COMPSCI 220 S1 2024

Algorithms and Data Structures

L01. Algorithm Definition & Basics

Slides adapted from Mark Wilson, Georgy Gimel'farb, Simone Linz and Tanya Gvozdeva

Agenda

- Algorithm
- Analysis of Algorithms
- Mathematical Induction
- Math Basis

Agenda

- Algorithm
- Analysis of Algorithms
- Mathematical Induction
- Math Basis

What is algorithm?

- An algorithm is a sequence of clearly stated rules that specify a step-by-step method for solving a given problem.
- The rules should be unambiguous and sufficiently detailed that they can be carried out without creativity.
- Examples of algorithms: primary school method for multiplication of decimal integers; quicksort.
- Algorithms predate electronic computers by thousands of years (example: Euclid's greatest common divisor algorithm gcd(a,b)=gcd(b, a mod b)).
- A program is a sequence of computer instructions implementing the algorithm.

Example algorithms

- Baking a cake (not a computational algorithm):
  - Explicit and precise step-by-step instructions on how to bake the cake from given ingredients
  - Question: Is the list of ingredients for a cake an algorithm?
- Finding the mean (average) of $\boldsymbol{n}$ numbers $\{\boldsymbol{a_0}, \boldsymbol{a_1}, \ldots, \boldsymbol{a_{(n-1)}}\}$:
  - Summing all the numbers and dividing the sum by n: $\frac{1}{n}(a_0 + a_1 + \ldots + a_{(n-1)})$.
  - Question: Is an example of calculating the mean an algorithm? E.g., (1+2+3)/3 = 2
- Searching for a certain entry in a database:
  - Explicit and precise computational steps are required to find whether a given entry is or is not in the database.
  - Question: Is an index of the database entries an algorithm?

Agenda

- Algorithm
- Analysis of Algorithms
- Mathematical Induction
- Math Basis

Analysis of algorithms

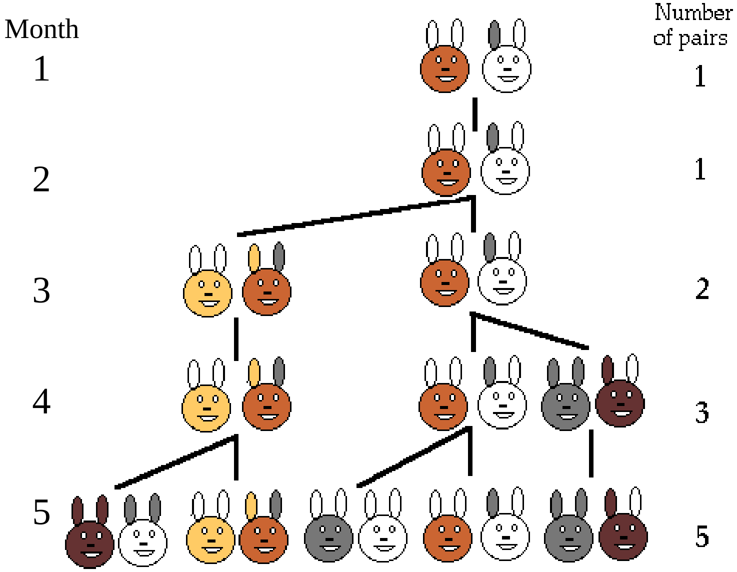
- What to analyse:
  - Domain of definition – what inputs are legal?
  - Correctness – does it solve the problem for all legal inputs?
  - Efficiency – its maximum or average requirements for resources:
    - Runtime
    - Memory space
    - Other resources
- There could be different implementations of the same algorithm: different programs, programming languages, computer platforms, operating systems, etc.
- The analysis should be isolated from a particular implementation.

Why analyse an algorithm?

- Experience shows that enormously more performance gains can be achieved by optimizing the algorithm than by optimizing other factors such as:
  - Processor
  - Language
  - Compiler
- The analysis process often results in us discovering simpler algorithms.
- Many algorithms have parameters that must be set before implementation. The analysis allows us to set the optimal values.
- Algorithms that have not been analysed for correctness often lead to major bugs in programs.

