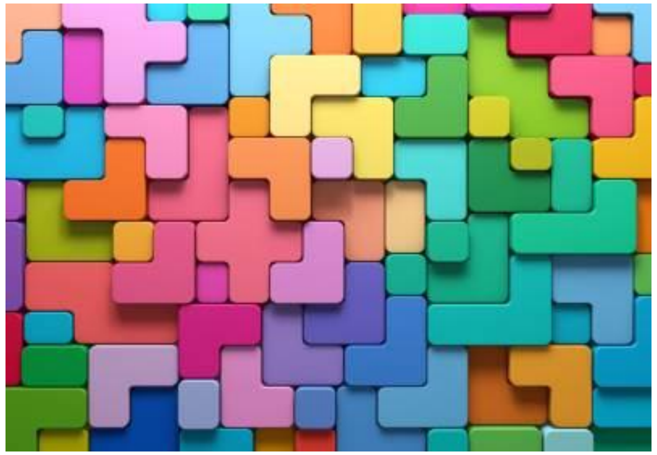
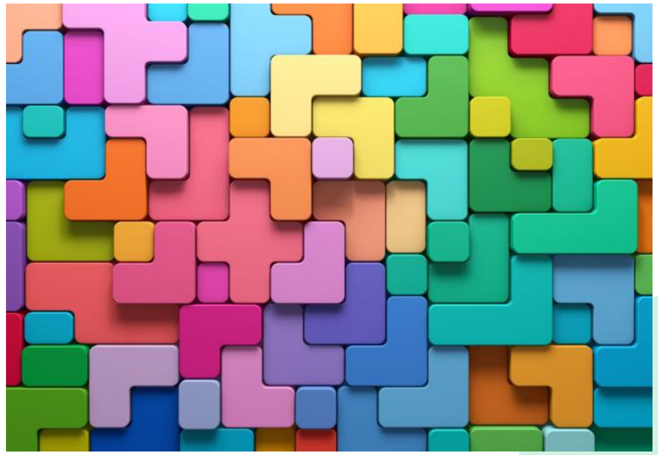
REFERANSER OG SORTERINGSALGORITMER

FORELESNING 16

FREDAG 17/10





(bilder generert av bing image creator)

Læremål: Datastrukturer og algoritmer

- Du har kanskje brukt både *arrays* og *lister* i Python
- Her skal vi gå litt under panseret og se hvordan de fungerer, for eksempel hvordan vi jobber med minnet på datamaskinen
- Dette danner en basis for å kunne lage egne dataobjekter på en god måte når det trengs
- Algoritmer inkluderer bl.a. smarte måter å sortere data på

Læremål: Algoritmeanalyse og optimalisering

- Å kunne analysere hvor rask en algoritme er (viktig når vi får enorme mengder data)
- (+ mer de kommende ukene)

Nytt begrep: amortisert kjøretid

- = gjennomsnittlig kjøretid
- Når både best case og worst case er sjeldne tilfeller som ikke er representative, kan det være det like nyttig å se på hvor lang tid en gjennomsnittlig operasjon tar
- Ser vi på summen av veldig mange operasjoner er det usannsynlig at alle vil være worst case (det vil ofte være raskere i praksis)
- Worst case er fortsatt viktig!

Amortisert kjøretid (prosjekt 2)

