Greg Timmons

CSC505 HW2 Question 3

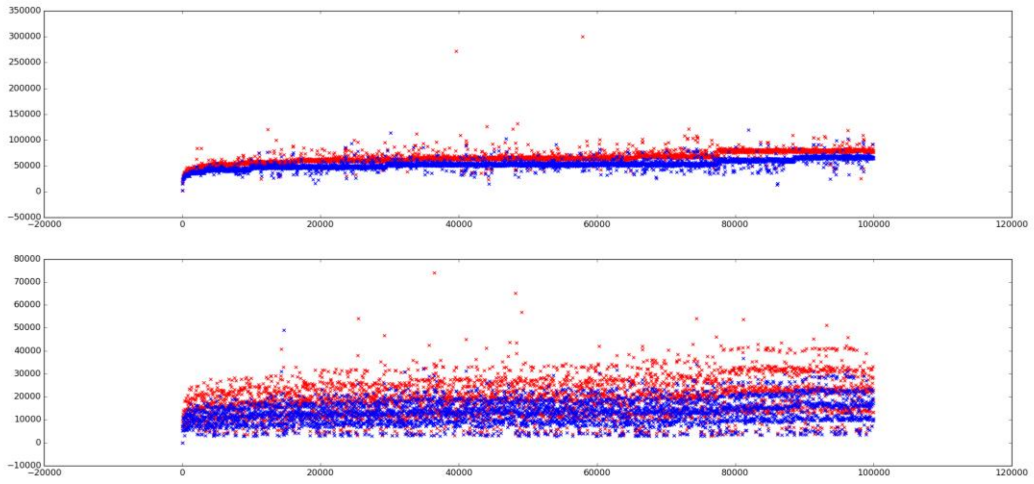
Method: In order to test the runtime of my method, a python decorator was devised to count the runtime of the two methods in question. The decorator will replace the function at load time with a wrapped version of the function the records an elapsed cpu time of the function in relation to the current size of the tree.

```
def timeFunction( func ) :
    def wrapper( *args, **kwargs ) :
        t0 = getTime()
        rt = func( *args, **kwargs )
        t1 = getTime()
        function_data[args[0].__class__][func][len(args[0])].append( t1 – t0)
        return rt
    return wrapper
...
Class Heap3 :
    ...
    @timeFunction
    def insert( self, x ) :
    ...
```

A testing script was then written to test the runtimes of the two heap implementations. Each heap was handed the same set of random inputs in a pattern to ensure one two inserts and one remove we recorded for every possibly size n.

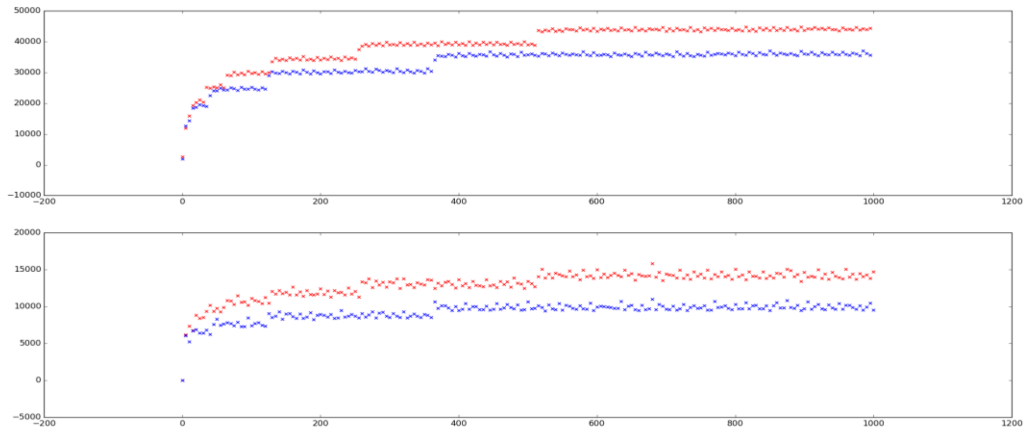
```
for i in xrange( test ) :
    heaps = [Heap2(), Heap3()]
    for j in xrange( rng ) :
        r1, r2 = randint(0,1000), randint(0,1000)
        for h in heaps :
            h.insert(r1)
            h.remove()
            h.insert(r2)
```

Analysis:
First a test run of the average of 2 tests of size 100,000. Blue is trinary heap, Red is the binary heap. The top graph is removals, and the bottom is inserts.



Interesting in this graph we can see some variation based on the number level each operation has to descend to complete. Higher times imply the tree having to descend deep to return. Removals are more consistent in runtime.

Second a test run of the average of 2000 tests of size 1000, this helped to even out the data.



Here we can see the average runtime increase as the depth of the tree increases, and we can see that the binary heap increased more often as the row sizes are smaller.

Questions:

a.) from the graphs above, both functions seem to be asymptotically similar following O(log n).

b.) The trinary constant would clearly have a smaller constant then the binary heap as it runs consistently faster with these two operations.