DCS/100: Procedural Programming

Week 11: Arrays, Sorting and Matrices

Queen Mary, University of London

Last Week

By now you should be able to:

- write programs that process bulk data using arrays
- write programs that search arrays for information
- explain how arrays are declared and used
- write programs using recursion
- implement divide and conquor algorithms
- explain concepts related to recursion and divide and conquor algorithms

Queen Mary
University of London

A program from last week

```
static int sumton(int n)
{ if (n==0)
       return 0;
  else
       return (n + sumton(n-1));
}
```

A program from a previous week

```
for (int i=0; i<length; i++)
{
   if (a[i]==target)
   {   out.writeln(i);
       break;
   }
}
```

This Week: Learning Outcomes

By the end of the week you should be able to:

- explain different ways to sort arrays of data into order
- write programs to sort arrays
- write programs containing matrices
- explain how to declare and use matrices

Arrays

Arrays are used to handle bulk data.
Typical things you do with arrays are:

- apply some operation to a single element
- apply some operation to each element of the array
- find some particular element
- sort the elements into some order

Sorting Arrays

- There are lots of sorting algorithms.
- Which one is best depends on the kind of data you have (size, how nearly sorted, how easy to copy, . . .).
- Some algorithms change the array so that it becomes sorted (sorting in place)
- Others give you a copy of the array, in sorted order.
- Here are a couple of examples of in-place algorithms.
- By using Divide and Conquer we can get very fast sort algorithms

Sorting: bubblesort

Go through the array swapping pairs that are out of order.
Go back and do it again.
Keep going till array sorted.

Sorting: bubblesort

```
boolean sorted=false;

while (!sorted)
{   sorted = true;// array potentially sorted

   //traverse array switching ill-ordered pairs
   for (int i=0; i < length-1; i++)
   {
     if (array[i] > array [i+1])
     { // swap them
       int tmp = array[i+1];
       array[i+1] = array[i];
       array[i] = tmp;

       // array wasn’t sorted
       sorted = false;
     }
   }
}
```

Sorting: insertion sort

Find the smallest element.
Put it in first place.
Find the second smallest element.
Put it in second place.
Keep going.

Sorting: insertion sort

Put another way (recursively):

Find the smallest element.
Put it in first place.
Sort the rest of the array (using insertion sort).

Divide and conquer: Sorting

Suppose you have a long array that you want to sort.
Divide and conquer says:
split it into smaller arrays and sort those
then combine the sorted arrays into the sorted large array.

Divide and Conquer: Mergesort

We could just split the array in the middle. Then we'd end up with two sorted arrays, and we'd have to shuffle them together. This leads to mergesort. Splitting the arrays is trivial, recombining them after sorting takes time.

```
Mergesort:
    split array in two halves
    Mergesort first half
    Mergesort second half
    Shuffle the two sorted halves together
```

Divide and Conquer: Quicksort

In quicksort we also split the array. But we put all the elements smaller than some suitably chosen size at one end, and all the elements bigger at the other first. That way we don’t have to do any shuffling at the end.

```
Quicksort:
    choose an element to partition round
    move elements smaller than it to its left
    move elements bigger than it to its right
    Split the array into two parts around
     the final position of the chosen element

    Quicksort first part
    Quicksort second half
    Put the two sorted halves together
```

Queen Mary
University of London

Quicksort

- Quicksort is a famous algorithm for sorting arrays invented by Tony Hoare.
- It is one of the algorithms most commonly used in real systems.
- It is generally very fast.
- We gave a rational reconstruction, but not its most efficient form.

Matrices

Sometimes the information you want to work with comes in two dimensions or more:

- pixels on a screen
- squares on chess board
- volume elements in space

Then you need a two or three-dimensional form of array.

Matrices

Java handles this by allowing arrays of arrays

```
int matrix[][] = new int[m][n];
```

This lets you store a matrix as an array of rows (or an array of columns).

Matrices

$$\begin{array}{ccccc}
M_{0,0} & M_{0,1} & M_{0,2} & \ldots & M_{0,m} \\
M_{1,0} & M_{1,1} & M_{1,2} & \ldots & M_{1,m} \\
 & \ldots & \ldots & \ldots & \\
M_{n,0} & M_{n,1} & M_{n,2} & \ldots & M_{n,m}
\end{array}$$

```
int M[][] = new int[n][m];
```

Matrices

$$\begin{array}{ccccc} M_{0,0} & M_{0,1} & M_{0,2} & \ldots & M_{0,m} \\ M_{1,0} & M_{1,1} & M_{1,2} & \ldots & M_{1,m} \\ & \ldots & \ldots & \ldots & \\ M_{n,0} & M_{n,1} & M_{n,2} & \ldots & M_{n,m} \end{array}$$

`int M[][] = new int[n][m];`
`M[1][]` is a row

(Row order).

Matrices

$$\begin{array}{ccccc}
M_{0,0} & M_{0,1} & M_{0,2} & \ldots & M_{0,m} \\
M_{1,0} & M_{1,1} & M_{1,2} & \ldots & M_{1,m} \\
 & \ldots & \ldots & \ldots & \\
M_{n,0} & M_{n,1} & M_{n,2} & \ldots & M_{n,m}
\end{array}$$

`int M[][] = new int[m][n];`
`M[][1]` is a column
(Column order).

Processing a matrix

```
int M[][] = new int[n][m];

for (int i=0; i<n; i++)
  for (int j=0; j<m; j++)
    PROCESS (a[i][j]);
```

Processing a matrix

```
int M[][] = new int[n][m];

for (int i=0; i<n; i++)
  for (int j=0; j<m; j++)
    out.writeln(a[i][j]);
```

Queen Mary
University of London

This Week: Learning Outcomes

By the end of the week you should be able to:

- explain different ways to sort arrays of data into order
- write programs to sort arrays
- write programs containing matrices
- explain how to declare and use matrices

Reading Computing Without Computers Chapter 14

Brinch Hansen: Chapter 8: Matricees.