Gruppövning 2 — Komplexitetsanalys — Lösningsförslag

5DV149

2023 LP3

I exemplen nedan har jag ibland ignorerat absolutteckнen. Det är okej bara om man är säker på att funktionen alltid är positiv, t.ex. för $T(n) = 10n + 7$ för $n > 0$. För enklare notation så skriver jag också $T(n)$ utan index (t.ex. ej $T_1(n)$) inom varje sektion då bara en T-funktion behandlas åt gången.

I exemplen så går jag igenom i detalj hur man visar t.ex. hur 2^n växer snabbare än n^3. Ni behöver inte alltid göra det. Om ej annat sägs så är det okej att anta att det är bevisat att

$$\log n < n < n \log n < n^2 < n^3 < \text{ polynom av högre gradtal i } n \ < 2^n < n! < n^n$$

för stora n.

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

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

Är $T(n)$ av $O(n)$?

1.1.1 Vi börjar med c

$$c = \lim_{n\to\infty} \left(\frac{T(n)}{g(n)}\right) = \lim_{n\to\infty} \left(\frac{10n+7}{n}\right) = \lim_{n\to\infty} \left(10 + \frac{7}{n}\right) = 10.$$

1.1.2 Därefter n_0

Vi vet att följande ska gälla:

$$T(n) \leq cg(n), \ \forall n \geq n_0.$$

Sök därför likhet och avrunda n_0 uppåt. Alltså:

$$\begin{aligned} T(n) &= cg(n), \\ 10n + 7 &= 10n. \end{aligned}$$

Saknar lösning för $n > 0$. Alltså duger inte $c = 10$.

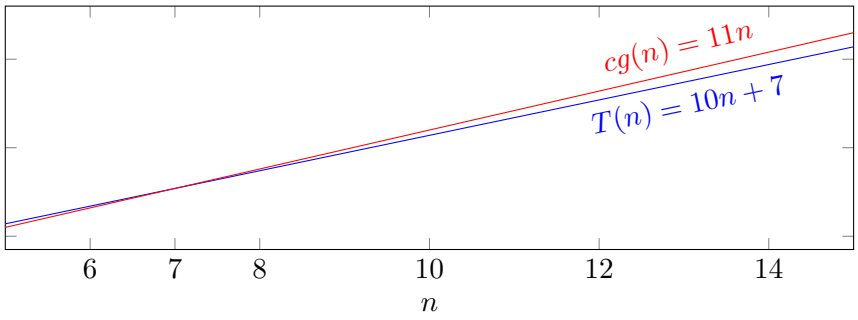

Figure 1: Funktionen $T(n) = 10n + 7$ är uppåt begränsad av $cg(n) = 11n$ för $n \geq n_0 = 7$.

1.1.3 Vi avrundar c uppåt

$$c = \lim_{n\to\infty}\left(\frac{T(n)}{g(n)}\right) + 1 = \lim_{n\to\infty}\left(\frac{10n+7}{n}\right) + 1 = \lim_{n\to\infty}\left(10 + \frac{7}{n}\right) + 1 = 11.$$

1.1.4 Därefter n_0

Vi vet att följande ska gälla:

$$T(n) \leq cg(n),\ \forall n \geq n_0.$$

Sök därför likhet och avrunda n_0 uppåt. Alltså:

$$\begin{aligned} T(n) &= cg(n), \\ 10n + 7 &= 11n, \\ 7 &= n. \text{ (Ingen avrundning behövdes.)} \end{aligned}$$

Alltså bör $T(n)$ vara uppåt begränsad av $11n$ för $n \geq 7$.

