Er uppgift

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 *worst-case* och ett *best-case* och hur de i så fall ser ut.

Algoritmer

Summera talen 1 till n

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

1.	`sum ← 0`	1 op
2.	`for i ← 1 to n do`	Initialisering 1 gång i = 1, 1 op Kontroll av loopvillkor n+1 ggr, i <= n, 3 op Ökning av loopvariabel n ggr, i = i + 1, 3 op
3.	`    sum ← sum + i`	4 op
4.	`return sum`	2 op

Denna algoritm har inget worst-case eller best-case. Loopen körs n ggr.

$$T(n) = 1 + 1 + 3(n+1) + 3n + 4n + 2 = 1 + 1 + 3n + 3 + 3n + 4n + 2 = 10n + 7$$

g(n) antas vara n

$$c = \lim_{n\to\infty} \frac{T(n)}{g(n)} + 1 = \lim_{n\to\infty} \frac{10n+7}{n} + 1 = 11 \qquad 10n + 7 \le 11n \text{ medför att } n_0 = 7$$

Slutsats: sumN är O(n) med c = 11 och n_0 = 7.

Summera alla udda tal mellan 1 och n

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

1.	`sum ← 0`	1 op
2.	`i ← 1`	1 op
3.	`while i <= n do`	Kontroll av loopvillkor n/2+1 ggr, i <= n, 3 op
4.	`    sum ← sum + i`	4 op
5.	`    i ← i + 2`	3 op
6.	`return sum`	2 op

Denna algoritm har inget worst-case eller best-case. Loopen körs n/2 ggr och raderna 4+5 blir 7 op tillsammans.

$$T(n) = 1 + 1 + 3\left(\frac{n}{2} + 1\right) + 7\frac{n}{2} + 2 = 5n + 7$$

g(n) antas vara n

$$c = \lim_{n\to\infty} \frac{T(n)}{g(n)} + 1 = \lim_{n\to\infty} \frac{5n+7}{n} + 1 = 6 \qquad 5n + 7 \le 6n \text{ medför att } n_0 = 7$$

Slutsats: sumN är O(n) med c = 6 och n_0 = 7.

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 op
`i ← 0`	1 op
`while (index == -1 and i < n)`	Kontroll av loopvillkor en gång mer än loopen körs ggr, 6 op
`    if v [ i] == num then`	5 op
`        index ← i;`	2 op
`    i ← i + 1`	3 op
`return index`	2 op

Bästa fallet: Elementet finns på första positionen. Loopen körs en gång och if-satsen är sann.

$T(n) = 1 + 1 + 2 * 6 + 1 * (5 + 2 + 3) + 2 = 26$

g(n) antas vara konstant (1)

$c = \lim_{n\to\infty} \frac{T(n)}{g(n)} + 1 = \lim_{n\to\infty} \frac{26}{1} + 1 = 27$ $26 \leq 27$ för alla n, medför att $n_0 = 1$

Slutsats: Bästa fallet är O(1) med c = 27 och $n_0 = 1$.

Sämsta fallet: Elementet finns inte. Loopen körs n gånger, if-sats aldrig sann.

$T(n) = 1 + 1 + 6(n + 1) + n(5 + 3) + 2 = 14n + 10$

g(n) antas vara n

$c = \lim_{n\to\infty} \frac{T(n)}{g(n)} + 1 = \lim_{n\to\infty} \frac{14n+10}{n} + 1 = 15$ $14n + 10 \leq 15$ medför att $n_0 = 10$

Slutsats: Sämsta fallet är O(n) med c = 15 och $n_0 = 10$.

Naiv bubblesort

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

`Repeat`	
`    swapped ← false`	1 op
`    for j ← low(arr) to high(arr)-1 do`	Init `j← low(arr)` 1 gång, 3 op Loopvillkor n gånger, 5 op `j<= high(arr)-1` Ökning av loopvariabel n-1 gånger, 3 op
`        if arr [ j] > arr [ j +1] then`	8 op
`            temp ← arr [ j]`	4 op
`            arr[ j] ← arr [ j +1]`	7 op
`            arr[ j +1] ← temp`	5 op
`            swapped ← true`	1 op
`until not swapped`	2 op
`return arr`	2 op

Bästa fallet: Redan sorterad lista, if-satsen blir då alltid falsk och raderna 5-8 körs 0 gånger. Repeat-until loopen körs 1 gång if-satsen (rad 4) körs n-1 ggr.

$T(n) = 1\,(1 + 3 + 5n + 3(n-1) + 8(n-1) + 2) + 2 = 16n - 3$

g(n) antas vara n

$c = \lim_{n\to\infty} \frac{T(n)}{g(n)} + 1 = \lim_{n\to\infty} \frac{16n-3}{n} + 1 = 17$ $\quad$ $16n - 3 \leq 17$ för alla n, medför att $n_0 = 1$

Slutsats: Bästa fallet är O(n) med c = 17 och $n_0 = 1$.

Sämsta fallet: Omvänt sorterad lista. If-satsen är då alltid sann och körs n-1 gånger första varvet i loopen, n-2 ggr andra varvet sen n-3,..., 1 ggr. Totalt körs den då 1+2+...n-1 = n(n-1)/2 gånger. Repeat-until-loopen körs n-1 gånger. Vi delar upp T(n) i två delar, en för raderna 5-8, T1(n) och en för allt annat T2(n)

$$T1(n) = \frac{n(n-1)}{2}(4 + 7 + 5 + 1) = \frac{17}{2}n^2 - \frac{17}{2}n$$

$$T2(n) = (n-1)(1 + 3 + 5n + 3(n-1) + 8(n-1)) + 2 = 16\mathrm{n}^2 - 23\mathrm{n} + 9$$

$$T(n) = T1(n) + T2(n) = \frac{17}{2}n^2 - \frac{17}{2}n + 16\mathrm{n}^2 - 23\mathrm{n} + 9 = 24.5\mathrm{n}^2 - 31.5n + 9$$

g(n) antas vara n^2

$$c = \lim_{n\to\infty} \frac{T(n)}{g(n)} + 1 = \lim_{n\to\infty} \frac{24.5\mathrm{n}^2 - 31.5n + 9}{\mathrm{n}^2} + 1 = 25.5$$

$24.5\mathrm{n}^2 - 31.5n + 9 \leq 25.5\mathrm{n}^2$ och krav på att $n_0 > 0$ medför att $n_0 = 1$

Slutsats: Sämsta fallet är $O(n^2)$ med c = 25.5 och $n_0 = 1$.