Chapter 3. The Fundamentals: Algorithms, the Integers, and Matrices

3.1 Sorting. Greedy Algorithms.

3.1 Sorting

Ordering the elements of a list is a problem that occurs in many contexts.

sorting is putting elements into a list in which the elements are in increasing (or decreasing) order.

3.1 Sorting

Ordering the elements of a list is a problem that occurs in many contexts.

sorting is putting elements into a list in which the elements are in increasing (or decreasing) order.

Example 1:
Given a list {1, 5, 2, 7, 3, 4}, the sorted list will be {1, 2, 3, 4, 5, 7}

Given a list {a, g, s, d, f, p} the sorted list will be {a, d, f, g, p, s}

3.1 Sorting

Ordering the elements of a list is a problem that occurs in many contexts.

sorting is putting elements into a list in which the elements are in increasing (or decreasing) order.

Example 1:
Given a list {1, 5, 2, 7, 3, 4}, the sorted list will be {1, 2, 3, 4, 5, 7}

Given a list {a, g, s, d, f, p} the sorted list will be {a, d, f, g, p, s}

There are many sorting algorithms. Some algorithms are easy to implement, some a more efficient, some take advantage of particular computer architecture, and so on.

Some of the names:
Bubble sort
Insertion sort
Merge sort
Selection sort
Quicksort

3.1 Bubble sort

Let's consider Bubble sort.
It is a simplest one, but not an efficient algorithm

idea: compares adjacent elements and interchanges them if necessary

3.1 Bubble sort

Let's consider Bubble sort.
It is a simplest one, but not an efficient algorithm

idea: compares adjacent elements and interchanges them if necessary

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)
for $i := 1$ **to** $n-1$
    for $j := 1$ **to** $n-i$
        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
$\{a_1, a_2, \ldots, a_n$ is in increasing order$\}$

3.1 Bubble sort

Let's consider Bubble sort.
It is a simplest one, but not an efficient algorithm

idea: compares adjacent elements and interchanges them if necessary

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)
for $i := 1$ **to** $n-1$
  for $j := 1$ **to** $n-i$
    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
{$a_1, a_2, \ldots, a_n$ is in increasing order}

summary: the bubble sort is done in n-1 passes.
During *each pass* we start at the beginning of the list and compare first and second elements: if the first element is larger that the second – we interchange them, and do nothing otherwise. Then we compare the second and the third elements (and interchange them if the second element is larger than the third one). And so on – till we reach the end of the list.

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

First pass (i=1):

a_5 0

a_4 5

a_3 7

a_2 1

a_1 3

j=1

$a_1 > a_2$?

3 > 1 ?

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

    for $j := 1$ **to** $n-i$

        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

First pass (i=1):

a_5	0	0
a_4	5	5
a_3	7	7
a_2	1	3
a_1	3	1
	j=1	j=2
	$a_1 > a_2$?	$a_2 > a_3$?
	3 > 1 ?	3 > 7 ?

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

  for $j := 1$ **to** $n-i$

    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

First pass (i=1):

a_5	0	0	0
a_4	5	5	5
a_3	7	7	7
a_2	1	3	3
a_1	3	1	1
	j=1	j=2	j=3
	$a_1 > a_2$?	$a_2 > a_3$?	$a_3 > a_4$?
	3 > 1 ?	3 > 7 ?	7 > 5 ?

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

  for $j := 1$ **to** $n-i$

    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

First pass (i=1):

a_5	0	0	0	0
a_4	5	5	5	7
a_3	7	7	7	5
a_2	1	3	3	3
a_1	3	1	1	1
	j=1	j=2	j=3	j=4=n-i
	$a_1 > a_2$?	$a_2 > a_3$?	$a_3 > a_4$?	$a_3 > a_4$?
	3 > 1 ?	3 > 7 ?	7 > 5 ?	7 > 5 ?