1.1.5 Kontrollera

Funktionen

$$u(n) = cg(n) - T(n)$$

visar hur mycket $cg(n)$ *överskattar* den faktiska tiden $T(n)$. Om överskattningen är ≥ 0 för n_0 och *inte minskar* för $n > n_0$ så vet vi att $cg(n) \geq T(n)$ för $n \geq n_0$, vilket krävs för att uppfylla ordo-definitionen. Vi behöver alltså kontrollera att

$$u(n_0) \geq 0,\ \text{(icke-negativ överskattning för } n = n_0), \tag{1a}$$
$$u'(n) \geq 0, \forall n \geq n_0.\ \text{(överskattningen är konstant eller ökar för } n \geq n_0) \tag{1b}$$

För vårt fall:

$$u(n) = 11n - (10n + 7) = n - 7, \tag{2a}$$
$$u(n_0) = 7 - 7 = 0, \tag{2b}$$
$$u'(n) = 1. \tag{2c}$$

Resultat (2b) uppfyller villkor (1a). Ekvation (2c) uppfyller villkor (1b). Alltså är $T(n) = 10n+7$ av $\boxed{O(n)}$ med konstanterna $\boxed{c = 11}$, $\boxed{n_0 = 7}$. Se också figur 1.

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 $g_3(n) = n^4$.

1.2.1 $g_1(n) = n^2$

$$\begin{aligned} c &= \lim_{n\to\infty}\left(\frac{T(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{4n^3 - 2n^2 + n + 12}{n^2}\right) \\ &= \lim_{n\to\infty}\left(4n - 2 + \frac{1}{n} + \frac{12}{n^2}\right) = \lim_{n\to\infty}(4n - 2) = \infty. \end{aligned}$$

Då c är obegränsad så kan inte $T_2(n)$ vara $O(n^2)$.

1.2.2 $g_2(n) = n^3$

$$\begin{aligned} c &= \lim_{n\to\infty}\left(\frac{T(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{4n^3 - 2n^2 + n + 12}{n^3}\right) \\ &= \lim_{n\to\infty}\left(4 - \frac{2}{n} + \frac{1}{n^2} + \frac{12}{n^3}\right) = 4. \end{aligned}$$

Vi provar med att söka likhet:

$$\begin{aligned} T(n) &= cg(n), \\ 4n^3 - 2n^2 + n + 12 &= 4n^3, \\ -2n^2 + n + 12 &= 0. \end{aligned}$$

Hmm... lite komplicerat. Vi vill egentligen att $cg(n) \geq T(n)$. Vi provar oss fram:

n	$4n^3$	$T(n)$
1	4	15
2	32	38
3	108	105
4	256	240

Alltså bör $T(n)$ vara uppåt begränsad av $4n^3$ för $n \geq n_0 = 3$. Vi kollar med överskattningen:

$$\begin{aligned} u(n) &= cg(n) - T(n) = 4n^3 - (4n^3 - 2n^2 + n + 12) = 2n^2 - n - 12. \\ u(n_0) &= u(3) = 3. \\ u'(n) &= 4n - 1. \\ u'(n_0) &= u'(3) = 11. \end{aligned}$$

$u'(n_0)$ är alltså positiv och då $4n$ växer med n så kommer $u'(n)$ aldrig att bli negativ för $n \geq n_0$. Alltså är $T_2(n) = 4n^3 - 2n^2 + n + 12$ av $\boxed{O(n^3)}$ med konstanterna $\boxed{c = 4}$, $\boxed{n_0 = 3}$. Se också figur 2.

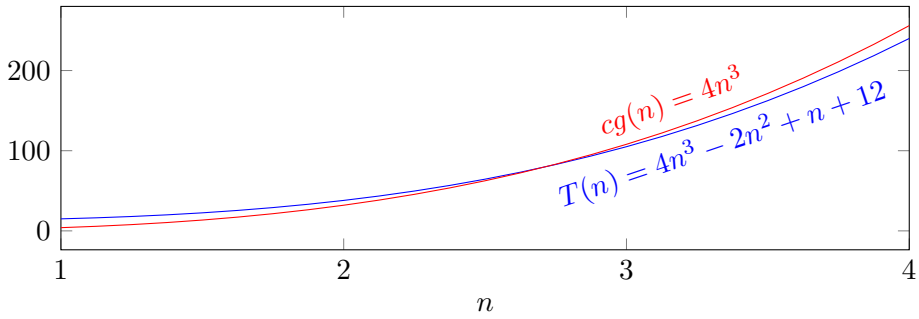

Figure 2: Funktionen $T(n) = 4n^3 - 2n^2 + n + 12$ är uppåt begränsad av $4n^3$ för $n \geq 3$.

