Searching in a sorted array

Fredrik Berzins

Fall 2023

Introduction

Searching for a key in an unsorted array is as you probably now know quite expensive. If the elements in the array are not sorted, the only way to find an element is to go through the whole data structure. As you will learn things will become much easier if the array is sorted.

Searching explanation

When searching for an element in a data set, their are multiple algorithms to choose from for example binary search or the simplest method linear search. To compare different algorithms Big O notation is used, this is a way of easily compare how long a search will take for different algorithms and data sizes.

Linear or Binary

When selecting a sorting algorithm the time complexity is important, its also important to know if the data set is sorted or not. If the data set is unsorted the quickest and only method is linear search while if it's sorted the quickest for large data sets is binary search. (If it's a small data set linear might be faster.)

Searching implementation

all algorithms relay on a loop to re-do a set number of instructions to get closer to the right answer usually by iteration thru the list, what effects the time complexity is how it iterates. Linear goes from beginning to end, binary goes from the middle to a new middle value.

Linear

A linear search algorithm test each element in a methodical way from one end of an array to another this means the algorithm has a O(n) time complexity.

Improved Linear

This is a linear search that is quicker at (except for real small data sizes) searching by jumping past any any value that is smaller then the key and if it gets to the point wear its larger it returns false instantly instead of going thru the hole loop like normal linear. This improved linear search still has a big O notation of O(n), this is the same as linear but it's still quicker.

Binary

A Binary search algorithm start in the middle of a data set and check if the target is smaller equal or larger, if it's equal its done. If it's not it will exclude the half it not in from the next search and move the middle pointer to the new search areas middle. Binary search has a O(n*log(n)) big o notation.

Duplicate searching implementation

When searching for duplicates you use the same functions in a for loop of one array and use it's values as keys for the normal searching algorithm. This means the linear duplicate search is the linear search in a for loop and same for binary.

Better Linear

The structure of the improved linear is special since it's working thru both arrays dependant on value sizes. This means it iterates thru array1 if the value of of array2 is larger and the opposite if array2 is larger. By running the if stamens in this order the leas amount of test will run the if equal statement and if all the three if statements fail it will return false. The big O notation of this algorithm is O(n).

```
for(int i = 0, n = 0; i < array1.length && n < array2.length;) {
    if (array1[i] > array2[n]) {
        n++;
        continue;
    }
    else if (array1[i] < array2[n]) {
        i++;
        continue;
    }
    else if (array1[i] == array2[n]) {
        return true;
    }
}
```

Benchmarks

The testing was done with data sets between 100 and 6400 values(doubling each time, 100,200,400...). Each time test was redone 1'000 times witch included searching for 10'000 different keys or thru 1'000 arrays for duplicates to get as accurate results as possible. This gives a min time showing avg from the group with the 1'000 quickest runs.

Result

Search

The result of for unsorted arrays doesn't differ from the sorted when doing a linear search, this is why its not included in the table below. The results in the tables below show that Binary search was faster then linear and better linear at data sets larger then 500 and 700 respectively. While both linear and better linear search algorithms wore faster at smaller data sets, it was significantly slower with larger data sets. The worst case big O notation of the different search algorithms are O(n) for linear, O(n) for better linear and O(log n) for binary.

Array length	**Linear**	**Improved Linear**	**Binary**
100	53	30	113
200	100	70	133
400	186	126	157
800	374	220	181
1600	726	421	207
3200	1397	843	232
6400	2781	1634	257

Table 1: Minimum time to search for a key value in an sorted array in nanoseconds.

Duplicates

The results from the duplicate search shows that the fastest every test is not related to the size of the array, since its up to random placement of variables in an array.

Size	linear	Better Linear	Binary
100	1228	477	2494
200	4454	922	5894
400	18451	2031	15567
800	72352	3797	33931
1600	281620	7553	74566
3200	1109841	15448	164035
6400	4458532	31128	360465

Table 2: Minimum time to search for duplicates in sorted arrays in nanoseconds.

Conclusion

Search

For unsorted data set you only have one option if you don't want to sort before hand witch is linear. If it's a sorted list the fastest option is dependant on the data set size it will handle, if the data sets are small linear or a improved linear method is the best but if the data set is large the best is to go with binary.

Duplicates

Similar situation for duplicate search wear linear is the only method that can handle two unsorted lists wile binary ca only handle one unsorted and one sorted and better linear can only handle two sorted lists. This also explains why the better linear is the fastest since it can assume the most about the data sets.