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

  for $j := 1$ **to** $n-i$

    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

First pass (i=1):

a_5	0	0	0	0	7
a_4	5	5	5	7	0
a_3	7	7	7	5	5
a_2	1	3	3	3	3
a_1	3	1	1	1	1
	j=1	j=2	j=3	j=4=n-i	
	$a_1 > a_2$?	$a_2 > a_3$?	$a_3 > a_4$?	$a_3 > a_4$?	
	3 > 1 ?	3 > 7 ?	7 > 5 ?	7 > 5 ?	

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

    for $j := 1$ **to** $n-i$

        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

Second pass (i=2):

a_5 7

a_4 0

a_3 5

a_2 3

a_1 1

j=1

$a_1 > a_2$?

1 > 3 ?

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

  for $j := 1$ **to** $n-i$

    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, \ldots, a_n$ is in increasing order}

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

Second pass (i=2):

	j=1	j=2
a_5	7	7
a_4	0	0
a_3	5	5
a_2	3	3
a_1	1	1
	$a_1 > a_2$?	$a_2 > a_3$?
	1 > 3 ?	3 > 5 ?

procedure *bubblesort*($a_1,...,a_n$:real numbers with $n \geq 2$)
for $i := 1$ **to** $n-1$
    for $j := 1$ **to** $n-i$
        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
{$a_1, a_2, ..., a_n$ is in increasing order}

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

Second pass (i=2):

a_5	7	7	7
a_4	0	0	0
a_3	5	5	5
a_2	3	3	3
a_1	1	1	1
	j=1	j=2	j=3=n-i
	$a_1 > a_2$?	$a_2 > a_3$?	$a_3 > a_4$?
	1 > 3 ?	3 > 5 ?	5 > 0 ?

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

  for $j := 1$ **to** $n-i$

    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

Second pass (i=2):

a_5	7	7	7	7
a_4	0	0	0	5
a_3	5	5	5	0
a_2	3	3	3	3
a_1	1	1	1	1
	j=1	j=2	j=3=n-i	
	$a_1 > a_2$?	$a_2 > a_3$?	$a_3 > a_4$?	
	1 > 3 ?	3 > 5 ?	5 > 0 ?	

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

    for $j := 1$ **to** $n-i$

        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, \ldots, a_n$ is in increasing order}

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

Third pass (i=3):

a_5	7
a_4	5
a_3	0
a_2	3
a_1	1

j=1

$a_1 > a_2$?

1 > 3 ?

procedure *bubblesort*($a_1,...,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

  for $j := 1$ **to** $n-i$

    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{a_1, a_2, ..., a_n is in increasing order}

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

Third pass (i=3):

a_5	7	7
a_4	5	5
a_3	0	0
a_2	3	3
a_1	1	1
	j=1	j=2=n-i
	$a_1 > a_2$?	$a_2 > a_3$?
	1 > 3 ?	3 > 0 ?

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

  for $j := 1$ **to** $n-i$

    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

Third pass (i=3):

a_5	7	7	7
a_4	5	5	5
a_3	0	0	3
a_2	3	3	0
a_1	1	1	1
	j=1	j=2=n-i	
	$a_1 > a_2$?	$a_2 > a_3$?	
	1 > 3 ?	3 > 0 ?	

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

  for $j := 1$ **to** $n-i$

    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

$\{a_1, a_2, \ldots, a_n$ is in increasing order$\}$

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

Fourth pass (i=4):

a_5 7

a_4 5

a_3 3

a_2 0

a_1 1

j=1=n-i

$a_1 > a_2$?

1 > 0 ?

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

  for $j := 1$ **to** $n-i$

    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

Fourth pass (i=4):

a_5	7	7
a_4	5	5
a_3	3	3
a_2	0	1
a_1	1	0

j=1=n-i

$a_1 > a_2$?

1 > 0 ?

procedure *bubblesort*($a_1,...,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

    for $j := 1$ **to** $n-i$

        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, ..., a_n$ is in increasing order}

3.1 Bubble sort

Example 2: Let's see the work of the Bubble sort on the list {3, 1, 7, 5, 0}

Fourth pass (i=4):

a_5	7	7	7
a_4	5	5	5
a_3	3	3	3
a_2	0	1	1
a_1	1	0	0

j=1=n-i

$a_1 > a_2$?

