3.1 Algorithms. Pseudocode. Searching Algorithms.

Outline:
- What is an Algorithm?
- Pseudocode
- Search Problems
- Sorting Problems (*next*)

3.1 Algorithms. Pseudocode. Searching Algorithms.

Examples of Mathematical Tasks:
• Find the largest integer in the sequence of integers.
• Given a sequence of integers put them in increasing order.

An algorithm is a *finite* set of precise instructions for performing a computation.

Given a mathematical task, we want to describe an algorithm that carries out the task.

Why would we want to do this?

3.1 Algorithms

Example 1: Describe an algorithm for finding the minimum value in a finite sequence of integers.

Possible Solution (algorithm):

1. Set the temporary minimum value (**min**) equal to the first integer in the sequence

2. Compare the next integer in the sequence to the temporary **min** . If it is smaller than the **min**, set **min** equal to this integer. Do nothing otherwise.

3. Repeat step 2 (if there are more integers in the sequence).

4. Stop when there are no integers left in the sequence. The current value of **min** is the minimum integer in the sequence.

This description of the algorithm is nice, but too many words.

3.1 Algorithms

We can describe an algorithm using a computer language, but then we are limited to the instructions permitted in the language + many programming languages are in common use (undesirable to chose only one)

A better solution: use pseudocode
(intermediate step between English and a programming language)

Example 1 (using pseudocode):

```
procedure $min(a_1,a_2,\ldots,a_n$: integers)
$min := a_1$
for $i := 2$ to $n$
    if $min > a_i$ then $min := a_i$
{$min$ is the smallest element}
```

Assume that there were 12 integers in the sequence.
How many comparison operations will be performed?

How many assignments/re-assignments will be performed?

3.1 Algorithms

Algorithm versus a Run of the algorithm

Algorithm: Set of instructions.

Run: The steps carried out by the algorithm on some specified input

Consider runs of the previous algorithm

Example 1 (using pseudocode):

```
procedure min($a_1, a_2, \ldots, a_n$: integers)
min := $a_1$
for i := 2 to n
    if min > $a_i$ then min := $a_i$
{min is the smallest element}
```

3.1 Algorithms

Properties of algorithms:

• *input*	an algorithm has input values from a specified set (usually)
• *output*	for each set of input values an algorithm produces output values (from a specified set)
• *definiteness*	the steps of the algorithm must be defined precisely
• *correctness*	an algorithm should produce the correct output values for each set of input values
• *finiteness*	an algorithm should produce the desired output after a finite number of steps for any input in the set
• *effectiveness*	it must be possible to perform each step of an algorithm exactly and in a finite amount of time
• *generality*	the procedure should be applicable for all problems of the desired form

3.1 Pseudocode

Our goals:

1) Understand pseudocode.
    a) Run the code and answer questions about the run
    b) Determine the output on specified input
    c) Count the number of steps performed in a run

2) Write pseudocode
    a) Given a task, write code to carry it out
    b) Only use “*standard pseudocode terminology*” unless told otherwise

3.1 Pseudocode

What do the lines of pseudocode do? How many "steps" in a run?

1)
```
x := 9
x := x + 3
```

2)
```
if (x < 0 and y < 0) then sign := -1
```

3)
```
if (x< 0 and y < 0) then sign := -1 else sign := 0
```

4)
```
for k := 3 to 9 by 2
    S := S + k
```

5)
```
S := K
while ( S > 0)
    S := S – 1
```

6)
```
n := 0
for i := 1 to 20
    n := n + i*i
```

3.1 Pseudocode

Write pseudocode to do following:

1) Change x to y if k is positive, and change x to z otherwise.

2) Swap the values of variables x and y.

3) Find the product of the first 1000 positive integers.

4) For a list a_1, …, a_k, find the product of the initial elements, up to and including the first negative number.

3.1 Search Algorithms

The problem of locating an element in an ordered list occurs in many contexts.
For instance, a program that checks the spelling of words, searches them in a dictionary (which is just an ordered list of words)

Problems of this kind are called search problems.

General search problem:
Locate an element x in a list of distinct elements $a_1, \ldots, a_n$, or determine that x is not in the list.

More precisely: If x is not in the list return 0, otherwise return i such that $a_i = x$.

We will study two search algorithms:
The Linear Search (Sequential Search)
The Binary Search

3.1 Linear Search (Sequential Search)

```
procedure linear search(x : integer; $a_1,\ldots,a_n$ : distinct integers)
i := 1
while ($i \le n$ and $x \ne a_i$)
    i : = i + 1
if $i \le n$ then location := i
else location := 0
{location is the index(subscript) of the term that equals x, or 0 if x is not found}
```

Example 2:
Assume that we have the following list **A** = (1, 43, 23, 17, 21, 90).
And we'd like to search for 23 in this list.