Example – Fibonacci numbers

- Italian mathematician, Leonardo Fibonacci (1170–1250). A problem of breeding rabbits.
- A pair of rabbits takes a month to become mature and start to have pairs of baby rabbits
- Each pair of newly born rabbits also takes a month to reach maturity.
- How many pairs of rabbits, $F(n)$ would there be after n months?



Fibonacci numbers: $F(n)$

$$F(n) = F(n-1) + F(n-2)$$
$$F(0) = 0, \qquad F(1) = 1$$

This immediately suggests a recursive algorithm

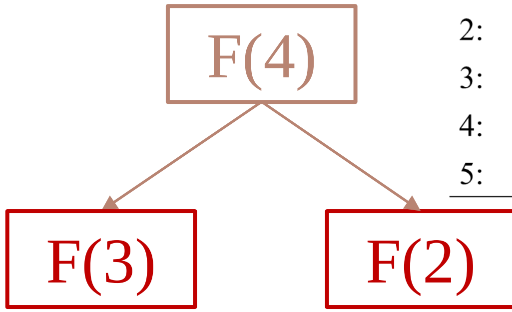
Example – Fibonacci numbers

Algorithm 1 Slow method for computing Fibonacci numbers

```
function SLOWFIB(integer n)
    if n < 0 then return 0
    else if n = 0 then return 0
    else if n = 1 then return 1
    else return SLOWFIB(n − 1) + SLOWFIB(n − 2)
```

- Correctness: The algorithm SLOWFIB is correct
- Efficiency: This algorithm is not efficient! It does a lot of repeated computation

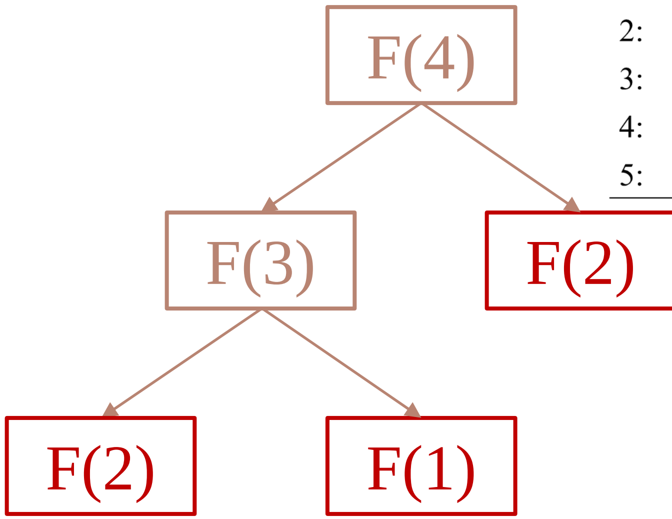
Example – F(4)



Algorithm 1 Slow method for computing Fibonacci numbers

```
function SLOWFIB(integer n)
    if n < 0 then return 0
    else if n = 0 then return 0
    else if n = 1 then return 1
    else return SLOWFIB(n − 1) + SLOWFIB(n − 2)
```