1 > 0 ?

Stop

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $i := 1$ **to** $n-1$

  for $j := 1$ **to** $n-i$

    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Bubble sort

procedure *bubblesort*($a_1,...,a_n$:real numbers with $n \geq 2$)
for $i := 1$ **to** $n-1$
    for $j := 1$ **to** $n-i$
        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
{$a_1, a_2, ..., a_n$ is in increasing order}

How many iterations (comparisons) are performed on an n-element list?

3.1 Bubble sort

procedure *bubblesort*($a_1,...,a_n$:real numbers with $n \geq 2$)
for $i := 1$ **to** $n-1$
    for $j := 1$ **to** $n-i$
        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
$\{a_1, a_2, ..., a_n$ is in increasing order$\}$

How many iterations (comparisons) are performed on an n-element list?
for i=1   n-1
for i=2   n-2
for i=3   n-3
....
for i=n-1   n-(n-1)

3.1 Bubble sort

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)
for $i := 1$ **to** $n-1$
    for $j := 1$ **to** $n-i$
        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
$\{a_1, a_2, \ldots, a_n$ is in increasing order$\}$

How many iterations (comparisons) are performed on an n-element list?

for i=1   n-1
for i=2   n-2
for i=3   n-3
....
for i=n-1   n-(n-1)

Therefore we have the following sum:
$(n-1) + (n-2) + (n-3) + (n-4) + \ldots + (n-(n-1)) =$

3.1 Bubble sort

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)
for $i := 1$ **to** $n-1$
    for $j := 1$ **to** $n-i$
        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
$\{a_1, a_2, \ldots, a_n$ is in increasing order$\}$

How many iterations (comparisons) are performed on an n-element list?

for i=1 n-1
for i=2 n-2
for i=3 n-3
....
for i=n-1 n-(n-1)

Therefore we have the following sum:

$$\underbrace{(n-1)}_{i=1} + \underbrace{(n-2)}_{i=2} + \underbrace{(n-3)}_{i=3} + \underbrace{(n-4)}_{i=4} + \ldots + \underbrace{(n-(n-1))}_{i=n-1} = (n-1)*n - (1+2+3+4+\ldots(n-1))=$$

3.1 Bubble sort

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)
for $i := 1$ **to** $n-1$
    for $j := 1$ **to** $n-i$
        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
$\{a_1, a_2, \ldots, a_n$ is in increasing order$\}$

How many iterations (comparisons) are performed on an n-element list?

for i=1   n-1
for i=2   n-2
for i=3   n-3
....
for i=n-1   n-(n-1)

arithmetic progression

Therefore we have the following sum:

$$\underset{i=1}{(n-1)} + \underset{i=2}{(n-2)} + \underset{i=3}{(n-3)} + \underset{i=4}{(n-4)} + \ldots + \underset{i=n-1}{(n-(n-1))} = (n-1)*n - \underbrace{(1+2+3+4+\ldots(n-1))}_{\text{arithmetic progression}}=$$

$$n^2 - n - (1+(n-1)) \cdot \frac{n-1}{2} =$$

3.1 Bubble sort

procedure *bubblesort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)
for $i := 1$ **to** $n-1$
    for $j := 1$ **to** $n-i$
        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
$\{a_1, a_2, \ldots, a_n$ is in increasing order$\}$

How many iterations (comparisons) are performed on an n-element list?

