PROGRAM 4 REPORT

Kyle Faith & Sam Dawson

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Front Push | Back Push | Front Pop | Back Pop |
| Deque | .00846 MS | .01094 MS | .002122 MS | .00368 MS |
| List | .00308 MS | .01068 MS | .00534 MS | .00364 MS |

The graph above shows that for pushing data to the front, the list is much faster, while the deque falls behind. For pushing data to the back of the list, both deque and list are relatively similar. In terms of popping data off the front of deque and list, the deque algorithm is much faster than the lists algorithm. Finally, for popping data off the back, both deque and list are similar in times. The underlying data structure of a deque is a vector, and a list is a doubly linked list. The Deque time complexity would be O(1) for everything, as it’s a double ended queue, in which you can modify values on both the front and the back without having to go through all n elements. The List’s time complexity should also be O(1), as all of the times are relatively similar to the Deques, and for the same reason you can modify values in the front and the back without having to go through all n elements.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Priority Queue | Merge Sort | Quick Sort |
| Time | 20.60576 MS | 29.11836 MS | 3.86292 MS |

The graph represents the average sort time of list 10,000 numbers ranging from 0 to 10,000 using three different sorting methods: Priority Queue, Merge Sort, and Quick Sort. The graph shows that quick sort was significantly faster than the other sorting methods. Quick Sort is faster because it simply compares each element using a pivot rather than splitting it up into multiple sub-arrays. Priority Queue was slightly faster than merge sort with its simple check and add onto the queue rather using constantly splitting up the data. The average time complexities are all the same at O(n\*log(n)); Quick Sort does however have a significantly worse, worst case time complexity at O(n^2). Merge Sort tends to be better in larger data structures, however it uses more space having a larger space complexity than Quick Sort.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Vector | List | Set | Unordered Set |
| Time | 0.03192 MS | 0.80645 MS | 0.00498 MS | 0.0028 MS |

Our graph above shows that the average finding time for the List takes an extraordinarily long time in comparison to the other three data structures. The underlying data structure for a vector is a dynamically resizing array, the list is a doubly linked list, and the sets are both binary search trees. The time complexity for the vector to find information would be worst case O(n). The time complexity for the list would be O(n), because to be able to get to the information it would have to start from either the beginning or the end of the list. The worst case time complexity of a set would be O(log(n)). Meanwhile, the worst case time complexity of an unordered set would be O(n), but the average is O(1), which is why it is slightly faster than the set.