5. Induction and Recursion

5.1 Mathematical Induction

Consider the sum of the first n positive odd numbers:

1=1, 1+3=4, 1+3+5=9, 1+3+5+7=16, 1+3+5+7+9=25

Is it n^2?

Induction is a powerful tool to prove assertions of this type.

Mathematical Induction:

Prove the theorem: P(n) is true $\forall n \in Z^+$

Proof by induction:

1. Basis step P(1) is shown to be true
2. Inductive step P(n) $\rightarrow$ P(n+1) is shown to be true $\forall n \in Z^+$

Mathematical Induction:

Prove the theorem: P(n) is true $\forall n \in Z^+$

Proof by induction:

1. Basis step — P(1) is shown to be true
2. Inductive step — P(n) $\rightarrow$ P(n+1) is shown to be true $\forall n \in Z^+$

Then if we apply the following rule of inference,

$$[P(1) \wedge \forall n\, (P(n) \rightarrow P(n+1))] \rightarrow \forall n\, P(n)$$

to conclude that P(n) is true $\forall n \in Z^+$

e.g.

P(n) : The sum of first n positive odd integers is n^2.

Prove P(n) is true $\forall n \in Z^+$.

e.g.

P(n) : The sum of first n positive odd integers is n^2.

Prove P(n) is true $\forall n \in Z^+$.

<u>Basis step:</u>

P(1): $1 = 1^2$ (True)

e.g.
P(n) : The sum of first n positive odd integers is n^2.
Prove P(n) is true $\forall n \in Z^+$.

<u>Basis step:</u>

P(1): $1 = 1^2$ (True)

<u>Inductive step:</u>

? $P(n) \rightarrow P(n+1) \quad \forall n \in Z^+$

e.g.

P(n) : The sum of first n positive odd integers is n^2.

Prove P(n) is true $\forall n \in Z^+$.

<u>Basis step:</u>

P(1): $1 = 1^2$ (True)

<u>Inductive step:</u>

? $P(n) \rightarrow P(n+1) \quad \forall n \in Z^+$

Suppose, for a fixed arbitrary n, P(n) is T, i.e., $1 + 3 + \ldots + (2n-1) = n^2$

Then show P(n+1) is also T.

$1 + 3 + \ldots + (2n-1) + (2n+1) = (n+1)^2$?

e.g.

P(n) : The sum of first n positive odd integers is n^2.

Prove P(n) is true $\forall n \in Z^+$.

<u>Basis step:</u>

P(1): $1 = 1^2$ (True)

<u>Inductive step:</u>

? $P(n) \rightarrow P(n+1)$ $\forall n \in Z^+$

Suppose, for a fixed arbitrary n, P(n) is T, i.e., $1 + 3 + \ldots + (2n-1) = n^2$

Then show P(n+1) is also T.

$1 + 3 + \ldots + (2n-1) + (2n+1) = (n+1)^2$?

$= n^2 + (2n+1)$ (T)

e.g.
P(n) : The sum of first n positive odd integers is n^2.
Prove P(n) is true $\forall n \in Z^+$.

<u>Basis step:</u>

P(1): $1 = 1^2$ (True)

<u>Inductive step:</u>

? $P(n) \rightarrow P(n+1) \qquad \forall n \in Z^+$

Suppose, for a fixed arbitrary n, P(n) is T, i.e., $1 + 3 + \ldots + (2n-1) = n^2$
Then show P(n+1) is also T.

$$1 + 3 + \ldots + (2n-1) + (2n+1) = (n+1)^2 \ ?$$
$$= n^2 + (2n+1) \qquad \text{(T)}$$

$\therefore$ P(n) is T $\forall n \in Z^+$

<u>Remark</u>: Here P(n) is called *inductive hypothesis* for a fixed arbitrary n.

e.g.
Prove $n < 2^n \quad \forall n \in Z^+$

<u>Basis step</u>:

P(1): $1 < 2^1 = 2$ (T)

<u>Inductive step</u>:

Show $P(n) \rightarrow P(n+1) \quad \forall n \in Z^+$.
$n < 2^n \rightarrow n+1 < 2^{(n+1)}$?

e.g.
Prove $n < 2^n \qquad \forall n \in Z^+$

<u>Basis step</u>:

P(1): $1 < 2^1 = 2 \qquad\qquad$ (T)

<u>Inductive step</u>:

Show $P(n) \rightarrow P(n+1) \quad \forall n \in Z^+$.
$n < 2^n \rightarrow \; n+1 < 2^{(n+1)}$?

$n < 2^n \;\Rightarrow n+1 < 2^n + 1 \leq 2^n + 2^n = 2^{n+1}$
$\therefore$ P(n+1) is T
$\therefore \; n < 2^n \;\; \forall n \in Z^+$

e.g. Inequality for Harmonic Numbers:

Harmonic number:
$H_k = 1 + 1/2 + 1/3 + \cdots + 1/k$, $\quad k = 1, 2, 3\ldots$

Show that $H_{2^n} \geq 1 + n/2 \quad \forall n$, n is a nonnegative integer.

e.g. Inequality for Harmonic Numbers:

Harmonic number:
$H_k = 1 + 1/2 + 1/3 + \cdots + 1/k, \qquad k = 1, 2, 3\ldots$

Show that $H_{2^n} \geq 1 + n/2 \quad \forall n$, n is a nonnegative integer.