for i=1   n-1
for i=2   n-2
for i=3   n-3
....
for i=n-1   n-(n-1)

arithmetic progression

Therefore we have the following sum:

$$\underbrace{(n-1)}_{i=1} + \underbrace{(n-2)}_{i=2} + \underbrace{(n-3)}_{i=3} + \underbrace{(n-4)}_{i=4} + \ldots + \underbrace{(n-(n-1))}_{i=n-1} = (n-1)*n - (1+2+3+4+\ldots(n-1)) =$$

$$n^2 - n - (1+(n-1)) \cdot \frac{n-1}{2} = n^2/2 - n/2$$ - quadratic

3.1 Insertion sort

Now, let's consider Insertion sort.
It is a simple algorithm, but still not an efficient one usually

3.1 Insertion sort

Now, let's consider Insertion sort.
It is a simple algorithm, but still not an efficient one usually

procedure *insertionsort*($a_1,...,a_n$:real numbers with $n \geq 2$)

for $j := 2$ **to** n

    $i := 1$

    while $a_j > a_i$

        $i := i+1$

    $m := a_j$

    for $k := 0$ **to** $j-i-1$

        $a_{j-k} := a_{j-k-1}$

    $a_i := m$

{$a_1, a_2, ..., a_n$ is in increasing order}

3.1 Insertion sort

Now, let's consider Insertion sort.
It is a simple algorithm, but still not an efficient one usually

procedure *insertionsort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i+1$
    $m := a_j$
    for $k := 0$ **to** $j-i-1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
$\{a_1, a_2, \ldots, a_n$ is in increasing order$\}$

<u>summary</u>: insertion sort starts with the second element.
It compares this element to the first one, and if it is smaller than the first one – inserts it in front of the first one (shifts the first one to the place of the second one), and does nothing otherwise.
Then it takes the third element and compares it with the first one and the second one and inserts it into a correct position (also shifts), if needed. And so on.

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=2:
i=1   7, 0, 3, 2, 6

procedure *insertionsort*($a_1,...,a_n$:real numbers with $n \geq 2$)
for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i+1$
    $m := a_j$
    for $k := 0$ **to** $j-i-1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{$a_1, a_2, ..., a_n$ is in increasing order}

3.1 Insertion sort

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=2:
i=1   7, 0, 3, 2, 6   7, 0, 3, 2, 6
m=0 $a_2=a_1$

procedure *insertionsort*($a_1,...,a_n$:real numbers with $n \geq 2$)
for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i+1$
    $m := a_j$
    for $k := 0$ **to** $j-i-1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
$\{a_1, a_2, ..., a_n$ is in increasing order$\}$

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=2:
i=1 7, 0, 3, 2, 6 7, 0, 3, 2, 6 7, 7, 3, 2, 6
m=0 $a_2=a_1$

procedure *insertionsort*($a_1,...,a_n$:real numbers with $n \geq 2$)
for $j := 2$ **to** n
$i := 1$
while $a_j > a_i$
$i := i+1$
$m := a_j$
for $k := 0$ **to** $j-i-1$
$a_{j-k} := a_{j-k-1}$
$a_i := m$
{$a_1, a_2, ..., a_n$ is in increasing order}

3.1 Insertion sort

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=2:

i=1   7, 0, 3, 2, 6   m=0   7, 0, 3, 2, 6 $a_2=a_1$ 7, 7, 3, 2, 6 $a_1=m=0$ 0, 7, 3, 2, 6

procedure *insertionsort*($a_1,...,a_n$:real numbers with $n \geq 2$)
for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i+1$
    $m := a_j$
    for $k := 0$ **to** $j-i-1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{$a_1, a_2, ..., a_n$ is in increasing order}

3.1 Insertion sort

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=2:
i=1   7, 0, 3, 2, 6   m=0   7, 0, 3, 2, 6 $a_2=a_1$ 7, 7, 3, 2, 6 a_1=m=0   0, 7, 3, 2, 6

j=3:
i=1   0, 7, 3, 2, 6

procedure *insertionsort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)
for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i+1$
    $m := a_j$
    for $k := 0$ **to** $j-i-1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Insertion sort

