# Exercises 1 datatypes and algoritme

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, hvid

Indhold genereret af kunstig intelligens kan være forkert.

Et billede, der indeholder tekst, skærmbillede, nummer/tal, linje/række

Indhold genereret af kunstig intelligens kan være forkert.

* We want to find the biggest problem size(n) each algorithm can solve in a given time.
* Micro seconds is one millionth of a second
* Et billede, der indeholder tekst, skærmbillede, software, Font/skrifttype

  Indhold genereret af kunstig intelligens kan være forkert.
  + To find the value for T = lg(n) (
    - if *x* = *by*, then *y* is the logarithm of *x* to base *b*, written *.* 
      * So in our example :
    - When T= 1sec = 1.000.000 micro seconds :
    - When T = 1 minut
    - **And simply continue with putting the value for a given time into the formula:**
    - T = 1 sec
    - T = 1 min
    - **And simply continue with putting the value for a given time into the formula:**
* For , we simply input the values for the time in micro seconds
* n lg n There is not a simply way to isolate n in , so we use trail and error:
  + Since logarithmic functions grow slowly, we can approximate x by trying different values until we get close to TTT.  
    Let’s say T=106, so we need to solve:
    - * Too small!
    - So, **x ≈ 6 × 104 when T = 106**
  + **And simply continue with putting the value for a given time into the formula**
* For n2 and n3 you simply take and to find the biggest value.
* For 2n = T you take = n
* For n!= T, we again need to use trail and error, since there is no straightforward way to isolate n

To do the same with nano seconds as processors use, we use the computation 1 sec = 1\*109 nano seconds

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, design

Indhold genereret af kunstig intelligens kan være forkert.Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, candelabrum

Indhold genereret af kunstig intelligens kan være forkert.

## Exercise session 2

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, algebra

Indhold genereret af kunstig intelligens kan være forkert.

So again we want to find the biggest problem size that can be calculated in the given time, 1 second or 1.000.000.000 nano seconds.   
To do this we need to isolate T in the formulas, and insert 1.000.000.000, in some cases we might brute force, and make qualified guesses, and insert values to test it.

* **n** is simply n = T, and T = 1,000,000,000, so **n = 1,000,000,000**
* **lg n** is the same as T = log2(n), so we isolate n here, by using the knowledge of logarithms:

We insert our value for T = 1,000,000,000

* **n log n** is the same as T = n\*log2(n). Here we can’t on a simple way isolate n, so we try with different values, that gives us a result close to 1,000,000,000.
  + By trying different values, we see that n = 39,000,000, gives the result:  
    39000000 \* log2(39000000) = **983,461,860.739**
    - This result Is close enough for us without spending to much time trying to estimate it.
* **n2** we simply isolate T in T = n2, and inserter out value for T:
* **n3** just like before:
* **2n**, isolate T again, by using the knowledge of logarithm:

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Indhold genereret af kunstig intelligens kan være forkert.

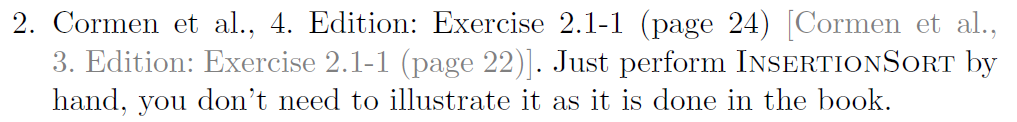
Et billede, der indeholder tekst, skærmbillede, display/skærm/fremvisning, Font/skrifttype

Indhold genereret af kunstig intelligens kan være forkert.



See the code “permutation\_list\_count\_circles.py” for more information

## Sheet 2



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| index | 1 | 2 | 3 | 4 | 5 | 6 |
| Elements | 5 | 8 | 4 | 2 | 3 | 9 |

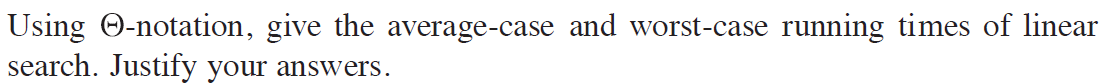
Given an array/list of elements, insertions sort takes the second value and compare it to the one before. If it is less than, the less element is taken out of the array and put in before the bigger one. This continues.

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Indhold genereret af kunstig intelligens kan være forkert.

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, hvid

Indhold genereret af kunstig intelligens kan være forkert.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

1 in programming n

0 in computerscience n-1

O-notation counts the amount of comparison (iterations the algorithm does)

* Best case: at the end of the list
  + O(1)
* worst case: at the end of the list
  + O(n) (could be just the constant )
* Average case:
  + **Average-case time complexity:** O(n)
* Runtime:
  + This is what the run time will be if we are looking for the n’th element
  + 1/n+2/n+3/n… n/n
    - Arithmetic summation

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede

Indhold genereret af kunstig intelligens kan være forkert.

