Sorted

Pontus Hector

Fall 2022

Introduction

This is the third assignment in the course ID1021. In this task one explores the difference between different search algorithms and why sorted arrays may be better to work with when searching for an element in an array.

Search through an unsorted array

In the first benchmark an unsorted array of size (n) is created and filled with random numeric values where no element is equivalent to one another. The array is then searched through by implementing a loop that iterates over all of the elements in that array, comparing them with a randomized key element between 0 and the highest element in the array. If an element in the array is found equivalent to the key element, the loop will break and the method will return true, otherwise the loop will continue the comparison until all elements are evaluated, and if none matches the key element the method will break and return false. The following measurements were obtained through the benchmark:

Iterations	**Array size (n)**	**Average runtime**
1000000	100	42ns
1000000	200	60ns
1000000	400	113ns
1000000	800	200ns
1000000	1600	533ns
1000000	3200	864ns

Table 1: Sequential search through an unsorted array without utilization

These measured values confirms that the time complexity is linear (O(n)).

Search through a sorted array

When searching through a sorted array one can start implementing more efficient search algorithms. In this task the same program is run, but a break is implemented in the search loop which triggers whenever the next element in the array is larger than the key element. This is because one can logically conclude the larger elements further in the array from being equivalent to the key element since the element array is sorted by size. The following measurements were obtained through the benchmark:

Iterations	Array size (n)	Average runtime
1000000	100	50ns
1000000	200	69ns
1000000	400	105ns
1000000	800	175ns
1000000	1600	334ns
1000000	3200	564ns

Table 2: Sequential search through a sorted array with utilization

Even though this program is slightly faster than the previous one, the time complexity of this program is still linear (O(n)) and for larger sized arrays this search algorithm will still take a considerable amount of time.

Binary search through a sorted array

In this task a new search algorithm is implemented where the target value is compared to the middle element of the element array. If they are not equal, the half in which the target element cannot lie is concluded, and the search continues on the remaining half, repeating this until the target value equivalent to the key element is found.

```
while ((last - first) > 1) {
            int index = ((first + last) / 2);
            if (array[index] == key) {
                return true;
            }
            if (array[index] < key) {
                first = index + 1;
                continue;
            }
            if (array[index] > key ) {
                last = index;
            }
```

If the search ends with the remaining half being empty, the target is not in the array and the search loop will be exited. The following measurements were obtained through the benchmark:

Iterations	Array size (n)	Average runtime
1000000	100	46ns
1000000	200	55ns
1000000	400	58ns
1000000	800	68ns
1000000	1600	75ns
1000000	3200	80ns

Table 3: Binary search through a sorted array

Now, this program is considerably faster (especially for larger arrays) than our previous programs and the time complexity for this search algorithm is (O(log(n)).

Improved search

In the last task one is supposed to implement sequential search (benchmark 1), binary search (benchmark 3) and a new type of search, and apply the algorithms to work over 2 different sorted arrays of the same size. The sequential and binary search are almost indifferently implemented as before with the adjustment to be able operate over two sorted sorted arrays (by implementing an extra loop in both algorithms which iterates over the new array).

The new search algorithm keeps track of the next element in both arrays and sequentially compares the elements between the arrays.

```
for (int k = 0; k < array.length - 1; k++) {
            if (array[j] == key[i]) {
                counter++;
                i++;
                j++;
                continue;
            }
            if (key[i + 1] > array[j + 1]) {
                j++;
                continue;
            }
            if (key[i + 1] <= array[j + 1]) {
                i++;
            }
```

Whenever an element in one array can be concluded from being equivalent to an element in the other, the array with the concluded value(s) is incremented,

repeating this until the all the target elements have been compared to the key elements in both of the arrays. The following measurements were obtained through benchmarks:

Iterations	**Array size (n)**	**Sequential**	**Binary**	**New**
1000000	100	1.9μs	1.5μs	0.9μs
1000000	200	6.9μs	2.7μs	1.7μs
1000000	400	23.9μs	8.4μs	3.3μs
1000000	800	96.4μs	23.3μs	6.6μs
1000000	1600	526μs	54.9μs	13.1μs
1000000	3200	1417μs	115μs	26μs

Table 4: Sequential, binary and the new search algorithm operating on 2 sorted arrays. Time presented is the average run time for each specific algorithm

These measurements prove a clear difference in run time between the different search algorithms. The time elapsed from running the sequential search algorithm is now growing at a quadratic rate in relation to the size of the arrays (time complexity is O(n^2)) and proves to be considerably inefficient compared to the other search algorithms, especially for larger arrays. The time elapsed from running the binary search algorithm are more favorable than from running the sequential search, however the time complexity of the binary search algorithm is O(n *log(n)*) and for larger sized arrays this search algorithm will still take a considerable amount of time. The new implemented search algorithm proves to be the most efficient algorithm of them all. The time complexity of this new algorithm is linear (O(n)) which makes the time consumption more manageable when operating on larger sized arrays, especially in comparison to the other search algorithms.

To conclude this report one can argue that working with sorted data is much more efficient than working with unsorted data and that different implementations of search algorithms provides a considerable difference in efficiency.