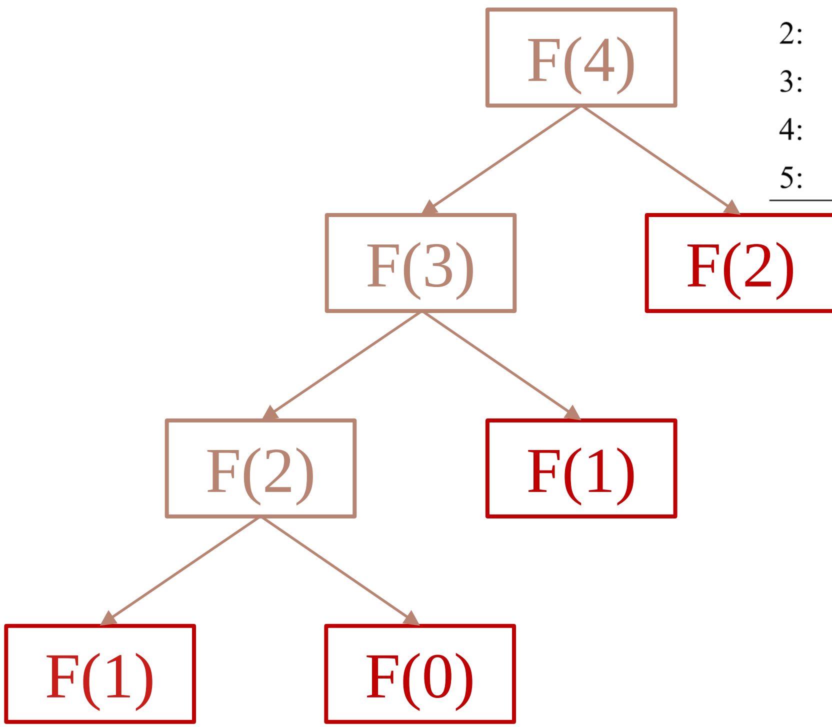
Example – F(4)



Algorithm 1 Slow method for computing Fibonacci numbers

```
function SLOWFIB(integer n)
    if n < 0 then return 0
    else if n = 0 then return 0
    else if n = 1 then return 1
    else return SLOWFIB(n − 1) + SLOWFIB(n − 2)
```

Example – F(4)



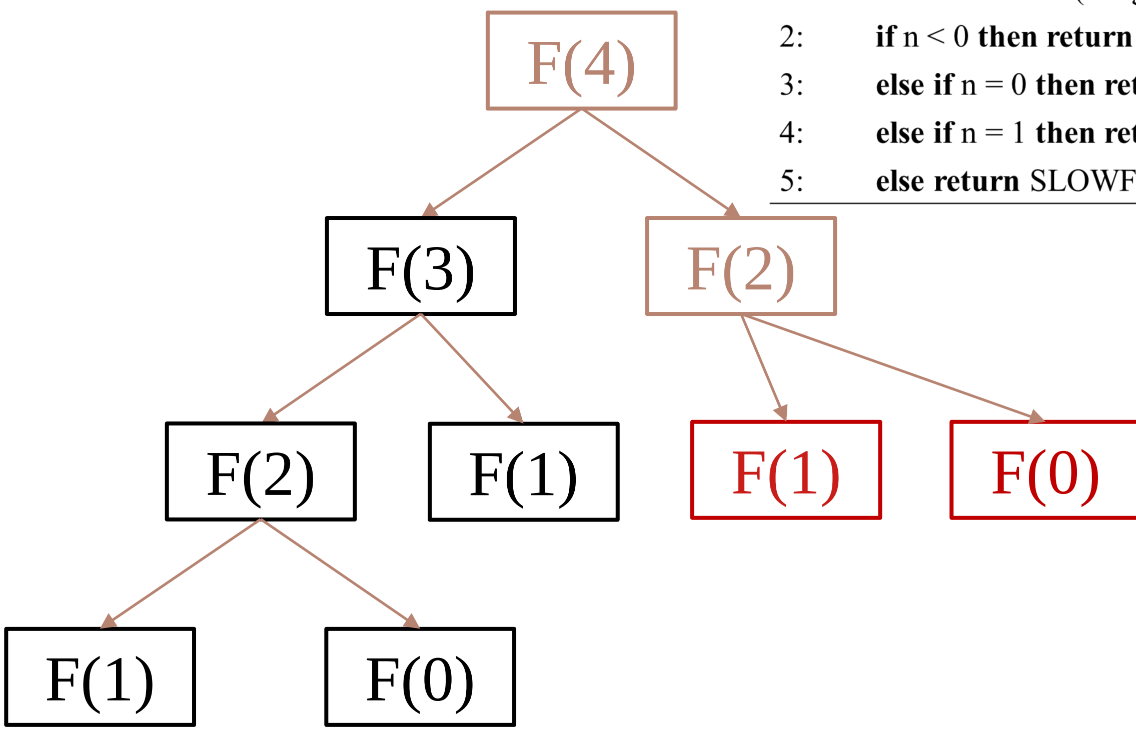
Algorithm 1 Slow method for computing Fibonacci numbers

