SC 612: Discrete Mathematics (MSc(IT)-Autumn 2023)

Lectures 7: Recursive Algorithms

Instructor: Gopinath Panda *August 23, 2023*

1 Algorithm

Definition 1.1

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

- The term algorithm is a corruption of the name al-Khowarizmi, a mathematician of the ninth century, whose book on Hindu numerals is the basis of modern decimal notation.
- Originally, the word algorism was used for the rules for performing arithmetic using decimal notation.
- Algorism evolved into the word algorithm by the eighteenth century.
- An algorithm can also be described using a computer language.
- Pseudocode provides an intermediate step between an English language description of an algorithm and an implementation of this algorithm in a programming language.

PROPERTIES OF ALGORITHMS

There are several properties that algorithms generally share. They are useful to keep in mind when algorithms are described. These properties are:

- 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.

1.1 Searching Algorithms

- THE LINEAR SEARCH The first algorithm that we will present is called the linear search, or sequential search, algorithm.

ALGORITHM 2 The Linear Search Algorithm.

```
procedure linear search($x$: integer, $a_1, a_2, \ldots, a_n$: 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
return location{location is the subscript of the term that equals $x$, or is 0 if $x$ is not found}
```

• THE BINARY SEARCH We will now consider another searching algorithm. This algorithm can be used when the list has terms occurring in order of increasing size

ALGORITHM 3 The Binary Search Algorithm.

```
procedure binary search ($x$: integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1${$i$ is left endpoint of search interval}
$j := n$ {$j$ is right endpoint of search interval}
while $i < j$
    $m := \lfloor (i + j)/2 \rfloor$
    if $x > a_m$ then $i := m + 1$
    else $j := m$
if $x = a_i$ then location := $i$
else location := 0
return location{location is the subscript $i$ of the term $a_i$ equal to $x$, or 0 if $x$ is not found}
```

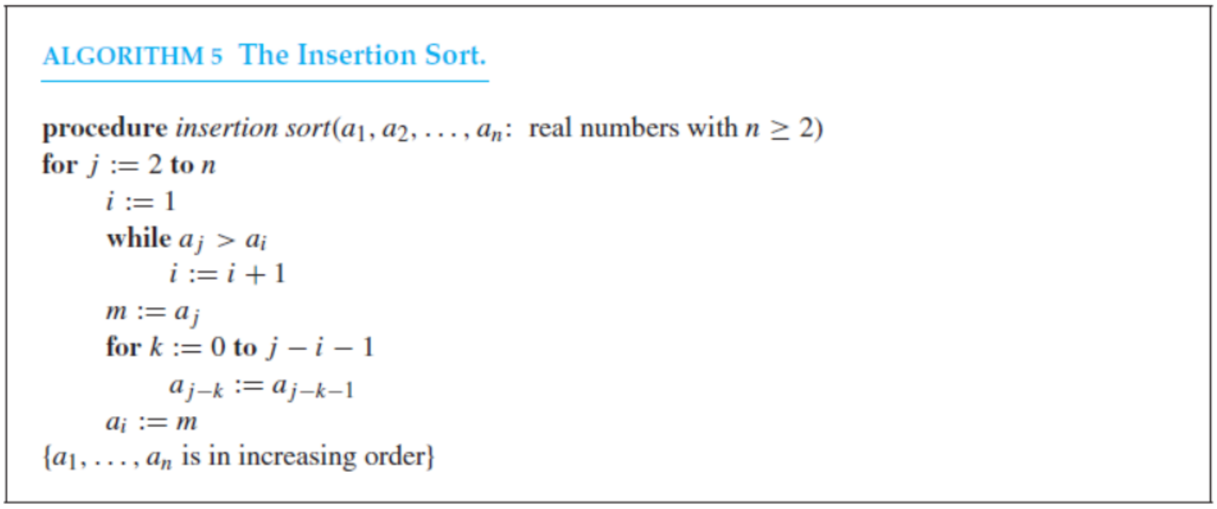
1.2 Sorting

• Ordering the elements of a list is a problem that occurs in many contexts.
• to produce a telephone directory it is necessary to alphabetize the names of subscribers.
• producing a directory of songs available for downloading requires that their titles be put in alphabetic order.
• THE BUBBLE SORT The bubble sort is one of the simplest sorting algorithms, that puts a list into increasing order by successively comparing adjacent elements, interchanging them if they are in the wrong order.

