

# CS70 Course Notes

Anmol Parande

Fall 2019 - Professors Alistair Sinclair and Yun Song

**Disclaimer:** These notes reflect CS60 when I took the course (Fall 2019). They may not accurately reflect current course content, so use at your own risk. If you find any typos, errors, etc, please report them on the GitHub repository.

## Contents

<b>1</b>	<b>Logic and Proofs</b>	<b>2</b>
1.1	Propositional Logic . . . . .	2
1.2	Proofs . . . . .	3
1.2.1	Direct Proof . . . . .	3
1.2.2	Proof by Contraposition . . . . .	3
1.2.3	Proof by Contradiction . . . . .	3
1.2.4	Proof by Cases . . . . .	4
1.2.5	Proof by Induction . . . . .	4
1.2.6	Strong Induction . . . . .	4
1.3	Case Study: Stable Marriage Algorithm . . . . .	4
1.3.1	The algorithm . . . . .	4
1.3.2	Proving the Algorithm . . . . .	5
<b>2</b>	<b>Polynomials</b>	<b>6</b>
2.1	Properties of Polynomials . . . . .	7
2.2	Polynomials over Finite Fields . . . . .	7
2.2.1	Case Study: Secret Sharing . . . . .	8
2.2.2	Case Study: Error-Correcting Codes . . . . .	8
<b>3</b>	<b>Countability</b>	<b>9</b>
3.1	Sets and Functions . . . . .	9
3.2	Countable Sets . . . . .	11
3.3	Uncountable Sets . . . . .	12
<b>4</b>	<b>Counting</b>	<b>12</b>
4.1	Ordered Objects, Sampling with replacement . . . . .	13
4.2	Ordered Objects, Sampling without replacement . . . . .	13

# 1 Logic and Proofs

## 1.1 Propositional Logic

**Definition 1** A proposition  $P$  is a statement that is either true or false

Propositions can depend on one or more variables. This is denoted by  $P(x, y, \dots)$

**Definition 2** A connective is an operator which joins two or more propositions together in some way

Connectives are fully defined by a **Truth Table** which enumerates the values of the connective given all possible combination of inputs.

**Definition 3** The base connectives are those which can be combined to create any other connective.

- $\wedge$ :  $P$  AND  $Q$
- $\vee$ :  $P$  OR  $Q$
- $\neg$ : NOT  $P$

One important connective is the **implies** connective.

P	Q	$P \implies Q$
T	T	T
T	F	F
F	T	T
F	F	T

Notice that  $P \implies Q$  has the same truth table as  $\neg P \vee Q$ . This means they are equivalent.

**Definition 4** The contrapositive of  $P \implies Q$  is  $\neg Q \implies \neg P$

**Notice:** The contrapositive is logically equivalent to the original statement

**Definition 5** The converse of  $P \implies Q$  is  $Q \implies P$

**Notice:** This is not always equivalent to the original statement The **if and only if** connective  $P \iff Q$  is equivalent to  $(P \implies Q) \wedge (Q \implies P)$

P	Q	$P \iff Q$
T	T	T
T	F	F
F	T	F
F	F	T

Quantifiers help introduce variables into our propositions.

- $\forall$ : For all

- $\exists$ : There exists

**Important:** The order of quantifiers matters.

- $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
- $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
- $\neg(\forall x P(x)) \equiv \exists x(\neg P(x))$
- $\neg(\exists x P(x)) \equiv \forall x(\neg P(x))$

## 1.2 Proofs

**Definition 6** A proof is a sequence of statements, each of which follows from the preceding ones by a valid law of reasoning.

**Definition 7** An Axiom is a basic fact which can be assumed without proof

### 1.2.1 Direct Proof

**Goal:** Prove  $P \implies Q$

**Approach:**

1. Assume  $P$
2. Deduce  $Q$  through logical steps

### 1.2.2 Proof by Contraposition

Since the contrapositive of a statement is logically equivalent to the original statement, we can prove the original statement by proving the contrapositive.

**Goal:** To prove  $P \implies Q$

**Approach:**

1. Assume  $\neg Q$
2. Deduce  $\neg P$  through logical steps

**Theorem 1** The pigeonhole principle says that if  $n$  objects are placed in  $k$  boxes and  $n > k$ , then some box contains  $\geq 1$  object.

