

One, two, three, we're...

Counting

Basic Counting Principles

Counting problems are of the following kind:

"How many different 8-letter passwords are there?"

"How many possible ways are there to pick 11 soccer players out of a 20-player team?"

Most importantly, counting is the basis for computing **probabilities of discrete events**.

("What is the probability of winning the lottery?")

Basic Counting Principles

The sum rule:

If a task can be done in n_1 ways and a second task in n_2 ways, and if these two tasks cannot be done at the same time, then there are $n_1 + n_2$ ways to do either task.

Example:

The department will award a free computer to either a CS student or a CS professor.

How many different choices are there, if there are 530 students and 15 professors?

There are $530 + 15 = 545$ choices.

Basic Counting Principles

Generalized sum rule:

If we have tasks T_1, T_2, \dots, T_m that can be done in n_1, n_2, \dots, n_m ways, respectively, and no two of these tasks can be done at the same time, then there are $n_1 + n_2 + \dots + n_m$ ways to do one of these tasks.

Basic Counting Principles

The product rule:

Suppose that a procedure can be broken down into two successive tasks. If there are n_1 ways to do the first task and n_2 ways to do the second task after the first task has been done, then there are $n_1 n_2$ ways to do the procedure.

Basic Counting Principles

Example:

How many different license plates are there that containing exactly three English letters ?

Solution:

There are 26 possibilities to pick the first letter, then 26 possibilities for the second one, and 26 for the last one.

So there are $26 \cdot 26 \cdot 26 = 17576$ different license plates.

Basic Counting Principles

Generalized product rule:

If we have a procedure consisting of sequential tasks T_1, T_2, \dots, T_m that can be done in n_1, n_2, \dots, n_m ways, respectively, then there are $n_1 \cdot n_2 \cdot \dots \cdot n_m$ ways to carry out the procedure.

Basic Counting Principles

The sum and product rules can also be phrased in terms of **set theory**.

Sum rule: Let A_1, A_2, \dots, A_m be disjoint sets. Then the number of ways to choose any element from one of these sets is $|A_1 \cup A_2 \cup \dots \cup A_m| = |A_1| + |A_2| + \dots + |A_m|$.

Product rule: Let A_1, A_2, \dots, A_m be finite sets. Then the number of ways to choose one element from each set in the order A_1, A_2, \dots, A_m is $|A_1 \times A_2 \times \dots \times A_m| = |A_1| \cdot |A_2| \cdot \dots \cdot |A_m|$.

Inclusion-Exclusion

How many bit strings of length 8 either start with a 1 or end with 00?

Task 1: Construct a string of length 8 that starts with a 1.

There is one way to pick the first bit (1),
two ways to pick the second bit (0 or 1),
two ways to pick the third bit (0 or 1),

⋮

two ways to pick the eighth bit (0 or 1).

Product rule: Task 1 can be done in $1 \cdot 2^7 = 128$ ways.

Inclusion-Exclusion

Task 2: Construct a string of length 8 that ends with 00.

There are two ways to pick the first bit (0 or 1),
two ways to pick the second bit (0 or 1),

⋮

two ways to pick the sixth bit (0 or 1),
one way to pick the seventh bit (0), and
one way to pick the eighth bit (0).

Product rule: Task 2 can be done in $2^6 = 64$ ways.

Inclusion-Exclusion

Since there are 128 ways to do Task 1 and 64 ways to do Task 2, does this mean that there are 192 bit strings either starting with 1 or ending with 00 ?

No, because here Task 1 and Task 2 can be done **at the same time**.

When we carry out Task 1 and create strings starting with 1, some of these strings end with 00.

Therefore, we sometimes do Tasks 1 and 2 at the same time, so **the sum rule does not apply**.

Inclusion-Exclusion

If we want to use the sum rule in such a case, we have to subtract the cases when Tasks 1 and 2 are done at the same time.

How many cases are there, that is, how many strings start with 1 **and** end with 00?

1xxxxx00

There is one way to pick the first bit (1), two ways for the second, ..., sixth bit (0 or 1), one way for the seventh, eighth bit (0).

Product rule: In $2^5 = 32$ cases, Tasks 1 and 2 are carried out at the same time.

