Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Johannes Lengler, David Steurer
Lucas Slot, Manuel Wiedmer, Hongjie Chen, Ding Jingqiu

2 October 2023

Algorithms & Data Structures — Exercise sheet 2 — HS 23

The solutions for this sheet are submitted at the beginning of the exercise class on 9 October 2023.

Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

You can use results from previous parts without solving those parts.

Exercise 2.1 *Induction.*

(a) Prove via mathematical induction that for all integers $n \geq 5$,

$$2^n > n^2 \,.$$

(b) Let x be a real number. Prove via mathematical induction that for every positive integer n, we have

$$(1+x)^n = \sum_{i=0}^{n} \binom{n}{i} x^i \,,$$

where

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \,.$$

We use a standard convention $0! = 1$, so $\binom{n}{0} = \binom{n}{n} = 1$ for every positive integer n.

Hint: *You* can use the following fact without justification: for every $1 \leq i \leq n$,

$$\binom{n}{i} + \binom{n}{i-1} = \binom{n+1}{i} \,.$$

Asymptotic Notation

When we estimate the number of elementary operations executed by algorithms, it is often useful to ignore constant factors and instead use the following kind of asymptotic notation, also called O-Notation. We denote by $\mathbb{R}^+$ the set of all (strictly) positive real numbers and by $\mathbb{N}$ the set of all (strictly) positive integers. Let N be a set of possible inputs.

Definition 1 (O-Notation)**.** For $f : N \to \mathbb{R}^+$,

$$O(f) \coloneqq \{g : N \to \mathbb{R}^+ \mid \exists C > 0 \, \forall n \in N \; g(n) \leq C \cdot f(n)\}.$$

We write $f \leq O(g)$ to denote $f \in O(g)$. Some textbooks use here the notation $f = O(g)$. We believe the notation $f \leq O(g)$ helps to avoid some common pitfalls in the context of asymptotic notation.

Instead of working with this definition directly, it is often easier to use limits in the way provided by the following theorem.

Theorem 1 (Theorem 1.1 from the script)**.** *Let N be an infinite subset of $\mathbb{N}$ and $f : N \to \mathbb{R}^+$ and $g : N \to \mathbb{R}^+$.*

- *If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f \leq O(g)$ and $g \not\leq O(f)$.*
- *If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = C \in \mathbb{R}^+$, then $f \leq O(g)$ and $g \leq O(f)$.*
- *If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, then $f \not\leq O(g)$ and $g \leq O(f)$.*

The following theorem can also be helpful when working with O-notation.

Theorem 2. *Let $f, g, h : N \to \mathbb{R}^+$. If $f \leq O(h)$ and $g \leq O(h)$, then*

1. *For every constant $c > 0$, $c \cdot f \leq O(h)$.*
2. *$f + g \leq O(h)$.*

Notice that for all real numbers $a, b > 1$, $\log_a n = \log_a b \cdot \log_b n$ (where $\log_a b$ is a positive constant). Hence $\log_a n \leq O(\log_b n)$. So you don't have to write bases of logarithms in asymptotic notation, that is, you can just write $O(\log n)$.

Exercise 2.2 *O-notation quiz.*

(a) For all the following functions the variable n ranges over $\mathbb{N}$. Prove or disprove the following statements. Justify your answer.

(1) $2n^5 + 10n^2 \leq O(\frac{1}{100}n^6)$

(2) $n^{10} + 2n^2 + 7 \leq O(100n^9)$

(3) $e^{1.2n} \leq O(e^n)$

(4)* $n^{\frac{2n+3}{n+1}} \leq O(n^2)$

(b) Find f and g as in Theorem 1 such that $f \leq O(g)$, but the limit $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ does not exist. This proves that the first point of Theorem 1 provides a sufficient, but not a necessary condition for $f \leq O(g)$.

Exercise 2.3 *Asymptotic growth of $\sum_{i=1}^{n} \frac{1}{i}$* **(1 point)**.