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=2:
i=1 7, 0, 3, 2, 6 m=0 7, 0, 3, 2, 6 $a_2=a_1$ 7, 7, 3, 2, 6 a_1=m=0 0, 7, 3, 2, 6

j=3:
i=1 0, 7, 3, 2, 6
i=2 m=3

procedure *insertionsort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)
for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i+1$
    $m := a_j$
    for $k := 0$ **to** $j-i-1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Insertion sort

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=2:
i=1   7, 0, 3, 2, 6   7, 0, 3, 2, 6   7, 7, 3, 2, 6   0, 7, 3, 2, 6
m=0 a_2=a_1 a_1=m=0

j=3:
i=1   0, 7, 3, 2, 6   0, 7, 3, 2, 6

procedure *insertionsort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)
for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i+1$
    $m := a_j$
    for $k := 0$ **to** $j-i-1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{$a_1, a_2, \ldots, a_n$ is in increasing order}

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=2:
i=1   7, 0, 3, 2, 6   m=0   7, 0, 3, 2, 6 $a_2=a_1$ 7, 7, 3, 2, 6 $a_1=m=0$ 0, 7, 3, 2, 6

j=3:
i=1   0, 7, 3, 2, 6   i=2   m=3   0, 7, 3, 2, 6 $a_3=a_2$ 0, 7, 7, 2, 6

procedure *insertionsort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)

for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i+1$
    $m := a_j$
    for $k := 0$ **to** $j-i-1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Insertion sort

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=2:

i=1	7, 0, 3, 2, 6	7, 0, 3, 2, 6	7, 7, 3, 2, 6	0, 7, 3, 2, 6
		m=0	$a_2=a_1$	a_1=m=0

j=3:

i=1	0, 7, 3, 2, 6	0, 7, 3, 2, 6	0, 7, 7, 2, 6	0, 3, 7, 2, 6
	i=2	m=3	$a_3=a_2$	a_2=m=3

procedure *insertionsort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)
for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i+1$
    $m := a_j$
    for $k := 0$ **to** $j-i-1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Insertion sort

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=4:
i=1 0, 3, 7, 2, 6

procedure *insertionsort*($a_1,...,a_n$:real numbers with $n \geq 2$)
for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i+1$
    $m := a_j$
    for $k := 0$ **to** $j-i-1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{$a_1, a_2, ..., a_n$ is in increasing order}

3.1 Insertion sort

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=4:

2^{nd} 1^{st}

i=1 0, 3, 7, 2, 6 0, 3, 7, 2, 6

i=2 m=2 $a_4=a_3, a_3=a_2$

procedure *insertionsort*($a_1,...,a_n$:real numbers with $n \geq 2$)

for $j := 2$ **to** n

  $i := 1$

  while $a_j > a_i$

    $i := i+1$

  $m := a_j$

  for $k := 0$ **to** $j-i-1$

    $a_{j-k} := a_{j-k-1}$

  $a_i := m$

{$a_1, a_2, ..., a_n$ is in increasing order}

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=4:

2nd 1st

i=1 0, 3, 7, 2, 6 0, 3, 7, 2, 6 0, 3, 3, 7, 6

i=2 m=2 $a_4=a_3, a_3=a_2$

procedure *insertionsort*($a_1,\dots,a_n$:real numbers with $n \geq 2$)

for $j := 2$ **to** n

$i := 1$

while $a_j > a_i$

$i := i+1$

$m := a_j$

for $k := 0$ **to** $j-i-1$

$a_{j-k} := a_{j-k-1}$

$a_i := m$

$\{a_1, a_2, \dots, a_n$ is in increasing order$\}$

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=4:
i=1 0, 3, 7, 2, 6 0, 3, 7, 2, 6 0, 3, 3, 7, 6 0, 2, 3, 7, 6
i=2 m=2 $a_4=a_3, a_3=a_2$ $a_2=m=2$

2nd 1st

j=5:
i=1 0, 2, 3, 7, 6

procedure *insertionsort*($a_1,\ldots,a_n$:real numbers with $n \geq 2$)
for $j := 2$ **to** n
  $i := 1$
  while $a_j > a_i$
    $i := i+1$
  $m := a_j$
  for $k := 0$ **to** $j-i-1$
    $a_{j-k} := a_{j-k-1}$
  $a_i := m$
{$a_1, a_2, \ldots, a_n$ is in increasing order}