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype

Indhold genereret af kunstig intelligens kan være forkert.

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Indhold genereret af kunstig intelligens kan være forkert.

1. (2,1),(3,1),(8,1),(6,1)
2. Inverted sorted list. Fx [5,4,3,2,1]  
   It has this amount of inversions:  
   n-1 + n-2 +n-3 … +1 =

Et billede, der indeholder tekst, Font/skrifttype, linje/række, hvid

Indhold genereret af kunstig intelligens kan være forkert.

* Et billede, der indeholder tekst, Font/skrifttype, håndskrift, hvid

  Indhold genereret af kunstig intelligens kan være forkert.
  + The run time for Insertions sort, which equals n2  
    Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, nummer/tal

    Indhold genereret af kunstig intelligens kan være forkert.

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, hvid

Indhold genereret af kunstig intelligens kan være forkert.

* Same principle as this one:

Et billede, der indeholder diagram, tegning, design, origami

Indhold genereret af kunstig intelligens kan være forkert.

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, hvid

Indhold genereret af kunstig intelligens kan være forkert. Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Indhold genereret af kunstig intelligens kan være forkert.

* On the first graph we see the average run time, for list’s with 400 – 900 elements, in the bestcase- acsending order, and worst case – descending order.
  + For best case we know time /input ratio should be constant no matter the input size

    - O(n) = 1
  + For worst case we also know time /input ratio should be constant no matter the input
    - size
    - O(n2 ) = 1

Et billede, der indeholder tekst, linje/række, Kurve, diagram

Indhold genereret af kunstig intelligens kan være forkert.

Figure 1 The graph shows the functions are constants, by only varience a bit

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede

Indhold genereret af kunstig intelligens kan være forkert.

Et billede, der indeholder tekst, skærmbillede, diagram, linje/række

Indhold genereret af kunstig intelligens kan være forkert.

Figure 2 We see that the random case falls between the best case and worse case, closes to the worst case

Et billede, der indeholder tekst, Font/skrifttype, linje/række, skærmbillede

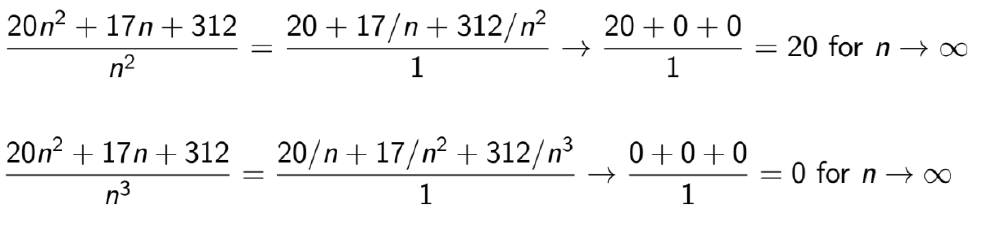
Indhold genereret af kunstig intelligens kan være forkert.

* For in the worst case, this rules out bruteforce, cheking each, since it would take . when it is something we lg n, then we know we need to split the set into halfs.
* First sort the set using merge-sort, that has a runtime of
  + Where you divide and conque. So splitting it in halfs untill one pieces. Merge two subarrays together by putting the smalles value first. and merge back together by putting the smallest value first
  + Then we check the element starting from the lowest, and see what values it needs to add up to x: x - S[i] = partner.
  + Than we use binary search to see if we can find the partner. Binary search split a sorted array in half and check if the value we search for is less then the “middelvalue” (can be found by rounding down or up), if so we repeat In the lower half of the array otherwise we repeat in the right half of the array.
  + The total run time is:
    - For merge sort, the divding in half part takes O(lg n), and at each level we seach n elements, so the final runtime for merge sort is O (n lg n)
    - Binary search is O(lg n), because we just divide and check only the value for one element.
    - IN your algorithm the run time is O(n lg n), because we need to check the sorted list for the first element (O(n)), and then find it’s partner. So we need to ad n to O(lg n), ending up with:
      * Merge sort O(n lg n) + binary search and linear search O(n lg n)
        + This would just be 2O(n lg n), and since we don’t care about the constant in big O notation we just drop the 2
        + O(n lg n).

