Assignment 3: Searching in a sorted array

Mahad Ahmed

Spring Fall 2022

Introduction

In this assignment the main purpose was to find out how quickly we could search trough a sorted array and a unsorted array, also discussion why using one is better than the other. Also we would try different techniques of sorting and searching through an array in the most time efficient manner.

Task 1, A first try

In this first task we were given an algorithm, searching through an unsorted array. Here is the algorithm below,

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

This is a simple algorithm just searching for the key/value in an unsorted array. The task was then to set up a benchmark of how long it takes for it to search through the array. We keep increasing the number of elements in the array to see how it will affect our algorithm.

Benchmark task 1

Here we have the benchmark for the algorithm in the first task.

Elements n	**Run time(ns)**
100	1500
200	1600
400	2000
800	5000
1600	10 000

Table 1: Benchmark of the algorithm *A first try*

As you can see the more elements we get the slower our algorithm is. The run time seems to double every time we double the size of the array. This suggest that we have a algorithm with the time complexity of *O(n)*. Meaning the time it takes for the search algorithm is the time of checking EVERY single element in the array. Not very time efficient.

A sorted array

If we have a sorted array we can do some optimizations without going through every single element in the array. One optimizations we can do is to stop searching once the next element is larger than the element we are looking for. This will save us quite a lot of time by just having the array sorted. So a long section cut short sorted array = love, unsorted array = "angry face emoji". Now when we have a sorted array we can do something better!

Task 2, Binary search

Binary search is a searching algorithm for finding "stuff" in a sorted array. This is a much more time efficient algorithm than a sequential search algorithm. But what is the difference between a sequential search algorithm and a binary search algorithm?

Let say you want to find a piece of information, for example someone in a phone-book. In the phone-book the last names are sorted in alphabetical order. To find the last name *Svensson*, you wouldn't look through every last name beginning with A until you find the name you were looking for. You would probably somewhere near the end of the book or in the middle. Then depending on the letter the last names begin with you would either move back or continue forward. This is essentially how a binary search works. A short illustration below,

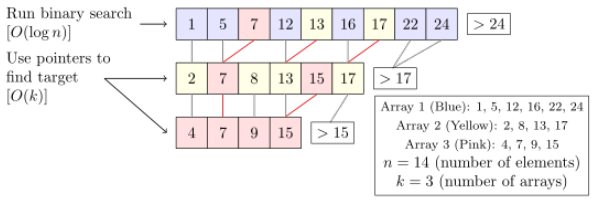

Figure 1: An illustration of how a binary search algorithm works

Implementation of the binary search algorithm

Here I will walk you through the code implementing the algorithm.

```
public static boolean binary_search(int[] array, int key) {

    int first = 0;
    int last = array.length-1;
    long t0 = System.nanoTime();
    while (true) {
        // jump to the middle
        int index = (first + last)/2;
        if (array[index] == key) {
            System.out.println(System.nanoTime() - t0 + "ns");
            return true;
        }
```

In this code section above we implemented the divide and conquer approach. Meaning we set to "pointer" to first and last element of the array. After that we just put another pointer in the middle most element in our array, hence divide.

```
        if (array[index] < key && index < last) {
            first = index + 1;
            continue;
        }
```

In the code section above we will compare if middle is smaller then our input element also if it smaller than the last element. If it is then we can disregard every element after the middle element, because those elements are greater.

```
            if (array[index] > key && index > first) {
                // The index position holds something that is larger
                    than
                // what we're looking for, what is the last possible
                    page?
                last = index - 1;
                continue;
            }
            //item does not exist in target array.
            break;
        }
        return false;
    }
```

In this code section above is almost the same as the one before but difference is we now disregard the left side of the middle of the array.

Benchmark, Binary search algorithm

In this benchmark we would do exactly the same as the for the unsorted array that was implementing the sequential sorting algorithm. We used the same number of elements so that we can compare the two algorithms.

Elements n	**Run time(ns)**
100	1900
200	2000
400	1600
800	1900
1600	1600
1 000 000	8000
16 000 000	15 000

Table 2: Benchmark of the binary algorithm

We also tried for a "huge" number of elements, 64 million. We estimated that it would take about 30 000 ns but it actually took about 11 000. Which is quite a difference. The algorithm is more efficient than we originally thought.

We can see that in the beginning with a "low" number elements this algorithm was quite slow, in fact slower than our sequential counterpart. But as we get more and more elements we can see that in fact it will first decrease and sometimes go up a little bit.But for huge number of elements we will of course have a slower run time but the penalty is not as large as it was for the sequential search algorithm. This algorithm will have a time complexity of *O(log(n))*.

Task 3, Even better

In the previous task we had to implement an algorithm sorting through an unsorted array to find duplicates. But now we have two arrays that are sorted and we want to search through the first list as before, sequentially and the second list we implemented binary search. The problem is the same as before, finding a duplicate. Is this even better?

Implementation of task: Even better

Here we implemented the binary on the second array.

```
public static int nLognAccess(int []array, int[] keys) {
    int duplicated = 0;

    for(int i = 0; i < keys.length; i++) {
        if(binary_search(array, keys[i]) == true)
        {
            duplicated++;
        }
    }
    return duplicated;
}
```

In this code above we just call our binary search method that we created before and apply it to one of the two arrays. So the first one will be searched through sequentially and the second will be search with the binary algorithm.

A quick benchmark of the method

Elements n	Run time(ms)
100	0,11
1000	1,2
10 000	6,5

Table 3: Benchmark of nLognAccess

The time complexity for this method is $O(n * log(n))$. So not as bad as n^2 but not as fast as we want. We can do better!

Better than "Even better"

Here we have done a even better implementation of searching through an array.

```
private static double betterAccess(int[] array, int[] keys) {
       int indexfirst = 0;
       int indexsecond = 0;

       int sum = 0;
       double t_total = 0;

       long t0 = System.nanoTime();
       while (indexfirst < keys.length && indexsecond <
           array.length) {
           if (array[indexsecond] < keys[indexfirst]) {
               indexsecond++;
           } else if (array[indexsecond] == keys[indexfirst]) {
               sum++;
               indexfirst++;
           } else if (array[indexsecond] > keys[indexfirst]) {
               indexfirst++;
           }
       }
       t_total += (System.nanoTime() - t0);
       System.out.println(t_total + "ns");

       return t_total;
   }
```

In the code above we are serching thhrough the arrays finding the duplicates without "resetting". Thus we can cut down our time to linear time O(n). An even better implementation!

Elements n	**Run time(ms)**
100	0,1
1000	1,3
10 000	1,5

Table 4: Benchmark of betterAccess

The cost of sorting

In real life sorted arrays are never a thing. Data is often then not unsorted. So we have to sort through and array first to then use a more time efficient algorithm to find the information we are looking for. We now use a very slow and time consuming sequential algorithm, but question is can we do it faster? It will be a question answered in the next assignment.