The purpose of this tutorial is to familiarise you with some fundamental concepts from discrete mathematics that are useful for designing and analysing data structures and algorithms. For a more complete account, including more detailed explanations and more examples, study Sections 1.2.1, 1.2.2 and 1.2.3 in the recommended reading "Algorithm Design and Applications" by Goodrich and Tamassia. The topics covered there, and which you should revise, are:

1. Summation notation Σ, and formulas for the sum of certain numeric sequences, e.g., $\Sigma_{i=1}^{n} i$ and $\Sigma_{i=1}^{n} 2^i$, etc.
2. Logarithms and exponents
3. Floor and ceiling functions
4. Justification by counterexample
5. Justification by contrapositive
6. Justification by contradiction
7. Justification by induction
8. Basic probability

In addition to that, you should be familiar with basic set notation and basic logic. A set is a group of objects represented as a unit. The objects in a set are called its members or elements. One way to describe a set is to list its elements inside braces, e.g., $S = \{7, 21, 57\}$. The symbols $\in$ and $\notin$ denote set membership and nonmembership. So $7 \in S$ but $8 \notin S$.

Induction:
Induction is a very useful proof technique for proving correctness and bound the running time of an algorithm. Most of the statements about the running times of algorithms that we'll see during this course will hold *for all* $n > 0$, where n is the size of the input. However, this is a claim about an infinite set of numbers and hence we can't prove it by proving it for each element separately.

This is where induction comes in. Induction proves the statement for a *base case*, the smallest element for which the claim must hold ($n = 1$). In some cases, it's helpful to prove an extended base case by explicitly showing that the claim holds for the first few elements, say $n = 1$, $n = 2$, and $n = 3$.

Next, we proceed to the *induction step*, which assumes that the claim holds for all values of n smaller than the current one. This assumption is called the *induction hypothesis*. Using the induction hypothesis, we then show that the claim holds for the next value of n. Typical applications of this are using the induction hypothesis for $n - 1$ to prove the claim for n or using the induction hypothesis for $n/2$ to prove the claim for n when n is even.

As a refresher, work out the problems in this tutorial sheet. If needed, refer to Section 1.2.3 in the class textbook ("Algorithm Design and Applications" by Goodrich and Tamassia) for more detail background on induction and more examples.

Warm-up

Problem 1. Examine the following formal descriptions of sets so that you understand which members they contain. Write a short informal English description of each set.

a) $\{1, 3, 5, 7, ...\}$

b) $\{..., -4, -2, 0, 2, 4, ...\}$

c) $\{n \mid n = 2m \text{ for some } m \in \mathbb{N}\}$

Solution 1. We do not include 0 in the set $\mathbb{N}$ of natural numbers.

a) The set of odd natural numbers.

b) The set of even integers.

c) The set of even natural numbers.

Problem 2. Write formal descriptions of the following sets.

a) The set containing the numbers 1, 10, and 100

b) The set containing all integers that are greater than 5

c) The set containing all natural numbers that are less than 5

d) The set containing nothing at all

Solution 2.

a) $\{1, 10, 100\}$

b) $\{n \mid n \in \mathbb{Z}, n > 5\}$

c) $\{n \mid n \in \mathbb{N}, n < 5\}$, or simply $\{0, 1, 2, 3, 4\}$.

d) $\emptyset$, or $\{\}$

Problem 3. Let A be the set $\{x, y, z\}$ and B be the set $\{x, y\}$.

a) Is A a subset of B?

b) Is B a subset of A?

c) What is $A \cup B$?

d) What is $A \cap B$?

e) What is $A \times B$?

f) What is the power set of B?

Solution 3.

a) No

b) Yes

c) $\{x, y, z\}$

d) $\{x, y\}$

e) $\{(x, x), (x, y), (y, x), (y, y), (z, x), (z, y)\}$

f) $\{\emptyset, \{x\}, \{y\}, \{x, y\}\}$

Problem solving

Problem 4. Use induction to show that $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$.

