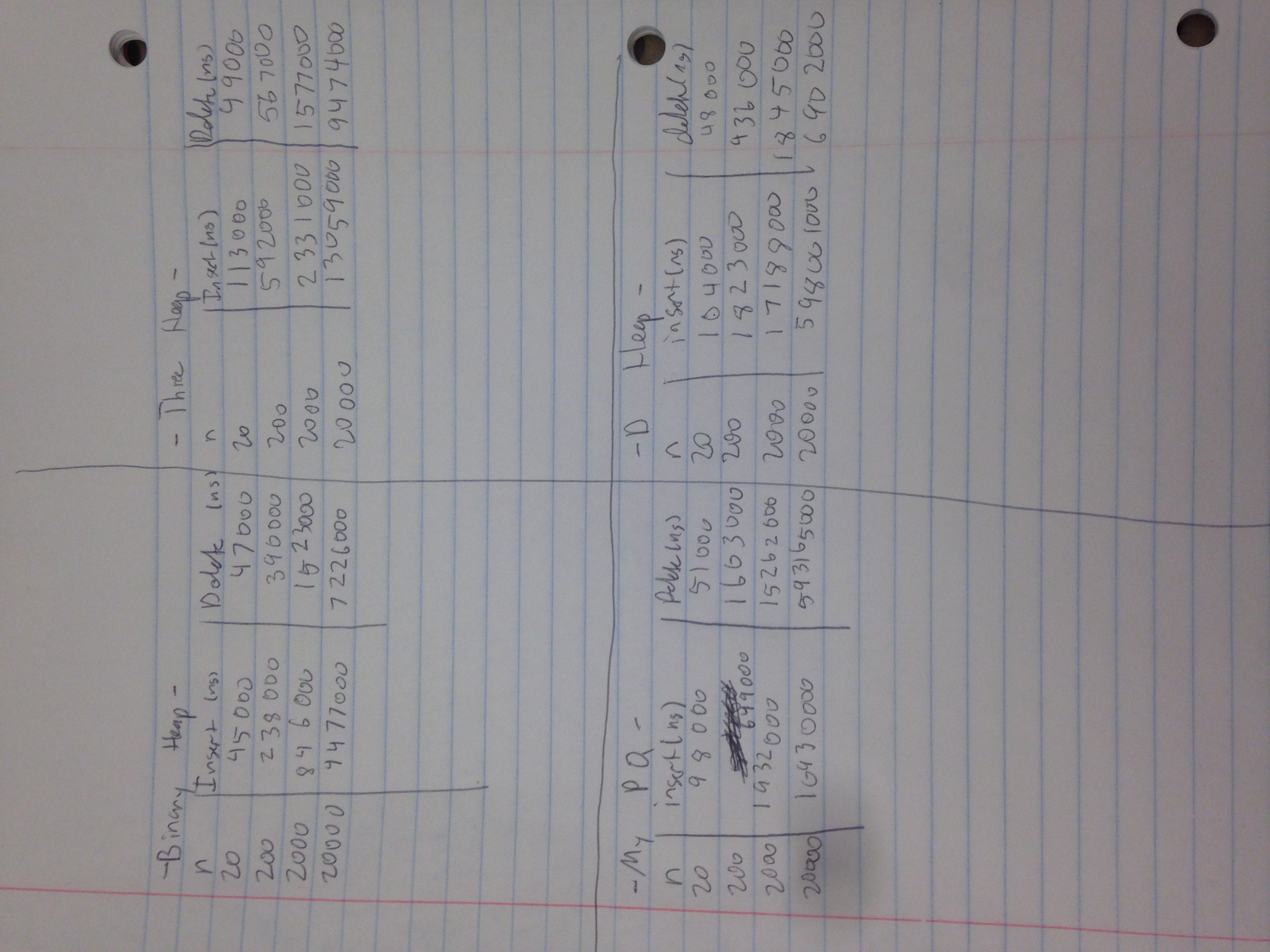
Sam Williams

CSE 373 HW3 Read Me

1. What is the worst case asymptotic running time of isEmpty, size, insert, findMin, and deleteMin operations on all three of your heap implementations? *For this analysis you should ignore the cost of growing the array. That is, assume that you have enough space when you are inserting a value.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Operation (vertical)*  *Heap (horizontal)* | *Binary* | *Three* | *MyPQ* | *D heap* |
| *IsEmpty* | *O(1)* | *O(1)* | *O(1)* | *O(1)* |
| *Size* | *O(1)* | *O(1)* | *O(1)* | *O(1)* |
| *Insert* | *O(log\_2(n))* | *O(log\_3(n))* | *O(1)* | *O(log\_D(n))* |
| *findMin* | *O(1)* | *O(1)* | *O(n)* | *O(1)* |
| *DeleteMin* | *O(log\_2(n))* | *O(log\_3(n))* | *O(n)* | *O(log\_D(n))* |

1. Timing your code: Values of N used : 20, 200, 2000, 20,000 \*\*\*NOTE\*\*\* In the included testing code the value appearing as 20 in the loop statements was increased by a factor of 10 after each respective compile and run. 
2. Compare what you see in your experiments, to what you expected to see based on a big-O analysis. (This is also similar to what you did in Homework 2.) In your discussion, answer these questions:

-Binary Heap

a) Big O asymptotic analysis is useful for predicting measured times because I can see that the times for insert and delete asymptotically grow as I expected.

b) N/A

-Three Heap

a) Big O asymptotic analysis is useful for predicting measured times because I can see that the times for insert and delete asymptotically grow as I expected.

b) N/A

-My PQ

a) Big O asymptotic analysis is useful for predicting measured times because I can see that the times for insert and delete asymptotically grow as I expected.

b) N/A

c) In most cases I would recommend a Binary Heap to someone who simply wanted to create a priority Queue. I would recommend a 3 heap to someone who was with a large number of items.

Or If we were restricted by disk space I would recommend a 3 Heap to an individual because D heaps have more values / key, therefore disk access is reduced, therefore disk access is optimized.

1. Briefly discuss how you went about testing your three heap implementations. Feel free to refer to your testing files, which you should submit.

I created a test file “HW3\_testing” in this file I tested the size, isEmpty, makeEmpty methods of all classes. I also inserted to the heaps elements in ascending order, and the I inserted the same elements in descending order, and made sure the heaps returned the appropriate minimum values when delete or findMin were called while maintaining the correct heap properties and order. I also tested the asymptotic analysis of these operations. \*\*\*NOTE\*\*\* In the included testing code the value appearing as 20 in the loop statements was increased by a factor of 10 after each respective compile and run as to test where n = 20, 200, 2000, 20,000.

5a)

\*\*\*Assuming\*\*\* That the first element (min element) in the Heap is at index 1.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Binary | i\*2 | i\*2 + 1 |  |  |  |  |  | |  |  |
| Three | i\*3 | i\*3 +1 | i\*3-1 |  |  |  | |  |  |  |
| Four | i\*4 | i\*4 +1 | i\*4-1 | i\*4-2 |  |  |  | |  |  |
| Five | i\*5 | i\*5+1 | i\*5-1 | i\*5-2 | i\*5-3 |  |  | |  |  |

5b) For a D-nary tree, \*\*\*Assuming\*\*\* That the first element (min element) in the Heap is at index 1, the leftmost child is as i\*d - (d – 2)