Inclusion-Exclusion

Since there are 128 ways to complete Task 1 and 64 ways to complete Task 2, and in 32 of these cases Tasks 1 and 2 are completed at the same time, there are

$128 + 64 - 32 = 160$ ways to do either task.

In set theory, this corresponds to sets A_1 and A_2 that are **not** disjoint. Then we have:

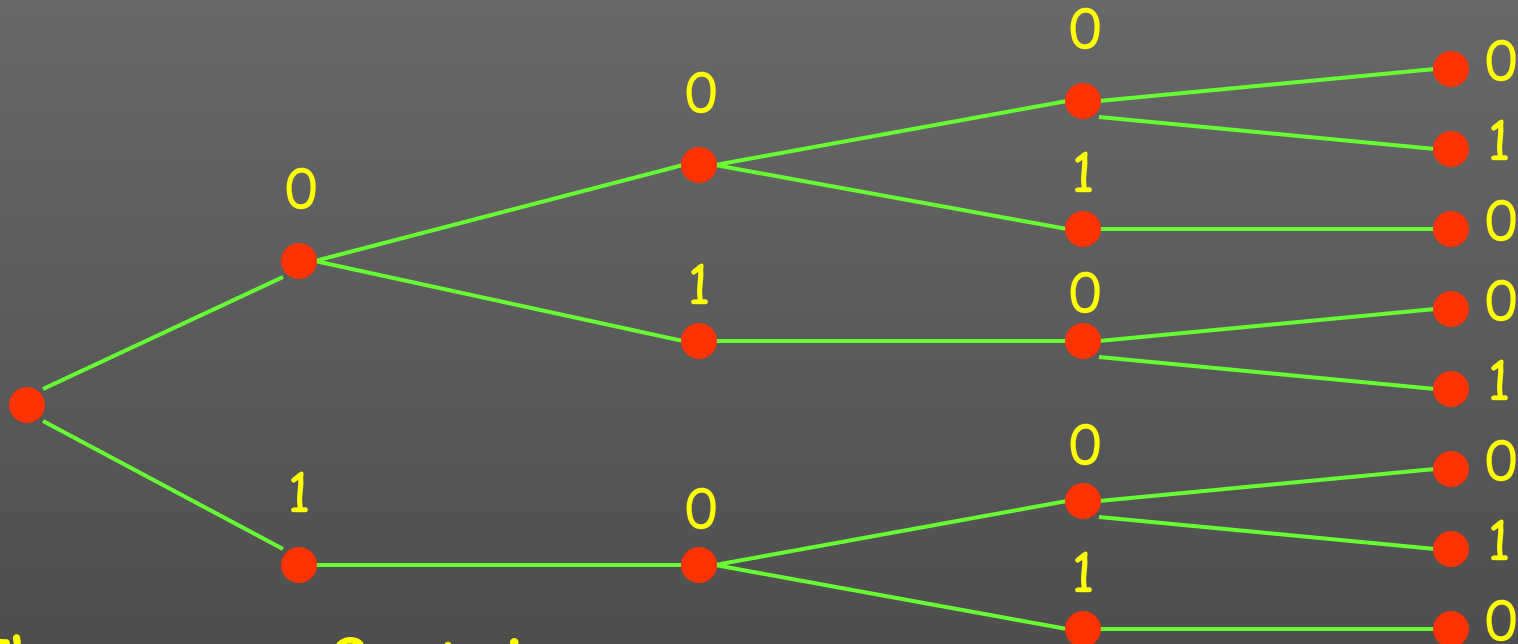
$$|A_1 \cup A_2| = |A_1| + |A_2| - |A_1 \cap A_2|$$

This is called the **principle of inclusion-exclusion**.

Tree Diagrams

How many bit strings of length four do not have two consecutive 1s?

Task 1 Task 2 Task 3 Task 4
(1st bit) (2nd bit) (3rd bit) (4th bit)



There are 8 strings.

The Pigeonhole Principle

The pigeonhole principle: If $(k + 1)$ or more objects are placed into k boxes, then there is at least one box containing **two or more** of the objects.

