Datastrukturer och algoritmer - laboration 1

Grupp 2 - Mats Högberg, Maxim Goretskyy

January 2016

1 Del 2 - komplexitetsanalys

1.1 Version 1

Metoden har tre nästlade for-loopar som alla på något sätt beror av n. Vi kan därför med en handviftning säga att komplexiteten för metoden är $O(n^3)$.

För en matematisk analys ställer vi upp följande summor, där varje summa motsvarar en loop.

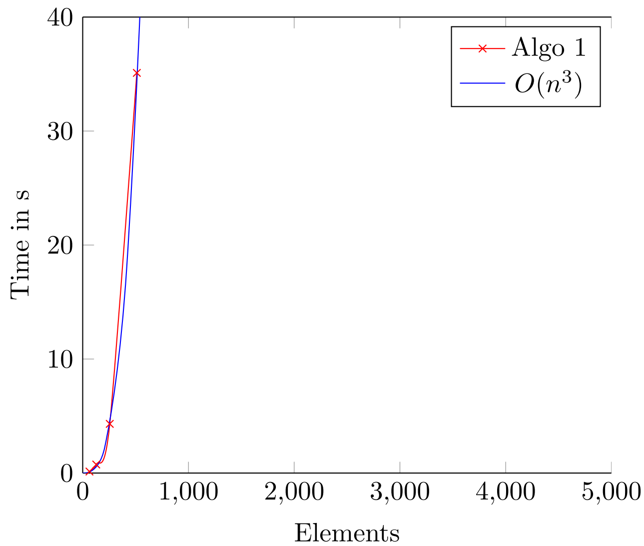
$$\sum_{i=0}^{n-1}(\sum_{j=i}^{n-1}(\sum_{k=i}^{j}1))$$

Vi kan sedan utveckla en summa i taget, och förenkla bort onödiga konstanter i varje steg.

$$\sum_{k=i}^{j}1 = j - 1 + 1 \approx j - i$$

$$\begin{aligned}
\sum_{j=i}^{n-1}(\sum_{k=i}^{j}1) &\approx \sum_{j=i}^{n-1}(j-i) = \sum_{j=i}^{n-1}j - \sum_{j=i}^{n-1}i \\
&= \frac{(n-i)(n+i-1)}{2} - i(n-i) \\
&= \frac{n^2 - i^2 - n + i}{2} - ni + i^2 \\
&= \frac{1}{2}n^2 + \frac{1}{2}i^2 - ni - \frac{1}{2}n + \frac{1}{2}i \\
&\approx n^2 + i^2 - ni - n + i
\end{aligned}$$

$$
\begin{aligned}
\sum_{i=0}^{n-1}(\sum_{j=i}^{n-1}(\sum_{k=i}^{j}1)) &\approx \sum_{i=0}^{n-1}(n^2+i^2-ni-n+i) \\
&= \sum_{i=0}^{n-1}n^2+\sum_{i=0}^{n-1}i^2-\sum_{i=0}^{n-1}ni-\sum_{i=0}^{n-1}n+\sum_{i=0}^{n-1}i \\
&= n^3+\frac{n(n-1)(2n-1)}{6}-\frac{n^2(n-1)}{2}-n^2+\frac{n(n-1)}{2} \\
&= n^3+\frac{2n^3-3n^2+n}{6}-\frac{n^3-n^2}{2}-n^2+\frac{n^2-n}{2} \\
&= n^3+\frac{1}{3}n^3-\frac{1}{2}n^2+\frac{1}{6}n-\frac{1}{2}n^3+\frac{1}{2}n^2-n^2+\frac{1}{2}n^2-\frac{1}{2}n \\
&= \frac{5}{6}n^3-\frac{1}{2}n^2-\frac{1}{2}n \in O(n^3)
\end{aligned}
$$

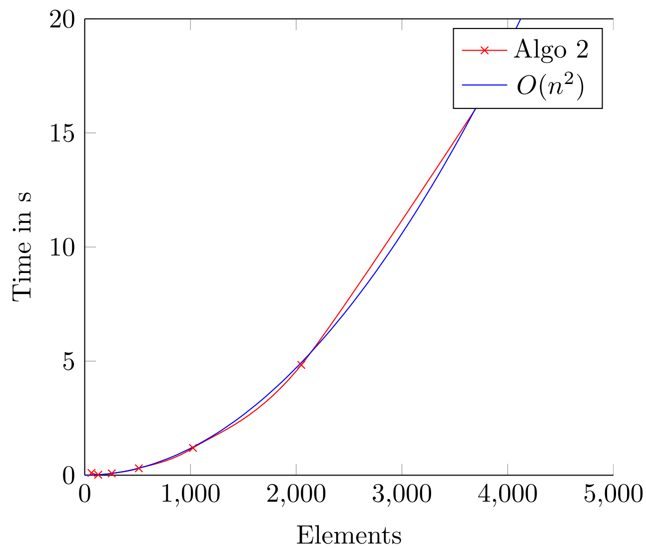


1.2 Version 2

Den här metoden har två nästlade loopar som båda beror av n. Vi får alltså komplexiteten $O(n^2)$.

För den matematiska analysen gör vi på samma sätt som för version 1, men den här gången har vi två summor.

$$
\begin{aligned}
\sum_{i=0}^{n-1}\left(\sum_{j=i}^{n-1} 1\right) &= \sum_{i=0}^{n-1}(n-1-i+1) \\
&= \sum_{i=0}^{n-1}(n-i) \\
&= \sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i \\
&= n^2 - \frac{n(n-1)}{2} \\
&= n^2 - \frac{1}{2}n^2 + \frac{1}{2}n \\
&= \frac{1}{2}n^2 + \frac{1}{2}n \in O(n^2)
\end{aligned}
$$



1.3 Version 3

Version 3 av metoden har endast en loop som går igenom hela arrayen en gång. Den körs alltså n gånger, och komplexiteten blir $O(n)$.

$$\sum_{j=0}^{n-1} 1 = n \in O(n)$$

För en pedantisk analys räknar vi hur många gånger alla operationer i metoden körs.

```java
public static int maxSubSum3(int[] a) {
```

```
    int maxSum = 0;                     // 1
    int thisSum = 0;                    // 1
    for(int i = 0,                      // 1
            j = 0;                      // 1
            j < a.length;               // (n + 1) * 1
            j++) {                      // n * 1
        thisSum += a[j];                // n * 1
        if(thisSum > maxSum) {          // n * 1
            maxSum = thisSum;           // n * 1
            seqStart = i;               // n * 1
            seqEnd = j;                 // n * 1
        } else if (thisSum < 0) {
            i = j + 1;
            thisSum = 0;
        }
    }
    return maxSum;                      // 1
}
```

$$T(n) = 1+1+1+1+(n+1)*1+n*1+n*1+n*1+n*1+n*1+n*1+1 = 7n+6$$

Här räknar vi att if-delen av if-satsen kommer exekveras i varje iteration av loopen. Det spelar dock ingen roll om if-delen eller else-if-delen körs, eftersom båda tar exakt 4 operationer att genomföra.