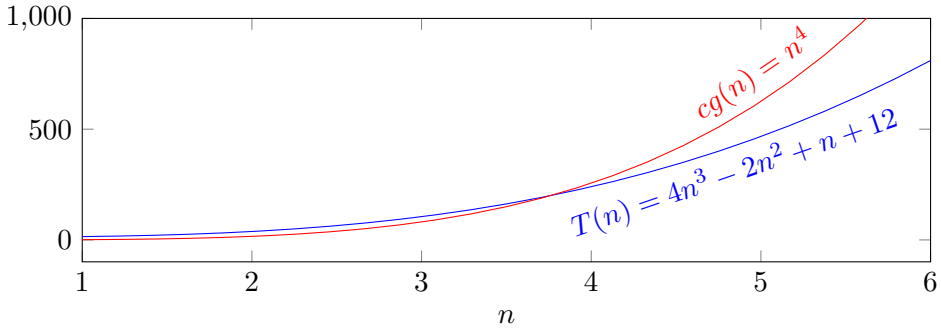

Figure 3: Funktionen $T(n) = 4n^3 - 2n^2 + n + 12$ är uppåt begränsad av n^4 för $n \geq 4$.

1.2.3 $g_3(n) = n^4$

$$\begin{aligned} c &= \lim_{n\to\infty}\left(\frac{T(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{4n^3 - 2n^2 + n + 12}{n^4}\right) \\ &= \lim_{n\to\infty}\left(\frac{4}{n} - \frac{2}{n^2} + \frac{1}{n^3} + \frac{12}{n^4}\right) = 0. \end{aligned}$$

Konstanten $c = 0$ duger inte enligt definitionen, vi avrundar uppåt till $c = 1$ i stället. Vi söker n_0 genom att söka likhet:

$$\begin{aligned} T(n) &= cg(n), \\ 4n^3 - 2n^2 + n + 12 &= n^4, \\ -n^4 - 4n^3 - 2n^2 + n + 12 &= 0. \end{aligned}$$

Nja, vi provar oss fram i stället:

n	n^4	$T(n)$
1	1	15
2	16	38
3	81	105
4	256	240
5	625	467

Alltså bör $T(n)$ vara uppåt begränsad av n^4 för $n \geq 4$. Vi kollar med överskattningen:

$$\begin{aligned} u(n) &= cg(n) - T(n) = n^4 - (4n^3 - 2n^2 + n + 12) = n^4 - 4n^3 + 2n^2 - n - 12. \\ u(n_0) &= u(4) = 16. \\ u'(n) &= 4n^3 - 12n^2 + 4n - 1, \\ u'(n_0) &= u'(4) = 79. \end{aligned}$$

$u'(n_0)$ är alltså positiv och då $4n^3$ växer fortare än $12n^2$ så kommer $u'(n)$ aldrig att bli negativ för $n \geq n_0$. Alltså är $T_2(n) = 4n^3 - 2n^2 + n + 12$ av $\boxed{O(n^4)}$ med konstanterna $\boxed{c = 1}$, $\boxed{n_0 = 4}$. Se också figur 3.

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 $O(n\log n)$?