Example 1: If there are 11 players in a soccer team that wins 12-0, there must be at least one player in the team who scored at least twice.

Example 2: If you have 6 classes from Monday to Friday, there must be at least one day on which you have at least two classes.

The Pigeonhole Principle

The generalized pigeonhole principle: If N objects are placed into k boxes, then there is at least one box containing at least $\lceil N/k \rceil$ of the objects.

Example 1: In our 60-student class, at least 12 students will get the same letter grade (A, B, C, D, or F).

The Pigeonhole Principle

Example 2: Assume you have a drawer containing a random distribution of a dozen brown socks and a dozen black socks. It is dark, so how many socks do you have to pick to be sure that among them there is a matching pair?

There are two types of socks, so if you pick at least 3 socks, there must be either at least two brown socks or at least two black socks.

Generalized pigeonhole principle: $\lceil 3/2 \rceil = 2$.

Example

Let us have 5 points P_1, P_2, P_3, P_4, P_5 in xy plane. If each of the points has integer co-ordinates, e.g., $(-3, 2)$, prove that at least one mid-point of the pair of points will have integer co-ordinates.

Solution:

There are only four options of having the integer co-ordinates:

(odd,odd), (odd,even), (even, odd) and (even, even).

Even if the four points have 4 different options, fifth point will also have to have one of these options. That means that one of the options will be repeated. The mid-point of two points with same option will have integer co-ordinates.

Permutations and Combinations

How many ways are there to pick a set of 3 people from a group of 6?

There are 6 choices for the first person, 5 for the second one, and 4 for the third one, so there are $6 \cdot 5 \cdot 4 = 120$ ways to do this.

This is not the correct result!

For example, picking person C, then person A, and then person E leads to the **same group** as first picking E, then C, and then A.

However, these cases are counted **separately** in the above equation.

Permutations and Combinations

So how can we compute how many different subsets of people can be picked (that is, we want to disregard the order of picking) ?

To find out about this, we need to look at **permutations**.

A **permutation** of a set of distinct objects is an ordered arrangement of these objects.

An ordered arrangement of r elements of a set is called an **r -permutation**.

Permutations and Combinations

Example: Let $S = \{1, 2, 3\}$.

The arrangement 3, 1, 2 is a permutation of S .

The arrangement 3, 2 is a 2-permutation of S .

The number of r -permutations of a set with n distinct elements is denoted by $P(n, r)$.

We can calculate $P(n, r)$ with the product rule:

$$P(n, r) = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot (n - r + 1).$$

(n choices for the first element, $(n - 1)$ for the second one, $(n - 2)$ for the third one...)

Permutations and Combinations

Example:

$$\begin{aligned} P(8, 3) &= 8 \cdot 7 \cdot 6 = 336 \\ &= (8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1) / (5 \cdot 4 \cdot 3 \cdot 2 \cdot 1) \end{aligned}$$

General formula:

$$P(n, r) = n! / (n - r)!$$

Knowing this, we can return to our initial question:

How many ways are there to pick a set of 3 people from a group of 6 (disregarding the order of picking)?

Permutations and Combinations

An **r-combination** of elements of a set is an unordered selection of r elements from the set. Thus, an r -combination is simply a subset of the set with r elements.

Example: Let $S = \{1, 2, 3, 4\}$.

Then $\{1, 3, 4\}$ is a 3-combination from S .

The number of r -combinations of a set with n distinct elements is denoted by $C(n, r)$.

Example: $C(4, 2) = 6$, since, for example, the 2-combinations of a set $\{1, 2, 3, 4\}$ are $\{1, 2\}$, $\{1, 3\}$, $\{1, 4\}$, $\{2, 3\}$, $\{2, 4\}$, $\{3, 4\}$.

Permutations and Combinations

How can we calculate $C(n, r)$?

Consider that we can obtain the r -permutation of a set in the following way:

First, we form all the r -combinations of the set (there are $C(n, r)$ such r -combinations).

Then, we generate all possible orderings in each of these r -combinations (there are $P(r, r)$ such orderings in each case).