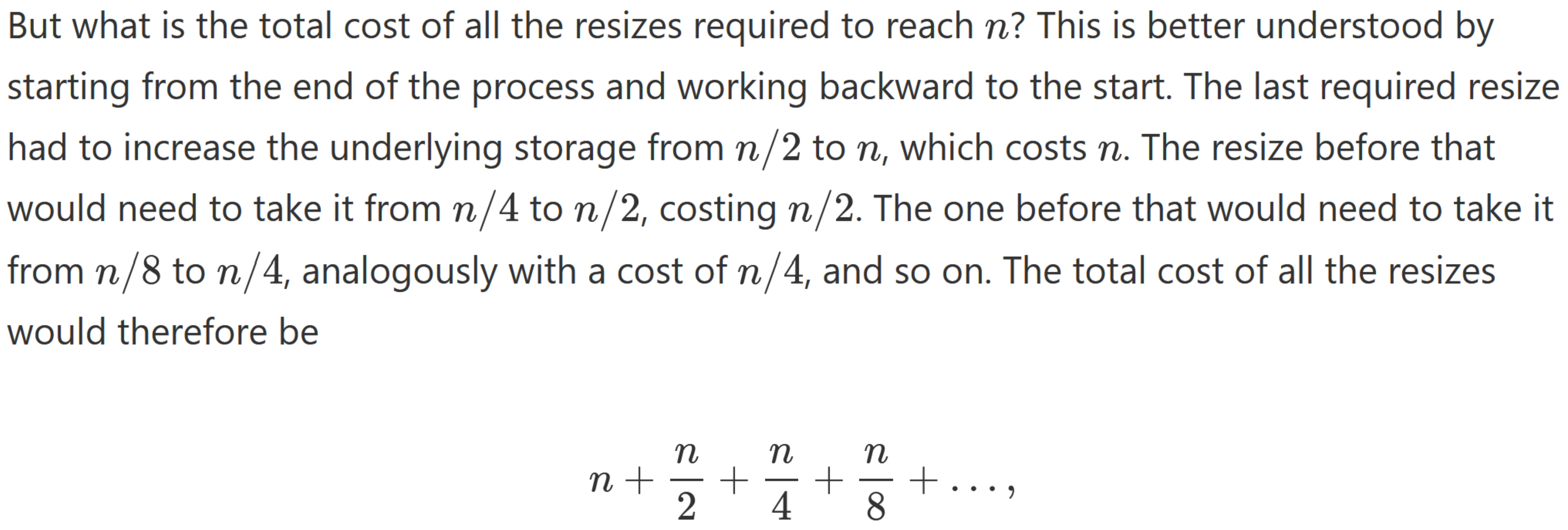
Instead of analyzing a single append. Let us say we start with an empty dynamic array and append n elements to it

```
ArrayList example;

for (int i = 0; i < n; i++)
{
    example.append(i);
}
```

What is the total cost of this operation, in big O, as a function n? Each of the n appends cost $\mathcal{O}(1)$, so n operations of $\mathcal{O}(1)$ will cost a total of $\mathcal{O}(n)$.

Amortisert kjøretid (prosjekt 2)



But what is the total cost of all the resizes required to reach n? This is better understood by starting from the end of the process and working backward to the start. The last required resize had to increase the underlying storage from $n/2$ to n, which costs n. The resize before that would need to take it from $n/4$ to $n/2$, costing $n/2$. The one before that would need to take it from $n/8$ to $n/4$, analogously with a cost of $n/4$, and so on. The total cost of all the resizes would therefore be

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots,$$

Amortisert kjøretid (prosjekt 2)

This sum is a geometric progression, which sums out to $2n$. The easiest way to see this is to draw out the series as several boxes