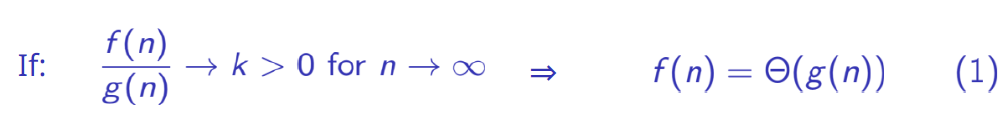
Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, linje/række

Indhold genereret af kunstig intelligens kan være forkert.

We use this example, as inspiration :



The first in the example is (big thetha), and the second is little o.

Proposition 1 says: 

**With values**

So we start by dividing the two functions:

What happens as n grows big?

Because 5/n gets smaller and smaller, and 25/n2 gets smaller even faster.

Our proposition says if we divide the two functions and get a constant, k bigger then 0, when n goes to inifinity, then . This means that and n2 grows at the same rate as n gets large.

**Proof**

We know that if and only if and , basically the function need to be less then or equal to O(g(n)) and more then or equal to .

*For , we know that it is the tight upper bound, meaning f(n) = (g(n)) and they have the same growth rate.*

*We see that the dominant term is*

*Proof for (big thetha)*

We know that if and only if and .  
We start by showing Big O:

*Then for all n holds that:*

*If we set c= 30, n0= 30.1  
n0 = 5, c = 6.1*

We see that the constant is the bigger then n, no matter what we plot in into n0.

Now we want to prove the second term of the definition

Proof for :

The definition of tells us:

We can write this as:

Because the runtime still needs to be bigger then or equal to zero.

Set c= , n0=

Then for all n holds that (turn the inequality around from before):

If we set c= 1, n0= 30.1

If we set c= 1/10, n0= 1

We just showed that there exists some positive constant c and some starting point n0 for which the inequality holds, for . And there for

Proposition 2:  
Et billede, der indeholder tekst, Font/skrifttype, håndskrift, linje/række

Indhold genereret af kunstig intelligens kan være forkert.

**With values:**

We use proposition 2:

Since we are divding by n, and n squred and cubed, when n grows the expression will be smaller, there by:

Which satisfy the condition for proposition two and tells us that f(n) = o(g(n3 )).

**Proof**

The definition of f(n) = o(g(n) says:

We insert our functions:

c = 1, n0= 30.1

c = 10, n0= 0.085

*Set c = 10*

*Then for all c > 0 and n holds that:*

*Unfinished*

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, linje/række

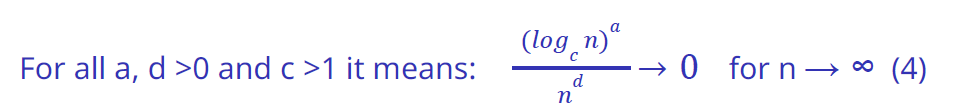
Indhold genereret af kunstig intelligens kan være forkert.

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, linje/række

Indhold genereret af kunstig intelligens kan være forkert.

Et billede, der indeholder tekst, Font/skrifttype, håndskrift, linje/række

Indhold genereret af kunstig intelligens kan være forkert.



We want to show f(n), g(n) => f(n) = o(g(n)) (stric upper bound). And we do this by using rule 2, and if the result is equal to 0, then it holds that g(n) is the upper bound, and the order holds.

We fist use rule number 4(not shown here), to determine if we can use rule number 2. If we get 0 at rule number 4, then we can preceed with rule number 2

* f(n) = 1, g(n) = log n
  + use rule 2
* f(n) = log n, g(n) =
  + use rule 2
* f(n) = , g(n) =
  + use rule 2
* Etc.

Still missing some solutions to exercise sheet 2

## Sheet 3

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, algebra

Indhold genereret af kunstig intelligens kan være forkert.

