Induction and Recursion
(Sections 4.1-4.3)
[Section 4.4 optional]

Based on Rosen and slides by K. Busch

Induction

Induction is a very useful proof technique

In computer science, induction is used to prove properties of algorithms

Induction and recursion are closely related
- Recursion is a description method for algorithms
- Induction is a proof method suitable for recursive algorithms
Use induction to prove that a proposition $P(n)$ is true:

**Inductive Basis:** Prove that $P(1)$ is true

**Inductive Hypothesis:** Assume $P(k)$ is true
(for any positive integer $k$)

**Inductive Step:** Prove that $P(k + 1)$ is true

In other words in inductive step we prove:

$$P(k) \rightarrow P(k + 1)$$

for every positive integer $k$
Inductive basis

\[ P(1) \]

True

Inductive Step

\[ P(k) \rightarrow P(k + 1) \]

True

Proposition true for all positive integers

\[ P(1) \rightarrow P(2) \rightarrow P(3) \rightarrow P(4) \rightarrow \cdots \]

Induction as a rule of inference:

\[ [P(1) \land \forall k(P(k) \rightarrow P(k + 1))] \rightarrow \forall nP(n) \]
Theorem: \( P(n): 1 + 2 + 3 + \cdots + n = \frac{n(n + 1)}{2} \)

Proof:

Inductive Basis: \( P(1): 1 = \frac{1(1+1)}{2} \)

Inductive Hypothesis: assume that it holds \( P(k): 1 + 2 + \cdots k = \frac{k(k + 1)}{2} \)

Inductive Step: We will prove \( P(k+1): 1 + 2 + \cdots k + (k+1) = \frac{(k+1)((k+1) + 1)}{2} \)

Inductive Step:

\[
P(k + 1): 1 + 2 + \cdots k + (k + 1) = \frac{k(k + 1) + 2(k + 1)}{2} = \frac{(k + 1)((k + 1) + 1)}{2}
\]

End of Proof
Harmonic numbers

\[ H_j = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{j} \]

\[ j = 1, 2, 3, \ldots \]

Example: \[ H_4 = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} = \frac{25}{12} \]

Theorem: \[ H_{2^n} \geq 1 + \frac{n}{2} \quad n \geq 0 \]

Proof:

Inductive Basis: \( n = 0 \)

\[ H_{2^n} = H_{2^0} = H_1 = 1 = 1 + \frac{0}{2} = 1 + \frac{n}{2} \]
Inductive Hypothesis: $n = k$

Suppose it holds: $H_{2^k} \geq 1 + \frac{k}{2}$

Inductive Step: $n = k + 1$

We will show: $H_{2^{k+1}} \geq 1 + \frac{k+1}{2}$

$$H_{2^{k+1}} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{2^k} + \frac{1}{2^k+1} + \cdots + \frac{1}{2^{k+1}}$$

$$\geq H_{2^k} + \frac{1}{2^k+1} + \cdots + \frac{1}{2^{k+1}}$$

from inductive hypothesis

$$\geq \left(1 + \frac{k}{2}\right) + 2^k \cdot \frac{1}{2^{k+1}}$$

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

$$= 1 + \frac{k+1}{2}$$

End of Proof
Theorem: \( H_{2^n} \leq 1 + n \quad n \geq 0 \)

Proof:

Inductive Basis: \( n = 0 \)

\[
H_{2^0} = H_0 = H_1 = 1 = 1 + 0 = 1 + n
\]

Inductive Hypothesis: \( n = k \)

Suppose it holds: \( H_{2^k} \leq 1 + k \)

Inductive Step: \( n = k + 1 \)

We will show: \( H_{2^{k+1}} \leq 1 + (k + 1) \)
\[ H_{2^{k+1}} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{2^k} + \frac{1}{2^k + 1} + \cdots + \frac{1}{2^{k+1}} \]

\[ = H_{2^k} + \frac{1}{2^k + 1} + \cdots + \frac{1}{2^{k+1}} \]

\[ \leq (1 + k) + \frac{1}{2^k + 1} + \cdots + \frac{1}{2^{k+1}} \]

\[ \leq (1 + k) + 2^k \cdot \frac{1}{2^k + 1} \]

\[ \leq (1 + k) + 1 \]

\[ = 1 + (k + 1) \]

**End of Proof**

---

**We have shown:**

\[ 1 + \frac{n}{2} \leq H_{2^n} \leq 1 + n \]

\[ 1 + \frac{\lfloor \log k \rfloor}{2} \leq H_k \leq 1 + \lceil \log k \rceil \]

\[ H_k \approx \log k \]

(for large \( k \))
Theorem: Every $2^n \times 2^n$, $n \geq 1$ checkerboard with one square removed can be tiled with triominoes.

Proof: Inductive Basis: $n = 1$
Inductive Hypothesis: \( n = k \)
Assume that a \( 2^k \times 2^k \) checkerboard can be tiled with the hole anywhere.

