ADT Implementations + Performance Shootout

Warmup ops: 15,000
Measured ops: 60,000
Trials: 7 (median reported)

All implementations passed sanity checks and produced identical checksums per workload.

Stack Results

Implementation	W1 Bulk (ns/op)	W2 Mixed (ns/op)
ArrayListStack	14.97	25.79
DLinkedListStack	18.15	80.18

Analysis

The ArrayListStack performed better, especially in the mixed workload. Although both implementations perform push and pop efficiently, the ArrayList stores elements in contiguous memory. This improves cache performance and avoids creating extra node objects.

The linked list version must allocate a node for every push and follow pointers during operations, which increases overhead.

Winner: ArrayListStack

Queue Results

Implementation	W1 Bulk (ns/op)	W2 Mixed (ns/op)
ArrayListQueue	14.34	30.73
DLinkedListQueue	14.20	38.95

Analysis

Both implementations perform similarly in the bulk workload. However, under mixed operations, the ArrayListQueue is faster.

The circular buffer keeps elements in contiguous memory and avoids extra object allocation. The linked list must allocate nodes and follow pointers, which adds overhead.

Winner: ArrayListQueue

Priority Queue Results

Implementation	W1 Bulk (ns/op)	W2 Mixed (ns/op)	W3 Skewed (ns/op)
SortedArrayListPQ	3965.72	4124.32	4467.06
SortedDLinkedListPQ	23542.88	21629.61	25662.27
BinaryHeapPQ	74.09	62.22	59.41

Analysis

The BinaryHeapPQ is dramatically faster than both sorted implementations.

Sorted priority queues keep all elements fully ordered, which makes insertion expensive as the structure grows. The heap maintains only partial ordering, which keeps operations efficient even with many elements.

Between the sorted versions, the ArrayList implementation performs much better than the linked list version due to contiguous memory and fewer allocations.

Under skewed priorities, the heap becomes slightly faster because it often requires fewer adjustments.

Winner: BinaryHeapPriorityQueue

Complete Benchmark Results

ADT	Implementation	Workload	Median (ns/op)
Stack	ArrayListStack	W1 Bulk	14.97
Stack	ArrayListStack	W2 Mixed	25.79
Stack	DLinkedListStack	W1 Bulk	18.15
Stack	DLinkedListStack	W2 Mixed	80.18
Queue	ArrayListQueue	W1 Bulk	14.34
Queue	ArrayListQueue	W2 Mixed	30.73
Queue	DLinkedListQueue	W1 Bulk	14.20
Queue	DLinkedListQueue	W2 Mixed	38.95
PriorityQueue	SortedArrayListPQ	W1 Bulk (uniform)	3965.72
PriorityQueue	SortedArrayListPQ	W2 Mixed (uniform)	4124.32
PriorityQueue	SortedArrayListPQ	W3 Skewed	4467.06
PriorityQueue	SortedDLinkedListPQ	W1 Bulk (uniform)	23542.88
PriorityQueue	SortedDLinkedListPQ	W2 Mixed (uniform)	21629.61

ADT	Implementation	Workload	Median (ns/op)
PriorityQueue	SortedDLinkedListPQ	W3 Skewed	25662.27
PriorityQueue	BinaryHeapPQ	W1 Bulk (uniform)	74.09
PriorityQueue	BinaryHeapPQ	W2 Mixed (uniform)	62.22
PriorityQueue	BinaryHeapPQ	W3 Skewed	59.41

Overall Conclusion

Across all three ADTs, array-based structures generally outperformed linked list implementations due to better memory layout and fewer object allocations.

The heap-based priority queue clearly dominates because it maintains order efficiently without the heavy cost of keeping the entire structure sorted.

When Each Is Best

- **ArrayListStack / ArrayListQueue:** Best overall performance for general use.
- **DLinkedList versions:** Acceptable, but typically slower due to pointer overhead.
- **BinaryHeapPriorityQueue:** Best choice for large or frequently updated priority queues.