### 1.2.3 Proof by Contradiction

**Goal:** Prove  $P \implies Q$  **Approach:**

1. Assume  $\neg P$
2. Deduce  $\neg P \implies (R \wedge \neg R)$

$R$  and  $\neg R$  are two facts which can be deduced from assumption  $\neg P$

### 1.2.4 Proof by Cases

**Goal:** Prove  $P \implies Q$

**Approach:**

1. Break  $P$  into cases  $1 \dots n$
2. Prove  $P$  holds in all cases

### 1.2.5 Proof by Induction

**Goal:** Prove  $\forall n, P(n)$  **Approach:**

1. Prove a based case
2. Assume  $P(k)$  (Induction hypothesis)
3. Prove  $P(k) \implies P(k+1)$  (Inductive Step)

### 1.2.6 Strong Induction

Because induction requires that  $P(k)$  be true to prove  $P(k+1)$ , it might be easier to just assume  $P(0) \wedge P(1) \wedge \dots \wedge P(k)$  when proving  $P(k+1)$ .

## 1.3 Case Study: Stable Marriage Algorithm

### 1.3.1 The algorithm

The stable marriage algorithm uses a list of preferences to make pairings from two disjoint groups of people.

Input :

- $n$  men,  $n$  women
- For each man, a ranked list of the women
- For each woman, a ranked list of the men

Goal: Pair men and women up in a way such that two people from different pairings would be happier together if they switched partners

Algorithm :

Each Day:

1. Each man proposes to the first woman on his list who has not yet rejected him
2. Each woman says "maybe" to the man she likes best among her proposals and rejects the others
3. Each man crosses off his list the woman who just rejected him

Repeat until there are no rejections

### 1.3.2 Proving the Algorithm

There are four things we must prove in order to make sure that the stable marriage algorithm works.

1. SMA terminates
2. SMA must output a pairing
3. The pairing outputted by SMA is stable

Beginning with proving the SMA terminates,

**Proof 1 (SMA Terminates)** *On each iteration, when SMA doesn't halt, some man got rejected. If a man gets rejected, he crossed a woman off his list. If every man gets rejected by every woman on his list, then there is a maximum of  $n^2$  rejections. This will take a maximum of  $n^2$  days, therefore the algorithm terminates after  $n^2$  days.*

Before proving that the SMA must output a pairing, it is first useful to prove something called the improvement lemma

**Lemma 1 (Improvement lemma)** *Suppose  $M$  proposes to  $W$  on iteration  $K$ . On every day  $J \geq K$ ,  $W$  has said "maybe" to a man she likes at least as much as  $M$*

**Proof 2 (Improvement Lemma)** *Proceed with induction on the number of days  $k$*

**Base Case:**  $J = K$

$M$  proposes to  $W$ , so the statement holds because even if  $M$  is her worst proposal, she will only say "maybe" to a better man.

**Induction Hypothesis:** *Assume the lemma is true for day  $J$*

**Inductive Step:**

*On day  $J$ ,  $W$  has said "maybe" to  $M'$  she likes at least as much as  $M$ .  $M'$  will propose again on day  $J + 1$ , so no matter who else proposes to  $W$ , she will at least have  $M'$  who is better than  $M$*

**Proof 3 (SMA outputs a pairing)** *Suppose the algorithm produces no pairing. Then there exists  $M$  who is rejected by all  $n$  women when the algorithm terminates. By the improvement lemma, all women at the end of the algorithm have a man whom they've said maybe to. So  $n$  women can only have rejected  $n - 1$  men which is a contradiction, therefore SMA must output a pairing.*

Before proving that the output of SMA is stable, we should define what stability means.

**Definition 8** *A pairing outputted by SMA is stable if there is not a man and a woman who would rather be with each other than their current partners (a rogue couple).*

**Proof 4 (SMA outputs a stable pairing)** Suppose the pairing outputted by SMA is  $\{...(M, W)...(M', W')\}$  and  $M$  likes  $W'$  more than  $W$ . Since  $M$  likes  $W'$  more than  $W$ ,  $M$  must have proposed to  $W'$ . By the improvement lemma,  $M'$  must be better than  $M$ , so  $(M, W')$  is not a rogue couple, so there are no rogue couples and the pairing is stable.

