Searching in a sorted array

Fatima Mohammad Ali

Fall 09-09-2023

Introduction

In this assignment, the searching algorithms *linear*, *binary*, and *two pointer*, methods were implemented and compared using benchmarks. The measurements were later analysed to determine the efficiency of each method when searching for a certain key or duplicates in a *sorted* or *unsorted* array.

Linear vs Binary Search

Finding a specific element (*key*) in a given array requires a searching method. In this task, linear search for sorted and unsorted arrays, as well as binary search for sorted arrays were implemented.

Method

The `Linear` search consists of a loop that iterates through a given array while searching for a certain key and returning `true` once the key is found. For sorted arrays, the linear search was slightly optimized by stopping the search once the next element in the array was larger than the key (compared to the unsorted linear search which went through the entire array until the key was found).

The `Binary` search iterates through a sorted array to find a specific element by constantly dividing each search interval in half. The fist and last indices are updated according to the middle index. If the element (at the middle index) is equal to the key, the search is done. If the element is larger than the key, the last page (bigger half) of the array is disregarded and only the first page is examined next and so on. Similarly, if the element is smaller than the key, then only the last page is regarded. Hence, the search interval will constantly be halved after each comparison.

The benchmarks used in this task measured the minimum value of 10 tries, each try calling the wanted method (`Linear.search` or `Binary.search`) 10000 times.

Results

Results of the first task are given in table 1.

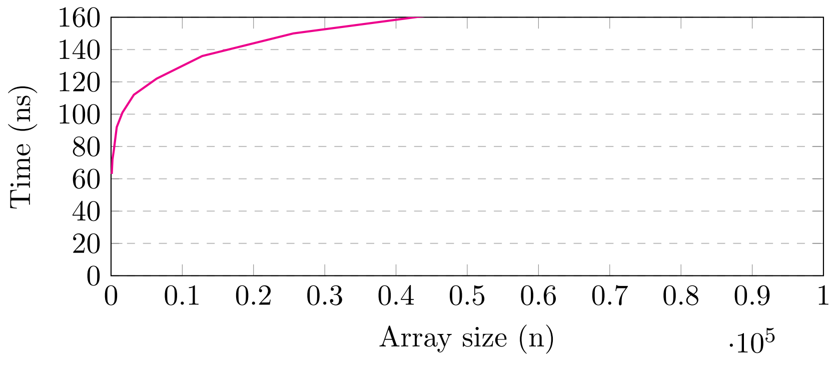
n	**Linear(sorted)**	**Linear(unsorted)**	**Binary**
100	51	42	63
200	78	82	72
400	140	170	78
800	260	320	92
1600	490	610	100
3200	1000	1200	110
6400	2000	2400	120
12800	4100	5000	140
25600	8300	10000	150
51200	16000	20000	170
1000000	320000	320000	280

Table 1: n is the size of the array, *Linear(sorted)*, *Linear(unsorted)*, and *Binary* shows the results in nanoseconds [ns] (rounded to two sign. fig.).

Searching for a specific key in an n sized, *sorted* array could be described with $t(n) = 2n$, which belongs to $O(n)$. As seen in table 1, each doubling of the size n results in the time doubling, which indicates a constant growth with the slope being approx. 2. The time-size relation for the linear search of an *unsorted* array could also be described with $t(n) = 2n$.

Time complexity of the binary search could be described with $O(log(n))$. Plotting the values of the binary search in the following graph (figure 1) further shows the logarithmic relation of this algorithm.

Figure 1. Binary search for sorted array



Discussion

As seen in table 1, the linear search showed drastically larger values compared to the binary search. This was to be expected since the algorithm goes through entire arrays, item by item, without any shortcuts or optimizations, which costs a lot in time. This could also be backed by comparing the time complexity, $O(n)$ versus $O(log(n))$, which also shows how the linear search grows larger and faster than the binary search.

Further analysis of the results shows that the sorted linear search showed a slight improvement in time compared to the unsorted linear search, but the change is less significant in the long run (see the timestamps for n = 1 000 000 in table 1). A reason for the slight (rather than large) improvement could be that the optimized version has to constantly check the additional if-statement, which still costs in time.

Duplicates

In this task, three algorithms, i.e. linear search (unsorted arrays), binary search (sorted arrays), and two pointer search (sorted arrays), were used to search for duplicates in two arrays of size n.

Method

To find duplicates in two unsorted arrays, each item in the first array was searched for in the second array. This was done by using the previously implemented linear search method on two randomly generated, unsorted arrays. Each duplicate that was found was added to a counter and at last returned as an integer.

The next duplicate searching method used binary search on two sorted arrays. For each item in the first array, a binary search was done in the second array and if a duplicate was found, a counter was incremented.

The last searching algorithm was *two pointer* search. The method consists of keeping track of the the next element in the first array, and compare it to the next element of the second array. If the element in the second array is smaller than the next in first, then the index (of the 2nd array) is moved forward. If it is equal or greater than, then the indices of both arrays are moved forward.

Benchmark values in this task measured the minimum value of 10 tries, each try calling the wanted method 10 times.

Result

Searching for duplicates in two unsorted arrays belongs to $O(n^2)$. This could be seen in table 2 where the time is approx. four times the size n.

The binary search iterates though the first array with n elements once and for each element, one binary search on the second array is done to find a duplicate. This could be described with $O(n * log(n))$.

n	Linear(unsorted) dup.	Binary dup.	Two Pointer dup.
100	31	9.5	8.1
200	15	5.6	1.3
400	46	12	2.7
800	170	31	6.5
1600	670	81	3.5
3200	2700	260	7.0
6400	11000	530	14
12800	46000	1100	28
25600	190000	2200	56
51200	750000	4600	112

Table 2: n is the size of the array, *Linear(sorted) dup.*, *Binary*, and *Two Pointer dup.* shows the results of the benchmarking given in nanoseconds*1000 [ns] (rounded to two sign. fig.).

The two pointer algorithm iterates through each array only once, making its time complexity constant, $O(n)$.

Discussion

The results of task 2 could be analysed by comparing the time complexity of each method and determining which big-o is stronger than the other. The linear search looks almost laughable compared to the binary search which in itself seems almost useless compared to the two pointer method. (In conclusion, linear search should die).

the cost of sorting

When searching for a key or duplicates, it would be worth it to sort the arrays first and then perform binary search procedure, as analysed in this rapport.

Reflections

To display the results as graphs would be much better than tables, but since there was a lack of time and knowledge to construct graphs, tables were used.