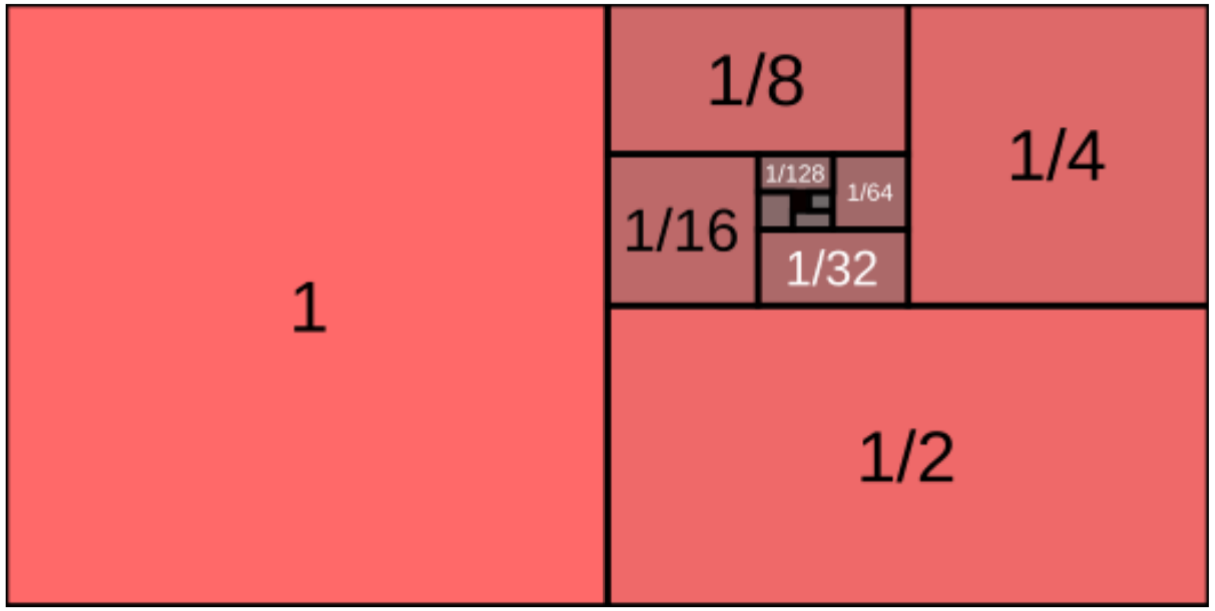


Fig. 36 Geometric progression convergence diagram.

Amortisert kjøretid (prosjekt 2)

In conclusion, while each resize might be costly, we carry out so few of them that the *total cost* of all the resizes also becomes $\mathcal{O}(n)$ and the total cost of appending n elements to an empty `DynamicArray` is

$$(\text{cost of } n \text{ appends}) = \mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n).$$

If doing n appends has a total cost of $\mathcal{O}(n)$, the average/amortized cost of a single operation must be

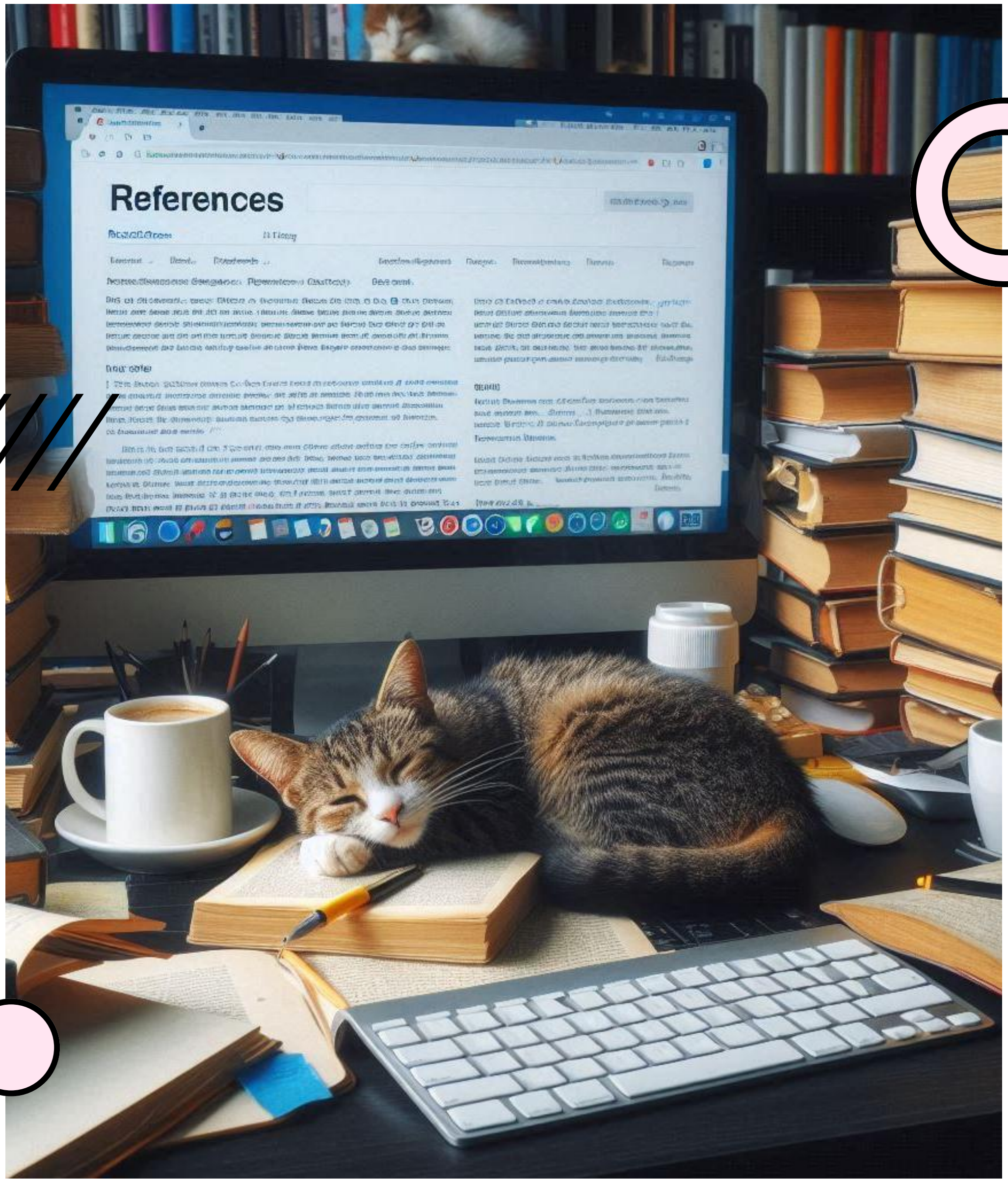
$$(\text{amortized cost of 1 append}) = \frac{1}{n} \cdot (\text{cost of } n \text{ appends}) = \frac{1}{n} \cdot \mathcal{O}(n) = \mathcal{O}(1).$$