**Definition 9** For a given  $M$  the optimal  $W$  is the highest woman on his list that he is paired with in any stable pairing

**Definition 10** The male optimal pairing is the one in which every man is paired with his optimal woman.

**Proof 5 (SMA is male-optimal)** Proceed by induction on  $k$  (the number of days).  $\forall k \geq 0$ , on day  $k$ , no man gets rejected by his optimal woman.

**Base Case:**  $k = 0$

Nothing to prove, no rejections have happened

**Inductive Step:**

Assume for all  $0 \leq k \leq j$ , no man is rejected by his optimal woman. Suppose for contradiction that on day  $j + 1$   $M$  gets rejected by his optimal woman  $W^*$  meaning some other man was accepted  $M^*$ . Since  $W^*$  is  $M$ 's optimal woman, there exists a stable pairing  $\{...(M, W^*)...(M^*, W')\}$  since  $W^*$  prefers  $M^*$  to  $M$  because she rejected  $M$ . By the inductive hypothesis,  $M^*$  has not yet been rejected by his optimal woman, so he likes  $W^*$  at least as much as  $W'$  because there is a stable pairing with  $W'$ . Hence  $(M^*, W^*)$  is rogue because they like each other better than their partners.

**Theorem 2** The male-optimal pairing is the female-pessimal pairing.

## 2 Polynomials

**Definition 11** A polynomial of a single variable  $x$  with degree  $d$  is a function of the form

$$P(x) = \sum_{i=0}^d a_i x^i$$

In simplest terms, a polynomial is simply a function. The degree of the polynomial is its highest exponent. The  $a_i$  are called the coefficients of the polynomial.

**Definition 12** A value  $a \in \mathbb{R}$  is a root of the polynomial  $P(x)$  if  $P(a) = 0$

We say that two polynomials are equal when they are equal at all values  $x$  ( $P(x) = Q(x) \forall x$ )

## 2.1 Properties of Polynomials

1. A nonzero polynomial of degree  $d$  has at most  $d$  roots
2. Given  $d + 1$  pairs  $(x_1, y_1) \dots (x_{d+1}, y_{d+1})$  with the  $x_i$  being distinct, there exists a unique polynomial of degree at most  $d$  such that  $P(x_i) = y_i, \forall i \in [1, d + 1]$

While seemingly complex, the second property tells us something very simple about polynomials: each polynomial of degree  $d$  is uniquely defined by  $d + 1$  points. Thus given a set of points, we can find the polynomial which yielded those points. This can be done by regression or solving a system of linear equations. A "faster" technique, however, is Lagrange interpolation.

Given points  $(x_1, y_1), \dots, (x_{d+1}, y_{d+1})$ ,

$$P(x) = \sum_{i=1}^{d+1} y_i \frac{\prod_{j \neq i} (x - x_j)}{\prod_{j \neq i} (x_i - x_j)}$$

This works because we need  $P(x_i) = y_i$ . Looking at our formula, whenever  $x = x_i$ , the fraction term becomes 1 only for the  $i$ th term and 0 otherwise (because  $x_i$  is a root for the numerator for all terms except the  $i$ th one). That forces  $P(x_i) = y_i$ . We can use Lagrange Interpolation to prove other interesting facts about polynomials. As an example,

**Lemma 2** *If  $a$  is a root of a degree of polynomial  $p$ , then we can write  $p(x) = (x - a)q(x)$  for a polynomial  $q(x)$  with degree  $d - 1$ .*

## 2.2 Polynomials over Finite Fields

Normally, we treat polynomials over the real numbers. However, we can also consider polynomials over a finite field. An particularly useful field is a Galois Field, which is the integers modulo  $p$ . This is denoted by  $GF(p)$ . This field contains only the values between 0 and  $p - 1$ .

**Definition 13** *A polynomial  $P(x)$  over a finite field  $GF(p)$  is equivalent the function  $Q(x) = P(x) \bmod p$*

Notice a couple things about polynomials about finite fields.

- There are  $p^{d+1}$  polynomials of degree  $\leq d$  over  $GF(p)$  because there are  $d + 1$  points and we have  $p$  choices for the values.
- Given  $k \leq d + 1$  points, we can find  $p^{d+1-k}$  unique polynomials which go through these points because we have  $d + 1 - k$  points for which we can choose values, and there are  $p$  values to choose

### 2.2.1 Case Study: Secret Sharing

One interesting use of polynomials over finite fields is to force people to collaborate in order to obtain a secret. Suppose that we have  $n$  generals with whom we want to share a secret encoded by an integer  $s$ . If a single general obtained the secret, they could destroy the other generals, so we want to make sure that at least  $k$  generals must work together to learn the secret (because then its mutually assured destruction). If any group of less than  $k$  generals works together, they should get no information about  $s$ .

