Algorithms

Chapter 3

Chapter Summary

- ✓Algorithms
  - Example Algorithms
  - Algorithmic Paradigms
- ✓Growth of Functions
  - Big-O and other Notation
- ✓Complexity of Algorithms

Algorithms

Section 3.1

Problems and Algorithms

- In many domains there are key general problems that ask for output with specific properties when given valid input.
  - to precisely state the problem, using the appropriate structures to specify the input and the desired output.
  - solve the general problem by specifying the steps of a procedure that takes a valid input and produces the desired output.
    - This procedure is called an *algorithm*.

Algorithms

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

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

Solution: Perform the following steps:

1. **Set the temporary maximum equal to the first integer in the sequence.**
2. **Compare the next integer in the sequence to the temporary maximum.**
   - **If it is larger than the temporary maximum, set the temporary maximum equal to this integer.**
3. **Repeat the previous step if there are more integers. If not, stop.**
4. **When the algorithm terminates, the temporary maximum is the largest integer in the sequence.**

Specifying Algorithms

- Algorithms can be specified in different ways. Their steps can be described in English or in pseudocode.
- Pseudocode is an intermediate step between an English language description of the steps and a coding of these steps using a programming language.
- The form of pseudocode we use is specified in Appendix 3. It uses some of the structures found in popular languages such as C++ and Java.
- Programmers can use the description of an algorithm in pseudocode to construct a program in a particular language.
- Pseudocode helps us analyze the time required to solve a problem using an algorithm, independent of the actual programming language used to implement algorithm.

Properties of Algorithms

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

Output: From the input values, the algorithm produces the output values from a specified set. The output values are the solution.

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 output after a finite number of steps for any input.

Effectiveness: It must be possible to perform each step of the algorithm correctly and in a finite amount of time.

Generality: The algorithm should work for all problems of the desired form.

Finding the Maximum Element in a Finite Sequence

- The algorithm in pseudocode:

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

- Does this algorithm have all the properties listed on the previous slide?

Some Example Algorithm Problems

- Three classes of problems will be studied in this section.

1. *Searching Problems*: finding the position of a particular element in a list.
2. *Sorting problems*: putting the elements of a list into increasing order.
3. *Optimization Problems*: determining the optimal value (maximum or minimum) of a particular quantity over all possible inputs.

Searching Problems

Definition: **The general searching problem**

Locate an element *x* in a list of distinct elements $a_1, a_2, \ldots, a_n$ or determine that it is not in the list.

- **The solution to a searching problem**
  - **the location of the term in the list that equals *x* (that is, *i* is the solution if $x = a_i$) or 0 if *x* is not in the list**
- **For example,**
  - **a library might want to check to see if a patron is on a list of those with overdue books before allowing him/her to checkout another book.**
- **Two different searching algorithms:**
  - **linear search and binary search**

(1)Linear Search or sequential search

- An algorithm that linearly searches a sequence for a particular element.

```
ALGORITHM   The Linear Search Algorithm.
Procedure linear search (x: integer,
        $a_1, a_2, \Lambda\ , a_n$ : 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 subscript of term that equals x ,or is 0 if x
  is not found}
```

(2) Binary Search

If the terms in a sequence are ordered, a binary search algorithm is more efficient than linear search.

Example, To search for 88 in the list

5 13 19 21 37 56 64 75 80 88 92 100.

Solution :

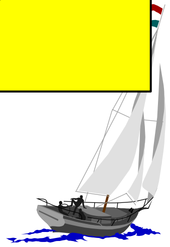
(1) **5 13 19 21 37 56 64 75 80 88 92 100**

(2) **64 75 80 88 92 100**

(3) **88 92 100**

(4) **88 92**

The binary search algorithm iteratively restricts the relevant search interval until it closes in on the position of the element to be located.



ALGORITHM **The Binary Search Algorithm.**

```
Procedure binary search (x: integer,
        $a_1, a_2, ..., a_n$ : increasing integers)
$i$ : = 1{$i$ is left endpoint of search interval }
$j$: =$n$ { $j$ is right endpoint of search interval }
while $i < j$
begin
    $m$: = $\lfloor (i+j)/2 \rfloor$
    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 term equal to x ,or 0 if x is
 not found }
```

Obviously, on sorted sequences, binary search is more efficient than linear search.

How can we analyze the efficiency of algorithms?

We can measure the

- **time (number of elementary computations) and**
- **space (number of memory cells) that the algorithm requires.**

These measures are called computational complexity and space complexity, respectively.

Sorting

- To sort the elements of a list is to put them in increasing order (numerical order, alphabetic, and so on).
- Sorting is an important problem because:
  - A nontrivial percentage of all computing resources are devoted to sorting different kinds of lists, especially applications involving large databases of information that need to be presented in a particular order (e.g., by customer, part number etc.).
  - An amazing number of fundamentally different algorithms have been invented for sorting. Their relative advantages and disadvantages have been studied extensively.
  - Sorting algorithms are useful to illustrate the basic notions of computer science.
- A variety of sorting algorithms are studied in this book: binary, insertion, bubble, selection, merge, quick, and tournament.

Greedy Algorithms

- Optimization problems minimize or maximize some parameter over all possible inputs.
- Among the many optimization problems we will study are:
  - Finding a route between two cities with the smallest total mileage.
  - Determining how to encode messages using the fewest possible bits.
  - Finding the fiber links between network nodes using the least amount of fiber.
- Optimization problems can often be solved using a greedy algorithm, which makes the "best" choice at each step.
- Making the "best choice" at each step does not necessarily produce an optimal solution to the overall problem, but in many instances, it does.
- After specifying what the "best choice" at each step is, we try to prove that this approach always produces an optimal solution, or find a counterexample to show that it does not.
- The greedy approach to solving problems is an example of an algorithmic paradigm, which is a general approach for designing an algorithm. We return to algorithmic paradigms in Section 3.3.