worst case: O(n)

Hva bør brukes til prosjekt 2?

- Worst case
- Amortisert
- Eventuelt begge

LIVEKODING: REFERANSER



Hva betyr int& for noe?

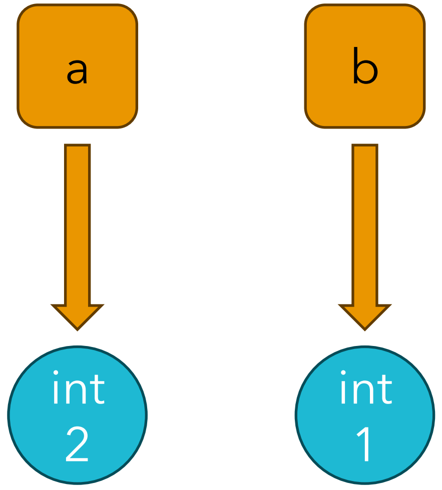
Symbol	Etter type	Før variabelnavn
(ikke noe symbol)	Variabel (heltall) **int x = …**	Verdi **… = x;**
*	Peker (minneadresse) **int* x = …**	Verdi på minneadressen **… = *x**
&	Referanse **int& x = …**	Minneadresse **… = &x**

Feilmelding hvis x ikke er en peker

- **int y = x** *kopierer* verdien til et annet sted i minnet (&y != &x)
- **int& y = x** gjør at x og y peker på samme verdi (&y == &x)

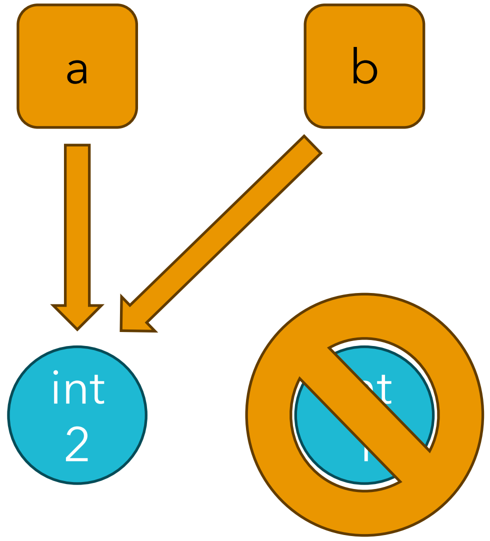
Vanlige (ikke-muterbare) variabler

C++	Python	Verdi av a	Verdi av b
int a = 1;	a = 1	1	
int b = a;	b = a	1	1
a = 2;	a = 2	2	1



C++: To måter å overskrive (tilordne) flere variabler samtidig på

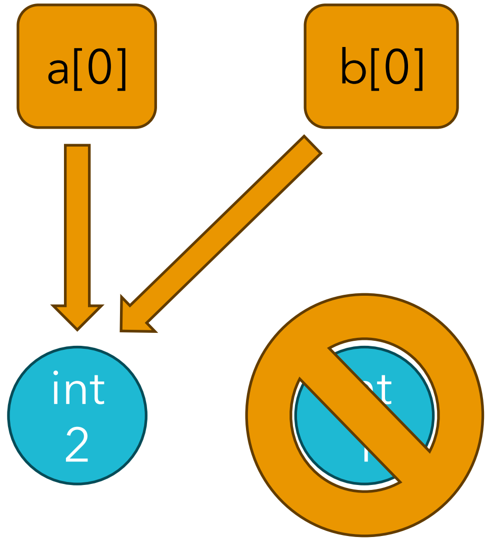
Med peker	Med referanse	Verdi av a	Verdi av *b/b
int a = 1;	int a = 1;	1	
int* b = &a;	int& b = a;	1	1
a = 2;	a = 2;	2	2