Therefore, we have:

$$P(n, r) = C(n, r) \cdot P(r, r)$$

Permutations and Combinations

$$\begin{aligned}C(n, r) &= P(n, r)/P(r, r) \\&= n!/(n-r)!/(r!/(r-r)!) \\&= n!/(r!(n-r)!)\end{aligned}$$

Now we can answer our initial question:

How many ways are there to pick a set of 3 people from a group of 6 (disregarding the order of picking)?

$$C(6, 3) = 6!/(3! \cdot 3!) = 720/(6 \cdot 6) = 720/36 = 20$$

There are 20 different ways, that is, 20 different groups to be picked.

Permutations and Combinations

Corollary:

Let n and r be nonnegative integers with $r \leq n$.

Then $C(n, r) = C(n, n - r)$.

Note that “picking a group of r people from a group of n people” is the same as “splitting a group of n people into a group of r people and another group of $(n - r)$ people”.

Permutations and Combinations

Example:

A soccer club has 8 female and 7 male members. For today's match, the coach wants to have 6 female and 5 male players on the grass. How many possible configurations are there?

$$\begin{aligned}C(8, 6) \cdot C(7, 5) &= 8!/(6! \cdot 2!) \cdot 7!/(5! \cdot 2!) \\&= 28 \cdot 21 \\&= 588\end{aligned}$$

Combinations

We also saw the following:

$$C(n, n-r) = \frac{n!}{(n-r)![n-(n-r)]!} = \frac{n!}{(n-r)!r!} = C(n, r)$$

This symmetry is intuitively plausible. For example, let us consider a set containing six elements ($n = 6$).

Picking two elements and leaving four is essentially the same as picking four elements and leaving two.

In either case, our number of choices is the number of possibilities to divide the set into one set containing two elements and another set containing four elements.

Combinations

Pascal's Identity:

Let n and k be positive integers with $n \geq k$.
Then $C(n + 1, k) = C(n, k - 1) + C(n, k)$.

How can this be explained?

What is it good for?

Combinations

Imagine a set S containing n elements and a set T containing $(n + 1)$ elements, namely all elements in S plus a new element a .

Calculating $C(n + 1, k)$ is equivalent to answering the question: How many subsets of T containing k items are there?

Case I: The subset contains $(k - 1)$ elements of S plus the element a : $C(n, k - 1)$ choices.

Case II: The subset contains k elements of S and does not contain a : $C(n, k)$ choices.

Sum Rule: $C(n + 1, k) = C(n, k - 1) + C(n, k)$.