3.1 Insertion sort

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=4:

2nd 1st

i=1	0, 3, 7, 2, 6	0, 3, 7, 2, 6	0, 3, 3, 7, 6	0, 2, 3, 7, 6
	i=2	m=2	$a_4=a_3, a_3=a_2$	$a_2=m=2$

j=5:

i=1	0, 2, 3, 7, 6	0, 2, 3, 7, 6	
	i=4	m=6	$a_5=a_4$

procedure *insertionsort*($a_1,\dots,a_n$:real numbers with $n \geq 2$)

for $j := 2$ **to** n

  $i := 1$

  while $a_j > a_i$

    $i := i+1$

  $m := a_j$

  for $k := 0$ **to** $j-i-1$

    $a_{j-k} := a_{j-k-1}$

  $a_i := m$

{$a_1, a_2, \dots, a_n$ is in increasing order}

3.1 Insertion sort

Example 3:
Let's see how insertion sort works on the list {7, 0, 3, 2, 6}

j=4:

i=1	0, 3, 7, 2, 6	0, 3, 7, 2, 6 (2nd, 1st)	0, 3, 3, 7, 6	0, 2, 3, 7, 6
	i=2	m=2	$a_4=a_3, a_3=a_2$	a_2=m=2

j=5:

i=1	0, 2, 3, 7, 6	0, 2, 3, 7, 6	0, 2, 3, 7, 7	0, 2, 3, 6, 7
	i=4	m=6	$a_5=a_4$	a_4=m=6

procedure *insertionsort*($a_1,...,a_n$:real numbers with $n \geq 2$)

for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i+1$
    $m := a_j$
    for $k := 0$ **to** $j-i-1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$

{$a_1, a_2, ..., a_n$ is in increasing order}

3.1 Greedy Algorithms

optimization problem – is a computational problem in which the goal is to find the "best" of all possible solutions.

"best" is different from problem to problem, for example:

- find a shortest route from city A to city B
- find a fastest route from city A to city B

3.1 Greedy Algorithms

optimization problem – is a computational problem in which the goal is to find the "best" of all possible solutions.

"best" is different from problem to problem, for example:
- find a shortest route from city A to city B
- find a fastest route from city A to city B

A simplest approach: *select the "best" choice at each step*

3.1 Greedy Algorithms

optimization problem – is a computational problem in which the goal is to find the "best" of all possible solutions.

"best" is different from problem to problem, for example:

- find a shortest route from city A to city B
- find a fastest route from city A to city B

A simplest approach: *select the "best" choice at each step*

Algorithms that make what seems to be the "best" choice at each step are called greedy algorithms.

- they often lead to a solution of optimization problem.

3.1 Greedy Algorithms

optimization problem – is a computational problem in which the goal is to find the "best" of all possible solutions.

"best" is different from problem to problem, for example:

- find a shortest route from city A to city B
- find a fastest route from city A to city B

A simplest approach: *select the "best" choice at each step*

Algorithms that make what seems to be the "best" choice at each step are called greedy algorithms.

- they often lead to a solution of optimization problem.

Once we know that a greedy algorithm finds a feasible solution, we need to determine whether it has found an *optimal solution*. To do this we:

- prove that the solution is optimal, or
- show that there is a counterexample where the algorithm yields a non-optimal solution.

Example 4:
Make *n* cents change with quarters (*q*), nickels (*c*), dimes (*d*), and pennies (*p*), using the least total number of coins.

Example 4:
Make *n* cents change with quarters (*q*), nickels (*c*), dimes (*d*), and pennies (*p*), using the least total number of coins.

A greedy algorithm:
Let's try to make the a *locally optimal choice* at each step: at each step we choose the coin of largest denomination possible to add to the pile of change without exceeding *n* cents.