Inductive Step: \( n = k + 1 \)
By inductive hypothesis $2^k \times 2^k$ squares with a hole can be tiled.

$$2^k \times 2^k \quad 2^k \times 2^k$$

add three artificial holes

$2^3 \times 2^3$ case:
Replace the three holes with a triomino
Now, the whole area can be tiled

2\(^3\) \times 2\(^3\) case:

End of Proof
Strong Induction

To prove $P(n)$:

**Inductive Basis:** Prove that $P(1)$ is true

**Inductive Hypothesis:**
Assume $P(1) \land P(2) \land \cdots \land P(k)$ is true

**Inductive Step:** Prove that $P(k + 1)$ is true

---

**Theorem:** Every integer $n \geq 2$

is a product of primes

(Fundamental Theorem of Arithmetic)

**Proof:** (Strong Induction)

**Inductive Basis:** $n = 2$

Number 2 is a prime

**Inductive Hypothesis:** $2 \leq n \leq k$

Suppose that every integer between 2 and $k$ is a product of primes
Inductive Step: \( n = k + 1 \)

If \( k + 1 \) is prime then the proof is finished

If \( k + 1 \) is not a prime then it is composite:

\[
k + 1 = a \cdot b \quad 2 \leq a, b \leq k
\]

By the inductive hypothesis:

\[
i, j \geq 1
a = p_1 p_2 \cdots p_i \quad \Rightarrow \quad k + 1 = a \cdot b = p_1 \cdots p_i q_1 \cdots q_j
\]

primes

End of Proof
Theorem: Every postage amount \( n \geq 12 \) can be generated by using 4-cent and 5-cent stamps.

Proof: (Strong Induction)

Inductive Basis: We examine four cases (because of the inductive step)

- \( n = 12 = 4 + 4 + 4 \)
- \( n = 14 = 5 + 5 + 4 \)
- \( n = 13 = 4 + 4 + 5 \)
- \( n = 15 = 5 + 5 + 5 \)

Inductive Hypothesis: \( 12 \leq n \leq k \)

Assume that every postage amount between 12 and \( k \) can be generated by using 4-cent and 5-cent stamps

\[ n = a \cdot 4 + b \cdot 5 \]

Inductive Step: \( n = k + 1 \)

If \( 12 \leq k \leq 14 \) then the inductive step follows directly from inductive basis.
Consider: \( k \geq 15 \) \quad \Rightarrow \quad k + 1 = (k - 3) + 4

\[
12 \leq (k - 3) \leq k
\]

Inductive hypothesis

\[
(k - 3) = a' \cdot 4 + b' \cdot 5
\]

\[
k + 1 = (k - 3) + 4 = (a' + 1) \cdot 4 + b' \cdot 5
\]

End of Proof

---

Recursion

Recursion is used to describe functions, sets, algorithms

**Example:** Factorial function \( f(n) = n! \)

**Recursive Basis:** \( f(0) = 1 \)

**Recursive Step:** \( f(n+1) = (n+1) \cdot f(n) \)
Recursive algorithm for factorial

\[
\text{factorial}(n) \{
\text{if } n = 1 \text{ then} \quad \text{ //recursive basis}
\text{ return 1}
\text{ else} \quad \quad \text{ //recursive step}
\text{ return } n \cdot \text{factorial}(n-1)
\}
\]

Fibonacci numbers

\[f_0, f_1, f_2, f_3, \ldots\]

Recursive Basis: \[f_0 = 0, \quad f_1 = 1\]

Recursive Step: \[f_n = f_{n-1} + f_{n-2}\]

\[n = 2, 3, 4, \ldots\]
\[ f_0 = 0 \]
\[ f_1 = 1 \]
\[ f_2 = f_1 + f_0 = 1 + 0 = 1 \]
\[ f_3 = f_2 + f_1 = 1 + 1 = 2 \]
\[ f_4 = f_3 + f_2 = 2 + 1 = 3 \]
\[ f_5 = f_4 + f_3 = 3 + 2 = 5 \]
\[ f_6 = f_5 + f_4 = 5 + 3 = 8 \]
\[ f_7 = f_6 + f_5 = 8 + 5 = 13 \]
\[ \vdots \]

**Recursive algorithm for Fibonacci function**

```c
fibonacci(n) {
    if \( n \in \{0,1\} \) then //recursive basis
        return n
    else //recursive step
        return fibonacci(n-1) + fibonacci(n-2)
}
```
Iterative algorithm for Fibonacci function

\[
\text{fibonacci}(n) \{ \\
\quad \text{if } n = 0 \text{ then } y \leftarrow 0 \\
\quad \text{else} \{ \\
\qquad x \leftarrow 0 \\
\qquad y \leftarrow 1 \\
\qquad \text{for } i \leftarrow 1 \text{ to } n-1 \text{ do } \{ \\
\qquad\quad z \leftarrow x + y \\
\qquad\quad x \leftarrow y \\
\qquad\quad y \leftarrow z \\
\qquad \} \\
\quad \text{return } y \\
\}\}
\]