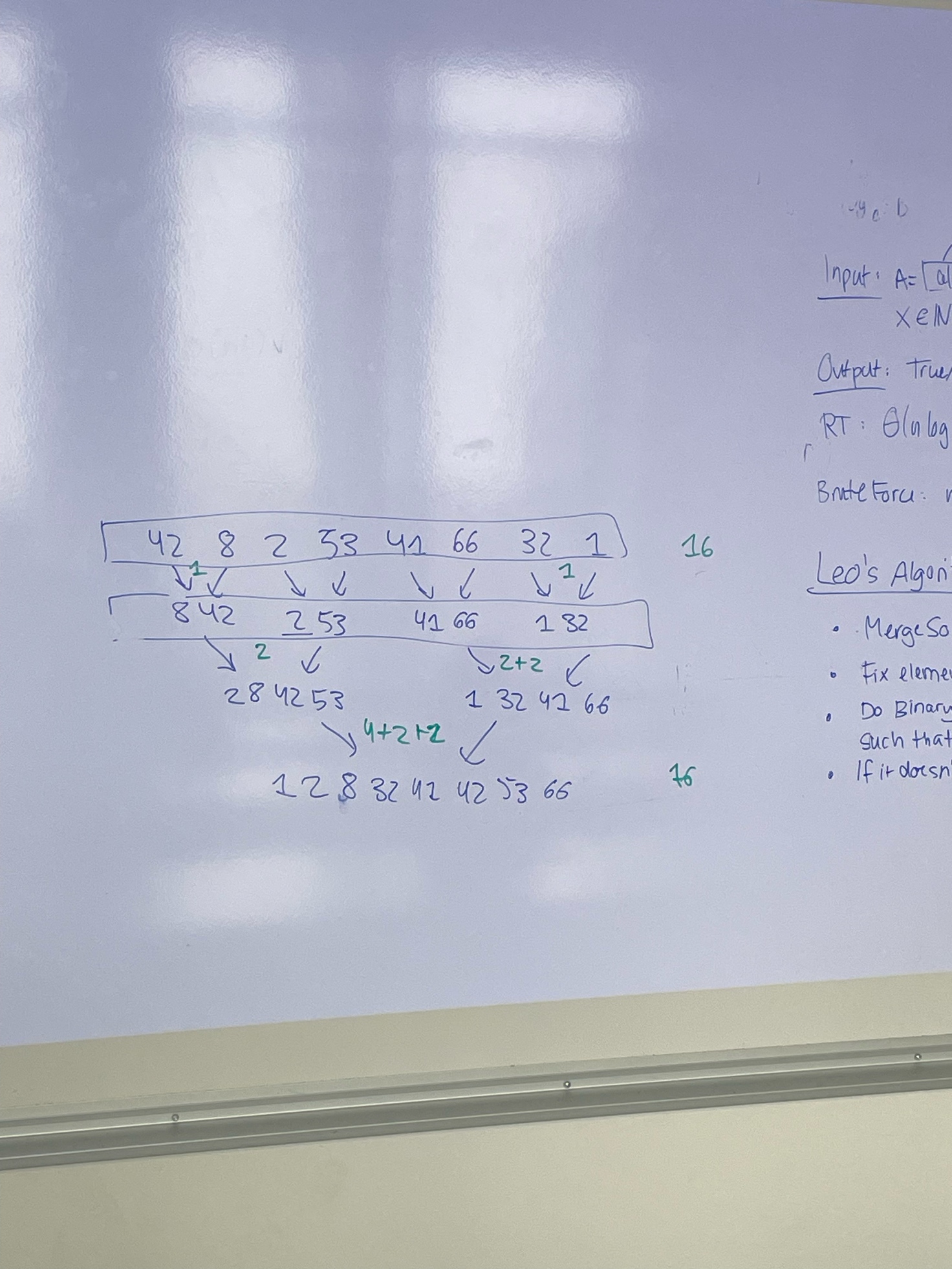
We use propertie two, that say if we divide f(n) by g(n), and it goes to zero, g(n) is indeed small o (strict upper bound) to f(n).  
Et billede, der indeholder tekst, Font/skrifttype, håndskrift, linje/række

Indhold genereret af kunstig intelligens kan være forkert.

* + We need to change the base, with the following formula, we can change the base to whatever we want, here we get a common, by having it on ln: ,

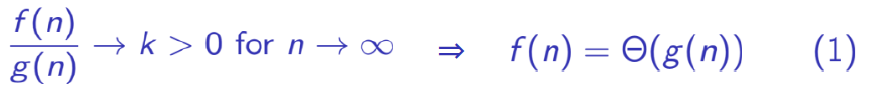
The right side is just a constant:

The higher value we plot into n, the bigger ln(n) becomes, and the smaller the fraction becomes. That means it holds, f(n) = o(g(n))

Inversion algorithm using merge count, and counting each change of place. Runtime n log n

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype

Indhold genereret af kunstig intelligens kan være forkert.

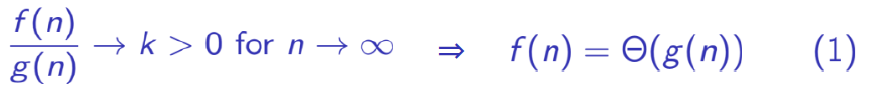
1. We use therom 1: 

What ever n is, 5/n will be a number that is non negative (since the input n can’t be negative), and there for 2 + 5/n will always be bigger then 0.   
There for meaning it grows at the same rate.   
  