Let's see how the algorithm will work:
n = 6

1. i = 1 $\quad 1 \le 6$ and $23 \ne 1 \leftarrow a_1$ $\quad$ **A** = (1, 43, 23, 17, 21, 90)

2. i = 2 $\quad 2 \le 6$ and $23 \ne 43 \leftarrow a_2$ $\quad$ **A** = (1, 43, 23, 17, 21, 90)

3. i = 3 $\quad 3 \le 6$ and $23 = 23 \leftarrow a_3$ $\quad$ **A** = (1, 43, 23, 17, 21, 90)

- condition of while is False

$3 \le 6$ therefore location := 3

3.1 Linear Search (Sequential Search)

```
procedure linear search($x$ : integer; $a_1,\ldots,a_n$ : distinct integers)
$i := 1$
while ($i \le n$ and $x \ne a_i$)
    $i := i + 1$
if $i \le n$ then location := $i$
else location := $0$
{location is the index(subscript) of the term that equals $x$, or $0$ if $x$ is not found}
```

Questions:

1) How many times does the while loop execute? (i.e. How many times do we run its body?)

2) Besides the while loop, how many other steps are there in a run of the algorithm?

3.1 Binary Search

! For this algorithm the elements of the set have to be *ordered from smallest to largest*.

How it works: It proceeds by comparing the element to be located to the middle term in the list. The list is then split into two smaller sublists of the same size (or one less). The appropriate sublist is taken and the comparison of the element to be located and the middle term in the sublist is done. And so on.

```
**procedure** *binary search*($x$ : integer; $a_1,\ldots,a_n$ : increasing integers)
$i := 1$ { $i$ is the left endpoint}
$j := n$ { $j$ is the right endpoint}
while $i < j$
    $m := \lfloor (i+j)/2 \rfloor$
    if $x > a_m$ then $i := m+1$
    else $j := m$
if $x = a_i$ then location := $i$
else location := 0
{location is the index(subscript) of the term that equals $x$, or $0$ if $x$ is not found}
```

3.1 Binary Search

Example 3:
Assume that we have the following list **A** = (1, 43, 23, 17, 21, 90).
And we'd like to search for 23 in this list. Let's use binary search algorithm this time.
! List **A** has to be ordered from smallest to largest.
So let **A** = (1, 17, 21, 23, 43, 90). *n=6*

1. *i = 1, j=6*

2. 1 < 6 ? True

$m = \lfloor (1+6)/2 \rfloor = \lfloor 3.5 \rfloor = 3$

$23 > a_3 = 21$? YES

i = 3+1 = 4

$\mathbf{A} = (\underline{\underset{\overset{\uparrow}{i}}{1}, 17, 21, 23, 43, \underset{\overset{\uparrow}{j}}{90}})$

3. 4 < 6 ? True

$m = \lfloor (4+6)/2 \rfloor = \lfloor 5 \rfloor = 5$

$23 > a_5 = 43$? NO

j = 5

$\mathbf{A} = (1, 17, 21, \underline{\underset{\overset{\uparrow}{i}}{23}, 43, \underset{\overset{\uparrow}{j}}{90}})$

4. 4 < 5 ? True

$m = \lfloor (4+5)/2 \rfloor = \lfloor 4.5 \rfloor = 4$

$23 > a_4 = 23$? NO

j = 4

$\mathbf{A} = (1, 17, 21, \underline{\underset{\overset{\uparrow}{i}}{23}, \underset{\overset{\uparrow}{j}}{43}}, 90)$

5. 4 < 4 ? False

$23 = a_4 (=23)$? YES location = 4

$\mathbf{A} = (1, 17, 21, \underline{\underset{\overset{\uparrow}{i\ j}}{23}}, 43, 90)$

Output: 4 (*23* is the fourth element in the ordered set).

3.1 Binary Search

```
procedure binary search($x$ : integer; $a_1,\ldots,a_n$ : increasing integers)
$i := 1$ { $i$ is the left endpoint}
$j := n$ { $j$ is the right endpoint}
while $i < j$
    $m := \lfloor (i+j)/2 \rfloor$
    if $x > a_m$ then $i := m+1$
    else $j := m$
if $x=a_i$ then location := $i$
else location := $0$
{location is the index(subscript) of the term that equals $x$, or $0$ if $x$ is not found}
```

Questions:

1) How many times does the while loop execute? (i.e. How many times do we run its body?)

2) Besides the while loop, how many other steps are there in a run of the algorithm?

3.1 Algorithms. Pseudocode. Searching Algorithms.

Good problems from book (Section 3.1)

1, 2, 3, 4, 5, 7, 9, 11, 13, 15, 17, 23