The goal of this exercise is to show that the sum $\sum_{i=1}^{n} \frac{1}{i}$ behaves, up to constant factors, as $\log(n)$ when n is large. Formally, we will show $\sum_{i=1}^{n} \frac{1}{i} \leq O(\log n)$ and $\log n \leq O(\sum_{i=1}^{n} \frac{1}{i})$ as functions from $\mathbb{N}_{\geq 2}$ to $\mathbb{R}^+$.
For parts (a) to (c) we assume that $n = 2^k$ is a power of 2. We will generalise the result to arbitrary n in part (d). For $j \in \mathbb{N}$, define

$$S_j = \sum_{i=2^{j-1}+1}^{2^j} \frac{1}{i}.$$

(a) For any $j \in \mathbb{N}$, prove that $S_j \leq 1$.

Hint: *Find a common upper bound for all terms in the sum and count the number of terms.*

(b) For any $j \in \mathbb{N}$, prove that $S_j \geq \frac{1}{2}$.

(c) For any $k \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$, prove the following two inequalities

$$\sum_{i=1}^{2^k} \frac{1}{i} \leq k + 1$$

and

$$\sum_{i=1}^{2^k} \frac{1}{i} \geq \frac{k+1}{2}.$$

Hint: *You can use that* $\sum_{i=1}^{2^k} \frac{1}{i} = 1 + \sum_{j=1}^{k} S_j$. *Use this, together with parts (a) and (b), to prove the required inequalities.*

(d)* For arbitrary $n \in \mathbb{N}$, prove that

$$\sum_{i=1}^{n} \frac{1}{i} \leq \log_2(n) + 2$$

and

$$\sum_{i=1}^{n} \frac{1}{i} \geq \frac{\log_2 n}{2}.$$

Hint: *Use the result from part (c) for* $k_1 = \lceil \log_2 n \rceil$ *and* $k_2 = \lfloor \log_2 n \rfloor$. *Here, for any* $x \in \mathbb{R}$, $\lceil x \rceil$ *is the smallest integer that is at least* x *and* $\lfloor x \rfloor$ *is the largest integer that is at most* x. *For example,* $\lceil 1.5 \rceil = 2$, $\lfloor 1.5 \rfloor = 1$ *and* $\lceil 3 \rceil = \lfloor 3 \rfloor = 3$. *In particular, for any* $x \in \mathbb{R}$, $x \leq \lceil x \rceil < x + 1$ *and* $x \geq \lfloor x \rfloor > x - 1$.

Exercise 2.4 *Asymptotic growth of* $\ln(n!)$.

Recall that the factorial of a positive integer n is defined as $n! = 1 \times 2 \times \cdots \times (n-1) \times n$. For the following functions n ranges over $\mathbb{N}_{\geq 2}$.

(a) Show that $\ln(n!) \leq O(n \ln n)$.

Hint: *You can use the fact that* $n! \leq n^n$ *for* $n \geq 1$ *without proof.*

(b) Show that $n \ln n \leq O(\ln(n!))$.

Hint: *You can use the fact that* $\left(\frac{n}{2}\right)^{\frac{n}{2}} \leq n!$ *for* $n \geq 1$ *without proof.*

Exercise 2.5 *Testing equations* **(2 points)**.

Your friend sends you a piece of code that computes his favorite function $f : \mathbb{N} \to \mathbb{N}$. For $n \in \mathbb{N}$, we want to test if the equation $f(a) + f(b) + f(c) = f(d)$ can be satisfied using positive integers $1 \leq a, b, c, d \leq n$. Your friend completed Algorithms and Data Structures last year, and so you may assume that his code computes $f(k)$ in $O(1)$ for any $k \in \mathbb{N}$. You may also assume simple arithmetic operations on integers can be performed in $O(1)$. Finally, you may initialize an array of size k in time $O(k)$.

(a) Design a simple $O(n^4)$ algorithm that outputs "YES" if there exist integers $1 \leq a, b, c, d \leq n$ such that $f(a) + f(b) + f(c) = f(d)$ and "NO" otherwise.

(b) Assume that $f(k) \leq k^3$ for all $k \in \mathbb{N}$. Modify your previous algorithm so that it works in time $O(n^3)$ under this assumption. Motivate briefly why it still works.

Hint: *You could use a helper array of size n^3 to get rid of one of the loops in your previous algorithm. The helper array could save which values the function f can take.*

(c)* Assume that $f(k) \leq k^2$ for all $k \in \mathbb{N}$. Modify your previous algorithm so that it works in time $O(n^2)$ under this assumption. Motivate briefly why it still works.

