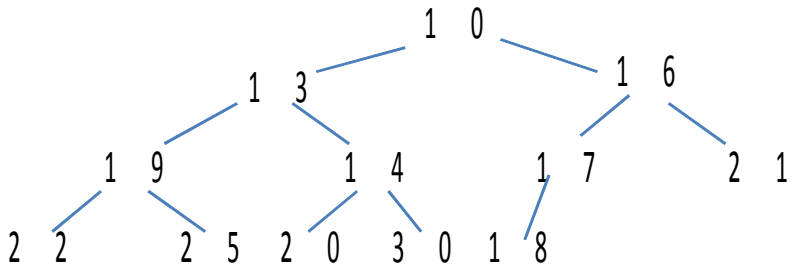
Def: Almost complete binary tree: A complete binary tree in which every level except possibly the last is full. The last level is filled from left to right, but may not be completely full. (Note: your textbook calls an almost complete binary tree a complete binary tree. See the picture on page 225).

For the following discussion we shall restrict our attention to heaps containing numbers, and when we envision a priority queue as a heap, we shall be concerned with the ordinary order relation on numbers and speak of min heaps or max heaps. This simply means that the number with the highest priority is the minimum value (or maximum value) under consideration. Thus we have the following definition:

Def: Min heap: A min heap is an almost complete binary tree in which the root of any tree (or subtree) is smaller than either of its children. Similarly, a max heap is an almost complete binary tree in which the root of any tree (or subtree) is greater than any of its children.

2. Implementation:

As it turns out, there are two possible implementations of a binary heap (we shall restrict our attention to min heaps for the duration of this discussion). Since a binary heap is a binary tree is a tree, we could use a nodes and references approach as we did earlier when we discussed binary search trees. This is a perfectly suitable approach. However, as it turns out, because of the requirement that the tree be almost complete, there is a far easier way to implement a binary heap. We may (as is so often the case) simply use a Python list. Consider the binary heap shown below:



If we start at 10, the root of the tree, and simply list the nodes of the tree from left to right in order by level, we obtain the following list:

h = [10,13,16,19,14,17,21,22,25,20,30,18]

We claim that the list we have named h contains sufficient information to reconstruct the binary heap above. Note that this is indeed a binary heap since in every case the root of the (sub)tree is less than either of its children. Note also that this is an almost complete binary tree. The bottom level need not be completely filled out, but must be filled completely from left to right as far as it goes.

But, if this list is to contain all of the information concerning the heap, we must be able to select any node and find both of its children (if they exist). Study the following table:

node	index of node	children	index of children
10	0	13,16	1,2
13	1	19,14	3,4
16	2	17,21	5,6
19	3	22,25	7,8
14	4	20,30	9,10
17	5	18	11

It is quite clear that we may find the children of the node at index k by calculating

index of left child = 2k+1
index of right child = 2k+2

Also quite clearly the index of the parent of the node at index k is (k-1)/2. (Note that his is integer arithmetic – no remainders)

Thus, by simply listing the nodes of the heap in order by level the list becomes a data structure for the heap. (Note to you textbook readers – this discussion is slightly different than that in the text. For some unknown reason your authors chose not to use the list position with index 0 as part of the tree).

Given an arbitrary list, we must be able to "heapify" it, that is transform it by interchanging elements so that it has the heap property. Note that this will not produce a sorted list – it will merely produce a list (heap) with the heap property that every node is smaller than its children.

This function (method of class) works as follows: You start at the right end of the list. If the element at the right end of the list is smaller than its parent, exchange the elements. Proceed to the next element to the left. Repeat the process. For example, suppose we wish to heapify the following list:

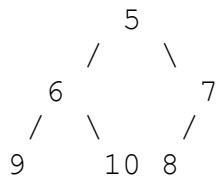
ls = [5, 9, 7, 6, 10, 8]
ind = [0, 1, 2, 3, 4, 5]

parent of 8 is 7. 7 is smaller than 8 so no interchange is necessary.
parent of 10 is 9. 9 is smaller than 10 so no interchange is necessary.
parent of 6 is 9. 6 is smaller than 9 so they must be interchanged. The list becomes

ls = [5,6,7,9,10,8]

parent of 7 is 5. 5 is smaller than 7 so no interchange is necessary.
parent of 6 is 5. 5 is smaller than 6 so no interchange is necessary.

ls = [5,6,7,9,10,8] is a heap.



Note that once you have a heap, the list is not in sorted order. But you do know that the smallest element is at the root of the tree.

Note also that if a heap is to represent a priority queue, the enqueue and dequeue operations must be provided. To insert (enqueue) an item into the heap, simply add the item to the end of the list, and then heapify the list. The new element will find its correct position in the priority queue. (This operation is called perc up in some of the literature).

3. Heapsort:

Now that we have taken an arbitrary list and constructed a binary heap from it, it is easy to use this representation of a priority queue to obtaining a sorting algorithm.

The algorithm proceeds as follows:

Since we know where the minimum element of the list is (it is a heap, so it must be at the root) we may simply pop this element from the list. But then there is a hole at the root. What should we use to replace it? The answer is we replace it with the last element of the list. This ensures that the list remains an almost complete binary tree.

But of course replacing the root of the tree with the last element of the list will surely violate the heap property. Thus, after popping the first element and replacing it with the last element we must rebuild the heap. We simply allow the root of the tree to fall down to its appropriate place by finding the min child and exchanging the root and the min child so that the smallest element of the tree is again at the root.

Once we have reconstructed the heap, we simply repeat the process. Pop the root, replace it with the last element, and allow the heap to be reconstructed as that element percolates down to its position.

4. Student Experiments:

Task One: Implement a Priority queue with all the appropriate queue operations, enqueue, dequeue, empty, peek, size. Use a binary heap implemented as a Python List.

Task Two: Use the binary heap to produce a heapsort. Use your heapsort algorithm to sort the test files you created for the earlier experiments. Compare heapsort with mergesort, quicksort. All of these

algorithms are O(n*ln(n)) algorithms, so should perform similarly. The heapsort proceeds in two steps, building the heap, then producing the sorted list from the heap. Return the sorted list as return value.

Consider the fact that the heapsort can also be implemented as an “in place” sort. That is, all of the operations can be done inside one list.

Note that producing an in place sort is desirable, since space overhead is also a consideration for some algorithms. We know we have a more than adequate amount of main memory in our machines, but this may not always be the case.

Recall that if you are able to perform the heapsort as an in place sort, then it becomes just another “exchange sort” like other sorts we have studied. We simply stir up the list in a certain way until it produces a sorted list.

Note that since heapsort proceeds in two steps, both of the steps count when considering the total time it takes to sort the list. It is claimed in the literature that building the heap can be done in O(n) time, which is in some sense insignificant when considering the overall time for the algorithm, which is O(n*ln(n)). However, it will add some time to the total execution time.

Task Three: (easy modification) provide a priority queue implemented as a max heap. (You may be able to use this to produce your inplace sort. Interchange the root (which is the largest element) with the last element of the list, reduce the size of the list by one (to make sure the last element stays in place) and rebuild the heap).

Task Four: Provide an excel (or similar) spreadsheet with the results of your sorting experiments. Provide also a concise discussion of your findings. What are your conclusions concerning heapsort? Is it sensitive to anomalies in the data? Why do you suppose that is? Would you consider heapsort a stable sort?

Make a folder and put everything you produced for this exercise in it. Name the folder

youruserid.heapsort

and copy it into my directory on or before the due date.

```
cp -r youruserid.heapsort /users/meggen/Public
```