

# MTH150 Discrete Mathematics

## Section 3.1. Algorithms

University of Rochester

October 7, 2009

**Definition 28.** An **algorithm** is a finite set of precise instructions for performing a computation or for solving a problem.

There are several well-known algorithms. For instance, we have algorithms for

1. Finding the greatest common divisor of two integers
2. Generating all the ordering of a finite set
3. Searching a list
4. Sorting the elements of a sequence
5. Finding the shortest path between two nodes in a network

We will consider a number of these algorithms and study their worst case **time/computational complexity**, using the big- $O$  and big- $\Theta$  notation. To specify algorithms, we will use a form of pseudocode, which loosely resembles a programming language.

**Example 67.** Describe an algorithm for finding the maximum value in a finite sequence of integers.

**Solution.** Let us consider the sequence of integers  $a_1, a_2, \dots, a_n$ .

1. Set the temporary maximum *max\_temp* equal to the first integer  $a_1$  in the sequence.
2. Compare the next integer  $a_2$  in the sequence to the temporary maximum.
3. If the integer  $a_2$  is larger than the temporary maximum, set the temporary maximum equal to this integer.
4. Repeat Step 2 if there are more integers in the sequence.
5. Stop when there are no integers left in the sequence.

The temporary maximum at this point is the largest integer in the sequence.

**Algorithm 1.** A procedure for finding the maximum element in a finite sequence.

```
procedure max( $a_1, a_2, \dots, a_n$  : integers)
  max_temp :=  $a_1$ 
  for  $i := 2$  to  $n$ 
    if max_temp <  $a_i$  then max_temp :=  $a_i$ 
  {max_temp is the largest element}
```

The **searching algorithms** locate an element in an ordered list.

The **searching problem** is the problem of locating an element  $x$  in a list of distinct elements  $a_1, a_2, \dots, a_n$  or determining that it is not in the list. The solution of this search problem is the location of the term in the list that equals  $x$  (i.e.,  $i$  is the solution of  $x = a_i$ ) and is 0 if  $x$  is not in the list.

The first search algorithm is called **linear search**, which can be described as follows: Given a sequence, compare  $x$  successively with each term of the sequence until a match is found.

1. Start by comparing  $x$  with  $a_1$ .
2. When  $x = a_1$  the solution is the location of  $a_1$ .
3. When  $x \neq a_1$ , compare  $x$  with  $a_2$ .
4. If  $x = a_2$ , then the solution is the location of  $a_2$ .
5. When  $x \neq a_2$ , compare  $x$  with  $a_3$ . And so on.

**Algorithm 2.** The linear search algorithm.

procedure *linearsearch*( $x$  : integer;  $a_1, \dots, a_n$  : distinct integers)

$i := 1$

while ( $i \leq n$  and  $x \neq a_i$ )

$i := i + 1$

if  $i \leq n$  then *location* :=  $i$

else *location* := 0

{*location* is the subscript of the term that equals  $x$ , or is 0 if  $x$  is not found}

The second search algorithm is the **binary search**, which can be described as follows.

1. Let  $a_1, a_2, \dots, a_n$  denote an ordered array of integers such that  $a_1 < a_2 < \dots < a_n$ .
2. Let  $x$  denote the integer for which we are searching.
3. Compare the value of  $x$  with the array entry at or near the center. The entry is  $a_{(n+1)/2}$  for  $n$  odd or  $a_{n/2}$  for  $n$  even. In either cases the array element is  $a_m$ , where  $m = \lfloor (n+1)/2 \rfloor$ . (Note that 1 is the value of the smallest subscript and  $n$  is the value of the largest subscript.)
4. If  $x = a_m$ , we are finished. If not then
  - i. If  $x > a_m$ , we search the subarray  $a_{m+1}, a_{m+2}, \dots, a_n$ .
  - ii. If  $x < a_m$ , we search the subarray  $a_1, a_2, \dots, a_{m-1}$ .

Note that the search is restricted to at most  $\lceil n/2 \rceil$  terms.

**Algorithm 3.** The binary search algorithm.

```
procedure binarysearch( $n$  : positive integer;  $x, a_1, \dots, a_n$  : integers)
 $i := 1$  { $i$  is the left endpoint of search interval}
 $j := n$  { $j$  is the right endpoint of search interval}
while  $i \leq j$ 
begin
     $m := \lfloor (i + j)/2 \rfloor$ 
    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 is 0 if  $x$  is
not found}
```

**Example 68.** Use the binary search to find 19 in the list

1, 2, 3, 5, 6, 7, 8, 10, 12, 13, 15, 16, 18, 19, 20, 22

**Solution.** Split the list into two smaller lists, each containing 8 terms:

1, 2, 3, 5, 6, 7, 8, 10 and 12, 13, 15, 16, 18, 19, 20, 22

Compare 19 and the largest term in the first list;  $10 < 19$ . So the search can be restricted to the second list. Split the second list into two smaller lists:

12, 13, 15, 16 and 18, 19, 20, 22

Since  $16 < 19$ , the search is restricted to the second list. Split the second list into two smaller lists:

18, 19 and 20, 22

Since  $19 \neq 19$ , the search is restricted to the first list. Split this list into two lists: 18 and 19. Since  $18 < 19$  the search is restricted to the second list. Now do a comparison to find that 19 is located as the 14th term in the original list.

The **sorting algorithms** order the elements of a list.

The **sorting problem** can be summarized as follows: Suppose we have a list of elements of a set. **Sorting** is putting the elements into a list in which the elements are in increasing order.

The simplest and most popular, though not the most efficient, method of sorting numeric data is the **bubble sort**. This algorithm successively compares adjacent elements and interchanges them if they are in the wrong order.

**Algorithm 4.** The bubble sort algorithm, version 1.

```
procedure bubblesort1( $a_1, \dots, a_n$  : 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  $a_{j+1}$ 
{ $a_1, \dots, a_n$  is in increasing order}
```

**Example 69.** Use the bubble sort in Algorithm 4 to put 3, 2, 4, 1, 5 into increasing order.

**Algorithm 5.** The bubble sort algorithm, version 2.

procedure *bubblesort2*( $a_1, \dots, a_n$  : reals with  $n \geq 2$ )

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

  for  $j := n$  down to  $i + 1$

    if  $a_j < a_{j-1}$  then interchange  $a_j$  and  $a_{j-1}$

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

**Example 70.** Use the bubble sort in Algorithm 5 to put

7, 9, 2, 5, 8

into increasing order.