Hint: *You could use a helper array again. Note that $f(a) + f(b) + f(c) = f(d)$ implies that $f(a) + f(b) = f(d) - f(c)$.*

Exercise 2.3 *Asymptotic growth of* $\sum_{i=1}^{n} \frac{1}{i}$ **(1 point).**

The goal of this exercise is to show that the sum $\sum_{i=1}^{n} \frac{1}{i}$ behaves, up to constant factors, as $\log(n)$ when n is large. Formally, we will show $\sum_{i=1}^{n} \frac{1}{i} \leq O(\log n)$ and $\log n \leq O(\sum_{i=1}^{n} \frac{1}{i})$ as functions from $\mathbb{N}_{\geq 2}$ to $\mathbb{R}^+$.

For parts (a) to (c) we assume that $n = 2^k$ is a power of 2. We will generalise the result to arbitrary n in part (d). For $j \in \mathbb{N}$, define

$$S_j = \sum_{i=2^{j-1}+1}^{2^j} \frac{1}{i}.$$

(a) For any $j \in \mathbb{N}$, prove that $S_j \leq 1$.

Hint: *Find a common upper bound for all terms in the sum and count the number of terms.*

Let $j \in \mathbb{N}$. To prove: $S_j = \sum_{i=2^{j-1}+1}^{2^j} \frac{1}{i} \leq 1$

Similar to last week's exercise! The hint tells you that as well. Now i gets bigger and bigger, thus $\frac{1}{i}$ gets smaller. Thus, the biggest $\frac{1}{i}$ will be is right at the start. $(i = 2^{j-1}+1)$ Remember, we look for a big term here, because we are interested in an upper bound.

$$S_j = \sum_{i=2^{j-1}+1}^{2^j} \frac{1}{i} \leq \sum_{i=2^{j-1}+1}^{2^j} \frac{1}{2^{j-1}+1}$$

How many terms are in the sum?

Since $2^j - (2^{j-1}+1) + 1 = 2^j - 2^{j-1} = 2^{j-1}(2-1) = 2^{j-1}$,

we have 2^{j-1} terms in the sum, thus

$$\sum_{i=2^{j-1}+1}^{2^j} \frac{1}{2^{j-1}+1} = 2^{j-1} \frac{1}{2^{j-1}+1}$$

Now notice that $\frac{1}{2^{j-1}+1} \leq \frac{1}{2^{j-1}}$, thus

$$2^{j-1} \frac{1}{2^{j-1}+1} \leq 2^{j-1} \cdot \frac{1}{2^{j-1}} = 1$$

Hence for arbitrary j we have

$$S_j \leq 1.$$

(b) For any $j \in \mathbb{N}$, prove that $S_j \geq \frac{1}{2}$.

Same idea as before. Now we are interested in a small term, since we want a lower bound.

$$S_j = \sum_{i=2^{j-1}+1}^{2^j} \frac{1}{i} \geq 2^{j-1} \cdot \frac{1}{2^j} = \frac{1}{2}$$

(c) For any $k \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$, prove the following two inequalities

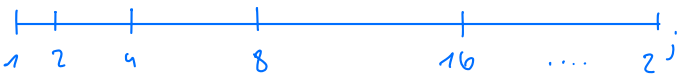
$$\sum_{i=1}^{2^k} \frac{1}{i} \leq k+1$$

and

$$\sum_{i=1}^{2^k} \frac{1}{i} \geq \frac{k+1}{2}.$$

Hint: *You can use that* $\sum_{i=1}^{2^k} \frac{1}{i} = 1 + \sum_{j=1}^{k} S_j$. *Use this, together with parts (a) and (b), to prove the required inequalities.*

Can you see why?



$S_1 = \sum_{i=2}^{2} \frac{1}{i} = \frac{1}{2}$, $S_2 = \sum_{i=3}^{4} \frac{1}{i} = \frac{1}{3} + \frac{1}{4}$, $S_3 = \sum_{i=5}^{8} \frac{1}{i} = \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8}$.