Men det gjelder *overskriving* av hele objektet, ikke *endring* av (muterbare) objekter

Må bruke peker for at dette skal funke i C++

C++	Python	Verdi av a[0]	Verdi av b[0]
int a[] = {1};	a = [1]	1	
int b* = a;	b = a	1	1
a[0] = 2;	a[0] = 2;	2	2

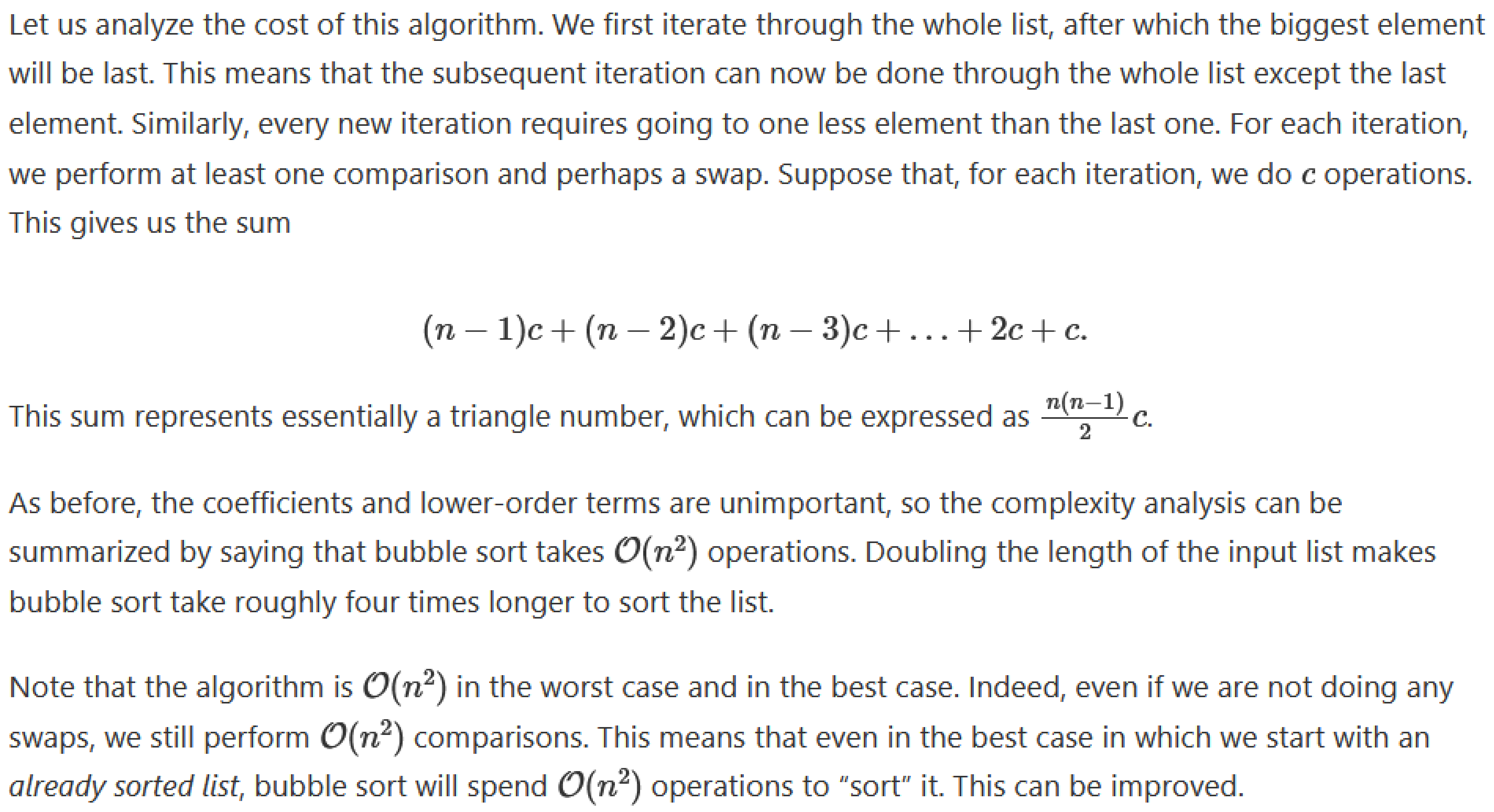


Boblesortering (bubble sort)