function SLOWFIB(integer n)
  if $n < 0$ **then return** 0
  else if $n = 0$ **then return** 0
  else if $n = 1$ **then return** 1
  else return SLOWFIB($n - 1$) + SLOWFIB($n - 2$)

Example – F(4)

Algorithm 1 Slow method for computing Fibonacci numbers

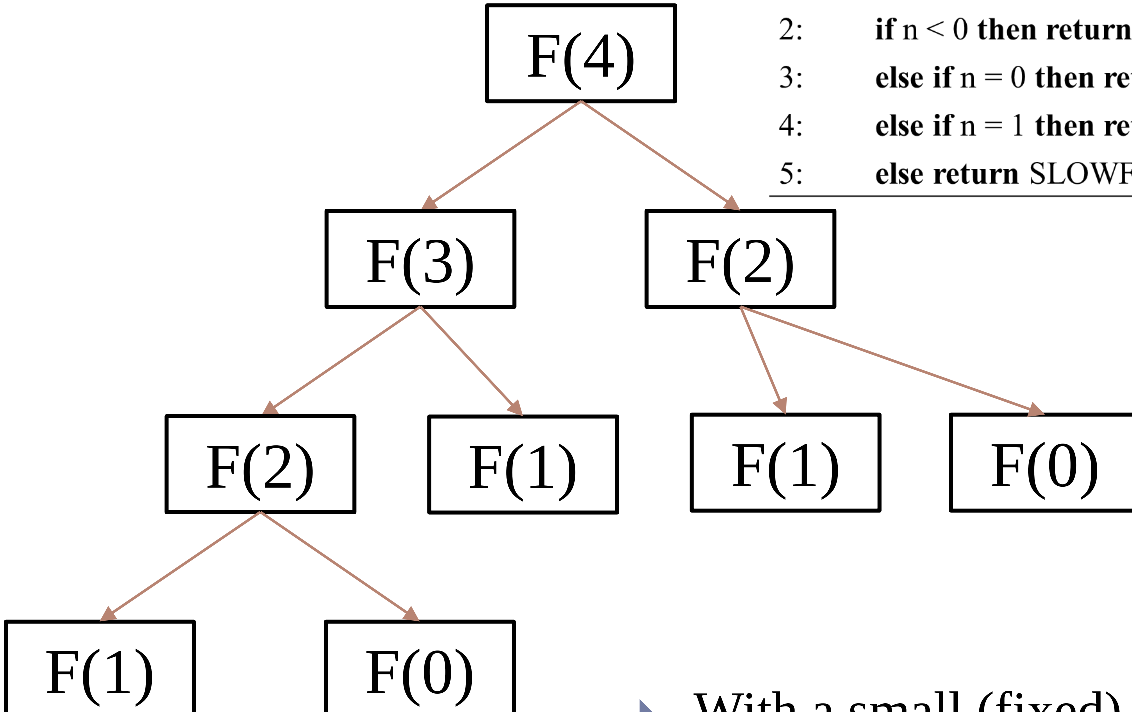
```
function SLOWFIB(integer n)
    if n < 0 then return 0
    else if n = 0 then return 0
    else if n = 1 then return 1
    else return SLOWFIB(n − 1) + SLOWFIB(n − 2)
```



Example – F(4)

Algorithm 1 Slow method for computing Fibonacci numbers

```
function SLOWFIB(integer n)
    if n < 0 then return 0
    else if n = 0 then return 0
    else if n = 1 then return 1
    else return SLOWFIB(n − 1) + SLOWFIB(n − 2)
```



- With a small (fixed) amount of extra space, we can do better, by working from the bottom up instead of from the top down.