1.3.1 $g_1(n) = n^3$

(Lägger på $+1$ direkt nu eftersom bägge termerna är större än noll och växande.)

$$\begin{aligned} c &= \lim_{n\to\infty}\left(\frac{T(n)}{g(n)}\right) + 1 = \lim_{n\to\infty}\left(\frac{4n\log n + 3n^3}{n^3}\right) + 1 \\ &= \lim_{n\to\infty}\left(\frac{4n\log n}{n^3} + 3\right) + 1 = \lim_{n\to\infty}\left(\frac{4\log n}{n^2}\right) + 4. \end{aligned}$$

Frågan här är vilken av täljaren $a(n) = 4\log n$ och nämnaren $b(n) = n^2$ som dominerar för stora n. (*Fusktabellen på första sidan säger att n^2 växer fortare. Vi kollar ändå.*) För täljaren gäller att ($\log n$ är den naturliga logaritmen $\ln n$ om ingen bas anges)

$$a'(n) = \frac{4}{n}.$$

Alltså kommer ökningstakten för täljaren att *avstanna* för stora n. För nämnaren gäller att

$$b'(n) = 2n.$$

Ökningstakten för nämnaren kommer alltså att *öka* för stora n. Alltså växer nämnaren fortare än täljaren, bråket går mot noll och

$$c = \lim_{n\to\infty}\left(\frac{4\log n}{n^2}\right) + 4 = 4.$$

Vi testar för att hitta n_0:

n	$4n^3$	$T(n)$
1	4	3
2	32	30
3	108	94

Alltså bör $T(n)$ vara uppåt begränsad av $4n^3$ redan för $n \geq 1$. Vi kollar med överskattningen:

$$\begin{aligned} u(n) &= cg(n) - T(n), \\ &= 4n^3 - (4n\log n + 3n^3) = n^3 - 4n\log n. \\ u(n_0) &= u(1) = 1. \\ u'(n) &= 3n^2 - 4(1\cdot\log n + n\cdot\frac{1}{n}) = 3n^2 - 4\log n - 4. \\ u'(n_0) &= u'(1) = -1. \end{aligned}$$

(Derivatan av $n\log n$ fås från produktregeln: $(fg)' = f'g + fg'$ med $f = n$, $g = \log n$.) För $n = 1$ så minskar alltså överskottet. Vi skulle kunna söka var överskottet är som minst (sätta $u'(n) = 0$), men då vi redan vet att $u(2)$ är positiv provar vi med $n_0 = 2$ i stället.

$$u'(n_0) = u'(2) = 5.23.$$

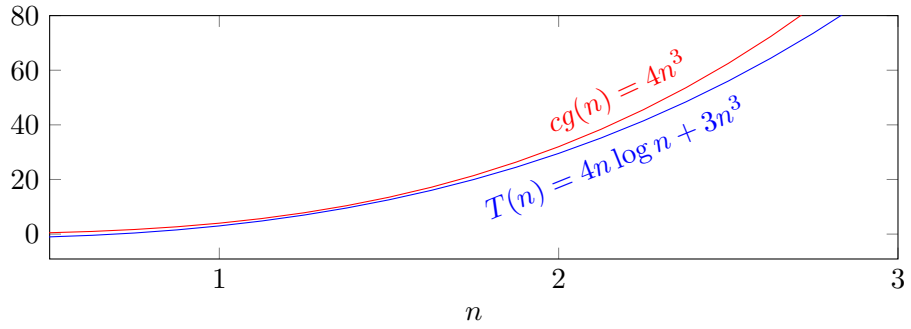

Figure 4: Funktionen $T(n) = 4n \log n + 3n^3$ är uppåt begränsad av $4n^3$ för $n \geq 1$.

$u'(n_0)$ är alltså positiv och då vi redan visat att n^2 växer fortare än $\log n$ så vet vi att ökningstakten kommer aldrig att bli negativ för $n \geq 2$. Alltså är $T_3(n) = 4n \log n + 3n^3$ av $\boxed{O(n^3)}$ med konstanterna $\boxed{c = 4}$, $\boxed{n_0 = 2}$. (Noggrannare kontroll skulle också visat att $n_0 = 1$ duger.) Se också figur 4.