Solution 4.

Base case: $n = 0$. So we have to show that $\sum_{i=0}^{0} 2^i = 2^1 - 1$. The sum consists of a single element $2^0 = 1$, which is equal to $2^1 - 1$.
Induction step: $n \geq 1$. Assume that the claim is true for $n' < n$. Consider n. We need to show that

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1.$$

First, we split off the last term of the sum

$$\sum_{i=0}^{n} 2^i = \sum_{i=0}^{n-1} 2^i + 2^n.$$

Since $n - 1 < n$, we can apply the induction hypothesis and the right-hand side can be rewritten to

$$2^n - 1 + 2^n,$$

which is equal to

$$2^{n+1} - 1.$$

Hence, we just showed that

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1.$$

Problem 5. Recall that one way of defining the Fibonacci sequence is as follows: $F(1) = 1$, $F(2) = 2$, and $F(n) = F(n-2) + F(n-1)$ for $n > 2$.
Use induction to prove that $F(n) < 2^n$.

Solution 5.
Base case: $n = 1$ and $n = 2$. By definition, we have that $F(1) = 1 < 2 = 2^1$ and $F(2) = 2 < 4 = 2^2$.
Induction step: $n > 2$. Assume that the claim is true for $n' < n$. Consider n. We need to show that $F(n) < 2^n$.
Since $n > 2$, the definition of the Fibonacci sequence gives us that

$$F(n) = F(n-2) + F(n-1).$$

Next, we apply the induction hypothesis on $F(n-2)$ and $F(n-1)$, which gives us

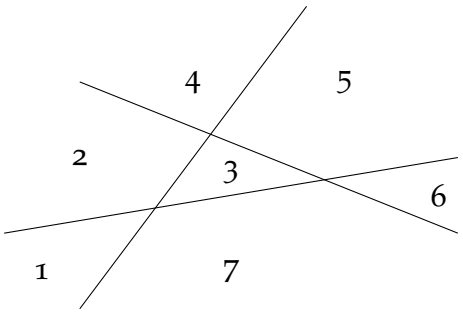
$$F(n) < 2^{n-2} + 2^{n-1}.$$

Using a bit of math it isn't hard to show that

$$F(n) < 2^{n-2} + 2^{n-1} < 2^{n-1} + 2^{n-1} = 2 \cdot 2^{n-1} = 2^n,$$

completing the proof.

Problem 6. One line divides the plane into two halves or regions, while two lines divide the plane into four regions. In this problem we study the number of regions in which n lines divide the plane. For simplicity, assume that no three lines intersect at a given point and that no two lines are parallel to one another. Here is a picture of the 7 regions defined by 3 lines.



Consider the following algorithm that builds the set of regions by starting from a single region encompassing the whole plane, and iteratively refines the set of regions by processing one line at a time.

```
1:  function BUILD_ARRANGEMENT(lines)
2:      regions ← [new region spanning whole plane]
3:      for line ∈ lines do
4:          old ← []
5:          new ← []
6:          for region ∈ regions do
7:              if line intersects region then
8:                  Append region to old
9:                  left, right ← split region through line
10:                 Append left to new
11:                 Append right to new
12:         for x ∈ old do
13:             Remove x from regions
```

```
14:         for x ∈ new do
15:             Append x to regions
16:     return regions
```

a) Prove that at the beginning of the ith iteration of the for loop in Line 3, the line cuts through exactly i regions from the previous iteration.

b) Prove that at the end of the ith iteration of the for loop in Line 3, we have $\frac{i^2+i+2}{2}$ regions.

Remember to use the fact that no three lines intersect in the same point!