S_1 ends at $i=2$, S_2 starts at $i=3$ and ends at $i=4$

S_3 starts at $i=5$ and ends at $i=8$ ...

Thus $\sum_{i=1}^{2^k} \frac{1}{i} = \sum_{j=1}^{k} S_j + 1$

since S_1 starts at $i=2$ we are missing $i=1 \Rightarrow \frac{1}{i} = \frac{1}{1} = 1$.

We use the hint, (a) and (b):

(1) $\sum_{i=1}^{2^k} \frac{1}{i} = 1 + \sum_{j=1}^{k} S_j \overset{(a)}{\leq} 1 + k \cdot 1 = 1 + k$.

(2) $\sum_{i=1}^{2^k} \frac{1}{i} = 1 + \sum_{j=1}^{k} S_j \geq 1 + k \cdot \frac{1}{2} \geq \frac{1}{2} + \frac{k}{2} = \frac{k+1}{2}$.

(d)* For arbitrary $n \in \mathbb{N}$, prove that

$$\sum_{i=1}^{n} \frac{1}{i} \leq \log_2(n) + 2$$

and

$$\sum_{i=1}^{n} \frac{1}{i} \geq \frac{\log_2 n}{2}.$$

Hint: *Use the result from part (c) for* $k_1 = \lceil \log_2 n \rceil$ *and* $k_2 = \lfloor \log_2 n \rfloor$. *Here, for any* $x \in \mathbb{R}$, $\lceil x \rceil$ *is the smallest integer that is at least* x *and* $\lfloor x \rfloor$ *is the largest integer that is at most* x. *For example,* $\lceil 1.5 \rceil = 2$, $\lfloor 1.5 \rfloor = 1$ *and* $\lceil 3 \rceil = \lfloor 3 \rfloor = 3$. *In particular, for any* $x \in \mathbb{R}$, $x \leq \lceil x \rceil < x + 1$ *and* $x \geq \lfloor x \rfloor > x - 1$.

Now that n doesn't have to be a power of 2, i.e. 2^k for some $k \in \mathbb{N}$, anymore we need to be careful. Notice that for arbitrary n it holds, that $\log_2 n \leq \lceil \log_2 n \rceil$ thus $n = 2^{\log_2 n} \leq 2^{\lceil \log_2 n \rceil}$. An analogous argument can be made for $\lfloor \log_2 n \rfloor$ by changing the $\leq$ sign to a $\geq$ sign.

Since $n \leq 2^{k_1}$ for $k_1 = \lceil \log_2 n \rceil$ we get

$$\sum_{i=1}^{n} \frac{1}{i} \leq \sum_{i=1}^{2^{k_1}} \frac{1}{i} \overset{(c)}{\leq} k_1 + 1 = \lceil \log_2 n \rceil + 1 \leq \log_2 n + 2.$$

Similarly, $n \geq 2^{k_2}$ for $k_2 = \lfloor \log_2 n \rfloor$ we get

$$\sum_{i=1}^{n} \frac{1}{i} \geq \sum_{i=1}^{2^{k_2}} \frac{1}{i} \overset{(c)}{\geq} \frac{k_2 + 1}{2} = \frac{\lfloor \log_2 n \rfloor + 1}{2} \geq \frac{\log_2 n}{2}$$

Exercise 2.5 *Testing equations* **(2 points).**

Your friend sends you a piece of code that computes his favorite function $f : \mathbb{N} \to \mathbb{N}$. For $n \in \mathbb{N}$, we want to test if the equation $f(a) + f(b) + f(c) = f(d)$ can be satisfied using positive integers $1 \leq a, b, c, d \leq n$. Your friend completed Algorithms and Data Structures last year, and so you may assume that his code computes $f(k)$ in $O(1)$ for any $k \in \mathbb{N}$. You may also assume simple arithmetic operations on integers can be performed in $O(1)$. Finally, you may initialize an array of size k in time $O(k)$.

also for code experts.

Tip #1: Sometimes descriptions are very wordy (i.e. a lot of words / beating around the bush). It's a great skill to have to boil it down to the most important bits of the text. Here

- we have access to a function f
- $1 \leq a, b, c, d \leq n$
- calculating $f(k)$ is in $O(1)$ for some $k \in \mathbb{N}$
- elementary operations like addition in $O(1)$
- we can initialize an array of size k in $O(k)$.