Example 4:
Make *n* cents change with quarters (*q*), nickels (*c*), dimes (*d*), and pennies (*p*), using the least total number of coins.

A greedy algorithm:
Let's try to make the a *locally optimal choice* at each step: at each step we choose the coin of largest denomination possible to add to the pile of change without exceeding *n* cents.

procedure *change*(*n*: positive integer; c_1, c_2, c_3, ..., c_r : values of denominations of coins, where $c_1 > c_2 > c_3 > ... > c_r$

for $i := 1$ **to** r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n := n - c_i$
{the pile has change of *n* cents}

Example 4:
Make *n* cents change with quarters (*q*), nickels (*c*), dimes (*d*), and pennies (*p*), using the least total number of coins.

A greedy algorithm:
Let's try to make the a *locally optimal choice* at each step: at each step we choose the coin of largest denomination possible to add to the pile of change without exceeding *n* cents.

procedure *change*(*n*: positive integer; $c_1, c_2, c_3, ..., c_r$: values of denominations of coins, where $c_1 > c_2 > c_3 > ... > c_r$

for *i := 1* **to** r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n := n - c_i$
{the pile has change of *n* cents}

- presented algorithm *leads to an optimal solution* (*solves optimization problem*) in the sense that it uses the least number of coins.

Let's see how the presented algorithm works for $n=85$

procedure *change*(n: positive integer; $c_1, c_2, c_3, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > c_3 > \ldots > c_r$

for $i := 1$ **to** r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n := n - c_i$

{the pile has change of n cents}

3.1 Greedy Algorithms

Let's see how the presented algorithm works for *n=85*

25

Total: 25

procedure *change*(*n*: positive integer; c_1, c_2, c_3, ..., c_r : values of denominations of coins, where $c_1 > c_2 > c_3 > ... > c_r$

for *i := 1* **to** r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n := n - c_i$
{the pile has change of *n* cents}

3.1 Greedy Algorithms

Let's see how the presented algorithm works for $n=85$

25 25

Total: 25 50

procedure *change*(n: positive integer; $c_1, c_2, c_3, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > c_3 > \ldots > c_r$

for $i := 1$ **to** r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n := n-c_i$

{the pile has change of n cents}

3.1 Greedy Algorithms

Let's see how the presented algorithm works for *n=85*

25 25 25

Total: 25 50 75

procedure *change*(*n*: positive integer; c_1, c_2, c_3, ..., c_r : values of denominations of coins, where $c_1 > c_2 > c_3 > ... > c_r$

for *i := 1* **to** r
  while $n \geq c_i$
    add a coin with value c_i to the change
    $n := n - c_i$

{the pile has change of *n* cents}

3.1 Greedy Algorithms

Let's see how the presented algorithm works for *n=85*

25 25 25 10

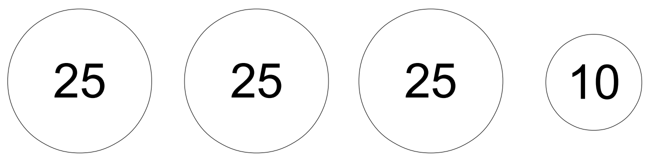
Total: 25 50 75 85

procedure *change*(*n*: positive integer; $c_1, c_2, c_3, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > c_3 > \ldots > c_r$

for *i := 1* **to** r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n := n - c_i$
{the pile has change of *n* cents}

3.1 Greedy Algorithms

Let's see how the presented algorithm works for *n=85*



Total: 25 50 75 85

Let's see how the presented algorithm works for *n=98*

procedure *change*(*n*: positive integer; $c_1, c_2, c_3, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > c_3 > \ldots > c_r$

for *i := 1* **to** r
  while $n \geq c_i$
    add a coin with value c_i to the change
    $n := n - c_i$
{the pile has change of *n* cents}