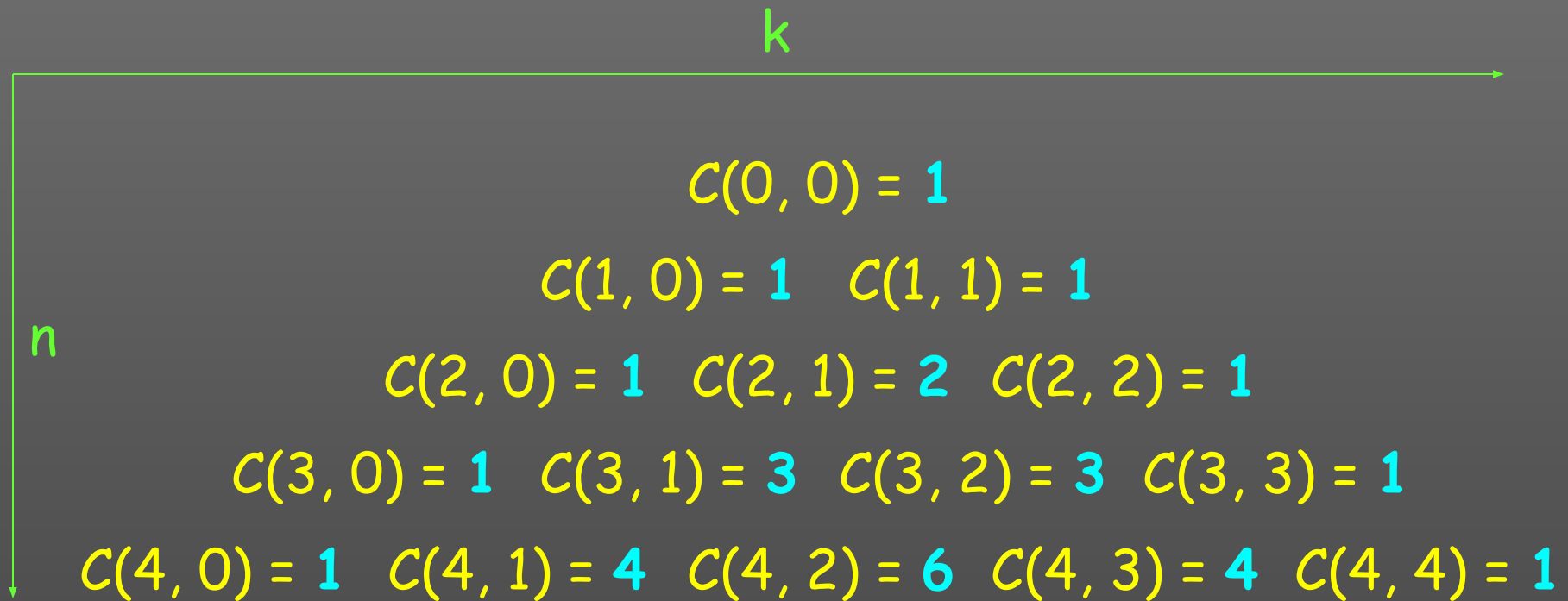
Pascal's Triangle

In Pascal's triangle, each number is the sum of the numbers to its upper left and upper right:



Pascal's Triangle

Since we have $C(n + 1, k) = C(n, k - 1) + C(n, k)$ and $C(0, 0) = 1$, we can use Pascal's triangle to simplify the computation of $C(n, k)$:



A diagram of Pascal's Triangle. A horizontal green arrow points to the right, labeled with a green 'k' above it. A vertical green arrow points downwards, labeled with a green 'n' to its left. The triangle is composed of rows of binomial coefficients $C(n, k)$. The values are displayed in yellow text, with the numbers 1, 2, 3, 4, and 6 highlighted in cyan. The rows are as follows:

$C(0, 0) = 1$
$C(1, 0) = 1$ $C(1, 1) = 1$
$C(2, 0) = 1$ $C(2, 1) = 2$ $C(2, 2) = 1$
$C(3, 0) = 1$ $C(3, 1) = 3$ $C(3, 2) = 3$ $C(3, 3) = 1$
$C(4, 0) = 1$ $C(4, 1) = 4$ $C(4, 2) = 6$ $C(4, 3) = 4$ $C(4, 4) = 1$

Binomial Coefficients

Expressions of the form $C(n, k)$ are also called **binomial coefficients**.

How come?

A **binomial expression** is the sum of two terms, such as $(a + b)$.

Now consider $(a + b)^2 = (a + b)(a + b)$.

When expanding such expressions, we have to form all possible products of a term in the first factor and a term in the second factor:

$$(a + b)^2 = a \cdot a + a \cdot b + b \cdot a + b \cdot b$$

Then we can sum identical terms:

$$(a + b)^2 = a^2 + 2ab + b^2$$

Binomial Coefficients

For $(a + b)^3 = (a + b)(a + b)(a + b)$ we have

$$(a + b)^3 = aaa + aab + aba + abb + baa + bab + bba + bbb$$

$$(a + b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$$

There is only one term a^3 , because there is only one possibility to form it: Choose **a** from all three factors: $C(3, 3) = 1$.

There is three times the term a^2b , because there are three possibilities to choose **a** from two out of the three factors: $C(3, 2) = 3$.

Similarly, there is three times the term ab^2 ($C(3, 1) = 3$) and once the term b^3 ($C(3, 0) = 1$).

Binomial Coefficients

This leads us to the following formula:

$$(a + b)^n = \sum_{j=0}^n C(n, j) \cdot a^{n-j} b^j \quad (\text{Binomial Theorem})$$

With the help of Pascal's triangle, this formula can considerably simplify the process of expanding powers of binomial expressions.

For example, the fifth row of Pascal's triangle (1 - 4 - 6 - 4 - 1) helps us to compute $(a + b)^4$:

$$(a + b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$$

Running time of algorithms

What is an algorithm?