Theorem: \( f_n > \delta^{n-2} \) for \( n \geq 3 \)

\[
\delta = \frac{1 + \sqrt{5}}{2} \quad \text{(golden ratio)}
\]

Proof: Proof by (strong) induction

Inductive Basis: \( n = 3 \) \quad \( n = 4 \)

\[
f_3 = 2 > \delta \\
f_4 = 3 > \delta^2
\]
Inductive Hypothesis: \( 3 \leq n \leq k \)

Suppose it holds \( f_n > \delta^{n-2} \)

Inductive Step: \( n = k + 1 \)

We will prove \( f_{k+1} > \delta^{(k-1)} \) for \( 4 \leq k \)

\( \delta \) \text{ is the solution to equation } \( x^2 - x - 1 = 0 \)

\[ \delta^2 = \delta + 1 \]

\[ \delta^{k-1} = \delta^2 \cdot \delta^{k-3} = (\delta + 1)\delta^{k-3} = \delta^{k-2} + \delta^{k-3} \]

\[ f_{k+1} = f_k + f_{k-1} \geq \delta^{k-2} + \delta^{k-3} = \delta^{k-1} \]

induction hypothesis

End of Proof
Euclidean Algorithm for Greatest Common Divisor

Recursive Basis: \( \gcd(a, 0) = a \)

Recursive Step: \( \gcd(a, b) = \gcd(b, a \mod b) \)

\( a > b \)

Recursive Euclidean algorithm for greatest common divisor

\[
\text{gcd}(a, b) \{ \\
\text{if } b = 0 \text{ then } \text{return } a \\
\text{else } \text{return } \gcd(b, a \mod b) \\
\}
\]
Algorithm Mergesort

\[
\begin{array}{c}
8 & 2 & 4 & 6 & 9 & 7 & 10 & 1 & 5 & 3 \\
\end{array}
\]

\[
\begin{array}{c}
split \\
8 & 2 & 4 & 6 & 9 & 7 & 10 & 1 & 5 & 3 \\
sort \\
2 & 4 & 6 & 8 & 9 & 1 & 3 & 5 & 7 & 10 \\
merge \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
\]

\[
\begin{array}{c}
sort(a_1,a_2,\ldots,a_n) \\
\text{if } n > 1 \text{ then } \\
\text{ } \\
\text{ } \\
\text{ } \\
\text{ } \\
\text{merge}(A,B) \\
\text{else return } a_1
\end{array}
\]
Input values of recursive calls

Input and output values of merging
merge( A, B ) { // two sorted lists
    L ← ∅
    while A ≠ ∅ and B ≠ ∅ do {
        Remove smaller first element of A, B from its list and insert it to L
    }
    if A ≠ ∅ or B ≠ ∅ then {
        append remaining elements to L
    }
    return L
}

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>L</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 4 8</td>
<td>6 9</td>
<td>2</td>
<td>2&lt;6</td>
</tr>
<tr>
<td>4 8</td>
<td>6 9</td>
<td>2 4</td>
<td>4&lt;6</td>
</tr>
<tr>
<td>8</td>
<td>6 9</td>
<td>2 4 6</td>
<td>6&lt;8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>3 4 6 8</td>
<td>8&lt;9</td>
</tr>
<tr>
<td>9</td>
<td>2 4 6 8 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The total number of comparisons to merge two lists $A, B$ is at most:

$$\# \text{comparisons} \leq |A| + |B|$$

**Merged size**

- Length of $A$
- Length of $B$

---

**Recursive invocation tree**

Assume $n = 2^k$
Recursive invocation tree

Elements per list

\[ n = 2^{\log n - 0} \]
\[ n / 2 = 2^{\log n - 1} \]
\[ n / 4 = 2^{\log n - 2} \]
\[ \vdots \]
\[ 4 = 2^{\log n - (\log n - 2)} \]
\[ 2 = 2^{\log n - (\log n - 1)} \]
\[ 1 = 2^{\log n - \log n} \]

Assume
\[ n = 2^k \]

#levels of tree = \(1 + \log n\)

merging tree
merging tree

Elements per list

\[ n \]

\[ \frac{n}{2} \]

\[ \frac{n}{4} \]

\[ \frac{n}{4} \]

\[ \frac{n}{4} \]

\[ \frac{n}{4} \]

\[ \frac{n}{4} \]

\[ \vdots \]

\[ 4 \]

\[ 4 \]

\[ 4 \]

\[ 4 \]

\[ 1 \]

\[ 1 \]

\[ 1 \]

\[ 1 \]

\[ 1 \]

\[ 1 \]

\[ 1 \]

Total cost: \( n \cdot (#\text{levels} - 1) = n \log n \)
If $n = 2^k$ the number of comparisons is at most $n \log n$

If $n \neq 2^k$ the number of comparisons is at most $m \log m < 2n \log 2n < c \cdot n \log n$

where $m = 2^{[\log n]} < 2n$

Therefore, worst-case running time of merge sort is $\approx n \log n$