ALGORITHM 4 The Bubble Sort.

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

• THE INSERTION SORT The insertion sort is a simple sorting algorithm, but it is usually not the most efficient. To sort a list with n elements, the insertion sort begins with the second element. The insertion sort compares this second element with the first element and inserts it before the first element if it does not exceed the first element and after the first element if it exceeds the first element.



ALGORITHM 5 The Insertion Sort.

```
procedure insertion sort($a_1, a_2, \ldots, a_n$: real numbers with $n \geq 2$)
for $j := 2$ to $n$
    $i := 1$
    while $a_j > a_i$
        $i := i + 1$
    $m := a_j$
    for $k := 0$ to $j - i - 1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{$a_1, \ldots, a_n$ is in increasing order}
```

1.3 Greedy Algorithms

These algorithms are designed to solve optimization problems.

2 The Growth of Functions

Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $O(g(x))$ if there are constants C and k such that

$$|f(x)| \leq C|g(x)|$$

whenever $x > k$. [This is read as " $f(x)$ is big-oh of $g(x)$.']

Remark: Intuitively, the definition that $f(x)$ is $O(g(x))$ says that $f(x)$ grows slower that some fixed multiple of $g(x)$ as x grows without bound.

2.1 Time Complexity

The time complexity of an algorithm can be expressed in terms of the number of operations used by the algorithm when the input has a particular size. The operations used to measure time complexity can be the comparison of integers, the addition of integers, the multiplication of integers, the division of integers, or any other basic operation.

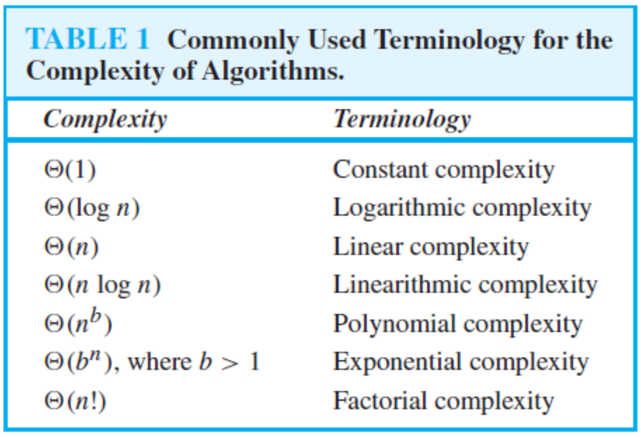


TABLE 1 Commonly Used Terminology for the Complexity of Algorithms.

Complexity	*Terminology*
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	Linearithmic complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

Write programs with these inputs and outputs.
Given a list of n integers,

1. find the largest integer in the list.
2. find the first and last occurrences of the largest integer in the list.
3. determine the position of an integer in the list using a linear search.

3 Recursive Algorithm

An algorithm is called recursive if it solves a problem by reducing it to an instance of the same problem with smaller input.

Algorithm 1: A Recursive Algorithm for Computing n!

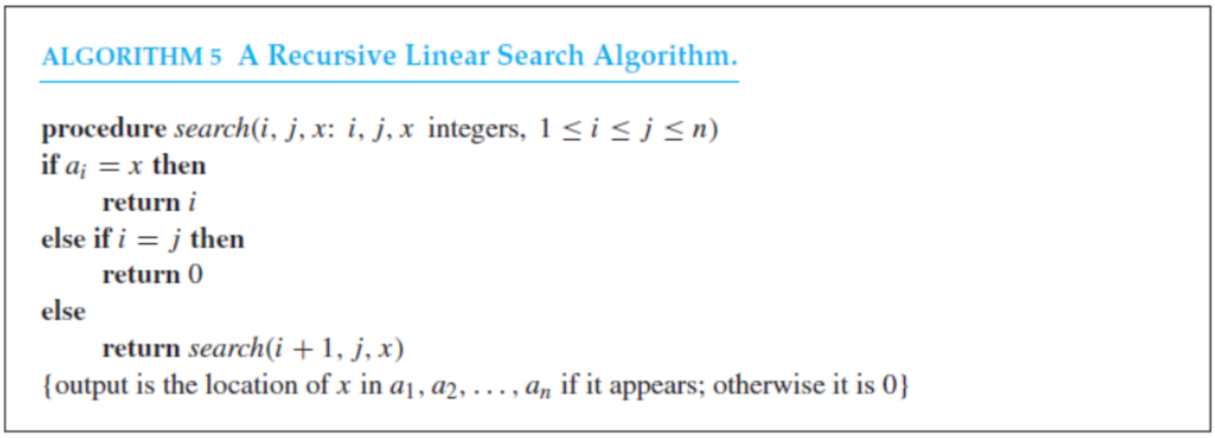
```
/* Python code to implement factorial                      */
// Factorial function
def  f(n)
    if n = 0 then
        return 1;                          /* Stop condition */
    else
        $nf(n-1)$;                    /* Recursive condition */
    end
```

