Write-up

Name: Shubham N. Amrelia

Motive of the assignment:

To understand how to use and implement priority queues and compare the running times of three different implementations.

1. Array of Min-Heap
2. An ordered Linked-List
3. Binary-tree version of min-heap

Below are the tables of running times for each value of n, for each implementation:

Min-Heap Implementation					
Run	n=50	n=100	n=1000	n=5000	n=10000
1	961100	1249700	8285000	24416600	37774300
2	624800	2016000	10133800	22520500	52112100
3	668300	1930200	9699900	26154000	48131500
Avg. Time	751400	1731967	9372900	24363700	46005966.7

PriorityQueue-LinkedList Implementation					
Run	n=50	n=100	n=1000	n=5000	n=10000
1	612900	1191400	13936400	97817900	476081000
2	591900	1025300	13606500	102265700	516676700
3	489400	1162600	10214300	93203500	432318900
Avg. Time	564733.3	1126433	12585733	97762367	475025533

BinaryTree-MinHeap Implementation					
Run	n=50	n=100	n=1000	n=5000	n=10000
1	1919800	2971800	28857900	496059600	2175741100
2	1580500	1902800	24936700	494000400	2241906900
3	2403900	1874600	26458400	555814400	2238845700
Avg. Time	1968067	2249733	26751000	515291467	2218831233

And below is a table that compares the running times of each of the 3 implementations:

All three Implementations					
Implementation Type	n=50	n=100	n=1000	n=5000	n=10000
Array MinHeap	743400	1507300	11213500	28154600	50172200
PriorityQueue_LinkedList	575700	1046800	10535300	94076100	487652200
BinaryTree_MinHeap	1728400	2301100	23880500	521320400	2128739400

So, after having this much information, we can comment on things such a time complexity, running times, difference between elapsed times of two different implementations, etc. Firstly, we notice that the most efficient implementation for testing small values (n -> [1,1000]), in terms of running times, is clearly the LinkedList implementation as it has the shortest time for n=50, n=100 and n=1000. In contrast, the BinaryTree - Min heap version is the slowest among the three implementations for any value of n because it's time for n=50 is clearly more than thrice the time of LinkedList at n=50. Most importantly, we can say that Array version of min-heap is the fastest, and most efficient, for the big values of n (n>1000) as it's elapsed time for n=10000 is less approximately 9 times less than that of LinkedList and approximately 40 times less than that of BinaryTree min-heap(Mind blowing, isn't it?).

Moreover, comparing the running times for different values of n in the same implementation reveals a very interesting discovery. To illustrate, comparing the time(n=100) and time(n=1000) for all three implementations shows that the running time of n=1000 is almost 10x more than that of n=100, hence it adds another digit when we shift from n=100 to n=1000. However, the array min-heap does not show such property when we move from n=1000 to n=10000 i.e., it does not add another digit to the running/elapsed time. This is due to the fact that the buildHeap() method has had a huge impact on the efficiency when testing the implementation for bigger values (n>1000). It has improved it drastically. Hence, we got to know that using the right data structure for general programs can improve efficiency not just in theory but also in practical running/elapsed times.