Assignment 3 - Searching in java

Jesper Hesselgren

August 21, 2024

Introduktion

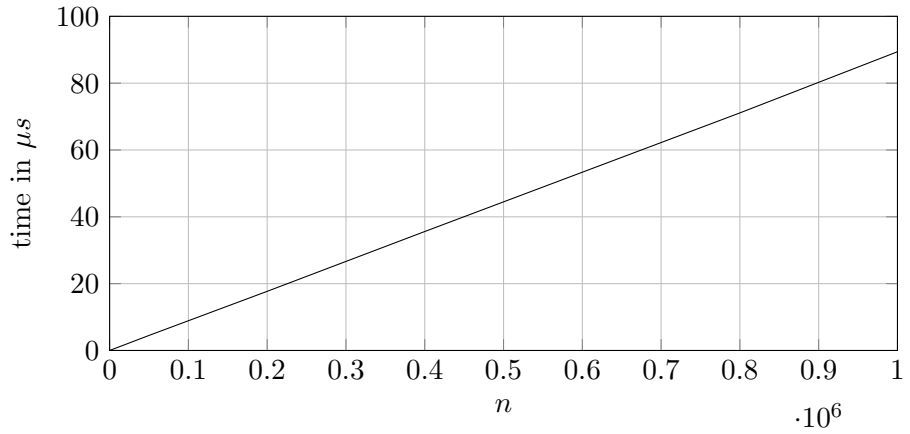
Syftet med denna uppgift är att undersöka hur effektivt vi kan söka efter en specifik nyckel i både osorterade och sorterade arrayer. Sökningen i den osorterade arrayen sker sekventiellt, vilket vi i tidigare uppgifter har sett är ineffektivt för stora datamängder. För att analysera prestandan i en sorterad array kommer vi att jämföra två olika sökmetoder, däribland den välkända binärsökningsalgoritmen. I slutet av rapporten ska även undersöka tekniken rekursiv programmering och se hur vi skulle kunna implentera binärsökning rekursivt.

Sökning i osorterad data

Att söka genom en osorterad array efter en nyckel är en tidsmässigt kostsamma operation som växer linjärt med tidkomplexitetn $O(n)$. I grafen och tabellen med mätvärden blir det väldigt tydligt hur operationen växer linjärt.

Antalet element	**Upplösning** (us)
50 000	4.5
100 000	8.9
200 000	17.7
400 000	35.6
800 000	71.1
1 000 000	89.4

Tabell 1: Mätning av nyckel i en osorterad array



Figur 1: Sökning efter en nyckel i en osorterad array med n stycken element

Sökning i sorterad data

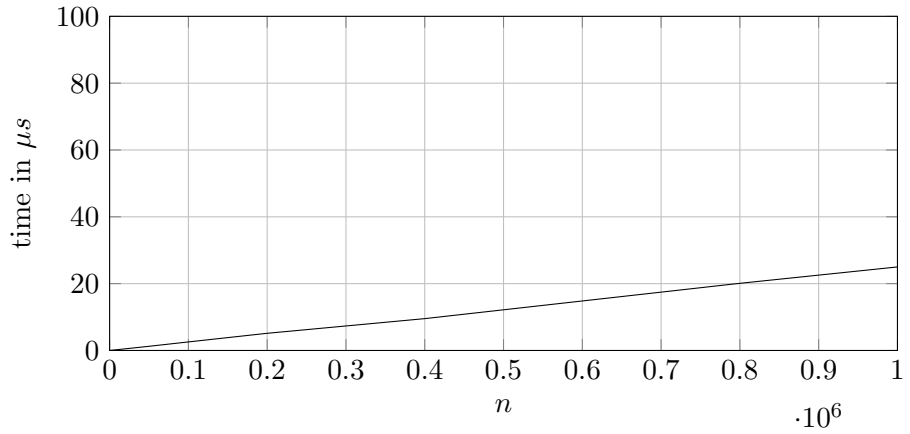
Genom att söka i en array där datan är sorterad kan man optimera sin kod på flera sätt för att göra sökningen mer effektiv. Den första förbättringen vi ska implementera är detta villkor:

```
if (array[index] > key) {
    return false;
}
```

Med denna implementering utnyttjar vi vetskapen om att datan är sorterad, vilket innebär att vi aldrig behöver söka längre i arrayen än nyckelvärdet. Enligt tabellen och grafen nedan kan vi se att vår implementation presterar bättre än den första sekventiella sökningen i osorterad data. Algoritmen har fortfarande en tidskomplexitet $O(n)$ men med en något planare lutning.

