Name: Daniel Wei-Hsuan. Chen
Class: CSE 363
Instructor: Aaron Bauer

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.***

PQ Implemtation Worse Case Asymptotic Running Time Comparison Table

	Binary Heap	Three Heap	Sorted Linked List
isEmpty	O(1)	O(1)	O(1)
Size	O(1)	O(1)	O(1)
Insert	O(logN)	O(logN)	O(N)
findMin	O(1)	O(1)	O(1)
deleteMin	O(logN)	O(logN)	O(1)

2. **Timing your code: Perform several timing experiments (similar to what you did in Homework 2, where you timed pieces of code), to examine the running time of all three of your heap implementations. An experiment will include running the same client code (that uses a Priority Queue) for your three different heap implementations *for at least four different values of N* and timing this. It is up to you to write and to determine what this client code should be. Just be sure that it exercises your `insert` and`deleteMin` operations in a reasonable manner, including eventually deleting everything that has been inserted into the heap. You are not required to explicitly measure calls to `findMin`, `size`, and `isEmpty`but feel free to do so if interested. Similar to Homework 2, graphing your results is recommended, but a table of results will work also. Please note that similar to Homework 2, you are required to turn in the code you used to do your timing experiments.**

	Binary Heap	Growth Rate	Three Heap	Growth Rate	Sorted LinkedList	Growth Rate
Insert (N)						
64000	6.40E+04		0.05829		0.059593	
128000	0.067212	0.0001%	0.07	117.1899%	0.06618	111.0533%
256000	0.064647	96.1837%	0.045222	66.2011%	0.071011	107.2998%
512000	0.094987	146.9318%	0.070819	156.6030%	0.469135	660.6512%
DeleteMin (N)						
64000	3.92E-02		0.039496		0.049251	
128000	0.033127	84.4926%	0.05	126.2001%	0.025811	52.4071%
256000	0.072295	218.2359%	0.076554	153.5872%	0.073315	284.0456%
512000	0.07253	100.3251%	0.069062	90.2134%	0.014866	20.2769%

3. **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:**

   a. How useful was the asymptotic analysis for predicting the measured run time of `insert`

and`deleteMin` for your three implementations?

Asymptotic analysis is able to approximately predict a few trends of the growth rate for the time required to run different values of input in some implementations.
In Insert with Binary Heap, the growth rate is growing larger as the input doubles.. mimicking the growth trend of log(n). It's not as obvious in DeleteMin with Binary Heap, but it's not too off the chart, the trend might be more obvious when we run the test with more than four values.
However, with threeheap's deleteMin we don't see the growth trend of Log(N). The time required seems to decrease with increasing input. We still see the trend asymptotic analysis predicted for Insertion, but not for delete Min.
For sorted linkedlist, asymptotic analysis predicted O(N) for insertion and O(1) for deleteMin. The trend shown by the test does not quite match that. For insertion, we should expect the growth rate to double when the input size doubled, but we do not see that trend here. With deleteMin, the growth rate seems to be fluctuating pretty randomly given the four input. We do not see asymptotic analysis to be too helpful predicting the trend of run time for sorted linkedlist. However, the deleteMin does have significant slower run time comparing to other operations, as predicted to have O(1) run time.

b. If your predictions differed substantially from your measured times, gives reasons why this might have occurred.

There could be different reasons why the predictions aren't matched by the actual results.

1) Sample size is too small so the trend is now shown with only 4 different input value.
2) Variability generated by hardware, processors, memory, OS, java version, libraries, drivers.
3) Other programs that were running could affect the run time

c. Which of your three implementations would you recommend to someone who needs to use a heap? Why is that one preferred? Are there any conditions under which you might suggest using your other implementations?

Threeheap had the best performance according to the testing results, but the run time is really similar to binary heap. So I would recommend either threeheaps or binary heaps. Definitely not sorted linked list due to it's slow run time with insertion.
However, if we don't care about the efficiency of insertion and care more about how much time it takes to deletemin from the priority queue, then sorted linkedlist might be a choice.

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

I am testing different cases with each of the implementation. At first, I am testing that each of the methods are working properly so I just use simple test codes by calling insert and delete min to make sure we are getting the highest priority number when we deleteMin. I also use the function size in the beginning and throughout the code to make sure the size function is working. I also use makeEmpty function and then use size to make sure makeEmpty size if working. I also tried some edge cases to see if exceptions are thrown when they are supposed to by calling more deleteMin than insertion or by calling Findmin and deletemin in the beginning of the code. I've also tested cases when I throw in a int instead of a double into the function to make sure it's not taking other type of data.

5. **You implemented a binary-heap and a three-heap. We could have also asked you to implement a four-heap, a five-heap, etc.**
   a. **In a short table, indicate for a binary heap, a three-heap, a four-heap and a five-heap, where the children for the node at array index `i` are. For example, the first row of the table would indicate that for a binary heap, the two children would be at `i*2` and `i*2+1`.**

Binary Heap	**i * 2**	**i * 2 + 1**			
Three Heap	i * 3 - 1	i * 3	i * 3 + 1		
Four Heap	i * 4 – 2	i * 4 – 1	i * 4	i * 4 + 1	
Five Heap	i *5 - 3	i *5 - 2	i *5 - 1	i *5	i *5 + 1

   b. **For a *d*-heap where *d* is a variable representing the number of children (like two, three, four, five, ...), give an arithmetic formula for calculating where the left-most child for the node at array index `i` are. For example, a *wrong* answer in the right format would be *i*d+14*. Naturally, your formula should produce the right answer for all the rows in your table from part (a).**

d * i – (d – 2)