An algorithm is a step by step specification of "how to do a Task"

A Program is a translation of an algorithm in a Programming Language

Analyzing programs means Analyzing algorithms or count the number of operations to be executed by the computer to complete the task written in the form of program

To evaluate the running time of a program, we study the pseudo code or the algorithm and then try to estimate the maximum number of operations required

Can we count exact number of operations/steps? Is it necessary?

Asymptotic analysis

It requires-

Writing program in pseudo code (more structured way of writing algorithm) (better than natural language but less formal than PL)

Characterize running time as a function of input size

Evaluating running time by studying a high level description of the algorithm without actually implementing it or running experiments on it

Evaluating relative performance independent of hardware and software environment

Asymptotic analysis

Example: Findmax (a, n) /* input an array of size n*/

Cmax = a[0]

For i = 1 to n-1

If cmax < a[i] then cmax = a[i]

Return cmax

If we examine above code, it has following operations:

Assignment

addition

Comparison

return

Asymptotic analysis cont..

Our objective is to analyze high level pseudo code
Every pseudo code is collection of certain set of basic instructions or primitive operations which are independent of programming language. For example:

1. *Assigning a value to a variable*
2. *Performing arithmetic operations*
3. *Comparing two numbers*
4. *Accessing an array through index*
5. *Calling a function*
6. *Returning from a function*
7. *Reading/writing.....*

A piece of code

```
// Here c is a constant  
for (int i = 1; i <= c; i++)  
{ // some  $O(1)$  expressions }
```

Complexity...

```
// Here c is a positive integer constant
for (int i = 1; i <= n; i += c)
{ // some O(1) expressions }

for (int i = n; i > 0; i -= c)
{ // some O(1) expressions }
```

$O(n)$ in both
cases ??

Example...

Selection sort

```
for i = 0 to n
{min = a[i]
Index = i
for j=i+1 to n
{
if a[i] > a[j]
{min = a[j]; index = j };
}
Swap(a[i],a[index]);
}
```

Asymptotic analysis cont..

A primitive operation corresponds to a low level instruction whose execution time depends on the hardware or software environment

Therefore, the number of primitive operations needed to execute the algorithm will give a high level estimate of the running time of that algorithm

Let us also assume that the running time of each primitive operation is approximately same

Hence, the no. of primitive operations will be directly proportional to the actual running time of algorithm

As a programmer and for a user, worst-case or average-case analysis is more important than best-case

It is more challenging to do average-case analysis in comparison to worst-case because it requires calculation over an input distribution

Worst-case requires the ability to identify the worst-case input only

So, we will specify running time in terms of worst-case

Example

Write an algorithm to find the maximum element of an array and find the complexity of it

Need for Asymptotic Notations

While evaluating the running time of simple algorithm `arraymax`, we had counted exactly how many primitive operations will be executed corresponding to each line of code

How important is this??

We do not want to calculate exact number of operations even for worst case

We want to capture "how the running time of an algorithm increases with the size of input in worst-case" or

we want to know how our program will behave for very large inputs (how much maximum time it will take)

We are not interested for small inputs

Asymptotic Notations

The "Big-Oh" notation or Big O-notation

Let $f(n)$ and $g(n)$ be functions, mapping nonnegative integers to real numbers.

We say that $f(n)$ is $O(g(n))$ if there exists a real constant $c > 0$ and an integer constant $n_0 \geq 1$, such that $f(n) \leq cg(n)$ for every integer $n \geq n_0$.

This definition is referred as " $f(n)$ is Big Oh of $g(n)$ " or " $f(n)$ is order of $g(n)$ "

Asymptotic Notations

Examine the behavior of $f(n) = n$, $f(n) = 2n+2$, $f(n) = n - 7$,
 $f(n) = 3n$..etc

We say that $7n-2$ is $O(n)$

To prove this, we have to find a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $7n-2 \leq cn$ for every $n \geq n_0$.

We can see that $c = 7$ and $n_0 = 1$ will fulfil this requirement

The big-Oh notation allows us to say that a function of n is less than or equal to another function, upto a constant factor and in asymptotic sense they are same as n grows to infinity

Cont...

