Introduktion.

Mattias Sandberg

30 Aug 2022

Introduction

Fråga 1: "You first task is to figure out the accuracy of this clock" KLocka 1:

Resultatet blev följnade:

```
resolution 167 nanoseconds
resolution 125 nanoseconds
resolution 83 nanoseconds
resolution 125 nanoseconds
resolution 83 nanoseconds
resolution 83 nanoseconds
resolution 83 nanoseconds
resolution 83 nanoseconds
resolution 84 nanoseconds
resolution 84 nanoseconds
```

Svar: Noggrannheten mellan mätningarna diffar mellan 83 till 167, noggrannheten är cirka 80 nanosekunder.

Fråga 2: "Will the clock be accurate enough to measure how long time it takes to perform a single array access" Klocka2:

```
resolution 292 nanoseconds
resolution 583 nanoseconds
resolution 125 nanoseconds
resolution 125 nanoseconds
resolution 250 nanoseconds
resolution 125 nanoseconds
resolution 250 nanoseconds
resolution 292 nanoseconds
resolution 209 nanoseconds
resolution 84 nanoseconds
```

Svar:Noggrannheten är ej tillräckligt bra för att bestämma den tid det tar för programmet att exikvera en array eftersom skillnaden när vid tio mätningar är så pass stor och att noggranheten på ca 80 nanosekunder är för stor. dvs att den är cirka lika stor den minsta uppmäta tiden.

Fråga 3: "What are we actually measuring? How do we avoid this - can we generate the random sequence before doing the array access?"

Svar: Vi mäter hur lång tid det tar för det första slumpmässiga tal att genereras, sedan jämför vi det med ett annat slumpässigt tal. felet är att det tar extra lång tid för att hela tiden generar ett nytt slumpässigt tal.

Fråga 4: "Could this piece of code help" Svar: detta förhindras genom att det görs en färdig lista med redan slumpade tal. Programmet kan nu hämta nya tal från samma lista.

Task1

Jag använde koden som gavs i PDF:en och implementerade kod där kommenterarna beskrev vad jag skulle göra och lade till main funktion, jag skriver i Java:

```
 // fill the indx array with random number from 0 to n (not including n)
            for (int i = 0; i < l; i++) {
                indx[i] = rnd.nextInt(n);
            }

 // fill the array with dummy values (why not 1)
            for (int i = 0; i < l; i++) {
                array[i] = 1;
            }


// access the array with the index given by indx[i] // sum up the result
                    sum += array[indx[i]];

public static void main(String[] args) {
            System.out.println(access(10000));
    }
```

Tabellen nedan beskriver tiden jämfört med mätvärdena mellan n-värdet 1-10000:

prgm	n-värde	Tid(ns)
Task1	1	2.6
Task1	10	0.56
Task1	100	0.056
Task1	1000	0.021
Task1	10000	0.004

Slutsatserna från testerna och tabellen är att programmet genererar ett linjärt polynom med ett negtivt k-värde (rikningskofficient). Big O (Orda) är därmed O(n). Det går snabbare för desto högre n-värden.

Task2

Jag använde koden som gavs i PDF:en och implementerade kod där kommenterarna beskrev vad jag skulle göra och lade till main funktion, jag skriver i Java:

```
for (int j = 0; j < k; j++) {
            // fill the keys array with random number from 0 to 10*n
            for (int i = 0; i < n; i++) {
                keys[i] = rnd.nextInt(10 * n);
            }
            // fill the array with with random number from 0 to 10*n
            for (int i = 0; i < m; i++) {
                array[i] = rnd.nextInt(10 * n);
            }

public static void main(String[] args){
        System.out.println(search(100,1000,2000));
    }
```

När jag har skrivit koden blandat med det jag fick i PDF:n så testar jag exekveringstiden då jag byter n-värdet mellan 100-1000000. Resultatet presenteras i tabellen nedan:

När jag har hittat rätt M och K-värden så ser jag att för olika n-värden får jag samma tid (0.5ms). Big O är därmed O(1) och är inte beroende av n-input. ett undantag för detta verkar vara väldigt små n-värden och jag är fundarsam över varför det är på det viset. Testerna är körda på en M1 Mac.

n	m	k	Tid(ms)
100	1000	2000	0.5
1000	1000	2000	0.5
10000	10000	2000	0.5
100000	10000	2000	0.5
1000000	10000	2000	0.5

Task3

Metoden för att hitta dubbletter verkar vara väldigt lik den förra uppgiften. För att göra det sätter vi "m" och "n" till samma värden. Om min kod är rätt vilket jag tror att den är så blir exekveringstiden nästan samma när n=m. Big O (Ordo) blir O(n^2), funktionen växer exponentiellt enligt tabellen nedan.

n	m	k	Tid(ms)
1	1	2000	0.2
10	10	2000	0.2
100	100	2000	0.3
1000	1000	2000	0.5
10000	10000	2000	1
100000	100000	2000	4.6
1000000	1000000	2000	47
10000000	10000000	2000	470