Generally, in "design an algorithm" tasks we are interested in the following:

- Algorithm itself (as Pseudo code or description)
- Proof of correctness (i.e. showing why it is correct)
- Runtime analysis (asymptotic complexity of the algorithm using O-Notation)

(a) Design a simple $O(n^4)$ algorithm that outputs "YES" if there exist integers $1 \leq a, b, c, d \leq n$ such that $f(a) + f(b) + f(c) = f(d)$ and "NO" otherwise.

Runtimes often give a lot of information about an algorithm.

Notice that $f(a) + f(b) + f(c) = f(d)$

(a, b, c, d) how many possible "tuples"?

$\Rightarrow$ $n \cdot n \cdot n \cdot n = n^4$ tuples.

```
for a = 1 ... n
    for b = 1 ... n
        for c = 1 ... n
            for d = 1 ... n
                ...
```

} testing all possibilities

Using four nested for loops we can check all n^4 tuples (a, b, c, d) by iterating over all integers in $[1, n]$ in each for loop.

For each iteration, we check if

$$f(a) + f(b) + f(c) = f(d)$$

in $O(1)$. If yes, return "Yes".

If after all iterations, there was no success, return "No".

Correctness follows trivially from the fact that we iterate over all possibilities.

Runtime: All n^4 iterations use 2 add operations, 1 comparison iteration and 4 times evaluating $f(k)$ for some $k \in \mathbb{N}$. All of these 7 operations are in $O(1)$, hence the algorithm finishes in $O(n^4)$.

Worst case? Best case? Quicksort example

(b) Assume that $f(k) \leq k^3$ for all $k \in \mathbb{N}$. Modify your previous algorithm so that it works in time $O(n^3)$ under this assumption. Motivate briefly why it still works.

Hint: *You could use a helper array of size n^3 to get rid of one of the loops in your previous algorithm. The helper array could save which values the function f can take.*

Now we have been given additional information of f, namely $f(k) \leq k^3$ for all $k \in \mathbb{N}$.

How does that help us?

Think of how we use f in our algorithm:

$$f(a) + f(b) + f(c) = f(d)$$
$$\leq a^3 \quad \leq b^3 \quad \leq c^3 \quad \leq d^3$$

what values can a, b, c, d take on?

remember: $1 \leq a, b, c, d \leq n$.

We see the largest value f can be in our algorithm is n^3 and since $f: \mathbb{N} \to \mathbb{N}$, the smallest possible value is 1.

What have we learned from this?

Since we test if $f(a) + f(b) + f(c) = f(d)$ and all values $f(x) \in [1, \dots, n^3]$ for $x \in \{a, b, c, d\}$ this test can only be true if

$$f(a) + f(b) + f(c) \in [1, \dots, n^3]$$

This is where the hint comes into play:

we initialise a helper array of size n^3

$$H[1, \dots, n^3] = [0, \dots, 0]$$

Now, using a for loop from $i = 1$ to n we calculate $f(i)$ and set $H[f(i)] = 1$.

Using three nested for loops, we check all tuples (a, b, c). Let $\alpha = f(a) + f(b) + f(c)$.

If $H[\alpha] = 1$, it means there exists a $d \in [1 \dots n]$, such that

$$f(a) + f(b) + f(c) = f(d).$$

solutions !

Thus, if $\alpha \leq n^3$ and $H[\alpha] = 1$ for any tuple (a, b, c) return "Yes", otherwise, if all tuples fail, return "No".

Correctness: Since any sum $f(a) + f(b) + f(c)$, in the case of success, must be equal to $f(d)$ for some $d \in [1 \dots n]$ and we test all tuples (a, b, c) as

well as determine all possible values $f(d)$, the algorithm must find (a, b, c, d) that fulfil the desired equation, if present.

Since $1 \leq f(d) \leq n^3$ for $d \in [1 \dots n^3]$ and we check if $a = f(a) + f(b) + f(c) \leq n^3$, out of bands errors are impossible.