Example – Fibonacci numbers

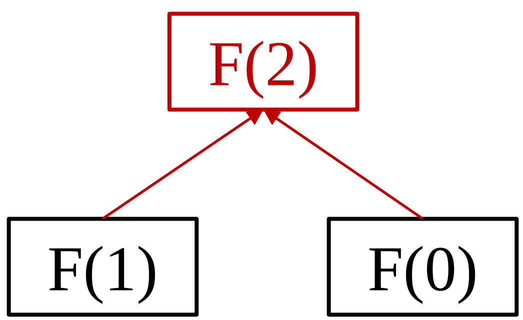
Algorithm 2 Fast method for computing Fibonacci numbers

function FASTFIB(integer n)
    if n $<$ 0 **then return** 0
    else if n = 0 **then return** 0
    else if n = 1 **then return** 1
    else
        $a \leftarrow 1$ stores $F(i-1)$ at bottom of loop
        b $\leftarrow$ 0   stores $F(i-2)$ at bottom of loop
        for $i \leftarrow 2$ **to** n **do**
            $t \leftarrow a$
            $a \leftarrow a + b$
            $b \leftarrow t$
    return a

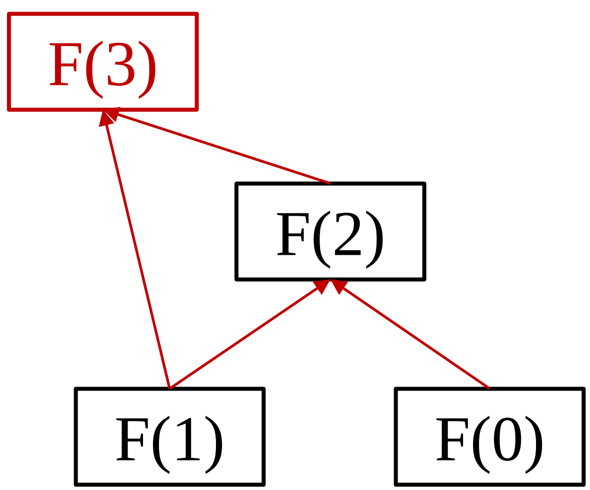
Example – F(4)

Algorithm 2 Fast method for computing Fibonacci numbers

```
function FASTFIB(integer n)
    if n < 0 then return 0
    else if n = 0 then return 0
    else if n = 1 then return 1
    else
        a ← 1                      stores F(i − 1) at bottom of loop
        b ← 0                      stores F(i − 2) at bottom of loop
        for i ← 2 to n do
            t ← a
            a ← a + b
            b ← t
    return a
```



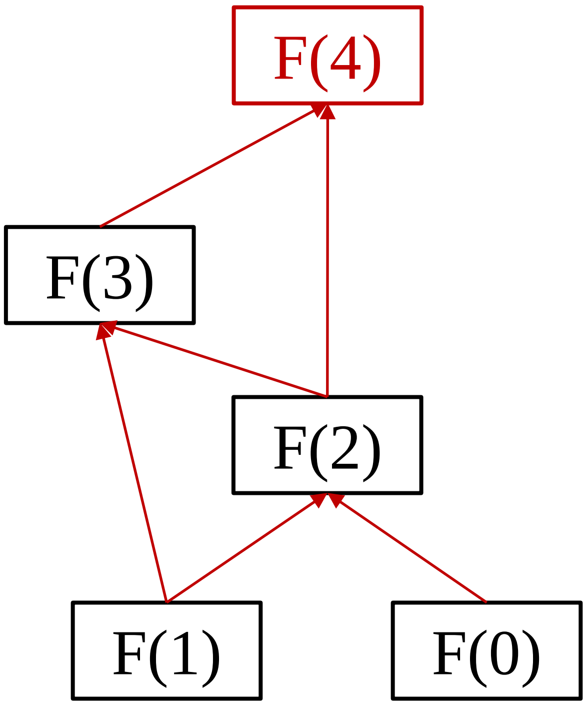
Example – F(4)