<u>Basis step</u>:

P(0) is true, since $H_{2^0} = H_1 = 1 \geq 1 + 0/2 = 1$.

<u>Inductive step</u>:

Assume P(n) is true $\Rightarrow \quad H_{2^n} \geq 1 + n/2$

$$\begin{aligned} H_{2^{n+1}} &= H_{2^n} + 1/(2^n + 1) + \ldots + 1/(2^{n+1}) \\ &\geq (1 + n/2) + 1/(2^n + 1) + \ldots. + 1/(2^{n+1}) \\ &\geq (1 + n/2) + 2^n (1/2^{n+1}) = 1 + (n+1)/2 \end{aligned}$$

$\therefore$ P(n+1) is true.

Hence by induction $H_{2^n} \geq 1 + n/2 \quad \forall n$

e.g.
Prove that $2^n < n!$ for $n = 4, 5, 6, \ldots$.

Let P(n): $2^n < n!$

e.g.
Prove that $2^n < n!$ for $n = 4, 5, 6, \ldots.$

Let P(n): $2^n < n!$

<u>Basis step</u>:

P(4) is true, since $2^4 = 16 < 4! = 24$

<u>Inductive step</u>:

Assume P(n) is true: $2^n < n!$

$$\begin{aligned} \Rightarrow \quad 2^{n+1} &< 2n! \\ &< (n+1)\, n! \\ &= (n+1)! \qquad\qquad \therefore \text{ P}(n+1) \text{ is true.} \end{aligned}$$

Hence by induction $2^n < n!$ for $n = 4, 5, 6, \ldots.$

5.2 Strong Induction

1. **Basis step:**
   Show that P(1) is true.

2. **Inductive step:**
   Show that $[P(1) \wedge P(2) \wedge \ldots \wedge P(n)] \rightarrow P(n+1)$ is true $\forall n \in Z^+$.

1. **Basis step:** Show that P(1) is true.
2. **Inductive step:** Show that $[P(1) \wedge P(2) \wedge \ldots \wedge P(n)] \rightarrow P(n+1)$ is true $\forall n \in Z^+$.

e.g. Show that if $n > 1$ integer, then n is either prime or can be written as a product of primes.
Let P(n) be "n is either prime or can be written as the product of primes".

1. **Basis step:** Show that P(1) is true.
2. **Inductive step:** Show that $[P(1) \wedge P(2) \wedge \ldots \wedge P(n)] \rightarrow P(n+1)$ is true $\forall n \in Z^+$.

e.g. Show that if $n > 1$ integer, then n is either prime or can be written as a product of primes.
Let P(n) be "n is either prime or can be written as the product of primes".
Basis step: P(2) is true since 2 is prime itself.

1. **Basis step:** Show that P(1) is true.
2. **Inductive step:** Show that $[P(1) \wedge P(2) \wedge \ldots \wedge P(n)] \rightarrow P(n+1)$ is true $\forall n \in Z^+$.

e.g. Show that if $n > 1$ integer, then n is either prime or can be written as a product of primes.
Let P(n) be "n is either prime or can be written as the product of primes".
<u>Basis step</u>: P(2) is true since 2 is prime itself.
<u>Inductive step</u>: Assume P(k) is true for all $2 \leq k \leq n$. Show P(n+1) is also true.

1. **Basis step:** Show that P(1) is true.
2. **Inductive step:** Show that $[P(1) \wedge P(2) \wedge \ldots \wedge P(n)] \rightarrow P(n+1)$ is true $\forall n \in Z^+$.

e.g. Show that if $n > 1$ integer, then n is either prime or can be written as a product of primes.

Let $P(n)$ be "n is either prime or can be written as the product of primes".

Basis step: P(2) is true since 2 is prime itself.

Inductive step: Assume $P(k)$ is true for all $2 \leq k \leq n$. Show $P(n+1)$ is also true.

If $(n+1)$ is prime then $P(n+1)$ is already true.

1. **Basis step:** Show that P(1) is true.
2. **Inductive step:** Show that $[P(1) \wedge P(2) \wedge \ldots \wedge P(n)] \rightarrow P(n+1)$ is true $\forall n \in Z^+$.

e.g. Show that if $n > 1$ integer, then n is either prime or can be written as a product of primes.
Let $P(n)$ be "n is either prime or can be written as the product of primes".
Basis step: P(2) is true since 2 is prime itself.
Inductive step: Assume $P(k)$ is true for all $2 \leq k \leq n$. Show $P(n+1)$ is also true.

If $(n+1)$ is prime then $P(n+1)$ is already true.
If $(n+1)$ is composite then $n+1 = a \cdot b$ s.t. $1 < a \leq b < n+1$.
Since we know that $P(a)$ and $P(b)$ are true by inductive hypothesis, a and b are either prime or can be written as product of primes.
$\Rightarrow a \cdot b$ can also be written as product of primes.
$\Rightarrow P(n+1)$ is also true.
$\therefore P(n)$ is T $\forall n > 1$ by **strong induction**

This completes the proof of the *Fundamental Theorem of Arithmetic* (see previous lectures, Ch. 4).

5.3 Recursive Definitions

A function can often be defined also recursively:

1. Specify the value of the function at the beginning, e.g., at zero.
2. Give a rule for finding its value based on its previous values.