For runtime:

- Array initialization: $O(n^3)$
- iterating over n^3 tuples: $O(n^3)$

$\Longrightarrow O(n^3)$.

(c)* Assume that $f(k) \leq k^2$ for all $k \in \mathbb{N}$. Modify your previous algorithm so that it works in time $O(n^2)$ under this assumption. Motivate briefly why it still works.

Hint: *You could use a helper array again. Note that* $f(a) + f(b) + f(c) = f(d)$ *implies that* $f(a) + f(b) = f(d) - f(c)$.

When looking at the hint, we see it's similar to (b).

The idea is now, to rule out more possible tuples, so our algorithm gets even faster.

How can we do that?

The hint suggest $f(a) + f(b) = f(d) - f(c)$.

When looking at the modified equation, one notices the following:

Calculating $f(a) + f(b)$ and $f(d) - f(c)$, each in $O(n^2)$, can be done in $O(n^2)$ total. Now, remembering how we used the helper array in (b), we derive the following solution:

$= [0 \dots 0]$

Let $H = [1 \dots 2n^2]$ be an array of size $2n^2$. Using two nested for loops, we can calculate the $f(a) + f(b)$ in $O(n^2)$, for all (a, b), where $1 \le a, b \le n$ and set $H[f(a) + f(b)] = 1$.

for $a = 1 \dots n$
  for $b = 1 \dots n$
    $H[f(a) + f(b)] = 1$.

Using another two nested for loops, we calculate $f(d) - f(c)$ for all (d, c), where $1 \le d, c \le n$, and if $H[f(d) - f(c)]$ is equal to 1, we return "Yes". If these two loops end and we haven't returned, return "No".

Correctness: Since $f(a) + f(b) + f(c) = f(d)$ implies that $f(a) + f(b) = f(d) - f(c)$ and we go over all possible

tuples for both sides of the equation. If (a, b, c, d) exist that fulfil $f(a) + f(b) + f(c) = f(d)$ the algorithm correctly returns "Yes" otherwise, correctly returns "No".

Since $f(d) - f(c) \leq 2n^2$ and

$f(a) + f(b) \leq 2n^2$

out of bounds errors can't occur.

Runtime:

- array initialization: $O(n^2)$
- first, second double loop: $O(n^2)$
  because elementary operations such as addition or subtraction as well as $f(k)$ for some $k \in \mathbb{N}$ is in $O(1)$.

Thus the total runtime is $O(n^2)$.

Exercise 2.1 *Induction.*

(a) Prove via mathematical induction that for all integers $n \geq 5$,

$$2^n > n^2 .$$

- **Base Case.**
  Let $n = 5$. Then:
  $$2^5 = 32 > 25 = 5^2 .$$

- **Induction Hypothesis.**
  Assume that the property holds for some positive integer $k \geq 5$, that is,
  $$2^k > k^2 .$$

$$2^{k+1} > 2\,2^k \overset{\text{I.H.}}{>} 2 \cdot k^2 = k^2 + k^2$$

Now, since $k \geq 5 \Rightarrow k^2 \geq 5 \cdot k$

$$\geq k^2 + 5k$$

$$= k^2 + 2k + 3k$$

Now, since $k \geq 5 \Rightarrow 3k \geq 15$

$$\geq k^2 + 2k + 15$$

$$> k^2 + 2k + 1$$

$$= (k+1)^2$$

This might seem tricky, but normally, you try out a lot of things before writing sum a clean solution.

(b) Let x be a real number. Prove via mathematical induction that for every positive integer n, we have

$$(1+x)^n = \sum_{i=0}^{n} \binom{n}{i} x^i ,$$

where

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} .$$

this is a variant of the binomial theorem.

We use a standard convention $0! = 1$, so $\binom{n}{0} = \binom{n}{n} = 1$ for every positive integer n.

Hint: You can use the following fact without justification: for every $1 \le i \le n$,

$$\binom{n}{i} + \binom{n}{i-1} = \binom{n+1}{i} .$$

IA: $n = 1$

$$(1+x)^1 = \binom{1}{0} x^0 + \binom{1}{1} x^1 = \sum_{i=0}^{1} \binom{1}{i} x^i \quad \checkmark$$

IH: Assume this property holds for some positive integer k, that is

$$(1+x)^k = \sum_{i=0}^{k} \binom{k}{i} x^i$$

IS: $k \rightsquigarrow k+1$

$$(1+x)^{k+1} = (1+x)(1+x)^k$$

$$\overset{\text{I.H.}}{=} (1+x) \sum_{i=0}^{k} \binom{k}{i} x^i$$

So far so good. Now what?