- Choose a large prime number  $p$ :  $p > s$ ,  $p > n$
- Construct a "random" polynomial  $q(x)$  w/  
degree  $(k-1) \bmod p$  such that  $q(0) = s$
- Give the  $i$ th general the value  $q(i) \bmod p$

With this scheme, if  $k$  generals get together, then they can use lagrange interpolation to recover  $q(x)$  and find  $s$  by computing  $q(0)$ . However if  $d < k$  polynomials get together, there are going to be  $p^{k-d}$  possible polynomials, so any secret value will be possible.

### 2.2.2 Case Study: Error-Correcting Codes

Another use for polynomials over finite fields is in creating error-correcting codes. Suppose we have a message which is a sequence of integers  $m_1, m_2, \dots, m_n$ . We are going to send this message over an unreliable channel, so we need to transmit some redundant information. Lets send code words  $c_1, c_2, \dots, c_m$  where  $m > n$  to create the redundancy we need. Some of these packets may be dropped or modified, so we will receive a message  $r_1, r_2, \dots, r_p$ .

Suppose we know the following about the channel:

- Up to  $k$  of the  $c_i$  will get erased, and we will know which ones do
- Up to  $k$  of the  $c_i$  will be corrupted and we don't know which ones.
- The maximum number of errors is  $k$

Since we are tolerating  $k$  errors, lets send  $n + 2k$  codewords over the channel. That way we are guaranteed  $n + k$  of the  $r_i$  are equal to the corresponding  $c_i$ .

1. Define  $q > n + 2k$  to be a large prime so we can encode our message as integers mod  $q$
2. Let  $P(x)$  be a unique degree  $n - 1$  polynomial over  $GF(q)$  such that  $P(i) = m_i$ ,  $i \in [1, n]$
3. Compute  $2k$  extra values  $p(n + 1) \dots p(n + 2k)$
4. Send all points  $c_i = p(i)$ ,  $i \in [1, n + 2k]$



We know we will receive at least  $n + k$  of our packets, but some of them may be modified. There must a degree  $n - 1$  polynomial  $p$  which goes through at least  $n + k$  of these points  $(i, r_i)$ . Suppose the location of the errors are  $e_1, e_2, \dots, e_k$ . We can define the error locator polynomial

$$E(x) = (x - e_1)(x - e_2) \dots (x - e_k) = x^k + \sum_{i=1}^{k-1} b_i x^i$$

Now notice that  $P(i)E(i) = r_i E(i)$  since  $P(i) = r_i$  for the non-error points, and  $E(i) = 0$  at the error points. Now define

$$Q(x) = P(x)E(x) = \sum_{i=0}^{n+k-1} a_i x^i$$

Since  $Q(i) = r_i E(i)$ , this gives us  $n + 2k$  equations which we can use to find the coefficients of  $Q(i)$  and  $E(i)$ . We can now simply find  $P(x) = \frac{Q(x)}{E(x)}$ .

## 3 Countability

### 3.1 Sets and Functions

A set is a collection of distinct objects. We say a set is finite if there are a non-infinite number of elements in the set.

**Definition 14** *The cardinality of a set  $X$ , denoted  $|X|$  is the number of elements in  $X$ .*

When it comes to counting, the cardinality of a set is what we care about most.