5.3 Recursive Definitions

A function can often be defined also recursively:

1. Specify the value of the function at the beginning, e.g., at zero.
2. Give a rule for finding its value based on its previous values.

e.g.

$$a_n = 2^n \qquad n = 0, 1, 2, \ldots$$

$$\Rightarrow \quad a_{n+1} = 2a_n \qquad a_0 = 1, \quad n = 0, 1, 2, \ldots.$$

5.3 Recursive Definitions

A function can often be defined also recursively:

1. Specify the value of the function at the beginning, e.g., at zero.
2. Give a rule for finding its value based on its previous values.

e.g.

$$a_n = 2^n \qquad n = 0, 1, 2, \ldots$$

$$\Rightarrow \quad a_{n+1} = 2a_n \qquad a_0 = 1, \quad n = 0, 1, 2, \ldots.$$

e.g.
$F(n) = n!$ can be defined recursively:

$F(0) = 1$

$F(n+1) = F(n)(n+1)$

e.g.
Fibonacci numbers:

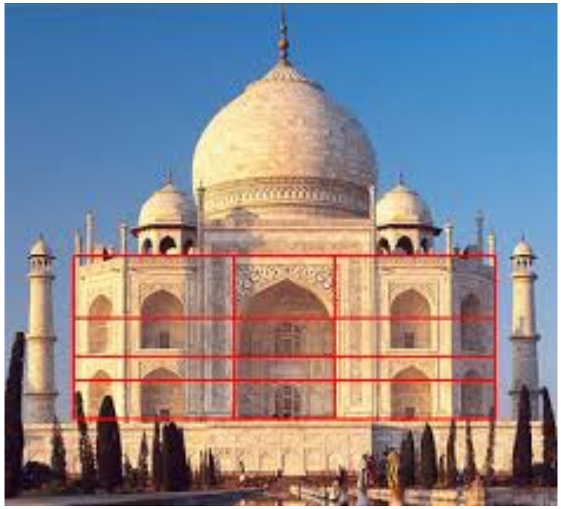
$$f_0 = 0, \quad f_1 = 1$$

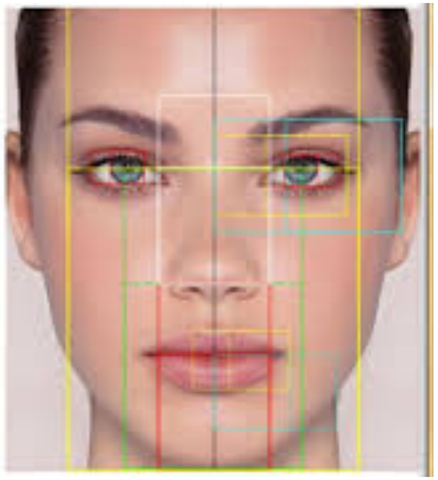
$$f_{n+1} = f_n + f_{n-1} \qquad n = 1, 2, \ldots$$

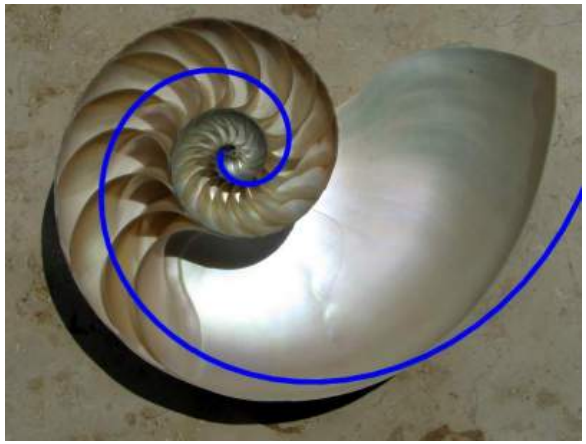
e.g.
Fibonacci numbers:

$$f_0 = 0, \quad f_1 = 1$$
$$f_{n+1} = f_n + f_{n-1} \qquad n = 1, 2, \ldots$$

$$\boxed{\lim_{n\to\infty} \frac{f_{n+1}}{f_n} = \frac{1+\sqrt{5}}{2} \approx \text{ Golden Ratio}}$$







Golden ratio, found in nature and used in art and architecture

e.g. Show that $f_n > \alpha^{n\text{-}2}$ $\forall n \geq 3$, where $\alpha = (1+\sqrt{5})/2$.
Note that α is a solution of $x^2 - x - 1 = 0$. Use **strong induction**.

e.g. Show that $f_n > \alpha^{n-2}$ $\forall n \geq 3$, where $\alpha = (1+\sqrt{5})/2$.
Note that α is a solution of $x^2 - x - 1 = 0$. Use **strong induction**.

Let P(n): $f_n > \alpha^{n-2}$

<u>Basis step</u>: $n = 3 \Rightarrow \alpha < 2 = f_3$ $\therefore$ P(3) is true

$n = 4 \Rightarrow \alpha^2 = (3+\sqrt{5})/2 < 3 = f_4$ $\therefore$ P(4) is true

e.g. Show that $f_n > \alpha^{n-2}$ $\forall n \geq 3$, where $\alpha = (1+\sqrt{5})/2$.
Note that α is a solution of $x^2 - x - 1 = 0$. Use **strong induction**.