Algorithm 2 Fast method for computing Fibonacci numbers

```
function FASTFIB(integer n)
    if n < 0 then return 0
    else if n = 0 then return 0
    else if n = 1 then return 1
    else
        a ← 1                      stores F(i − 1) at bottom of loop
        b ← 0                      stores F(i − 2) at bottom of loop
        for i ← 2 to n do
            t ← a
            a ← a + b
            b ← t
    return a
```

Example – F(4)



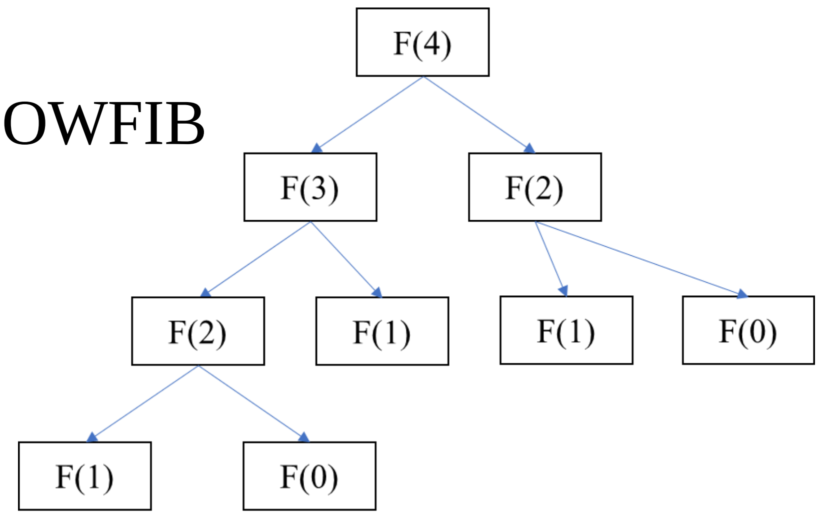
Algorithm 2 Fast method for computing Fibonacci numbers

```
function FASTFIB(integer n)
    if n < 0 then return 0
    else if n = 0 then return 0
    else if n = 1 then return 1
    else
        a ← 1                    stores F(i − 1) at bottom of loop
        b ← 0                    stores F(i − 2) at bottom of loop
        for i ← 2 to n do
            t ← a
            a ← a + b
            b ← t
    return a
```

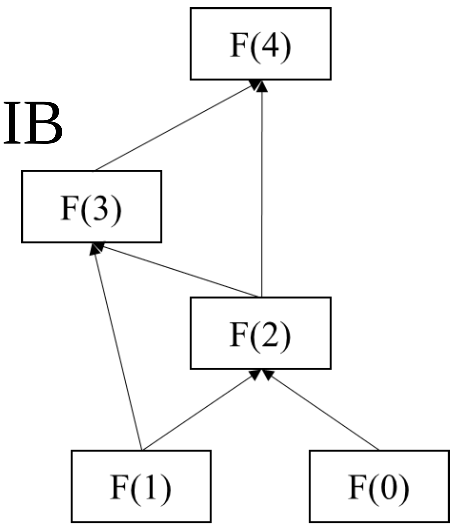
Analysis of the fast algorithm

- It is easy to see that the number of additions, function calls, etc needed by FASTFIB to compute $F(n)$ has the form $An + B$ for some constants A, B.

SLOWFIB



FASTFIB



- Proving the correctness of FASTFIB is done by mathematical induction on n. We will prove this later.