Example. *Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.*

Solution: We can base a recursive algorithm on the recursive definition of a^n.
- Basis step: $a^0 = 1$
- Recursive step: $a^{n+1} = a \times a^n$ for $n > 0$

To find a^n, successively use the recursive step to reduce the exponent until it becomes zero.

$$\begin{aligned} a^5 &= a \times a^4 \\ &= a \times a \times \boldsymbol{a^3} \\ &= a \times a \times \boldsymbol{a} \times \underline{\boldsymbol{a^2}} \\ &= a \times a \times \boldsymbol{a} \times \underline{\boldsymbol{a \times a}} \end{aligned}$$



ALGORITHM 5 A Recursive Linear Search Algorithm.

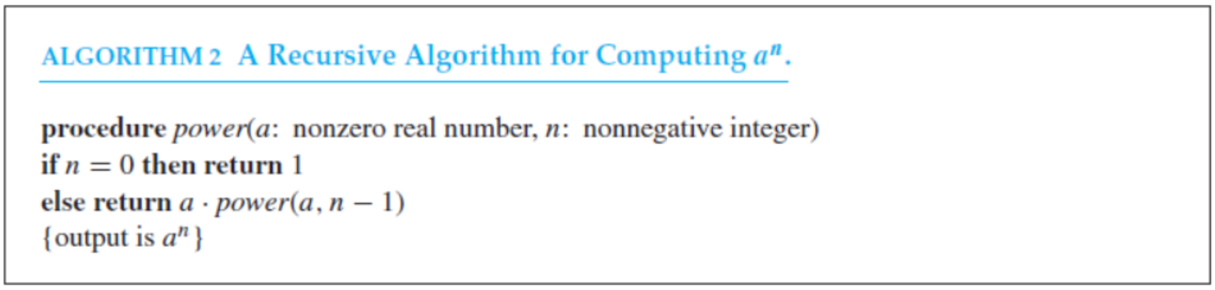
```
procedure search($i, j, x$: $i, j, x$ integers, $1 \le i \le j \le n$)
if $a_i = x$ then
    return $i$
else if $i = j$ then
    return 0
else
    return search($i + 1, j, x$)
{output is the location of $x$ in $a_1, a_2, \dots, a_n$ if it appears; otherwise it is 0}
```

To search for the first occurrence of x in the sequence $a_1, a_2, \dots, a_n$, at the ith step of the algorithm, x and a_i are compared.

• If x equals a_i , then the algorithm returns i, the location of x in the sequence.
• Otherwise, the search for the first occurrence of x is reduced to a search in a sequence with one fewer element, namely, the sequence $a_{i+1}, \ldots, a_n$.
• The algorithm returns 0 when x is never found in the sequence after all terms have been examined.

4 Proving Recursive Algorithms Correct

Mathematical induction, and its variant strong induction, can be used to prove that a recursive algorithm is correct.
Prove that the recursive search algorithm is correct.



ALGORITHM 2 A Recursive Algorithm for Computing a^n.

```
procedure power(a: nonzero real number, n: nonnegative integer)
if $n = 0$ then return 1
else return $a \cdot power(a, n - 1)$
{output is $a^n$}
```

Solution: We use mathematical induction on the exponent n.