Let P(n): $f_n > \alpha^{n-2}$

<u>Basis step</u>: $n = 3 \Rightarrow \alpha < 2 = f_3$ $\therefore$ P(3) is true

$n = 4 \Rightarrow \alpha^2 = (3+\sqrt{5})/2 < 3 = f_4$ $\therefore$ P(4) is true

<u>Inductive step</u>: Assume $f_k > \alpha^{k-2}$ $\forall k, 3 \leq k \leq n$ where $n \geq 4$

$f_{n+1} > \alpha^{n-1}$?

e.g. Show that $f_n > \alpha^{n-2}$ $\forall n \geq 3$, where $\alpha = (1+\sqrt{5})/2$.
Note that α is a solution of $x^2 - x - 1 = 0$. Use **strong induction**.

Let P(n): $f_n > \alpha^{n-2}$

Basis step: $n = 3 \Rightarrow \alpha < 2 = f_3$ $\quad\therefore$ P(3) is true

$n = 4 \Rightarrow \alpha^2 = (3+\sqrt{5})/2 < 3 = f_4$ $\quad\therefore$ P(4) is true

Inductive step: Assume $f_k > \alpha^{k-2}$ $\quad\forall k, 3 \leq k \leq n$ where $n \geq 4$

$f_{n+1} > \alpha^{n-1}$?

Since α is a solution of $x^2 - x - 1 = 0$
$\therefore\ \alpha^2 = \alpha+1$
$\Rightarrow \alpha^{n-1} = \alpha^2 \cdot \alpha^{n-3} = (\alpha+1)\ \alpha^{n-3} = \alpha^{n-2} + \alpha^{n-3}$

e.g. Show that $f_n > \alpha^{n-2}$ $\forall n \geq 3$, where $\alpha = (1+\sqrt{5})/2$.
Note that α is a solution of $x^2 - x - 1 = 0$. Use **strong induction**.

Let P(n): $f_n > \alpha^{n-2}$

Basis step: $n = 3 \Rightarrow \alpha < 2 = f_3$ $\therefore$ P(3) is true

$n = 4 \Rightarrow \alpha^2 = (3+\sqrt{5})/2 < 3 = f_4$ $\therefore$ P(4) is true

Inductive step: Assume $f_k > \alpha^{k-2}$ $\forall k$, $3 \leq k \leq n$ where $n \geq 4$

$f_{n+1} > \alpha^{n-1}$?

Since α is a solution of $x^2 - x - 1 = 0$

$\therefore \alpha^2 = \alpha + 1$

$\Rightarrow \alpha^{n-1} = \alpha^2 \cdot \alpha^{n-3} = (\alpha+1)\,\alpha^{n-3} = \alpha^{n-2} + \alpha^{n-3}$

By inductive hypothesis, $f_n > \alpha^{n-2}$ and $f_{n-1} > \alpha^{n-3}$

e.g. Show that $f_n > \alpha^{n-2}$ $\forall n \geq 3$, where $\alpha = (1+\sqrt{5})/2$.
Note that α is a solution of $x^2 - x - 1 = 0$. Use **strong induction**.

Let P(n): $f_n > \alpha^{n-2}$
Basis step: $n = 3 \Rightarrow \alpha < 2 = f_3$ $\therefore$ P(3) is true
$n = 4 \Rightarrow \alpha^2 = (3+\sqrt{5})/2 < 3 = f_4$ $\therefore$ P(4) is true

Inductive step: Assume $f_k > \alpha^{k-2}$ $\forall k, 3 \leq k \leq n$ where $n \geq 4$

$f_{n+1} > \alpha^{n-1}$?

Since α is a solution of $x^2 - x - 1 = 0$
$\therefore \alpha^2 = \alpha + 1$
$\Rightarrow \alpha^{n-1} = \alpha^2 \cdot \alpha^{n-3} = (\alpha+1)\,\alpha^{n-3} = \alpha^{n-2} + \alpha^{n-3}$

By inductive hypothesis, $f_n > \alpha^{n-2}$ and $f_{n-1} > \alpha^{n-3}$
$\Rightarrow f_{n+1} = f_n + f_{n-1} > \alpha^{n-2} + \alpha^{n-3} = \alpha^{n-1}$ $\therefore$ $f_n > \alpha^{n-2}$ $\forall n \geq 3$ by strong induction.

e.g.
Show that $f_n < (5/3)^n \ \forall n \geq 0$. Exercise (use strong induction).

Theorem: *Lamé's Theorem*
Let $a,b \in Z^+$ s.t $a \le b$.
The number n of division steps used by **Euclidean algorithm** to find gcd(a,b) $\le$ 5 times the number of decimal digits in a, that is, $n \le 5(\lfloor \log_{10} a \rfloor + 1)$.

Hence the complexity of Euclidean algorithm is O(log a).

Recall the code: (Euclidean Algorithm, $a \le b$)

```
int gcd(int a, int b)
{
    int x,y,r;
    x = b;
    y = a;
    while (y != 0){
        r = x % y;
        x = y;
        y = r;
    }
    return x;
}
```

Example:

$155 = 125 \cdot 1 + 30$
$125 = 30 \cdot 4 + 5$
$30 = 5 \cdot 6$
$\therefore$ gcd (155, 125) = 5

Proof: Let $b = r_0$ and $a = r_1$

$$r_0 = r_1 q_1 + r_2 \qquad 0 \le r_2 < r_1$$

$$r_1 = r_2 q_2 + r_3 \qquad 0 \le r_3 < r_2$$

$$\vdots$$

$$r_{n-2} = r_{n-1} q_{n-1} + r_n \qquad 0 \le r_n < r_{n-1}$$

$$r_{n-1} = r_n q_n$$

n division steps to find r_n , and $q_i \ge 1$ and $q_n \ge 2$,

Example:

$155 = 125 \cdot 1 + 30$

$125 = 30 \cdot 4 + 5$

$30 = 5 \cdot 6$

$\therefore \gcd(155, 125) = 5$

<u>Proof</u>: Let $b = r_0$ and $a = r_1$

$r_0 = r_1 q_1 + r_2 \qquad 0 \le r_2 < r_1$

$r_1 = r_2 q_2 + r_3 \qquad 0 \le r_3 < r_2$

$\vdots$

$r_{n-2} = r_{n-1} q_{n-1} + r_n \qquad 0 \le r_n < r_{n-1}$

$r_{n-1} = r_n q_n$

n division steps to find r_n , and $q_i \ge 1$ and $q_n \ge 2$,

$r_n \ge 1 = f_2$ (f_n : nth Fibonacci number)

$r_{n-1} \ge 2r_n \ge 2f_2 = f_3$

<u>Example</u>:

$155 = 125 \cdot 1 + 30$

$125 = 30 \cdot 4 + 5$

$30 = 5 \cdot 6$

$\therefore$ gcd (155, 125) = 5

<u>Proof</u>: Let $b = r_0$ and $a = r_1$

$$r_0 = r_1 q_1 + r_2 \qquad 0 \le r_2 < r_1$$
$$r_1 = r_2 q_2 + r_3 \qquad 0 \le r_3 < r_2$$
$$\vdots$$
$$r_{n-2} = r_{n-1} q_{n-1} + r_n \qquad 0 \le r_n < r_{n-1}$$
$$r_{n-1} = r_n q_n$$

n division steps to find r_n , and $q_i \ge 1$ and $q_n \ge 2$,

$r_n \ge 1 = f_2$

$r_{n-1} \ge 2r_n \ge 2f_2 = f_3$

$r_{n-2} \ge r_{n-1} + r_n \ge f_3 + f_2 = f_4$

<u>Example</u>:

$155 = 125 \cdot 1 + 30$

$125 = 30 \cdot 4 + 5$

$30 = 5 \cdot 6$

$\therefore$ gcd (155, 125) = 5

Proof: Let $b = r_0$ and $a = r_1$

$$\begin{array}{ll} r_0 = r_1 q_1 + r_2 & 0 \le r_2 < r_1 \\ r_1 = r_2 q_2 + r_3 & 0 \le r_3 < r_2 \\ \vdots & \\ r_{n-2} = r_{n-1} q_{n-1} + r_n & 0 \le r_n < r_{n-1} \\ r_{n-1} = r_n q_n & \end{array}$$

n division steps to find r_n , and $q_i \ge 1$ and $q_n \ge 2$,

$r_n \ge 1 = f_2$

$r_{n-1} \ge 2r_n \ge 2f_2 = f_3$

$r_{n-2} \ge r_{n-1} + r_n \ge f_3 + f_2 = f_4$

$\vdots$

$r_2 \ge r_3 + r_4 \ge f_{n-1} + f_{n-2} = f_n$

$a = r_1 \ge r_2 + r_3 \ge f_n + f_{n-1} = f_{n+1}$ $\quad\quad \therefore\ a \ge f_{n+1}$

Example:

$155 = 125 \cdot 1 + 30$

$125 = 30 \cdot 4 + 5$

$30 = 5 \cdot 6$

$\therefore$ gcd (155, 125) = 5

So we have $a \geq f_{n+1}$.

We also know that $f_{n+1} \geq \alpha^{n-1}$ for $n > 2$, where $\alpha = (1 + \sqrt{5}) / 2$.

$\Rightarrow a \geq \alpha^{n-1}$

Recall that n is the number of division steps required by the Euclidean algorithm.

So we have $a \geq f_{n+1}$.

We also know that $f_{n+1} \geq \alpha^{n-1}$ for $n > 2$,
where $\alpha = (1 + \sqrt{5}) / 2$.

$\Rightarrow a \geq \alpha^{n-1}$
$\Rightarrow \log_{10} a \ \geq\ (n-1)\log_{10}\alpha \ >\ (n-1) / 5$
$\therefore\ n-1 < 5\log_{10} a \Rightarrow n < 5\log_{10} a + 1$

Recall that n is the number of division steps required by the Euclidean algorithm.

So we have $a \geq f_{n+1}$.

We also know that $f_{n+1} \geq \alpha^{n-1}$ for $n > 2$,
where $\alpha = (1 + \sqrt{5}) / 2$.

Recall that n is the number of division steps required by the Euclidean algorithm.

$\Rightarrow a \geq \alpha^{n-1}$
$\Rightarrow \log_{10} a \;\geq\; (n-1)\log_{10}\alpha \;>\; (n-1) / 5$
$\therefore\; n-1 < 5\log_{10} a \Rightarrow n < 5\log_{10} a + 1$
$\Rightarrow n < 5(\lfloor \log_{10} a \rfloor + 1) + 1$ since $\lfloor \log_{10} a \rfloor + 1 > \log_{10} a$
$\Rightarrow n \leq 5(\lfloor \log_{10} a \rfloor + 1)$

So we have $a \geq f_{n+1}$.

We also know that $f_{n+1} \geq \alpha^{n-1}$ for $n > 2$,
where $\alpha = (1 + \sqrt{5}) / 2$.