Agenda

- Algorithm
- Analysis of Algorithms
- Mathematical Induction
- Math Basis

Mathematical induction

A useful tool to prove a math statement is true for all integers $n \geq n_0$, where n_0 is a non-negative integer, it has three key steps:

1. **Basis**: Prove that the statement is true for n_0.
2. **Induction hypothesis**: Assume that the statement is true for some $n = k$.
3. **Inductive step from $\boldsymbol{n = k}$ to $\boldsymbol{k + 1}$**: If the induction hypothesis holds, prove that the statement is also true for $k + 1$

Mathematical induction

- Example 1: Prove the correctness of Gauss formula $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ for $n \geq 1$
- Basis: for n=n_0=1, the statement is true: $\frac{1(1+2)}{2} = 1$
- Induction hypothesis: Assume that the statement is true for some n=k
- Inductive step from $\boldsymbol{k}$ to $\boldsymbol{k+1}$:
  - By hypothesis $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$, then when $n = k + 1$

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

Mathematical induction

- Example 2: Prove that $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ for $n \geq 1$
- Basis: for $n = n_0 = 1$, the statement is true: $\frac{1(1+1)(2+1)}{6} = 1$
- Induction hypothesis: Assume that the statement is true for some n=k
- Inductive step from $\boldsymbol{k}$ to $\boldsymbol{k}$**+1**:
  - By hypothesis $\sum_{i=1}^{k} i^2 = \frac{k(k+1)(2k+1)}{6}$, then when $n = k + 1$

$$\sum_{i=1}^{n} i^2 = \sum_{i=1}^{k+1} i^2 = \sum_{i=1}^{k} i^2 + (k+1)^2 = \frac{k(k+1)(2k+1)}{6} + (k+1)^2$$

$$= \frac{k(k+1)(2k+1)+6(k+1)^2}{6} = \frac{(k+1)\left(2k^2+k+6k+6\right)}{6}$$

$$= \frac{(k+1)\left(2k^2+7k+6\right)}{6} = \frac{(k+1)(k+2)(2k+3)}{6}$$

$$= \frac{(k+1)\big((k+1)+1\big)(2(k+1)+1)}{6} = \frac{n(n+1)(2n+1)}{6}$$

Mathematical induction

- Example 3: Prove the correctness of FASTFIB
  - **Basis**: for $n = 1$ and $n = 0$, FASTFIB is correct.
  - **Induction hypothesis**: Assume it is correct for some $n = k$
  - **Inductive step from k to $k + 1$**:
    - By hypothesis a stores $F(k)$ and b stores $F(k-1)$ after running the k-th iteration.
    - Then, in the $(k+1)$-th iteration $i = k + 1$, a is updated to the value of $a + b = F(k) + F(k-1) = F(k+1)$, and b is updated to the value of $F(k)$.

Algorithm 2 Fast method for computing Fibonacci numbers

```
function FASTFIB(integer n)
    if n < 0 then return 0
    else if n = 0 then return 0
    else if n = 1 then return 1
    else
        a ← 1                    ▷ stores F(i − 1) at bottom of loop
        b ← 0                    ▷ stores F(i − 2) at bottom of loop
        for i ← 2 to n do
            t ← a
            a ← a + b
            b ← t
    return a
```

Agenda

- Algorithm
- Analysis of Algorithms
- Mathematical Induction
- Math Basis

Sets

- **Definition (Sets):** A set X is an **unordered** collection of zero or more elements
- Examples:

  $X = \{3, 4, 5, 6, 7\}$ – the set of integers from 3 to 7.
  $X = \{$Thor, Iron Man, Captain America$\}$ – heroes in "The Avengers".
  $X = \{A, \alpha, B, \beta, \ldots, \Omega, \omega\}$ – the Greek alphabet.

- By unordered, we mean: $\{3, 4, 5, 6, 7\} = \{5, 3, 7, 6, 4\}$
- Some other specifications $X = \{x | \ldots\}$ or $X = \{x: \ldots\}$:
  - Example: $X = \{x \mid x$ is an integer and $x \geq 3$ and $x \leq 7\}$
  - Meaning: X is the set all x such that x is an integer and is in the range of $[3, 7]$.