**Definition 15** *The power set of a set  $A$ , denoted by  $\mathcal{P}(A)$  is the set of all subsets of  $A$*

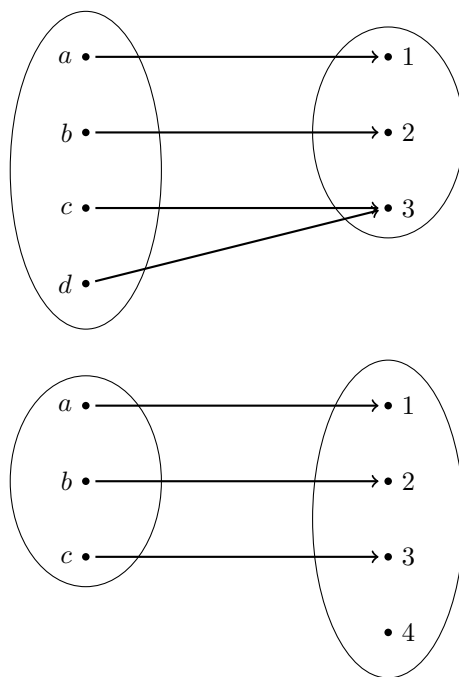
Notice the cardinality of  $\mathcal{P}(A)$  is  $2^{|A|}$  if  $A$  is finite. One thing we can do with sets is set up functions which map between the elements of two sets.

**Definition 16** *A function  $f : A \rightarrow B$  is a mapping from elements in the domain  $A$  to elements in the codomain  $B$  where each element in  $A$  is mapped to a single element in  $B$*

Functions come in a variety of shapes and forms. One of these forms is what is known as a surjection.

**Definition 17** *A surjection  $f : A \rightarrow B$  is function where  $\forall y \in B, \exists x \in A$  such that  $f(x) = y$ .*

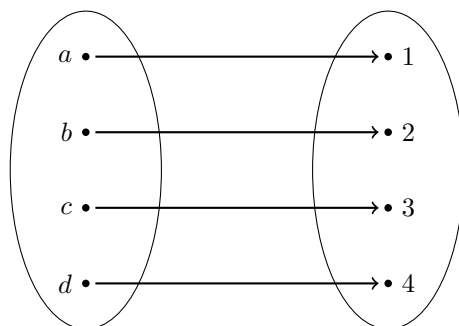
Another name for a surjection is an "onto" function because the we are mapping  $A$  onto the entirety of  $B$ .



**Definition 18** An injection  $f : A \rightarrow B$  is a function where  $\forall (x_1, x_2) \in A, ((x_1 \neq x_2) \implies f(x_1) \neq f(x_2))$

In English, this means that every element in the codomain corresponds to at most 1 element in the domain. This is why injections are also called "one-one" functions. Notice that not all elements in  $B$  have to have a corresponding element in  $A$ .

**Definition 19** A function  $f : A \rightarrow B$  is a bijection if  $f$  is both an injection and a surjection.



Bijections are nice because every element in the codomain corresponds to a single element in the domain (one-one) and every element in the codomain is

covered by an element in the domain (onto). This gives us an interesting way to check if two sets are of equal size. Let's say from sets  $X$  and  $Y$ , we pair up  $(x_i, y_i)$ . If you can be done no elements left over, then the two sets are the same size. If you have elements from either set which are left over after pairing the elements up, the sets must be different sizes. Notice that this is the same thing as forming a bijection between the two sets. This leads us to an important theorem.

**Theorem 3** *Two sets have the same cardinality if and only if there exists a bijection between them*

It turns out that sometimes we don't even have to find the actual bijection to prove that two sets have the same cardinality because of the following theorem.

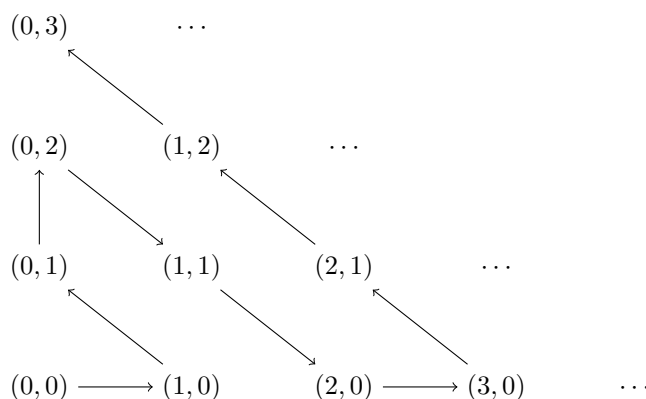
**Theorem 4** *If there exists an injection  $f : A \rightarrow B$  and an injection  $g : B \rightarrow A$ , then there exists a bijection  $h : A \rightarrow B$*

## 3.2 Countable Sets

By using the bijection method of measuring cardinalities, we can "count" sets with an infinite number of elements because all we have to do is establish a bijection between this set and another set we know is countable. What makes a set countable?