3.1 Greedy Algorithms

Let's see how the presented algorithm works for *n=85*

25 25 25 10

Total: 25 50 75 85

Let's see how the presented algorithm works for *n=98*

25

Total: 25

procedure *change*(*n*: positive integer; $c_1, c_2, c_3, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > c_3 > \ldots > c_r$

for $i := 1$ **to** r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n := n - c_i$
{the pile has change of *n* cents}

3.1 Greedy Algorithms

Let's see how the presented algorithm works for $n=85$

25 25 25 10

Total: 25 50 75 85

Let's see how the presented algorithm works for $n=98$

25 25

Total: 25 50

procedure *change*(n: positive integer; $c_1, c_2, c_3, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > c_3 > \ldots > c_r$

for $i := 1$ **to** r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n := n - c_i$
{the pile has change of n cents}

3.1 Greedy Algorithms

Let's see how the presented algorithm works for *n=85*

25 25 25 10

Total: 25 50 75 85

Let's see how the presented algorithm works for *n=98*

25 25 25

Total: 25 50 75

procedure *change*(*n*: positive integer; $c_1, c_2, c_3, ..., c_r$: values of denominations of coins, where $c_1 > c_2 > c_3 > ... > c_r$

for *i := 1* **to** r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n := n - c_i$
{the pile has change of *n* cents}

3.1 Greedy Algorithms

Let's see how the presented algorithm works for $n=85$

25 25 25 10

Total: 25 50 75 85

Let's see how the presented algorithm works for $n=98$

25 25 25 10

Total: 25 50 75 85

procedure *change*(n: positive integer; $c_1, c_2, c_3, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > c_3 > \ldots > c_r$

for $i := 1$ **to** r
  while $n \geq c_i$
    add a coin with value c_i to the change
    $n := n - c_i$
{the pile has change of n cents}

3.1 Greedy Algorithms

Let's see how the presented algorithm works for *n=85*

25 25 25 10

Total: 25 50 75 85

Let's see how the presented algorithm works for *n=98*

25 25 25 10 10

Total: 25 50 75 85 95

procedure *change*(*n*: positive integer; $c_1, c_2, c_3, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > c_3 > \ldots > c_r$

for *i := 1* **to** r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n := n - c_i$

{the pile has change of *n* cents}

Let's see how the presented algorithm works for *n=85*

	25	25	25	10
Total:	25	50	75	85

Let's see how the presented algorithm works for *n=98*

	25	25	25	10	10	1	1	1
Total:	25	50	75	85	95	96	97	98

procedure *change*(*n*: positive integer; c_1, c_2, c_3, ..., c_r : values of denominations of coins, where $c_1 > c_2 > c_3 > ... > c_r$

for *i := 1* **to** r

    while $n \geq c_i$

        add a coin with value c_i to the change

        $n := n - c_i$

{the pile has change of *n* cents}

3.1 Greedy Algorithms

procedure *change*(*n*: positive integer; c_1, c_2, c_3, ..., c_r : values of denominations of coins, where $c_1 > c_2 > c_3 > ... > c_r$

for *i* := *1* **to** r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n := n-c_i$
{the pile has change of *n* cents}

- presented algorithm *leads to an optimal solution* (*solves optimization problem*) in the sense that it uses the least number of coins.

It is not enough to present few examples to show that the algorithm leads to an optimal solution. We should present a *proof* (which we won't see in here, but if you are curious – see the book, pages 175-176) .

3.1 Greedy Algorithms

procedure *change*(*n*: positive integer; $c_1, c_2, c_3, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > c_3 > \ldots > c_r$

for *i* := *1* **to** r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n := n - c_i$
{the pile has change of *n* cents}

- presented algorithm *leads to an optimal solution* (*solves optimization problem*) in the sense that it uses the least number of coins.

It is not enough to present few examples to show that the algorithm leads to an optimal solution. We should present a *proof* (which we won't see in here, but if you are curious – see the book, pages 175-176) .

! There are sets of coins (for example, quarters, dimes and pennies) for which the presented greedy algorithm doesn't produce change using the fewest coins possible.