3.1 Algorithms (continued)

Sorting

We have already seen that lists are easier to search if they are in order. This is what sorting does, so its an important topic.

A simple sorting algorithm is **bubble sort**. It goes through a list switching pairs that are out of order. This process is repeated until the list is sorted.

Example (1) Show how bubble sort puts

52, 90, 13, 40

into increasing order.

Solution: First pass

52	52	52	52
90	90	13	13
13	13	90	40
40	40	40	$\boxed{90}$

Second pass

52	13	13
13	52	40
40	40	$\boxed{52}$
90	90	$\boxed{90}$

Third pass

13	13
40	$\boxed{40}$
52	$\boxed{52}$
90	$\boxed{90}$

Smaller numbers bubble to top and bigger ones sink. Ordered version 13, 40, 52, 90.

Here is the algorithm in pseudocode.

```
procedure bubble sort (a_1, ..., a_n real numbers)

for i := 1 to n-1
    for j := 1 to n-i
        if a_j > a_{j+1} then switch a_j, a_{j+1}
```

The variable i stores the number of the pass. Then j is used to step through the list on each pass.

A second sorting algorithm is insertion sort. It works by sorting the first part of the list and then inserting the next element in the right place. The right place is found using the linear search algorithm we already saw.

Example ② Show how insertion sort puts 52, 90, 13, 40 into increasing order.

Solution: At the start we say the first number 52 is already "sorted" and insert the 2nd number 90 into this list

$$\boxed{52,}\ 90,\ 13,\ 40$$

Now 52, 90 are sorted and we insert 13

$\boxed{52, 90,}$ 13, 40

Finally insert 40 $\boxed{13, 52, 90,}$ 40

to get 13, 40, 52, 90.

This is more complicated to write in pseudocode. In the outer loop we want to insert a_j into the right place

sorted

$\boxed{a_1, a_2, \ldots, a_{i-1}, a_i, \ldots, a_{j-1}}, a_j, \ldots, a_n$

insert here
if $a_i \geq a_j$

```
procedure insertion sort (a_1,...,a_n: reals)
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
```

The for loop with k moves up the list between a_i and $a_j = m$

$a_i, a_{i+1}, a_{i+2}, \ldots, a_{j-1}, a_j.$

m

Greedy algorithms

Suppose a company is trying to make as much money as possible from its customers within certain constraints. This is called an optimization problem. Suppose they also made an algorithm that solves this problem using 100 steps. This algorithm is called a greedy algorithm if it tries to get the most money possible at each step.

This greedy strategy might not give the optimal (best) solution though.

Example ③ Give a greedy algorithm to make n cents in change using as few coins as possible (out of quarters, dimes, nickels, pennies).

Solution: Step one, start with quarters and use as many as possible. Then use as many dimes as possible for what's left. Step three nickels and step four pennies.

Instead of 25, 10, 5, 1 we could use other size coins

$$c_1 > c_2 > c_3 > \cdots > c_{r-1} > c_r$$

and make n cents in change using them. Let d_1 be the number of size c_1 coins needed, d_2 the number of c_2 coins etc.

procedure change($c_1, \ldots, c_r, n$ positive integers $c_1 > c_2 > \cdots > c_r$)

for $i := 1$ to r
  $d_i := 0$
  while $n \geq c_i$
    $d_i := d_i + 1$
    $n := n - c_i$

Example (4) Show the steps used by the change procedure to make 70 cents in change using coins of size
$$c_1 = 12, \quad c_2 = 7, \quad c_3 = 1.$$

Solution: The procedure starts with $n = 70$, $i = 1$, $d_1 = 0$. In the while loop $n \geq c_1$ is true $(70 \geq 12)$ so
$$d_1 = 0 + 1 = 1$$
and $$n = 70 - 12 = 58$$

The condition for the while loop is still true $(58 \geq 12)$ so
$$d_1 = 1 + 1 = 2$$
$$n = 58 - 12 = 46$$

loop repeats $d_1 = 3$, $n = 34$; $d_1 = 4$, $n = 22$; $d_1 = 5$, $n = 10$; now $n \geq c_1$ false

next $i = 2$, $d_2 = 0$, $n \geq c_2$ $(10 \geq 7)$ true

so $d_2 = 1$, $n = 10 - 7 = 3$, loop ends

finally $i = 3$ and the last loop ends with $n = 0$, $d_3 = 3$.
So $70 = 5 \cdot 12 + 1 \cdot 7 + 3 \cdot 1$ in change.