$3n+2 = O(n)$ as $3n+2 \leq 4n$ for all $n \geq 2$

$3n+3 = O(n)$ as $3n+3 \leq 4n$ for all $n \geq 3$

$110n+6 = O(n)$ as $110n+6 \leq 101n$ for all $n \geq 6$

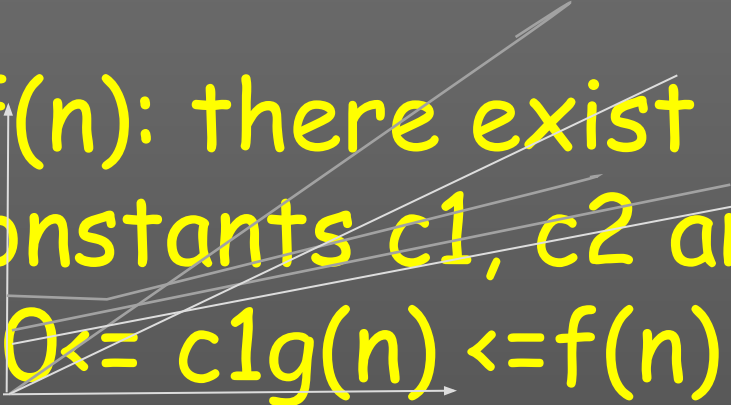
$10n^2 + 4n + 2 = O(n^2)$ as $10n^2 + 4n + 2 \leq 11n^2$ for all $n \geq 5$

$6 \cdot 2^n + n^2 = O(2^n)$ as $6 \cdot 2^n + n^2 \leq 7 \cdot 2^n$ for $n \geq 4$

Other Asymptotic notations

Θ notation : asymptotically tight bound

$\Theta(g(n)) = \{ f(n): \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}$



Cont....

Ω notation: lower bound

o notation: tight upper bound

Asymptotic Examples

$$20n^3 + 10n + 5$$

$$\log n + 5$$

$$n^2 + 10$$

$$5n - 20$$

$$5n + 100$$

$$4000$$

$$5n/2 + 100$$

```
for (int i = 1; i <=n; i += c)
{ for (int j = 1; j <=n; j += c)
  { // some O(1) expressions } }
for (int i = n; i > 0; i += c)
{ for (int j = i+1; j <=n; j += c)
  { // some O(1) expressions }
```

$O(n^2)$ and $O(n^2)$??

```
for (int i = 1; i <= n; i *= c)
{ // some O(1) expressions }
  for (int i = n; i > 0; i /= c)
  { // some O(1) expressions }
```

$$\log_c(n)$$

Working with nested loops..

1. Simple loops(n times and m times)

```
for (i = 0; i < N; i++)  
  { sequence of statements } X statements  
for (j = 0; j < M; j++)  
  { sequence of statements } Y statements
```

2. A nested loop followed by a non-nested loop: (one nested loop and another one simple loop)

```
for (i = 0; i < N; i++)  
  { for (j = 0; j < N; j++) { sequence of statements } }  
  for (k = 0; k < N; k++) { sequence of statements }
```

3. A nested loop in which the number of times the inner loop executes depends on the value of the outer loop index:

```
for (i = 0; i < N; i++) { for (j = N; j > i; j--) { sequence of statements } }
```

Cont...

The first loop is $O(N)$ and the second loop is $O(M)$. Since you don't know which is bigger, you say this is $O(N+M)$. This can also be written as $O(\max(N,M))$. In the case where the second loop goes to N instead of M the complexity is $O(N)$. You can see this from either expression above. $O(N+M)$ becomes $O(2N)$ and when you drop the constant it is $O(N)$. $O(\max(N,M))$ becomes $O(\max(N,N))$ which is $O(N)$.

The first set of nested loops is $O(N^2)$ and the second loop is $O(N)$. This is $O(\max(N^2, N))$ which is $O(N^2)$.

This is similar to earlier example of a nested loop where the number of iterations of the inner loop depends on the value of the index of the outer loop. The only difference is that in this example the inner-loop index is counting down from N to $i+1$. The inner loop executes N times, then $N-1$, then $N-2$, etc, so the total number of times the innermost "sequence of statements" is $O(N^2)$.