**Definition 20** *A set  $S$  is countable if there exists a bijection  $f : S \rightarrow \mathbb{N}$  or some subset of  $\mathbb{N}$*

We can think of the natural numbers, a.k.a the counting numbers, as the most fundamental countably infinite set. There are many different ways which we can construct bijections. One way is to explicitly define the mathematical relationship which forms the bijection. Another method is to do this pictorially. For example, when checking if  $\mathbb{N} \times \mathbb{N}$  is countable, we might draw the following picture: Notice that this enumerates all of the possible tuples in  $\mathbb{N} \times \mathbb{N}$ . We



can also use the theorem presented above where if we can inject  $A$  into  $B$  and  $B$  into  $A$ , then there is bijection between the two sets. This is powerful because we don't actually have to find the bijection.

### 3.3 Uncountable Sets

Just like  $\mathbb{N}$  is the "fundamental" countable infinite set,  $\mathbb{R}$  is the "fundamental" Uncountable set.

**Theorem 5**  $\mathbb{R}$  is uncountable, and every interval in  $\mathbb{R}$  is uncountable.

One way to prove a set is uncountable is to prove a bijection exists to  $\mathbb{R}$ . However, sometimes this can be quite difficult. In that case, a proof technique called Cantors Diagonalization can be useful. To see how it works, let's apply it to prove the interval  $[0, 1] \in \mathbb{R}$  is uncountable.

**Proof 6 (Cantors Diagonalization)** We can write  $r \in [0, 1]$  as an infinite decimal with no trailing 0s. Suppose for contradiction that there exists a bijection from  $\mathbb{N}$  to  $[0, 1]$ . This bijection might look as follows:

$f(0) = 0.$	3	7	2	6	5	9	...
$f(1) = 0.$	9	9	9	9	9	9	...
$f(2) = 0.$	1	0	1	0	1	0	...
							$\vdots$

We can now construct a real number  $s \in [0, 1]$  that differs from  $f(i)$  in the  $i$ th digit  $\forall i \in \mathbb{N}$  (a.k.a the diagonal elements). This number must be enumerated by  $f(n)$  for  $n \in \mathbb{N}$ . However, by construction,  $s$  differs from  $f(n)$  in the  $n$ th digit. This is a contradiction, therefore no such bijection can exist, so the interval is uncountable.

All diagonalization proofs layout in that same fashion.

## 4 Counting

A very useful skill to have is counting all of the possible ways we can do something. In general, counting problems fall in 4 broad categories.

1. Ordered objects, sampling with replacement
2. Ordered objects, sampling without replacement
3. Unordered objects, sampling with replacement
4. Unordered objects, sampling without replacement

When we say "ordered objects" that means ordering matters in the different possibilities. For example, putting  $n$  people into a line is fundamentally ordered because their position in line matters. When we say sampling with replacement, that means the same object can show up in one of our possibilities twice. Without replacement means each object in our possibility is distinct. When we consider how to count each of these scenarios, we use the first and second rules of counting.

**Theorem 6 (First Rule of Counting)** *Suppose an object is formed by a succession of  $k$  choices. The total number of distinct objects we can form is  $\prod_{i=1}^k n_k$  where  $n_k$  is the number of ways the  $i$ th choice can happen.*

**Theorem 7 (Second Rule of Counting)** *Suppose an object is formed by a succession of  $k$  choices, but the order of those choices does not matter. The number of possible objects is the number of objects where order doesn't matter divided by the number of ways we can rearrange the elements in each object.*

$$\frac{\prod_{i=1}^k n_k}{k!}$$

The first and second rules correspond to counting ordered and unordered objects respectively. Let's see how they can be applied to 4 different types of counting problems.

#### 4.1 Ordered Objects, Sampling with replacement

We can apply the first rule of counting. We are making  $k$  choices, and there are  $n$  objects to choose from for each choice we make. This means there are  $n^k$  total choices.

#### 4.2 Ordered Objects, Sampling without replacement

Once again, we can apply the first rule of counting. We are making  $k$  choices. For the first choice, there are  $n$  options. For the second choice, there are  $n - 1$  options because we can't pick the same element twice. Likewise, for the 3rd choice, we have  $n - 2$  options. This gives the product

$$n(n-1)(n-2)\dots(n-k+1) = \frac{n!}{(n-k)!}$$