Recall that n is the number of division steps required by the Euclidean algorithm.

$\Rightarrow a \geq \alpha^{n-1}$
$\Rightarrow \log_{10} a \;\geq\; (n-1)\log_{10}\alpha \;>\; (n-1) / 5$
$\therefore\; n-1 < 5\log_{10} a \Rightarrow n < 5\log_{10} a + 1$
$\Rightarrow n < 5(\lfloor \log_{10} a \rfloor + 1) + 1$ since $\lfloor \log_{10} a \rfloor + 1 > \log_{10} a$
$\Rightarrow n \leq 5(\lfloor \log_{10} a \rfloor + 1)$
Hence the complexity of Euclidean algorithm is O(log a), which is easy to show using the definition of big-O notation.

So we have $a \geq f_{n+1}$.

We also know that $f_{n+1} \geq \alpha^{n-1}$ for $n > 2$,
where $\alpha = (1 + \sqrt{5}) / 2$.

Recall that n is the number of division steps required by the Euclidean algorithm.

$\Rightarrow a \geq \alpha^{n-1}$
$\Rightarrow \log_{10} a \ \geq \ (n-1)\log_{10}\alpha \ > \ (n-1) / 5$
$\therefore\ n-1 < 5\log_{10} a \Rightarrow n < 5\log_{10} a + 1$
$\Rightarrow n < 5(\lfloor \log_{10} a \rfloor + 1) + 1$ since $\lfloor \log_{10} a \rfloor + 1 > \log_{10} a$
$\Rightarrow n \leq 5(\lfloor \log_{10} a \rfloor + 1)$
Hence the complexity of Euclidean algorithm is O(log a), which is easy to show using the definition of big-O notation.

Note also that (# of decimal digits in a) = $\lfloor \log_{10} a \rfloor + 1$
$\therefore\ n \leq 5\cdot$(# of decimal digits in a) (which is *Lamé's Theorem*)

5.4 Recursive Algorithms

Definition:
An algorithm is called *recursive* if it solves a problem by reducing it to an instance of the same problem with smaller input.

e.g. Compute GCD(a,b), $a \le b$

```
int GCD(int a, int b){
    int gcdAB;
    if (a == 0) gcdAB = b;
    else gcdAB = GCD(b % a, a);
    return gcdAB;
}
```

Example:

$155 = 125 \cdot 1 + 30$
$125 = 30 \cdot 4 + 5$
$30 = 5 \cdot 6$
$\therefore$ gcd (125, 155) = gcd (30,125)
= gcd (5,30) = gcd (0,5) = 5

e.g. Compute a^n, $a \in R$ and $n \in N$

$a^0 = 1, \quad a^{n+1} = aa^n$

```
int power(int a, int n){
    int p;
    if (n == 0) p = 1;
    else p = a*power(a,n-1);
    return p;
}
```

e.g. Product of two integers

```
int product(int m,int n){
    int prod;
    if (n==1) prod = m;
    else prod = m + product(m, n-1);
    return prod;
}
```

e.g. Linear search x in $\{a_0, a_1, \ldots, a_{n-1}\}$

```
int search(int x, int a[], int n, int i){
    int location;
    if (a[i] == x)
       location = i;
    else if (i == n-1)
        location = -1;
    else
        location = search(x,a,n,i+1);
    return location;
}
```

Initially $i = 0$.

e.g. Binary search pseudocode

```
function binary search(x,a,i,j)
    m = ⌊(i + j)/2⌋;
    if (x == a[m])  loc = m;
    else if (x < a[m] and i < m)
        loc = binary search(x,a, i, m-1);      (search in the first half)
    else if (x > a[m] and j> m)
        loc = binary search(x,a,m+1, j);       (search in the second half)
    else
        loc = -1;
    return loc;
```

Initially $i = 0$ and $j = n - 1$.

Recursion vs. Iteration:

e.g. Computing factorial

Recursive:

```
function factorial(n: positive integer)
if (n == 1)fact = 1;
else fact = n*factorial (n-1);
return fact;
```

Iterative:

```
function factorial(n: positive integer)
fact = 1;
for (i = 1; i ≤ n; i++)
      fact = fact * i;
return fact;
```

ALL recursive algorithms have an iterative equivalent, and vice versa.

Recursion vs. Iteration:

e.g. Computing factorial

Recursive:

```
function factorial(n: positive integer)
if (n == 1)fact = 1;
else fact = n*factorial (n-1);
return fact;
```

Iterative:

```
function factorial(n: positive integer)
fact = 1;
for (i = 1; i ≤ n; i++)
      fact = fact * i;
return fact;
```

Let $T(n)$ be the time complexity of the recursive solution (in terms of comparison and arithmetic operations).

$T(n) = T(n-1) + 3$ for $n \geq 2$ with $T(1) = 1$.

$$
\begin{aligned}
\text{Then } T(n) &= T(n-2) + 3 + 3 \\
&= T(n-3) + 3 + 3 + 3 \\
&\ \ \vdots \\
&= T(1) + 3 + 3 + \ldots + 3 \\
&= 3(n-1)+1 = 3n-2
\end{aligned}
$$

Hence $T(n)$ is $O(n)$.

