## *Section* 2.2 – Algorithms

### *Introduction*

### *Definition*

An ***algorithm*** is a finite sequence of precise instructions for performing a computation or for solving a problem.

- A program is one type of algorithm
    - All programs are algorithms
    - Not all algorithms are programs!
- Directions to somebody's house is an algorithm
- A recipe for cooking a cake is an algorithm
- The steps to compute the cosine of 90° is an algorithm

### Properties of Algorithms

***Input***: An algorithm has input values from a specified set.

***Output***: From each set of input values an algorithm produces output values from a specified set. The output values are the solution to the problem.

***Definiteness***: The steps of an 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 (but perhaps large) 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, not just for a particular set of input values.

### *Algorithm* 1 – Finding the Maximum Element in a Finite Sequence

Given a list, how do we find the maximum element in the list?

To express the algorithm, we'll use pseudocode

- ✓ Pseudocode is kinda like a programming language, but not really

## *Section* 2.2 – Algorithms

### *Introduction*

### *Definition*

An ***algorithm*** is a finite sequence of precise instructions for performing a computation or for solving a problem.

- ➢ A program is one type of algorithm
    - o All programs are algorithms
    - o Not all algorithms are programs!
- ➢ Directions to somebody's house is an algorithm
- ➢ A recipe for cooking a cake is an algorithm
- ➢ The steps to compute the cosine of 90° is an algorithm

### Properties of Algorithms

***Input***: An algorithm has input values from a specified set.

***Output***: From each set of input values an algorithm produces output values from a specified set. The output values are the solution to the problem.

***Definiteness***: The steps of an 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 (but perhaps large) 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, not just for a particular set of input values.

### *Algorithm* 1 – Finding the Maximum Element in a Finite Sequence

Given a list, how do we find the maximum element in the list?

To express the algorithm, we'll use pseudocode

- ✓ Pseudocode is kinda like a programming language, but not really

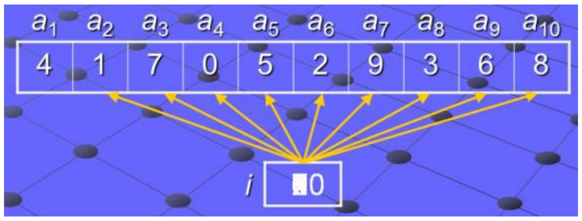
## *Example*

Show that Algorithm 1 for finding the maximum element in a finite sequence of integers has all the properties listed.

***Solution***

The input to Algorithm 1 is a sequence of integers. The output is the largest integer in the sequence.
Each step of the algorithm is precisely defined, because only assignments, a finite loop, and conditional statements occur.
The values of the variable *max* equals the maximum terms when the algorithm terminates.
The initial value of max is the first term; as successive terms of the sequence are examined. This argument shows that when all the terms have been examined, max equal the value of the largest term and it will take *n* steps.
Algorithm 1 is general, because it can be used to find the maximum of any finite sequence of integers.

```
Procedure max $\{a_1, a_2, \ldots, a_n\}$
max:= $a_1$
for i := 2 to n
    if max < $a_i$ then max := $a_i$
return max{max is the largest element}
```



$a_1$ $a_2$ $a_3$ $a_4$ $a_5$ $a_6$ $a_7$ $a_8$ $a_9$ $a_{10}$
4 1 7 0 5 2 9 3 6 8
i
10

## Searching Algorithms

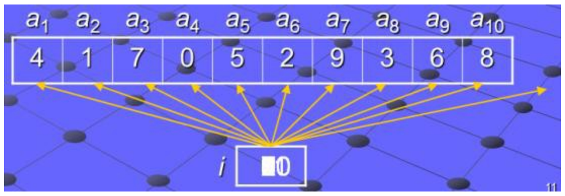
Given a list, find a specific element in the list. There are two types:

1. Linear search
2. Binary search

## *Algorithm 2* – Linear Search

Given a list which does not have to be sorted, find element in the list

```
procedure linear_search (x: integer; $a_1, a_2, \ldots, a_n$ : integers)
i := 1
while ( $i \le n$ and $\left(i \le n \text{ and } x \ne a_i\right)$
    $i := i + 1$
if $i \le n$ then location := i
else location := 0
{location is the subscript of the term that equals x, or it is 0 if x is not found}
```

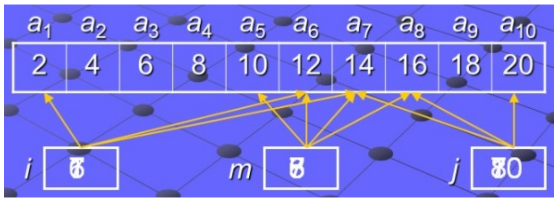


$a_1$ $a_2$ $a_3$ $a_4$ $a_5$ $a_6$ $a_7$ $a_8$ $a_9$ $a_{10}$
4 1 7 0 5 2 9 3 6 8
i
10
11

## *Algorithm* 3 – Binary Search

Given a list which *must* be sorted, find element in the list

```
procedure linear_search (x: integer; $a_1, a_2, \dots, a_n$ : increasing integers)
i := 1        { i is left endpoint of search interval }
j := n        { j is right endpoint of search interval }
while $i < j$
begin
    $m := \lfloor (i+j)/2 \rfloor$        {m is the point in the middle }
    if $x > a_m$ then $i := m+1$
    else $j := m$
end
if $x = a_i$ then location := i
else location := 0
{location is the subscript of the term that equals x, or it is 0 if x is not found}
```



$a_1$ $a_2$ $a_3$ $a_4$ $a_5$ $a_6$ $a_7$ $a_8$ $a_9$ $a_{10}$
2 4 6 8 10 12 14 16 18 20
i
m
j

$x = \boxed{14}$ *location* $\boxed{7}$

## *Sorting*

*Ordering* the elements of a list is a problem that occurs in many contexts. Suppose that we have a list of elements of a set. Suppose that we have a way to order elements of the set. ***Sorting*** is putting these elements into a list in which the elements are in increasing order.