1.3.2 $g_2(n) = n \log n$

(Lägger på $+1$ direkt nu eftersom bägge termerna är större än noll och växande.)

$$\begin{aligned} c &= \lim_{n\to\infty} \left(\frac{T(n)}{g(n)} \right) + 1 = \lim_{n\to\infty} \left(\frac{4n \log n + 3n^3}{n \log n} \right) + 1 \\ &= \lim_{n\to\infty} \left(4 + \frac{3n^3}{n \log n} \right) + 1 = \lim_{n\to\infty} \left(\frac{3n^2}{\log n} \right) + 5. \end{aligned}$$

Som vi tidigare visat så växer n^2 fortare än $\log n$. Alltså blir

$$c = \lim_{n\to\infty} \left(\frac{3n^2}{\log n} \right) + 5 = \infty.$$

$T(n) = 4n \log n + 3n^3$ är alltså **inte** av $O(n \log n)$.

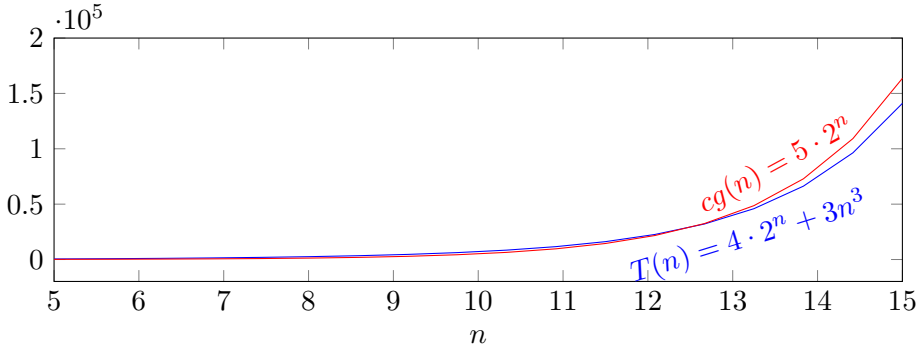

Figure 5: Funktionen $T(n) = 4 \cdot 2^n + 3n^3$ är uppåt begränsad av $5 \cdot 2^n$ för $n \geq 13$.

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 $O(n^3)$?

1.4.1 $g_1(n) = n^3$

Frågan här vilken av $a(n) = 2^n$ och $b(n) = n^3$ som växer fortare. Fusktabellen säger att $a(n)$ växer fortare. $T(n) = 4 \cdot 2^n + 3n^3$ är alltså **inte** av $O(n^3)$.

1.4.2 $g_2(n) = 2^n$

(Lägger på $+1$ direkt eftersom bägge termerna är större än noll och växande.)

$$\begin{aligned} c &= \lim_{n\to\infty}\left(\frac{T(n)}{g(n)}\right) + 1 = \lim_{n\to\infty}\left(\frac{4\cdot 2^n + 3n^3}{2^n}\right) + 1 \\ &= \lim_{n\to\infty}\left(\frac{4\cdot 2^n}{2^n} + \frac{3n^3}{2^n}\right) + 1 = \lim_{n\to\infty}(4+0) + 1 = 5. \end{aligned}$$

Vi testar för att hitta n_0:

n	$5 \cdot 2^n$	$T(n)$
1	10	11
5	160	503
10	5120	7096
15	163840	141197

Alltså är $T(n)$ uppåt begränsad av $5 \cdot 2^n$ för $n \geq n_0 = 15$. $T_4(n) = 4 \cdot 2^n + 3n^3$ är alltså av $\boxed{O(2^n)}$ med konstanterna $\boxed{c = 5}$, $\boxed{n_0 = 15}$. (Noggrannare beräkning skulle givit $n_0 = 13$.) Se också figur 5.