Recursion vs. Iteration:

e.g. Computing factorial

Recursive:

```
function factorial(n: positive integer)
if (n == 1)fact = 1;
else fact = n*factorial (n-1);
return fact;
```

Iteration:

```
function factorial(n: positive integer)
fact = 1;
for (i = 1; i ≤ n; i++)
      fact = fact * i;
return fact;
```

Let $T(n)$ be the time complexity of the recursive solution (in terms of comparison and arithmetic operations).

$T(n) = T(n-1) + 3$ for $n \geq 2$ with $T(1) = 1$.

$$
\begin{aligned}
\text{Then } T(n) &= T(n-2) + 3 + 3 \\
&= T(n-3) + 3 + 3 + 3 \\
&\;\;\vdots \\
&= T(1) + 3 + 3 + \ldots + 3 \\
&= 3(n-1)+1 = 3n-2
\end{aligned}
$$

Hence $T(n)$ is $O(n)$.

Iterative solutions are usually faster than recursive solutions, though in this example both algorithms are of $O(n)$ complexity.

e.g. Computing Fibonacci numbers

Recursive:

```
function Fibonacci(n: nonnegative integer)
if (n==0) fibonacci = 0;
else if (n==1) Fibonacci = 1;
else fibonacci = Fibonacci(n-1) + Fibonacci(n-2);
return fibonacci;
```

Iterative:

```
function Fibonacci(n: nonnegative integer)
if (n==0) y = 0;
else{
      x=0;
      y=1;
      for (i=1; i < n; i++){
            z = x+y;
            x = y;
            y = z;
      }
}
return y;
```

Complexity analysis for recursive algorithms:

Recursive solution for computing Fibonacci numbers seems more clever and compact; however the iterative solution is much more efficient!!!

Let T(n) be the time complexity of the recursive solution (in terms of comparison and arithmetic operations).

Then T(n) = T(n–1) + T(n–2) + 5 for $n \geq 2$ with T(0) = 1 and T(1) = 2.

(The term 5 counts for 2 comparisons plus 1 addition plus 2 subtractions).

```
function Fibonacci(n: nonnegative integer)
if (n==0) fibonacci = 0;
else if (n==1) Fibonacci = 1;
else fibonacci = Fibonacci(n-1) + Fibonacci(n-2);
return fibonacci;
```

Complexity analysis for recursive algorithms:

Recursive solution for computing Fibonacci numbers seems more clever and compact; however the iterative solution is much more efficient!!!

Let T(n) be the time complexity of the recursive solution (in terms of comparison and arithmetic operations).

Then $T(n) = T(n-1) + T(n-2) + 5$ for $n \geq 2$ with $T(0) = 1$ and $T(1) = 2$.

(The term 5 counts for 2 comparisons plus 1 addition plus 2 subtractions).

Since $f_n = f_{n-1} + f_{n-2}$, $T(n) \geq f_n \ \forall n$ and

$f_n > \alpha^{n-2} \ \forall n \geq 3$, $\alpha = (1+\sqrt{5})/2$,

$\Rightarrow T(n) > \alpha^{n-2}$

Complexity analysis for recursive algorithms:

Recursive solution for computing Fibonacci numbers seems more clever and compact; however the iterative solution is much more efficient!!!

Let T(n) be the time complexity of the recursive solution (in terms of comparison and arithmetic operations).
Then $T(n) = T(n-1) + T(n-2) + 5$ for $n \geq 2$ with $T(0) = 1$ and $T(1) = 2$.
(The term 5 counts for 2 comparisons plus 1 addition plus 2 subtractions).
Since $f_n = f_{n-1} + f_{n-2}$, $T(n) \geq f_n \ \forall n$ and
$f_n > \alpha^{n-2} \ \forall n \geq 3$, $\alpha = (1+\sqrt{5})/2$,
$\Rightarrow T(n) > \alpha^{n-2}$ $\therefore$ exponential complexity $\Theta(\alpha^n)$ or worse.

(Or you can use big-Omega notation: $T(n)$ is $\Omega(\alpha^n)$.)

Note that complexity of the iterative solution is $\Theta(n)$ (big-Theta).

Moral of the story: Don't compute anything more than once!

Correctness of recursive algorithms:

We can use mathematical induction also to show that a given recursive algorithm produces the correct (i.e., the desired) output.

Consider the recursive factorial example:

```
function factorial(n: positive integer)
if (n == 1)fact = 1;
else fact = n*factorial (n-1);
return fact;
```

Is it correct?

Correctness of recursive algorithms:

We can use mathematical induction also to show that a given recursive algorithm produces the correct (i.e., the desired) output.
Consider the recursive factorial example:

```
function factorial(n: positive integer)
if (n == 1)fact = 1;
else fact = n*factorial (n-1);
return fact;
```

Is it correct?

We have to show that the recursive code works correctly for all positive n, i.e., P(n) is true $\forall n$, where P(n): "The output of the algorithm is n!"

Basis step:

P(1): The output is 1 = 1! (T)

Correctness of recursive algorithms:

We can use mathematical induction also to show that a given recursive algorithm produces the correct (i.e., the desired) output.
Consider the recursive factorial example:

```
function factorial(n: positive integer)
if (n == 1)fact = 1;
else fact = n*factorial (n-1);
return fact;
```

Is it correct?

We have to show that the recursive code works correctly for all positive n, i.e., P(n) is true $\forall n$, where P(n): "The output of the algorithm is n!"

Basis step:

P(1): The output is 1 = 1! (T)

Inductive step:

Show $P(n) \rightarrow P(n+1)\ \forall n > 0$ or equivalently show $P(n-1) \rightarrow P(n)\ \forall n > 1$.