Let's manipulate the expression further:

$$= \sum_{i=0}^{k} \binom{k}{i} x^i + \sum_{i=0}^{k} \binom{k}{i} x^{i+1}$$

My guess is, that some students might have difficulties from here on out.

First, observe that $\sum_{i=0}^{k} \binom{k}{i} x^{i+1} = \sum_{i=1}^{k+1} \binom{k}{i-1} x^i$.

How? Notice that i goes from 0 to k in the left sum, meaning we go from $\binom{k}{0} x^0$ up to $\binom{k}{k} x^k$. We have to preserve this, otherwise the sum won't stay the same. Now in the right sum, i goes from 1 to $k+1$, therefore, we need to subtract 1 from all occurences of i.

$$= \sum_{i=0}^{k} \binom{k}{i} x^i + \sum_{i=0}^{k} \binom{k}{i} x^{i+1}$$

$$= \sum_{i=0}^{k} \binom{k}{i} x^i + \sum_{i=1}^{k+1} \binom{k}{i-1} x^i$$

How am I supposed to notice this?

Well, ideally we want to use the hint:

$$\binom{n}{i} + \binom{n}{i-1} = \binom{n+1}{i}$$

So playing around with the sum and looking for patterns is the way to go here.

Now it would be nice to combine the sums, but the indices don't match i.e. one starts at 0 and goes to k the other from 1 to $k+1$.

$$= \sum_{i=0}^{k} \binom{k}{i} x^i + \sum_{i=1}^{k+1} \binom{k}{i-1} x^i$$

$$= \binom{k}{0} x^0 + \sum_{i=1}^{k} \binom{k}{i} x^i + \sum_{i=1}^{k} \binom{k}{i-1} x^i + \binom{k}{k} x^{k+1}$$

$$= \binom{k}{0} x^0 + \sum_{i=1}^{k} \left(\binom{k}{i} + \binom{k}{i-1} \right) x^i + \binom{k}{k} x^{k+1}$$

Hint $$= \binom{k}{0} x^0 + \sum_{i=1}^{k} \binom{k+1}{i} + \binom{k}{k} x^{k+1}$$

Now we use $\binom{n}{0} = \binom{n}{n} = 1$, thus

$$\binom{k}{0} = \binom{k+1}{0} = \binom{k}{k} = \binom{k+1}{k+1} = 1$$

$$= \binom{k+1}{0} x^0 + \sum_{i=1}^{k} \binom{k+1}{i} + \binom{k+1}{k+1} x^{k+1}$$

$$= \sum_{i=0}^{k+1} \binom{k+1}{i}$$

This might seem difficult, and it takes some time to get comfortable with sum notation. With practice, you will get there! :)

There exist other Σ-tricks, more in later classes.

Exercise 2.2 *O-notation quiz.*

(a) For all the following functions the variable n ranges over $\mathbb{N}$. Prove or disprove the following statements. Justify your answer.

refer to official solutions.

(b) Find f and g as in Theorem 1 such that $f \leq O(g)$, but the limit $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ does not exist. This proves that the first point of Theorem 1 provides a sufficient, but not a necessary condition for $f \leq O(g)$.

I find the description to be confusing, so here is another way to put this:

Find f and g, such that $f \leq O(g)$, but the limit $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ doesn't exist.

We define the following two functions $f, g : \mathbb{N} \to \mathbb{R}^+$. Let $f(n) = 2 + (-1)^n$ and $g(n) = 1$. We have $\frac{f(n)}{g(n)} = \frac{2+(-1)^n}{1} = 2 + (-1)^n$, which has no limit when $n \to \infty$. However, for any $n \in \mathbb{N}$, $f(n) \leq 3g(n)$ and thus $f \leq O(g)$.

If you don't know what a limit is, or what it means for a limit to not exist, refer to my guide or look online.