6 5 3 1 8 7 2 4

LIVEKODING: BUBBLE SORT

Analyse av bubble sort



Let us analyze the cost of this algorithm. We first iterate through the whole list, after which the biggest element will be last. This means that the subsequent iteration can now be done through the whole list except the last element. Similarly, every new iteration requires going to one less element than the last one. For each iteration, we perform at least one comparison and perhaps a swap. Suppose that, for each iteration, we do c operations. This gives us the sum

$$(n-1)c + (n-2)c + (n-3)c + \ldots + 2c + c.$$

This sum represents essentially a triangle number, which can be expressed as $\frac{n(n-1)}{2}c$.

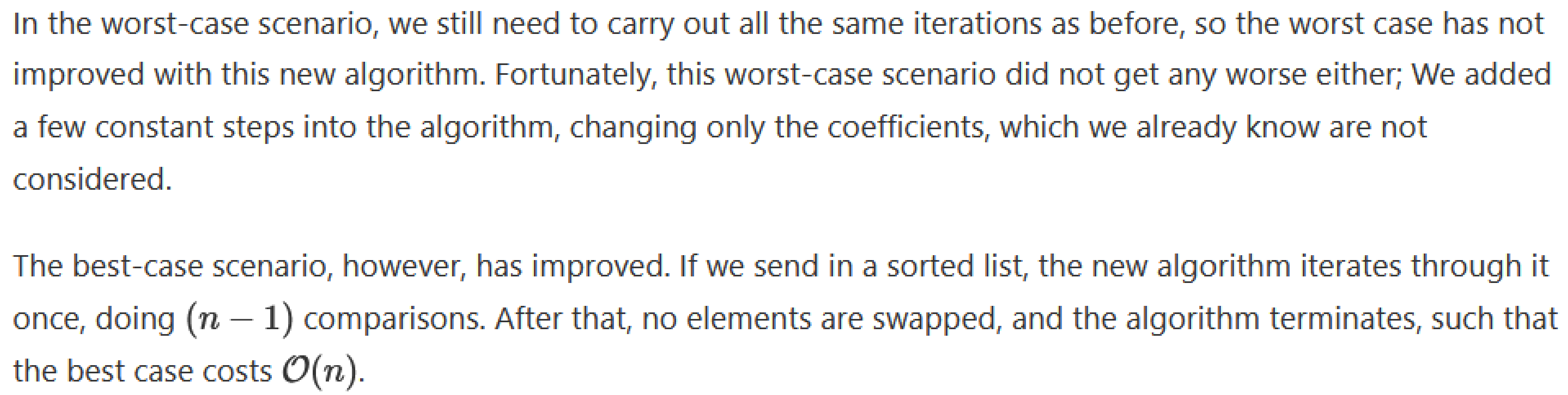
As before, the coefficients and lower-order terms are unimportant, so the complexity analysis can be summarized by saying that bubble sort takes $\mathcal{O}(n^2)$ operations. Doubling the length of the input list makes bubble sort take roughly four times longer to sort the list.

Note that the algorithm is $\mathcal{O}(n^2)$ in the worst case and in the best case. Indeed, even if we are not doing any swaps, we still perform $\mathcal{O}(n^2)$ comparisons. This means that even in the best case in which we start with an *already sorted list*, bubble sort will spend $\mathcal{O}(n^2)$ operations to "sort" it. This can be improved.

LIVEKODING: BUBBLE SORT

(DEL 2)

Analyse av forbedret bubble sort



In the worst-case scenario, we still need to carry out all the same iterations as before, so the worst case has not improved with this new algorithm. Fortunately, this worst-case scenario did not get any worse either; We added a few constant steps into the algorithm, changing only the coefficients, which we already know are not considered.

The best-case scenario, however, has improved. If we send in a sorted list, the new algorithm iterates through it once, doing $(n-1)$ comparisons. After that, no elements are swapped, and the algorithm terminates, such that the best case costs $\mathcal{O}(n)$.

Etter forelesningen

- Vurderingsrubrikken for prosjekt 2 er nå publisert
- Sjekk også oppgaveteksten for oppdateringer