We assume `factorial(n-1)` correctly returns $(n-1)!$ and show `factorial(n)` returns $n!$.

Correctness of recursive algorithms:

We can use mathematical induction also to show that a given recursive algorithm produces the correct (i.e., the desired) output.
Consider the recursive factorial example:

```
function factorial(n: positive integer)
if (n == 1)fact = 1;
else fact = n*factorial (n-1);
return fact;
```

Is it correct?

We have to show that the recursive code works correctly for all positive n, i.e., P(n) is true $\forall n$, where P(n): "The output of the algorithm is n!"

Basis step:

P(1): The output is $1 = 1!$ (T)

Inductive step:

Show $P(n) \rightarrow P(n+1)\ \forall n > 0$ or equivalently show $P(n-1) \rightarrow P(n)\ \forall n > 1$.

We assume `factorial(n-1)` correctly returns $(n-1)!$ and show `factorial(n)` returns $n!$.

For n, the output is, as seen from the code, $n\cdot$ `factorial(n-1)` $= n\cdot(n-1)! = n!$

$\therefore$ P(n) is T $\therefore$ P(n) is true $\forall n \in Z^+$

e.g. Show that the following recursive function correctly computes the n-th Fibonacci number (Exercise: use **strong** induction).

```
function Fibonacci(n: nonnegative integer)
if (n==0) fibonacci=0;
else if (n==1) fibonacci=1;
else fibonacci=Fibonacci(n-1)+Fibonacci(n-2);
return fibonacci;
```

e.g. Show that the following recursive function correctly returns gcd(a, n), $a \le n$, for all positive integers n.

```
int GCD(int a, int n){
    int gcdAB;
    if (a == 0) gcdAB = n;
    else gcdAB = GCD(n % a, a);
    return gcdAB;
}
```

Let $P(n)$: The function GCD(a,n) correctly computes gcd(a, n) when $a \le n$, $\forall n>0$.
Basis step: $P(1)$ is true because the only possible cases are $a = 0$ and $a = 1$. Looking into the code, we see that the algorithm returns 0 in the first case and returns $n = 1$ in the latter, as expected.

e.g. Show that the following recursive function correctly returns gcd(a, n), $a \leq n,$ for all positive integers n.

```
int GCD(int a, int n){
    int gcdAB;
    if (a == 0) gcdAB = n;
    else gcdAB = GCD(n % a, a);
    return gcdAB;
}
```

Let $P(n)$: The function GCD(a,n) correctly computes gcd(a, n) when $a \leq n$, $\forall n>0$.
<u>Basis step</u>: $P(1)$ is true because the only possible cases are $a = 0$ and $a = 1$. Looking into the code, we see that the algorithm returns 0 in the first case and returns $n = 1$ in the latter, as expected.
<u>Inductive step:</u> Assume $P(k)$ for all $1 \leq k \leq n$ and check whether GCD(a, n+1) returns gcd(a, n+1) for any $a \leq n$.

e.g. Show that the following recursive function correctly returns gcd(a, n), $a \le n,$ for all positive integers n.

```
int GCD(int a, int n){
    int gcdAB;
    if (a == 0) gcdAB = n;
    else gcdAB = GCD(n % a, a);
    return gcdAB;
}
```

Let $P(n)$: The function GCD(a,n) correctly computes gcd(a, n) when $a \le n$, $\forall n>0$.
<u>Basis step</u>: $P(1)$ is true because the only possible cases are $a = 0$ and $a = 1$. Looking into the code, we see that the algorithm returns 0 in the first case and returns $n = 1$ in the latter, as expected.
<u>Inductive step:</u> Assume $P(k)$ for all $1 \le k \le n$ and check whether GCD(a, n+1) returns gcd(a, n+1) for any $a \le n$. We see that in this case the function calls itself with input (c, a) where $c = (n+1)\%a$, and this call generates gcd((n+1)% a, a) correctly because $P(a)$ is true by inductive hypothesis since $a \le n$.

e.g. Show that the following recursive function correctly returns gcd(a, n), $a \leq n$, for all positive integers n.

```
int GCD(int a, int n){
    int gcdAB;
    if (a == 0) gcdAB = n;
    else gcdAB = GCD(n % a, a);
    return gcdAB;
}
```

Let $P(n)$: The function GCD(a,n) correctly computes gcd(a, n) when $a \leq n$, $\forall n>0$.
Basis step: $P(1)$ is true because the only possible cases are $a = 0$ and $a = 1$. Looking into the code, we see that the algorithm returns 0 in the first case and returns $n = 1$ in the latter, as expected.
Inductive step: Assume $P(k)$ for all $1 \leq k \leq n$ and check whether GCD(a, n+1) returns gcd(a, n+1) for any $a \leq n$. We see that in this case the function calls itself with input (c, a) where $c = (n+1)\%a$, and this call generates gcd((n+1)% a, a) correctly because $P(a)$ is true by inductive hypothesis since $a \leq n$. Since gcd(a, n+1) = gcd((n+1)% a, a) from what we've learned in number theory, GCD(a, n+1) returns gcd(a, n+1). (Note also that $a = n+1$ case is trivial to check). So by strong induction we conclude that $P(n)$ is true for all n, that is, GCD(a, n) returns gcd(a, n) whenever $a \leq n$, $\forall n>0$.