Write Up:

Benchmark Output:

```
== Stack: ArrayListStack ==
 Workload1 bulk push+pop            median:     34.07 ns/op   checksum: 449985000
 Workload2 mixed steady-state       median:     52.87 ns/op   checksum: -1055495428

== Stack: DLinkedListStack ==
 Workload1 bulk push+pop            median:     42.64 ns/op   checksum: 449985000
 Workload2 mixed steady-state       median:    226.14 ns/op   checksum: -1055495428

== Queue: ArrayListQueue ==
 Workload1 bulk enq+deq             median:     33.73 ns/op   checksum: 449985000
 Workload2 mixed steady-state       median:     83.15 ns/op   checksum: 8738648310

== Queue: DLinkedListQueue ==
 Workload1 bulk enq+deq             median:     75.61 ns/op   checksum: 449985000
 Workload2 mixed steady-state       median:    117.36 ns/op   checksum: 8738648310

== PriorityQueue: SortedArrayListPQ ==
 Workload1 bulk enq+deq (uniform priorities)  median:   47851.39 ns/op   checksum: 449985000
 Workload2 mixed steady-state (uniform priorities)  median:   83263.62 ns/op   checksum: -101944034770
 Workload3 skewed priorities (bulk)     median:   57702.88 ns/op   checksum: 449985000

== PriorityQueue: SortedDLinkedListPQ ==
 Workload1 bulk enq+deq (uniform priorities)  median:   60436.94 ns/op   checksum: 449985000
 Workload2 mixed steady-state (uniform priorities)  median:   52971.76 ns/op   checksum: -101944034770
 Workload3 skewed priorities (bulk)     median:   49454.45 ns/op   checksum: 449985000

== PriorityQueue: BinaryHeapPQ ==
 Workload1 bulk enq+deq (uniform priorities)  median:    249.04 ns/op   checksum: 449985000
 Workload2 mixed steady-state (uniform priorities)  median:    176.34 ns/op   checksum: -106337492798
 Workload3 skewed priorities (bulk)     median:    151.82 ns/op   checksum: 449985000
```

The fastest stack was the Arraylist which performed much faster in Workload2. This is probably because of the constant memory allocation of creating and reallocating Nodes in the

DLinkedListStack. The fastest Queue was again the ArrayListQueue. This is probably due to the same reason as before; however, Workload2 performed much slower on each test. This is because of the loss of branch predictability, which led to more cache misses. Last, the BinaryHeapPQ performed exponentially better than the other two. This is due to the fact that the sort algorithm for the heap is much more efficient than the others. The heap did not have to search through the entire array unlike the other PriorityQueues. This especially plays out with larger workloads.