Stephen Tsung-Han Sher
ID: 7555500940

CS 104 Assignment 7

Question 2

Part A:
The heap property is defined such that for a d-ary tree, any given node, except the root node, the value must be:

For max heap: at most the value of its parent
For min heap: at least the value of its parent

while maintaining the characteristics of a complete tree.

Part B:
Assuming: index of root node is 0

For any node at index k for any given d-ary tree:

Its parent node will be $\left\lfloor \frac{(k-1)}{d} \right\rfloor$
Its child nodes will be $(k \bullet d) + i$ for $i = \{i \in \mathbb{R} \mid 1 \leq i \leq d\}$

Part C:
Let: height 0 mean the root node and only the root node:

Because each node will have a maximum of d children,
therefore the number of nodes at any given height *h* is d^h

The total number of nodes *n* at any given height *h* will be the sum of all the nodes previously:

$$\sum_{i=0}^{h} d^i = n$$

$$\because \quad sum \quad of \quad geometric \quad sequence \quad = \frac{u_1(r^n - 1)}{r-1}$$

$$\therefore \sum_{i=0}^{h} d^i = \frac{d^h - 1}{d-1} = n$$

$$\frac{d^h - 1}{d-1} = n$$

$$d^h - 1 = n(d-1)$$

$$d^h = nd - n + 1$$

$$h = \log_d(nd - n + 1)$$

$$for \quad any \quad node \quad n \ : \ h \quad = \ \lfloor \log_d(nd - n + 1) \rfloor$$

Part D:
For trickleUp and trickleDown functions, the function will run itself equal to the number of times as the height of the tree, therefore the run times of trickle up and trickle down are both $O(h) = O(\log_d(n))$

Both add and remove functions require one trickle up or trickle down, therefore the run time of both add and remove functions are $O(\log_d(n))$

The peek function only looks at the root, therefore the run time is $\Theta(\log_d(n))$

Part E:
As you increase d, the peek function remains the same speed, while the add and delete function both increase in speed.

Question 3

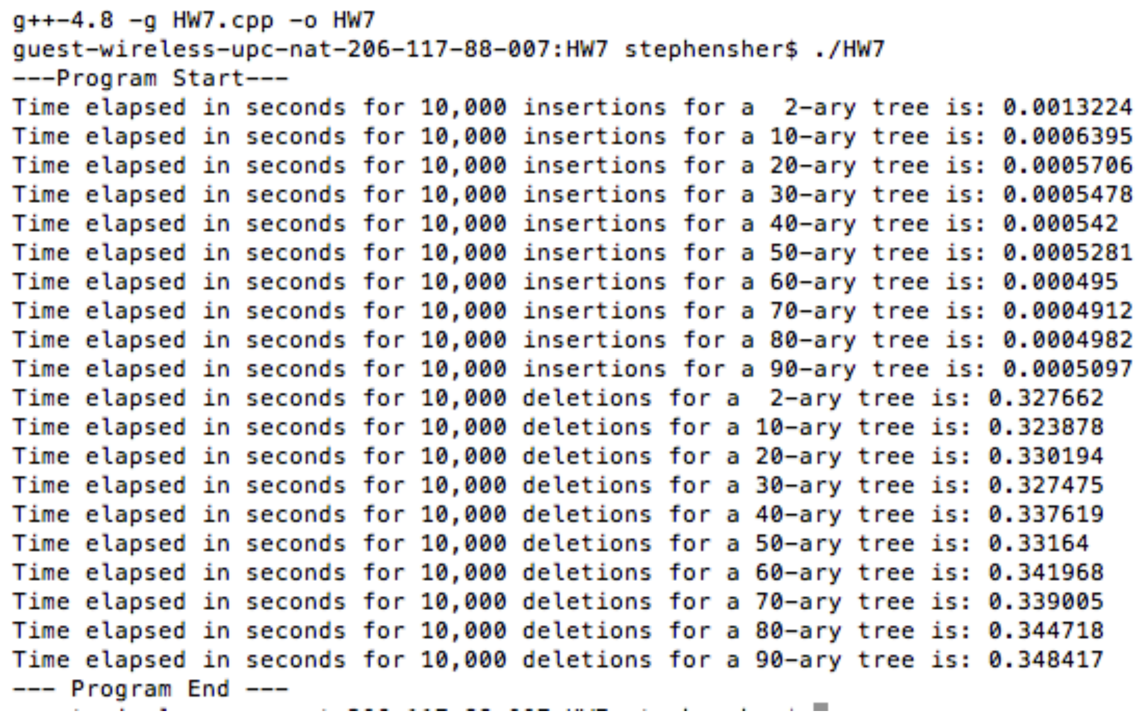
Part A:
I used the doubleSize array because:

1. It has the fastest run time for large quantities
2. Heaps are likely to store large quantities of data

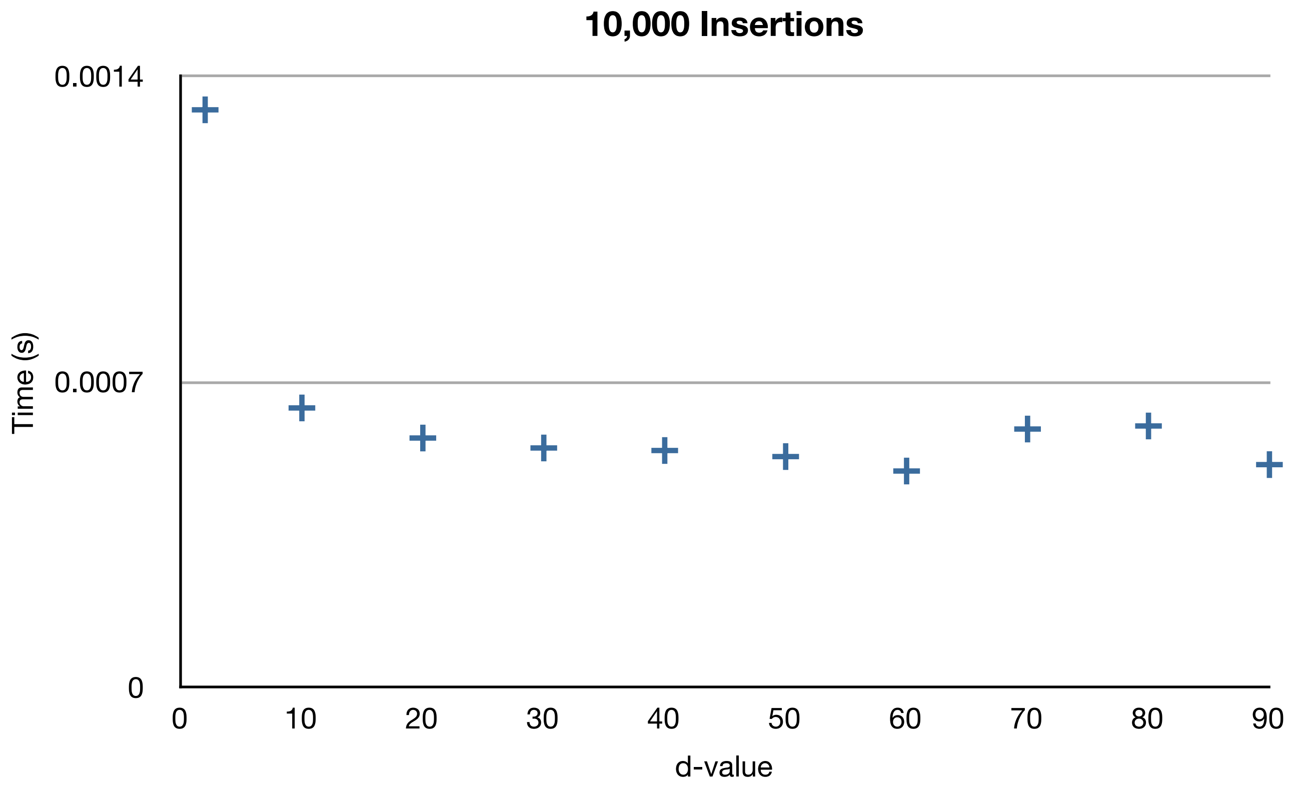
Part B:
Look at heap.h and heap.hpp

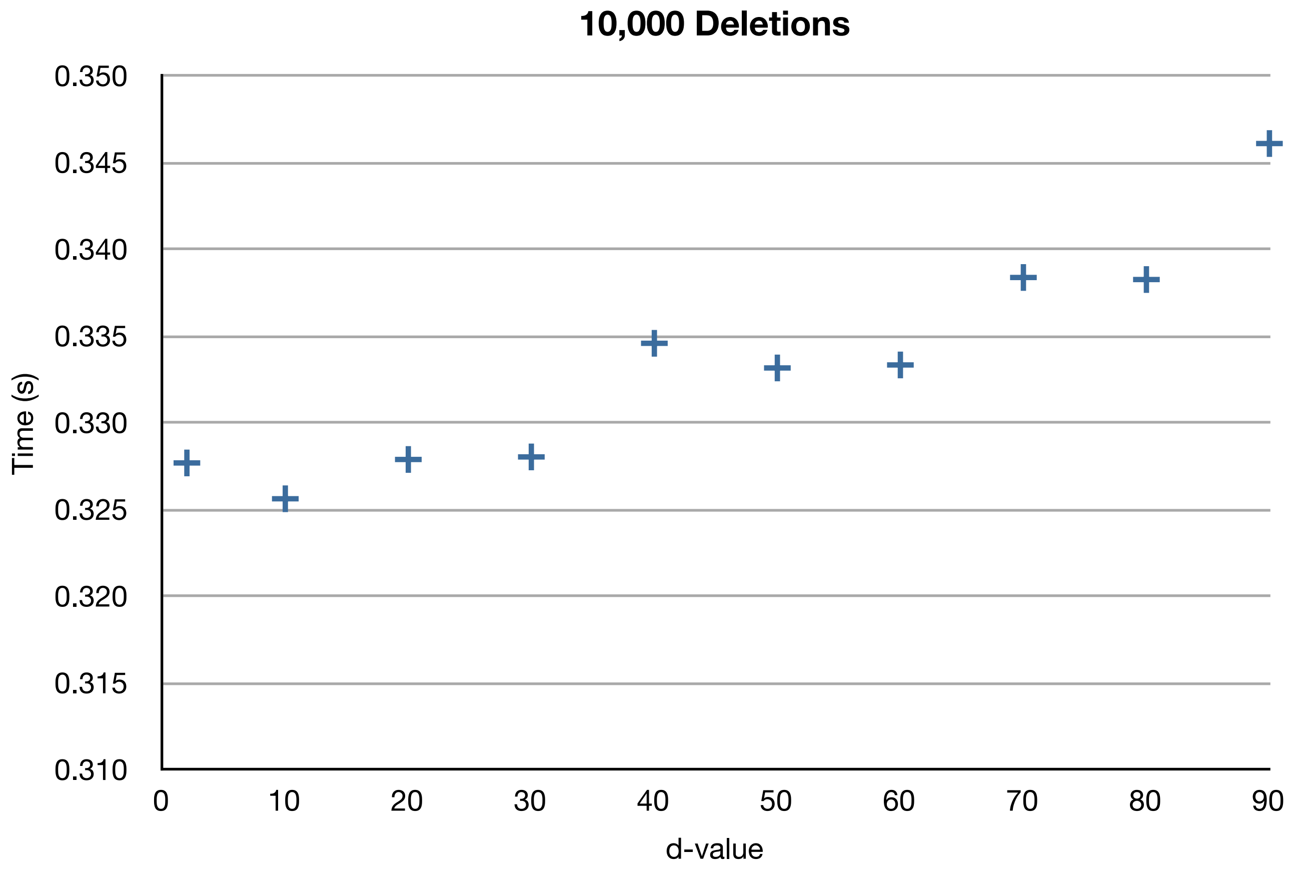
Question 4

Part A and B
Look at HW7.cpp



```
g++-4.8 -g HW7.cpp -o HW7
guest-wireless-upc-nat-206-117-88-007:HW7 stephensher$ ./HW7
---Program Start---
Time elapsed in seconds for 10,000 insertions for a  2-ary tree is: 0.0013224
Time elapsed in seconds for 10,000 insertions for a 10-ary tree is: 0.0006395
Time elapsed in seconds for 10,000 insertions for a 20-ary tree is: 0.0005706
Time elapsed in seconds for 10,000 insertions for a 30-ary tree is: 0.0005478
Time elapsed in seconds for 10,000 insertions for a 40-ary tree is: 0.000542
Time elapsed in seconds for 10,000 insertions for a 50-ary tree is: 0.0005281
Time elapsed in seconds for 10,000 insertions for a 60-ary tree is: 0.000495
Time elapsed in seconds for 10,000 insertions for a 70-ary tree is: 0.0004912
Time elapsed in seconds for 10,000 insertions for a 80-ary tree is: 0.0004982
Time elapsed in seconds for 10,000 insertions for a 90-ary tree is: 0.0005097
