(For my d-heap, I tested with d=20)

1.

	`isEmpty()`	`size()`	`insert()`	`findMin()`	`deleteMin()`
Binary Heap	O(1)	O(1)	$O(\log_2 N)$	O(1)	$O(\log_2 N)$
Three Heap	O(1)	O(1)	$O(\log_3 N)$	O(1)	$O(\log_3 N)$
d-Heap	O(1)	O(1)	$O(\log_d N)$	O(1)	$O(\log_d N)$

2.

`insert()` times

	N=250,000	N=500,000	N=750,000	N=1,000,000
Binary Heap	22.998 ms	40.003 ms	47.542 ms	69.362 ms
Three Heap	18.436 ms	30.991 ms	48.750 ms	63.559 ms
d-Heap	14.445 ms	32.232 ms	41.589 ms	54.181 ms

`deleteMin()` times

	N=25,000	N=50,000	N=75,000	N=100,000
Binary Heap	6.785 ms	13.403 ms	23.440 ms	40.033 ms
Three Heap	7.445 ms	14.934 ms	25.225 ms	41.030 ms
d-Heap	14.369 ms	29.918 ms	49.056 ms	69.282 ms

3.

a) Useful; it gave me a general idea of what time each priority queue should take to insert and delete N elements

b) Predictions did not differ too much except for my d-heap. This slightly strange behavior (taking longer than a binary heap and three-heap) was probably because the compiler made optimizations to binary and three-heap that it couldn't to the d-heap.

c) Three heap would be my recommendation because it is the fastest of the three. The binary heap could be better because it is much easier to create, and the time improvements are very marginal.

4. I created three methods to test:

- `void testFunctions()`
  - First, tested `isEmpty()` and `size()` functions
  - Next, tested `insert()` by checking validity after adding 100 elements
  - Then, tested `deleteMin()` by checking validity after deleting 10 elements
  - Finally, tested the exception handling by calling `findMin()` and `deleteMin()` on an empty PQ
- `void timeAdd(int num)`
  - `num` = number of elements to test the `insert()` function
  - Times the milliseconds it takes to add `num` elements to the PQ
  - Takes the average of 30 test cases
- `void timeRemove(int num)`

- `num` = number of elements to test the `deleteMin()` function
- Times the milliseconds it takes to delete `num` elements to the PQ
- Takes the average of 30 test cases

5. (Assuming head of tree is index 0)

	Children Index				
Binary Heap	i*2+1	i*2+2			
Three Heap	i*3+1	i*3+2	i*3+3		
Four Heap	i*4+1	i*4+2	i*4+3	i*4+4	
Five Heap	i*5+1	i*5+2	i*5+3	i*5+4	i*5+5

For a d-Heap: i*d+1 is the left-most child index