OutputExecutionTimes

```
n = 50

linked pq start: 15843537663459
linked pq end: 15843547638202
9974743
Array heap  start: 15843547638992
Array heap end: 15843552937433
5298441
Tree heap start: 15843552937827
Tree heap end: 15843561766983
88291560

Process finished with exit code 0

n = 100

linked pq start: 15907410173622
linked pq end: 15907415823792
5650170
Array heap  start: 15907415824187
Array heap end: 15907420839192
5015005
Tree heap start: 15907420839192
Tree heap end: 15907427415464
6576272

Process finished with exit code 0

n = 1000

linked pq start: 15974199136427
linked pq end: 15974247159124
48022697
Array heap  start: 15974247159914
Array heap end: 15974279237153
32077239
Tree heap start: 15974279237943
Tree heap end: 15974359042601
79804658

Process finished with exit code 0

n = 5000

linked pq start: 16051090972359
linked pq end: 16051092088340
1115981
Array heap  start: 16051092088340
```

OutputExecutionTimes

```
Array heap end: 16051092816274
727934
Tree heap start: 16051092816274
Tree heap end: 16051093917648
1101374

Process finished with exit code 0

n = 10000

linked pq start: 16098890504413
linked pq end: 16099161799038
271294625
Array heap  start: 16099161799827
Array heap end: 16099248922113
87122286
Tree heap start: 16099248922902
Tree heap end: 16099800514606
551591704

Process finished with exit code 0
```

Explanation:

The simple linked priority queue is of complexity sorted o(1) insertion but o(n) when searching. However the heap offers o(logn) performace for insertion and deletion
due to its stucture. The Binary tree useage also offers o(logn) and this is why we see
similar values between the array heap and tree heap. My results reflect this and show that a simple
linked list priority queue may be a naive implementation when dealing with large data sets.
In each of my tests the linked implementation was double if not more which shows this.