|  |  |  |  |
| --- | --- | --- | --- |
|  | **Best case**  **(target is the first item in the array)** | **Average case**  **(target is the middle item in the array)** | **Worst case**  **(target is the last item in Array)** |
| **Linear Search** | 0.0002ms | 0.0567ms | 0.1755ms |
| **Binary Search** | 0.0003ms | 0.00036ms | 0.0252ms |
| **Using Lambda** | 0.0006ms | 0.2054ms | 0.6835ms |

Based on the bast case, average case, and worst case timings over 3 repetitions for each case, the best algorithm here is the Iterative Binary Search algorithm. Even though its best case is slower than the linear search (by 0.0001ms), it definitely beats both the linear search, as well as the searching through lambda expression by a significant margin, especially in the average case. The reason it is faster, especially in average and worst case, is because the linear search has a time complexity of O(n/2) and O(n) in average and worst case respectfully. Meanwhile, the binary search in average case has a time complexity of O(log n) which is much faster. In the end, the hardware where the code is being ran also affects the results.

Binary search:

The binary search algorithm has both iterative and recursive types. The only prerequisite for this algorithm is that the array must be sorted. The algorithm works by divide and conquer technique. It repeatedly keeps dividing the array into halves. The checking starts at the middle element. If it matches the target, it ends the search successfully. If the target is smaller than the middle element, it continues to search the left half of the array, otherwise if it is larger, it moves to the right half of the array. This process is then repeated until the target is either found or the search range becomes empty, meaning the target is not in the array. With using this, it makes it much faster for large datasets.