BASIS STEP: If $n = 0$, the first step of the algorithm tells us that power$(a, 0) = 1$. This is correct because $a^0 = 1$ for every nonzero real number a. This completes the basis step.
INDUCTIVE STEP: The inductive hypothesis is the statement that power $(a, k) = a^k$ for all $a \neq 0$ for an arbitrary nonnegative integer k. That is, the inductive hypothesis is the statement that the algorithm correctly computes a^k.
To complete the inductive step, we show that if the inductive hypothesis is true, then the algorithm correctly computes a^{k+1}. Because $k+1$ is a positive integer, when the algorithm computes a^{k+1}, the algorithm sets power $(a, k+1) = a \cdot$ power(a, k).
By the inductive hypothesis, we have power $(a, k) = a^k$, so power $(a, k+1) = a \cdot$ power$(a, k) = a \cdot a^k = a^{k+1}$. This completes the inductive step.
We have completed the basis step and the inductive step, so we can conclude that Algorithm 2 always computes a^n correctly when $a \neq 0$ and n is a nonnegative integer.

5 Program Correctness

Program verification, the proof of correctness of programs, uses the rules of inference and proof techniques, including mathematical induction.

- A program is said to be correct if it produces the correct output for every possible input.
- A proof that a program is correct consists of two parts.
  - The first part shows that the correct answer is obtained if the program terminates.
    - ∗ This part of the proof establishes the partial correctness of the program.
  - The second part of the proof shows that the program always terminates.
- To specify what it means for a program to produce the correct output, two propositions are used.

  - The first is the initial assertion, which gives the properties that the input values must have.
  - The second is the final assertion, which gives the properties that the output of the program should have, if the program did what was intended.

- The appropriate initial and final assertions must be provided when a program is checked.

Definition 5.1

A program, or program segment, S is said to be partially correct with respect to the initial assertion p and the final assertion q if whenever p is true for the input values of S and S terminates, then q is true for the output values of S. The notation $p\{S\}q$ indicates that the program, or program segment, S is partially correct with respect to the initial assertion p and the final assertion q.

Note. • The notation p{S}q is known as a Hoare triple. Tony Hoare introduced the concept of partial correctness.
• the notion of partial correctness has nothing to do with whether a program terminates; it focuses only on whether the program does what it is expected to do if it terminates.

A simple example illustrates the concepts of initial and final assertions.

Show that the program segment

$y := 2$
$z := x + y$

is correct with respect to the initial assertion p: $x = 1$ and the final assertion q: $z = 3$.

Solution: Suppose that p is true, so that $x = 1$ as the program begins. Then y is assigned the value 2, and z is assigned the sum of the values of x and y, which is 3. Hence, S is correct with respect to the initial assertion p and the final assertion q. Thus, $p\{S\}q$ is true. ◀

Rules of Inference

A useful rule of inference proves that a program is correct by splitting the program into a sequence of subprograms and then showing that each subprogram is correct.

Suppose that the program S is split into subprograms S_1 and S_2. Write $S = S_1; S_2$ to indicate that S is made up of S_1 followed by S_2. Suppose that the correctness of S_1 with respect to the initial assertion p and final assertion q, and the correctness of S_2 with respect to the initial assertion q and the final assertion r, have been established. It follows that if p is true and S_1 is executed and terminates, then q is true; and if q is true, and S_2 executes and terminates, then r is true. Thus, if p is true and $S = S_1; S_2$ is executed and terminates, then r is true. This rule of inference, called the **composition rule**, can be stated as

$$\begin{array}{l} \quad p\{S_1\}q \\ \quad q\{S_2\}r \\ \hline \therefore\ p\{S_1; S_2\}r. \end{array}$$

This rule of inference will be used later in this section.

Next, some rules of inference for program segments involving conditional statements and loops will be given. Because programs can be split into segments for proofs of correctness, this will let us verify many different programs.

Conditional Statements

First, rules of inference for conditional statements will be given. Suppose that a program segment has the form

```
if condition then
    S
```

where S is a block of statements. Then S is executed if *condition* is true, and it is not executed when *condition* is false. To verify that this segment is correct with respect to the initial assertion p and final assertion q, two things must be done. First, it must be shown that when p is true and *condition* is also true, then q is true after S terminates. Second, it must be shown that when p is true and *condition* is false, then q is true (because in this case S does not execute).

This leads to the following rule of inference:

$$\begin{array}{l} \quad (p \wedge \mathit{condition})\{S\}q \\ \quad (p \wedge \neg\mathit{condition}) \rightarrow q \\ \hline \therefore\ p\{\textbf{if } \mathit{condition} \textbf{ then } S\}q. \end{array}$$