Solution 6.

a) Let ℓ be the line processed in the ith iteration, and let R be a region that ℓ intersects. Let us orient ℓ so that we can talk about ℓ entering or leaving R. Notice that ℓ enters or leaves R through an edge of R (otherwise we would have three lines going through a point) and that edges of R are defined by lines processed in previous iterations.
Except for the first region that ℓ intersects, ℓ enters each region it intersects through an edge, which in turn corresponds to a line. Given that there are no parallel lines, every two lines intersect at exactly one point. Thus the total number of regions ℓ intersects must be i.

b) Let us prove this statement by induction on i:
Base case: $i = 1$. After inserting ℓ, we have 2 regions, so the statement holds.
Induction step: $i > 1$. Assume the claim holds, for $i-1$, so at the start of the ith iteration, we have $\frac{(i-1)^2+(i-1)+2}{2}$ regions. In this iteration the algorithm replaces the i regions that intersect ℓ with $2i$ regions; resulting in

$$\frac{(i-1)^2+(i-1)+2}{2} + 2i - i = \frac{(i^2-i+2)}{2} + i = \frac{i^2+i+2}{2},$$

regions, as desired.

Advanced problem solving

Problem 7. Consider the the classical algorithm of Euclid to compute the greatest common divisor of two positive integers a and b

```
1: function EUCLID(a, b)
2:     a, b ← max(a, b), min(a, b)
3:     while b ≠ 0 do
4:         a, b ← b, a mod b
5:     return a
```

a) Prove that the algorithm outputs the greatest common divisor of a and b.

b) Prove that $\max(a, b)$ must at least halve in every other iteration.

c) Use the previous fact to argue that the number of iterations is at most $2 \log \max(a, b)$.

Solution 7.

a) Let us prove the correctness by induction on the number n of iterations of the while loop.
Base case: $n = 0$, namely, we do not iterate. This happens when $b = 0$, in which case $gcd(a, b) = a$, as 0 is divisible by any number. This is precisely what the algorithm returns, so the base case holds.
Induction step: $n > 0$, namely, we iterate at least once. After one iteration we replace a and b with $a' = b$ and $b' = a \mod b$. Notice that `euclid(a', b')` returns the same as `euclid(a, b)` using one fewer iteration. Thus, by induction we can assume that the algorithm returns $gcd(a', b')$. Notice that $a = b' + ka' = a \mod b + kb$ for some $k > 0$. Thus, any number that divides a' and b' also divides a and b, and any number that divides a and b must also divide a' and b'. Therefore, $gcd(a, b) = gcd(a', b')$. This finishes up the proof of correctness.

b) Let us assume that $a > b$, otherwise switch the argument around. After one iteration we replace a and b with $a' = b$ and $b' = a \mod b$. If $b < a/2$ then $\max(a', b') = b \leq a/2$. Otherwise, if $b > a/2$ then $b' < a/2$; in which case in the next iteration we replace a' and b' with $a'' = b'$ and $b'' = a' \mod b'$, in which case $\max(a'', b'') = b' < a/2$. Either case, the statement holds.

c) Let a' and b' be the values after $2i$ iterations. We prove that $\max(a', b') \leq \max(a, b)/2^i$.
Base case: $n \leq 1$, namely, we iterate at most once. If $n = 0$ the statement is trivially true. If $n = 1$ then we iterate once so $a > 1$ or $b > 1$, which means $\log \max(a, b) \geq 1$, so again the statement is trivially true.
Induction step: $n > 1$, namely, we iterate at least twice. Let a'' and b'' be the value after $2(i - 1)$ iteration. By inductive hypothesis, $\max(a'', b'') \leq \max(a, b)/2^{i-1}$. We just showed that after two more iterations we must have $\max(a', b') \leq \max(a'', b'')/2$. Thus, $\max(a', b') \leq \max(a, b)/2^i$.
It follows that after $2 \log \max(a, b)$ the algorithm must terminate since that many iterations would imply $\max(a, b) \leq 1$, which can only happen if $a = b = 1$, which in turn would cause the while loop to terminate.