2 Algoritmer

I lösningsförslagen nedan har jag inte räknat några operationer för själva loopandet i loopar eller hopp i if-satser. Det går att motivera att även detta räknas. Viktigare att räkna konsekvent än att de får exakt samma resultat som nedan. Ordo—uttrycken bör givetvis stämma, annars har man gjort fel, och så mycket lär resultatet inte misstämma. Börjar man ha t.ex. dubbelt så höga c-värden så kan man kanske fundera på om man gjort fel.

Tänk på att svaren kan variera en del beroende på hur man valt att räkna olika operationer. Jag har t.ex. räknat på att uppräkning av räknare tar 3 operationer ($i = i + 1$), men det går att motivera att det t.ex. borde räknas som 2.

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                  1 (tilldelning)
   for i ← 1 to n do        1 (initial tilldelning)
                            + 3(n+1) (kontroll av loopvillkor, motsv. i<=n)
                            + 3n (ökning av loopvariabeln, motsv. i←i+1)
                            + n·( innehållet i loopen )
      sum ← sum + i           1 (avläsning av sum) + 1 (avläsning av i)
                              + 1 (operation +) + 1 (tilldelning)
   return sum               1 (avläsning av sum) + 1 (return).
```

Not: Jag har låtit "avläsning" av konstanter vara gratis.

2.1.1 Sammanfattning

$$T(n) = 1 + 1 + 3(n+1) + 3n + n \cdot (1+1+1+1) + 2 = 10n + 7.$$

$$T(n) = \boxed{O(n)}.$$

$$c = \lim_{n\to\infty} \frac{10n+7}{n} + 1 = \lim_{n\to\infty} \frac{10n}{n} + \frac{7}{n} + 1 = 10 + 0 + 1 = \boxed{11}.$$

$$10n + 7 \leq 11n \to 7 \leq n \to \boxed{n_0 = 7}.$$

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

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

   sum ← 0                 1 (tilldelning)
   i ← 1                   1 (tilldelning)
   while i <= n do         3(⌊n/2⌋ + 1) (villkorskoll avrundat nedåt)
                           + ⌊n/2⌋ · ( innehållet i loopen )
      sum ← sum + i           1 (avläsning av sum) + 1 (avläsning av i)
                              + 1 (operation +) + 1 (tilldelning)
      i ← i + 2               1 (avläsning av i) + 1 (oper. +) + 1 (tilldelning)
   return sum              1 (avläsning av sum) + 1 (return).
```

2.2.1 Sammanfattning

$$
\begin{aligned}
T(n) &= 1 + 1 + 3\left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right) + \left\lfloor \frac{n}{2} \right\rfloor \cdot (4+3) + 2 \\
&\leq 4 + 3\left(\frac{n}{2} + 1\right) + 7\frac{n}{2} = \frac{10n}{2} + 7 = 5n + 7. \\
T(n) &= \boxed{O(n)}. \\
c &= \lim_{n\to\infty} \frac{5n+7}{n} + 1 = \lim_{n\to\infty} \frac{5n}{n} + \frac{7}{n} + 1 = 5 + 0 + 1 = \boxed{6}. \\
5n + 7 \leq 6n &\to 7 \leq n \to \boxed{n_0 = 7}.
\end{aligned}
$$

