Gruppövning 2 — Komplexitetsanalys

5DV149

2023 LP3

1 Stora Ordo

Först lite uppvärming med givna $T(n)$:

1.1 $T_1(n) = 10n + 7$

Bestäm c och n_0 för $g(n) = n$ och

$$T_1(n) = 10n + 7.$$

för $g(n) = n$. Är $T(n)$ av $O(n)$?

1.2 $T_2(n) = 4n^3 - 2n^2 + n + 12$

Bestäm c och n_0 för

$$T_2(n) = 4n^3 - 2n^2 + n + 12.$$

och $g_1(n) = n^2$, $g_2(n) = n^3$, samt $g_3(n) = n^4$.

1.3 $T_3(n) = 4n \log n + 3n^3$

Om

$$T_3(n) = 4n \log n + 3n^3,$$

är $T_3(n)$ av $O(n^3)$? Är $T_3(n)$ av $O(n \log n)$?

1.4 $T_4(n) = 4 \cdot 2^n + 3n^3$

Om

$$T_4(n) = 4 \cdot 2^n + 3n^3,$$

är $T_4(n)$ av $O(2^n)$? Är $T_4(n)$ av $O(n^3)$?

2 Algoritmer

Er uppgift är att bestämma tidskomplexiteten för de nedanstående algoritmerna. Detta ska göras genom att räkna antalet *primitiva operationer* som utförs i varje algoritm. Utifrån detta ska sedan definitionen av *Ordo* användas för att ge uttryck för varje algoritms tidskomplexitet. Konstanterna c och n_0 skall bestämmas. För ett exempel på hur en analys kan se ut titta på föreläsningsanteckningarna. Tänk på att fundera kring om det finns ett *värsta-fall* och ett *bästa-fall* och hur de i så fall ser ut.

2.1 Algoritm 1: Summera talen 1 till n

```
Algorithm sumN(n)
input:  A number n.
output:  The sum of the numbers 1 to n.

    sum ← 0
    for i ← 1 to n do
        sum ← sum + i
    return sum
```

2.2 Algoritm 2: Summera alla udda tal mellan 1 och n

```
Algorithm sumNodd(n)
input:  A number n.
output:  The sum of the odd numbers between 1 and n.

    sum ← 0
    i ← 1
    while i <= n do
        sum ← sum + i
        i ← i + 2
    return sum
```

2.3 Algoritm 3: Linjär sökning

```
Algorithm linearSearch(v, n, num)
input:  A vector v containing numbers.
n is the length of the vector v.
A number num to be found in v.
output:  The index of num in the vector v or -1 if not found.

    index ← -1
    i ← 0
    while (index == -1 and i < n) do
        if v[i] == num then
            index ← i
        i ← i + 1
    return index
```

2.4 Algoritm 4: Naiv bubblesort

```
Algorithm bubblesort(arr)
Input:  An array to be sorted.
Output:  The sorted array.

    repeat
        swapped ← false
        for j ← low(arr) to high(arr)-1 do
            if arr[j] > arr[j+1] then
                temp ← arr[j]
                arr[j] ← arr[j+1]
                arr[j+1] ← temp
                swapped ← true
    until not swapped
    return arr
```

2.5 Algoritm 5: Beräkna $x^n y^m$

Nedan finns det två olika algoritmer `math1` och `math2` som båda löser samma problem. Om du granskar de två algoritmerna (du behöver inte räkna operationerna i denna uppgift utan se på det mer övergripande), vilken av algoritmerna har lägst tidskomplexitet? Varför?

```
Algorithm math1(x, y, n, m)
Input:  x, y numbers to be multiplied.
n, m how many multiplications that should be done.
Output:  x^n*y^m.

    res ← 1
    for i ← 1 to n do
        res ← res * x
    for i ← 1 to m do
        res ← res * y
    return res
```

```
Algorithm math2(x, y, n, m)
Input:  x, y numbers to be multiplied.
n, m how many multiplications that should be done.
Output:  x^n*y^m.

    return pow(x, n)*pow(y, m)
```

```
Algorithm pow(x, n)
Input:  x number to be multiplied.
n how many multiplications that should be done.
Output:  x^n

    if n = 0 then
        return 1
    temp ← pow(x, n/2)
    if (n % 2 = 1) then
        return x*temp*temp
    else
        return temp*temp
```

Not: divisionen `n/2` ska tolkas som heltalsdivision (som i C), dvs. resultatet är det heltal som är närmast mindre än n/2. Annan notation: $\lfloor n/2 \rfloor$. Exempel: $\lfloor 4/2 \rfloor = \lfloor 5/2 \rfloor = 2$.