Time elapsed in seconds for 10,000 deletions for a  2-ary tree is: 0.327662
Time elapsed in seconds for 10,000 deletions for a 10-ary tree is: 0.323878
Time elapsed in seconds for 10,000 deletions for a 20-ary tree is: 0.330194
Time elapsed in seconds for 10,000 deletions for a 30-ary tree is: 0.327475
Time elapsed in seconds for 10,000 deletions for a 40-ary tree is: 0.337619
Time elapsed in seconds for 10,000 deletions for a 50-ary tree is: 0.33164
Time elapsed in seconds for 10,000 deletions for a 60-ary tree is: 0.341968
Time elapsed in seconds for 10,000 deletions for a 70-ary tree is: 0.339005
Time elapsed in seconds for 10,000 deletions for a 80-ary tree is: 0.344718
Time elapsed in seconds for 10,000 deletions for a 90-ary tree is: 0.348417
--- Program End ---
```


10,000 Insertions
Time (s)
0.0014
0.0007
0
0
10
20
30
40
50
60
70
80
90
d-value


10,000 Deletions
Time (s)
0.350
0.345
0.340
0.335
0.330
0.325
0.320
0.315
0.310
0
10
20
30
40
50
60
70
80
90
d-value

Part C:
For the insertion / add function, it agrees with my theoretical calculations. As the value of d increases, the base of the logarithmic run-time increases, and therefore there will be a large initial decrease in run time for small values of d, but as d increases to larger values, due to the nature of logarithms, the run time will somewhat level out.

This can be seen in the graph as for the first point, d = 2, the run time is relatively high, the there is a sharp drop in run time, and a gradual, but minimal decrease in the general trend of the run time as the value of d increases

For the deletion / removal function, it does not agree with my theoretical calculations. The run time actually increases as the value of d increases. I came to the realization that my theoretical calculation is wrong as I was writing the trickleDown function.

For the trickleDown function, I had to dynamically create an array of size d to store the values of all the child nodes of the current node, then after I need to go through the array and find the largest or smallest value to determine which node to swap. This means that for larger and larger values of d, I will need to dynamically create larger arrays, and be computing with larger and larger arrays, and therefore this constant variable will override the logarithmic factor of the run time, therefore making the run time increase.

Question 5:
See studentRecords.cpp