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                                  1 (tilldelning)
i ← 0                                       1 (tilldelning)
while (index == -1 and i < n) do            6(k+1)+k·( loopens innehåll )
   if v[i] == num then                         5 (avläsning i, indexering, array-
                                                  avläsning, avläsn. num, jämförelse)
      index ← i                                2 (avläsning, tilldelning)
   i ← i + 1                                 3 (avläsning, operation, tilldelning)
return index                                2 (avläsning + return)
```

Not: k är antalet gånger loopen körs.

2.3.1 Bästa fallet

Bästa fallet: Loopen körs en gång ($k = 1$).

$$T_{\min}(n) = 1 + 1 + 6 \cdot 2 + 1(5 + 2 + 3) + 2 = 26 = \boxed{O(1)}.$$
$$c = \boxed{26}.$$
$$n_0 = \boxed{1}.$$

2.3.2 Värsta fallet

Värsta fallet: Loopen körs alla gånger ($k = n$). Testet är då alltid falskt, dvs. tilldelningen `index ← i` körs aldrig.

$$T_{\max}(n) = 1 + 1 + 6 \cdot (n + 1) + n(5 + 3) + 2 = 14n + 10 = \boxed{O(n)}.$$
$$c = \boxed{15}.$$
$$n_0 = \boxed{10}.$$

2.4 Algoritm 4: Naiv bubblesort

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

```
repeat                                          k · ( loopens innehåll )
   swapped ← false                                 1 (tilldelning)
   for j ← low(arr) to high(arr)-1 do              3 (avläsning arr, funktionsanrop
                                                     low, initial tilldelning j)
                                                   + 4n (stopptest = avläsning j,
                                                        avläsning arr, funktionsarop
                                                        high, jämförelse)
                                                   + 3(n − 1) (uppräkning av j)
                                                   + (n − 1) · ( loopens innehåll )
      if arr[j] > arr[j+1] then                       8 (avläsning j x 2, operation,
                                                        indexering x 2,
                                                        avläsning arr x 2, jämförelse)
         temp ← arr[j]                                4 (avläsning, indexering,
                                                          avläsning arr, tilldelning)
         arr[j] ← arr[j+1]                            7 (avläsning j x 2,
                                                          indexering x 2, operation,
                                                          avläsning arr x 1,
                                                          tilldelning)
         arr[j+1] ← temp                              5 (avläsning temp + j,
                                                          tilldelning, operation,
                                                          indexering)
         swapped ← true                               1 (tilldelning)
until not swapped                               2k (avläsning, test)
return arr                                      2 (avläsning, return)
```

Not: k är antalet gånger den yttre loopen körs, n är antalet element i `arr`.

2.4.1 Bästa fallet

Bästa fallet: if-satsen blir aldrig sann och $k = 1$.

$$T_{\min}(n) = 1 + 3 + 4n + 3(n-1) + (n-1)(8) + 2 + 2 = 15n - 3 = \boxed{O(n)}.$$

$$c = \boxed{15}.$$

$$n_0 = \boxed{1}.$$

2.4.2 Värsta fallet

Värsta fallet: Varje ytter-loop hittar ett ”nästa” minsta element och kör totalt $k = n - 1$ gånger. If-satsen är först sann $n - 1$ gånger, sen $n - 2$, . . . , och sista varvet 1 gång. Dvs. den inre loopen körs $1 + 2 + \cdots + n - 1 = n(n-1)/2$ gånger. Then-delen av if-satsen tar alltså

$$T_{\text{then}}(n) = \frac{(n-1)n}{2}(4 + 7 + 5 + 1) = 8.5n^2 - 8.5n.$$

Resten som körs varje gång i den inre loopen tar

$$\begin{aligned} T_{\text{rest}}(n) &= (n-1)(1+3+4n+3(n-1)+(n-1)\cdot 8+2)+2 \\ &= (n-1)(15n-5)+2 = 15n^2-15n-5n+5+2 = 15n^2-20n+7. \end{aligned}$$

Tillsammans:

$$T_{\max}(n) = 8.5n^2-8.5n+15n^2-20n+7 = 23.5n^2-28.5n+7 = \boxed{O(n^2)}.$$

$$c = \boxed{24}.$$

$$23.5n^2-28.5n+7 \leq 24n^2 \rightarrow \boxed{n_0 = 1}.$$

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
```

2.5.1 Lösning

`Math1` är $\boxed{O(n+m)}$, `Math2` är $\boxed{O(\log n + \log m)}$. `Math2` är alltså snabbare.