Sets

- **Definition (cardinality):** $|X|$ is the number of elements, or the **cardinality** of a set X

  |{3, 4, 5, 6, 7}| = 5

  |{Thor, Iron Man, Captain America}| = 3

  For an empty set Ø, it's cardinality is 0

- **Definition (Subsets & Superset):** A set Y whose elements are all also elements in X is a **subset** of X, denoted by $Y \subseteq X$. Meanwhile X is a **superset** of Y. If $Y \neq X$ and $Y \subseteq X$, Y is called a **strict subset** of X denoted by $Y \subset X$ and X is called a **strict superset** of Y.
  - $\{3, 4\}$ is a **subset** of $\{3, 5, 6, 7, 4\}$ and {3,5,6,7,4} is a **superset** of $\{3,4\}$
  - $\{3, 7, 9\}$ is not a subset of $\{3, 4, 5, 6, 7\}$
  - If $Y \subseteq X$, $|Y| \leq |X|$

Sets

- **Definition (union):** The **union** of two sets, X and Y, is the set of elements in **at least one of** X and Y. $X \cup Y = \{x \mid x \in X$ **OR** $x \in Y\}$.
  - If $X = \{3, 5, 6, 7, 4\}$ and $Y = \{2, 5, 7\}$, then $X \cup Y = \{2, 3, 4, 5, 6, 7\}$.
- **Definition (Intersection):** The **intersection** of two sets, X and Y, is the set of elements in **both** X and Y. $X \cap Y = \{x \mid x \in X$ **AND** $x \in Y\}$.
  - If $X = \{3, 5, 6, 7, 4\}$ and $Y = \{2, 5, 7\}$, then $X \cap Y = \{5, 7\}$.
- **Definition (Complement):** The **complement** of a **subset** Y of X, is the set of elements in X **but not in** Y. $X \backslash Y = \{x \mid x \in X$ **AND** $x \notin Y\}$.
  - If $X = \{3, 5, 6, 7, 4\}$ and $Y = \{3, 5, 7\}$, then $X \backslash Y = \{4, 6\}$.

Rounding

- **Definition (Rounding):** Rounding is an operation to replace a real number x with the closest integer.
- **Ceil** notation $\lceil x \rceil$: rounds up to the nearest integer larger than or equal to x
- **Floor** notation $\lfloor x \rfloor$: rounds down to the nearest integer smaller than or equal to x
- Examples:
  - $\lceil 1.2 \rceil = 2$
  - $\lfloor 1.2 \rfloor = 1$
  - $\lceil 3 \rceil = \lfloor 3 \rfloor = 3$

Exponential functions

- **Definition (Exponential functions):** An exponential function of x can be written in the following form: $f(x) = a^{bx+c}$, where a is called the base, and a, b, c are constants.
- Simple rules for exponential functions:
  - $a^{-x} = \frac{1}{a^x}$; $a^{x+y} = a^x \cdot a^y$; $a^{x-y} = \frac{a^x}{a^y}$; $(a^x)^y = (a^y)^x = a^{xy}$
  - The derivative $\frac{da^x}{dx} = a^x \ln a$
  - When $a = e$, where $e \approx 2.718$ is the Euler's constant, the derivative of e^x is itself.

Logarithms

- **Definition (Logarithm):** Logarithm is the inverse of exponential function – if $y = a^x$, then $x = \log_a y$. We say "x is logarithm to the base a of y".
- Commonly used logarithms:
  - "logarithm to base 2" (in computing)
  - "logarithm to base 10" (in engineering)
  - "logarithm to base e" (the "natural logarithm", denoted by $\ln \equiv \log_e$)
- Simple rules for logarithms:
  - $\log_a y = \log_b y \cdot \log_a b$ $\Rightarrow$ The base can be switched by a constant. So, we often don't care the base and use **log** for simplicity.
  - $\log(x \cdot y) = \log x + \log y$; $\log \frac{x}{y} = \log x - \log y$; $\log x^y = y \log x$