Searching in Arrays

Abdinasir Abdi

September 2024

1 Introduction

This report concludes the third assignment in course ID1021 taken fall of 2024. The report compares how the runtime is affected when searching through sorted and unsorted arrays using different searching algorithms.

2 Sorted search in an array

The first part of the assignment measured how long it would take to search for a random value in an unsorted array. An unsorted array of size (n) was created and filled with random integer values. The following code was provided and used as the search method for the array.

```
 public static boolean unsorted_search(int[] array, int key) {
 for (int index = 0; index < array.length ; index++) {
 if (array[index] == key) {
 return true;
   }
 }
 return false;
}
```

It searches the array and compares all values with a randomized key element. If the key element is present the search will break and return true otherwise, break and return false. With this, a simple benchmark was constructed and used to measure how long it would take to search through different-sized arrays.

```
long totalDuration = 0;
   for (int i = 0; i < iterations; i++) {
      int key = random.nextInt();  // Random key to search for
```

```
        long startTime = System.nanoTime();
        unsorted_search(array, key);
         long endTime = System.nanoTime();

       totalDuration += (endTime - startTime);
   }

    // Calculate and display the average time per iteration
    double averageDuration = totalDuration / (double) iterations;
```

The benchmark did not only measure the performance time for a single search but the average of multiple. By including an iteration parameter, it would not stop after one search but continuously make a new search until the desired iteration. The final time would then be divided by the number of iterations. With this, a more accurate assessment can be made of the performance time for each array size. The following results were obtained through the benchmark for an unsorted array:

Iterations	Size (n)	Average Runtime
1000000	100	42
1000000	200	79
1000000	400	102
1000000	800	195
1000000	1600	355
1000000	3200	635
1000000	6400	1135
1000000	12800	1998

Table 1: Sequential search through an unsorted array, measured in ns

For each test the size of the array doubles and the time increase is proportional. The results only support the expected pattern and confirm the time complexity to be linear $O(n)$

3 Unsorted search in an array

In the next part of the assignment, a similar task was done. The only difference is that it would search through a sorted array. The same benchmark was used as in the previous task and a new method for creating a sorted array was implemented:

```
private static int[] sorted(int n) {
        Random rnd = new Random();
        int[] array = new int[n];
        int nxt = 0;
        for (int i = 0; i < n; i++) {
            nxt += rnd.nextInt(10) + 1;
            array[i] = nxt;
        }
        return array;
    }
```

It generates a sorted array unaffected by the size of the array, with no duplicates, ensuring that the search is complete and correct. The following results were obtained through the benchmark for a sorted array:

Iterations	**Size (n)**	**Average Runtime**
1000000	100	55
1000000	200	75
1000000	400	105
1000000	800	202
1000000	1600	332
1000000	3200	587
1000000	6400	894
1000000	12800	1532

Table 2: Sequential search through a sorted array, measured in ns

The runtime is now slightly faster but the difference could be considered negligible. The search is still sequential and from the results it can be concluded that time complexity is linear $O(n)$

4 Binary search

In this part of the assignment, a new search algorithm was introduced. It starts by comparing the target value with the middle element in the array. If that value is less than the target value it will discard the right half of the array and if greater it will discard the left half. If the middle value is equal to the target value, the run is successful and the search will end. The algorithm is simple and works by repeatedly dividing the search range in half until the target value is reached. This search algorithm is called binary search and translates into the following code:

```
      int firstPos = 0;
    int lastPos = array.length-1;
    while (firstPos <= lastPos ){

        int middlePos = firstPos + (lastPos - firstPos)/2;
         int middleValue = array[middlePos];

        if (target == middleValue)
               return middlePos;

        if( middleValue < target) {
            firstPos = middlePos + 1;
         }

        if (middleValue > target)
               lastPos = middlePos -1;
    }
     return -1;

}
```

The following results were obtained through the benchmark for a sorted array, but this time using the binary search algorithm instead of the sequential search:

Iterations	**Size (n)**	**Average Runtime**
1000000	100	63
1000000	200	69
1000000	400	88
1000000	800	88
1000000	1600	78
1000000	3200	91
1000000	6400	89
1000000	12800	102

Table 3: Binary search through a sorted array, measured in ns

For smaller sets of data, the difference between binary search and sequential search is minimal. It is only when the data gets bigger that it becomes noticeable. As seen from the results in Table 3, the difference in runtime as the array gets bigger only seems to become less every time the array is doubled. This algorithm performs considerably faster, even for bigger sets of data and the time for this type of search algorithm is $O(log(n))$

Binary search only works on sorted arrays by the constructed logic it follows. Therefore if binary search is used on an unsorted array it will not work or the given times are inaccurate. The only way is to sort the data first and then run the binary search. The question is how will this affect the performance time?

Iterations	Size (n)	Average Runtime
1000000	100	198
1000000	200	125
1000000	400	168
1000000	800	189
1000000	1600	192
1000000	3200	184
1000000	6400	160
1000000	12800	178

Table 4: Binary search through an array that gets sorted first, measured in ns

Sorting the array takes approximately as much time as the search itself. Even with this extra step, the runtime is still noticeably better with binary search.

5 Recursive search

The final task of the assignment studied how the previous search algorithms performed over two arrays of the same size. Additionally, a different method of applying binary search was introduced. The main objective of the task was to compare two arrays of the same size and search for any duplicates. The new method used the logic of recursion to implement a binary search. Recursion is a commonly used programming technique where you can recall the function to solve smaller instances of the same problem. It can be described in two key steps:

- Base case: The condition in which recursion stops, if not present the recursion will continue indefinitely, leading to stack overflow.
- Recursive case: This is the part where the function calls itself with modified or set parameters to work towards the base case.

The following results were obtained through benchmarks:

Iterations	**Size (n)**	**Sequential**	**Binary**	**Recursive**
1000000	100	2.2	1.3	0.7
1000000	200	5.5	2.4	2.1
1000000	400	28.1	9.7	2.9
1000000	800	102	22.3	7.8
1000000	1600	580	48.4	11.0
1000000	3200	1560	97.5	25.3
1000000	6400	2904	198	32.1

Table 5: : Sequential, binary, and the new search algorithm operating on 2 sorted arrays. measured in μs

The new search method and binary search perform relatively well compared to the sequential search. The time complexity seems to also have changed from a linear to a quadratic $O(n^2)$, now that it is running over two arrays. The runtime for the binary search was a little slower and from the results seemed to now follow the time complexity of $O(n * log(n))$. The new search method performed best far exceeding the two others and had a linear time complexity of $O(n)$.

To conclude the report one can argue that sorted arrays are easier to work with than unsorted arrays, but that the performance mostly depends on the search algorithm. For sequential search, the runtime difference between the two could be considered negligible. Only first when binary search was used did the difference become noteworthy. While it is true that sorting the array first takes a considerable amount of time, the performance of the two methods that used the logic of binary search performed significantly better.