Antalet element	**Upplösning** (us)
50 000	1.25
100 000	2.56
200 000	6.62
400 000	9.53
800 000	20.1
1 000 000	25.0

Tabell 2: Mätning av nyckel i en sorterad array



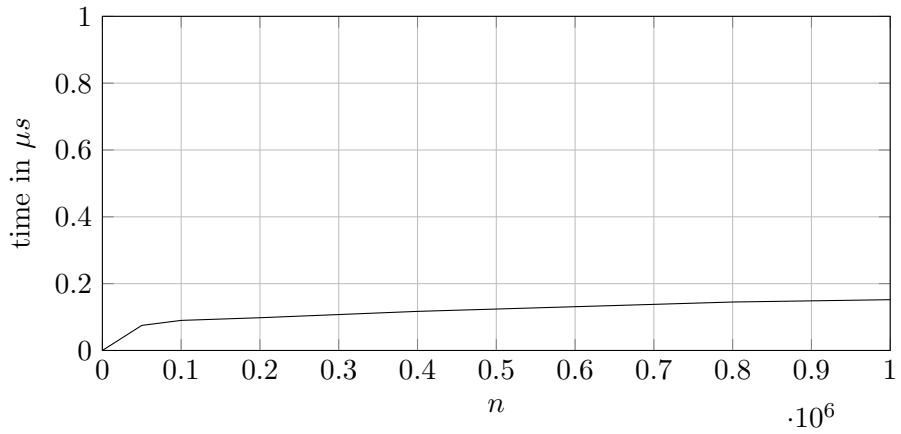
Figur 2: Sökning efter en nyckel i en sorterad array med n stycken element

Binär sökning

Den andra optimerade sökmetoden vi undersöker är den välkända algoritmen binärsökning. Binärsökning fungerar genom att snabbt halvera mängden data som måste undersökas. Istället för att söka igenom varje element i en array ett i taget, börjar binärsökningen med att titta på elementet i mitten av arrayen. Om detta element är större än nyckelvärdet vi letar efter, vet vi att nyckeln måste finnas i den första halvan av arrayen, och vi kan ignorera den andra halvan. Om elementet i mitten är mindre än nyckeln, fokuserar vi istället på den andra halvan av arrayen.Genom att upprepa denna process, halverar vi sökområdet varje gång, tills vi antingen hittar nyckeln eller fastställer att den inte finns i arrayen. Detta gör binärsökning extremt mycket mer effektiv än sekventiell sökning som vi kan se tabellen och grafen nedan.Binär sökning har en tidskomplexitet på $O(log(n))$.

Antalet element	**Upplösning** (us)
50 000	0.075
100 000	0.090
200 000	0.098
400 000	0.117
800 000	0.145
1 000 000	0.152

Tabell 3: Mätning av nyckel i en sorterad array med binärsöknings algoritm



Figur 3: Sökning efter en nyckel i en sorterad array med n stycken element med binärsöknings algoritm

Rekursiv programmering

Ett alternativ till att implementera binärsökning med en while-loop är att använda rekursion. Istället för loopen anropar funktionen sig själv med nya gränsvärden tills nyckeln hittas eller sökningen är klar. Mittvärdet beräknas, och beroende på om nyckeln är större eller mindre, smalnas sökområdet av.

```
public static boolean recursive(int[] arr, int key, int first, int last) {

    int mid = first + ((last - first) / 2);

    if (arr[mid] == key) {
        return true;
    }
    if ((arr[mid] > key) && (first < mid)) {
        last = mid - 1;
        return recursive(arr, key, first, last);
    }
    if ((arr[mid] < key) && (mid < last)) {
        first = mid + 1;
        return recursive(arr, key, first, last);
    }
    return false;
};
```

Diskussion/slutsats

I denna uppgift har vi undersökt prestandan hos olika sökalgoritmer. Sekventiell sökning i osorterade arrayer är som tidigare sätt ineffektivt, särskilt för stora datamängder, med en linjär tidskomplexitet $O(n)$. Genom att sortera arrayen och använda en enkel optimering kunde vi förbättra prestandan något, men den största förbättringen uppnåddes med binärsökning. Binärsökningen, med sin tidskomplexitet på $O(\log(n))$, visade sig vara överlägsen och betydligt snabbare, särskilt vid stora datamängder.Sammanfattningsvis är binärsökning den mest effektiva metoden för att söka i sorterade arrayer.