Also When considering asymptotic behavior, constant factors and lower-order terms are ignored. In (2n+5), the dominant term is (2n), which is linearly proportional to (n). Therefore, (2n+5) is indeed Theta of (n)(**true)**.

1. By looking at it we know that n^2 grows faster than, but we show it with formula 2:  
   Et billede, der indeholder Font/skrifttype, håndskrift, tekst, linje/række

   Indhold genereret af kunstig intelligens kan være forkert.

And indeed when n grows 1/n, becomes smaller, and thereby going to 0. So n^2 is small o of n, **it is true**.

1. By looking at it we know n, would never grow faster then n2, there for n = O(n2). We can calculate it by using the property o(g(n)) => O(g(n)), therefor if it is small o, it is also big O. We just showed last that n = o(n2), so we know it is also big O, **and it is true**.
2. Just by looking at it, it seems like n would not grow at the same rate as n2. But we use formula 1, to solve this:  
   

We see that this goes to zero, and not a constant bigger then zero. Therefor **it does not hold.**

1. We know that n = O(n2) ⬄ n2 = . Which is truly not the same as n = . There by it **is False.**
2. We know that ⬄, and to solve this we can use the second formula:  
     
   Et billede, der indeholder Font/skrifttype, håndskrift, tekst, linje/række

   Indhold genereret af kunstig intelligens kan være forkert.

When n grows, n of cause grows aswell. Meaning it does not go to zero. There for does not hold, and **does not hold** either.

1. , again we use the knowledge that if it is small o, then it is also big O, and we use formula 2:  
   Et billede, der indeholder Font/skrifttype, håndskrift, tekst, linje/række

   Indhold genereret af kunstig intelligens kan være forkert.

Since log n, grows as n goes to infinity, it is not small o. However the relationship between small o, and big O is a implication, therefor it could still be big O, without being small o.  
  
We try to change the nominator and denominator:

And we see that when n grows, it goes to zero. Hence, . can not be big O of , when is little o of . That would be a contradiction. So it is **False**

1. , we saw before that , and we know that . Hence it is **true**
2. , again we see if n is little o of , cause if it is it implies that it is also big O:

We see that n grows faster then , similar, n grows faster then log n. Meaning that goes to zero, as n goes to infinity. Which is exactly what we need for n to be little o, and thereby also be Big o. **Hence it holds**

1. . We do just as before:  
   By plotting numbers in we see that:

We can see that when n grows, the whole expression gets smaller and smaller. And that is excalty what is need for f(n) = o(g(n), which => f(n)= O(g(n)). There for **it holds** .

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, algebra

Indhold genereret af kunstig intelligens kan være forkert.

1. We know that, when a function is big O, it means:  
   so we can rewrite:

And Since the LHS is less then or equal, their sum must also be less than or equal to the RHS:

We choose the biggest constant(let’s say c1), since we know this will respect the less than or equal:

We can sum both sides up, to get one new function, since the sum of two functions is just a new function:

Hence it holds. We can also look at it logically, if we add two smaller things, and add two bigger things, the relationship still holds.

1. We can rewrite this as : , then we add them, and the inequality still holds:

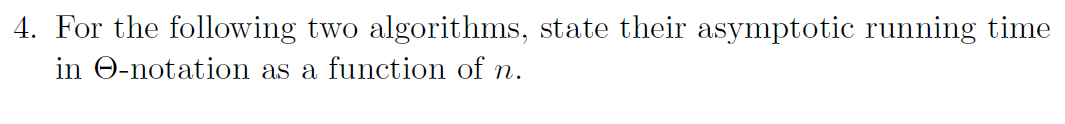
Selecting the smallest constant to ensure the less than or qual holds (min(c1,c2):

Again we just have the sum of two functions, with is just a third function, so we can write:

Hence it holds. We can also look at it logically, if we add two smaller things, and add two bigger things, the relationship still holds.

1. We can see that, now it is the relationship between two function that determines if it is big O, or not. Logically this does not hold for all, because you could have two functions that are both big O, but grows at very different rates.  
   let:   
     
   this gives us:

We see that when n grows, the demoninator on the RHS, grows exponentially, times a constant. This means that the whole fraction is going to zero. So the RHS shrinks as the LHS grows. No matter what the two constans are. Hence it is a counter example.



Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, hvid

Indhold genereret af kunstig intelligens kan være forkert.

# Exercise 5

Et billede, der indeholder tekst, skærmbillede, software, Multimediesoftware

Indhold genereret af kunstig intelligens kan være forkert.

Et billede, der indeholder tekst, skærmbillede, Multimediesoftware, diagram

Indhold genereret af kunstig intelligens kan være forkert.