There are two types:

- ✓ ***Bubble sort***
- ✓ ***Insertion sort***

## Bubble Sort

The *bubble sort* is one of the simplest sorting algorithms, but not one of the most efficient. It takes successive elements and "***bubbles***" them up the list.

### *Example*

Use the bubble sort to put 3, 2, 4, 1, 5 into increasing order.

***Solution***

| First Pass | 3 | 2 | 2 | 2 |
|---|---|---|---|---|
| | 2 | 3 | 3 | 3 |
| | 4 | 4 | 4 | 1 |
| | 1 | 1 | 1 | 4 |
| | 5 | 5 | 5 | 5 |

| Second Pass | 2 | 2 | 2 |
|---|---|---|---|
| | 3 | 3 | 1 |
| | 4 | 1 | 3 |
| | 1 | 4 | 4 |
| | $\boxed{5}$ | $\boxed{5}$ | $\boxed{5}$ |

| Third Pass | 2 | 1 |
|---|---|---|
| | 1 | 2 |
| | 3 | 3 |
| | $\boxed{4}$ | $\boxed{4}$ |
| | $\boxed{5}$ | $\boxed{5}$ |

| Fourth Pass | 1 |
|---|---|
| | 2 |
| | $\boxed{3}$ |
| | $\boxed{4}$ |
| | $\boxed{5}$ |

## *Algorithm* **4 – Bubble**

```
procedure bubblesort ($a_1, a_2, \ldots, a_n$ : real numbers with $n \ge 2$)
 for i := 1 to n – 1
     for j := 1 to n – i
         if $a_j > a_j + 1$
             then interchange $a_j$ and $a_j + 1$
```

## Bubble sort running time

***Outer*** for loop does $n-1$ iterations

***Inner*** for loop does:

$n-1$ iterations the first time

$n-2$ iterations the second time

***Total***: $(n-1)+(n-2)+\ldots+2+1=\frac{n^2-n}{2}$

The bubble sort will take about $n^2$ time.

## Insertion sort

The *insertion sort* is another simple sorting algorithm, but inefficient. It starts with a list with one element, and inserts new elements into their proper place in the sorted part of the list

## *Algorithm* 5 – Insertion sort

```
procedure insertion_sort $(a_1, a_2, \ldots, a_n)$
        for $j := 2$ to $n$                    take successive elements in the list
    begin
        $i := 1$                               find where that element should be
        while $a_j > a_i$                      in the sorted portion of the list
            $i := i + 1$
        $m := a_i$                             move all elements in the sorted portion of the list
        for $k := 0$ to $j-i-1$                that are greater than the current element up by one
            $a_{j-k} := a_{j-k-1}$
        $a_i := m$                             put the current element into it's proper place
    end $\{a_1, a_2, \ldots, a_n$ are sorted$\}$        in the sorted portion of the list
```

The *insertion* sort will take about $n^2$ time.

## *Comparison* of Running Times

***Searches***

- *Linear*: $n$ steps
- *Binary*: $\log_2 n$ steps
- *Binary* search is about as fast as you can get

**Sorts**

- *Bubble:* $n^2$ steps
- *Insertion:* $n^2$ steps
- There are other, more efficient, sorting techniques
  - In principle, the fastest are heap sort, quick sort, and merge sort
  - These each take $n \bullet \log_2 n$ steps
  - In practice, quick sort is the fastest, followed by merge sort

## *Algorithm* 6 – Greedy Change-Making Algorithm

```
procedure change ($c_1, c_2, \ldots, c_r$ : values of denominations of coins, where $c_1 > c_2 > \ldots > c_r$ ; n: a
                  positive integer)
  for i := 1 to r
      $d_i := 0$                          $d_i$ counts the coins of denomination $c_i$ used
      While $n \ge c_i$
          $d_i := d_i + 1$                Add a coin of denomination $c_i$
          $n := n - c_i$
{ $d_i$ is the number of coins of denomination $c_i$ in the change for i = 1, 2, …, r}
```

### *Definition*

If *n* is a positive integer, then *n* cents in change using quarters, dimes, nickels, and pennies using the fewest coins possible has at most two dimes, at most one nickel, at most four pennies, and cannot have two dimes and a nickel. The amount of change in dimes, nickels, and pennies cannot exceed 24 cents

### *Theorem*

The greedy algorithm (–6) produces change using the fewest coins possible.

## *Exercises* *Section* 2.2 – Algorithms

**1.** List all the steps used by the Algorithm 1 to find the maximum of the list

1, 8, 12, 9, 11, 2, 14, 5, 10, 4.

**2.** Devise an algorithm that finds the sum of all the integers in a list.

**3.** Describe an algorithm that takes as an input a list of $n$ integers and produces as output the largest difference obtained by subtracting an integer in the list from the one following it.

**4.** Describe an algorithm that takes as an input a list of $n$ integers in non-decreasing order and produces the list of all values that occur more than once.

**5.** Describe an algorithm that takes as an input a list of $n$ integers and finds the location of the last even integer in the list or returns 0 if there are no even integers in the list.

**6.** Describe an algorithm that interchanges the values of the variables $x$ and $y$, using only assignments. What is the minimum number of assignment statements needed to do this?

**7.** List all the steps used to search for 9 in the sequence 1, 3, 4, 5, 6, 7, 9, 11 using

***a***) a linear search *b*) a binary earch

**8.** Describe an algorithm that inserts an integer $x$ in the appropriate position into the list $a_1, a_2, \ldots, a_n$ of integers that are in increasing order.