Ch 3.1 Algorithms (R)

Algorithms

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

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.

Searching Algorithms

The Linear Search (Sequential Search)

Start comparing with the first term, see if it matches, and go on.

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

Split into two lists, compare the largest term in the first list, eliminate one list, repeat until one list with one term is left.

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}

Sorting

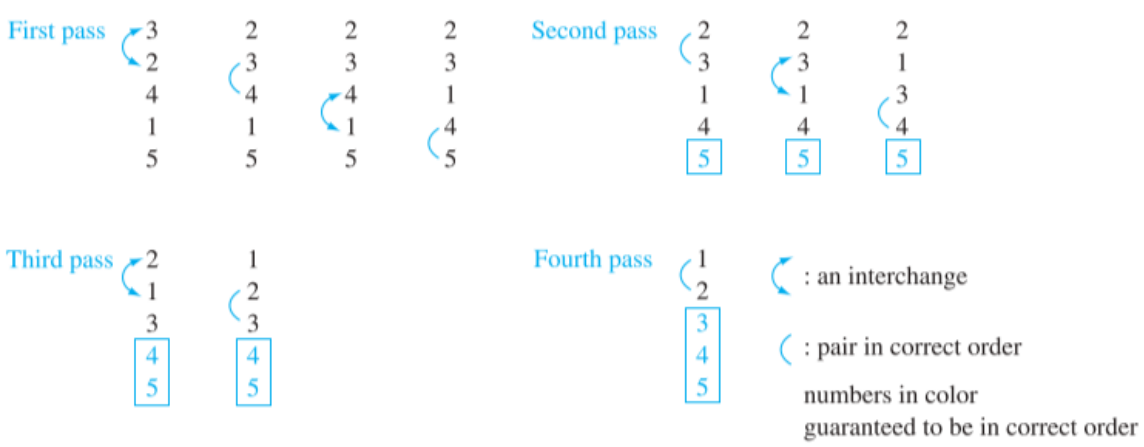
Bubble Sort

One of the simplest but not one of the most efficient

Successively comparing adjacent elements, interchange them if in 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}



Insertion Sort

A simple sorting algorithm, but usually not the most efficient.

Begins with the second element, compare the second to the first, if exceed, insert before the first element if does not exceed, vice versa. And on with the third... elements.

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}

String Matching

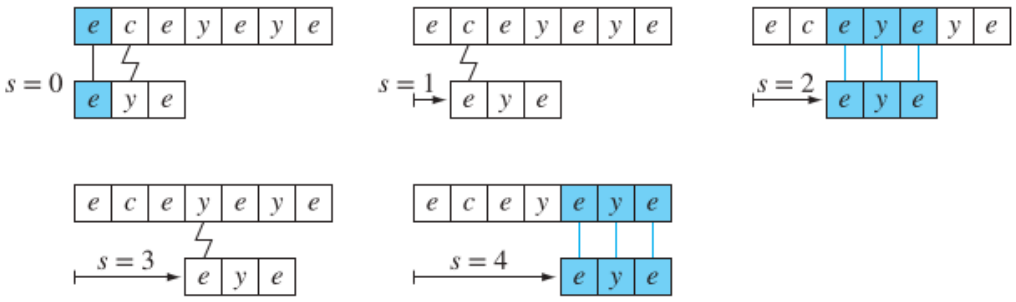
→ Finding where a pattern occurs in a text string

Ex. Find whether 101 is in 11001011, whether CAG is in CATCACAGAGA.

Naive String Matcher

ALGORITHM 6 Naive String Matcher.

procedure *string match* (n, m: positive integers, $m \leq n$, $t_1, t_2, \ldots, t_n, p_1, p_2, \ldots, p_m$: characters)
  for $s := 0$ **to** $n - m$
    $j := 1$
    while ($j \leq m$ and $t_{s+j} = p_j$)
      $j := j + 1$
    if $j > m$ **then print** "s is a valid shift"



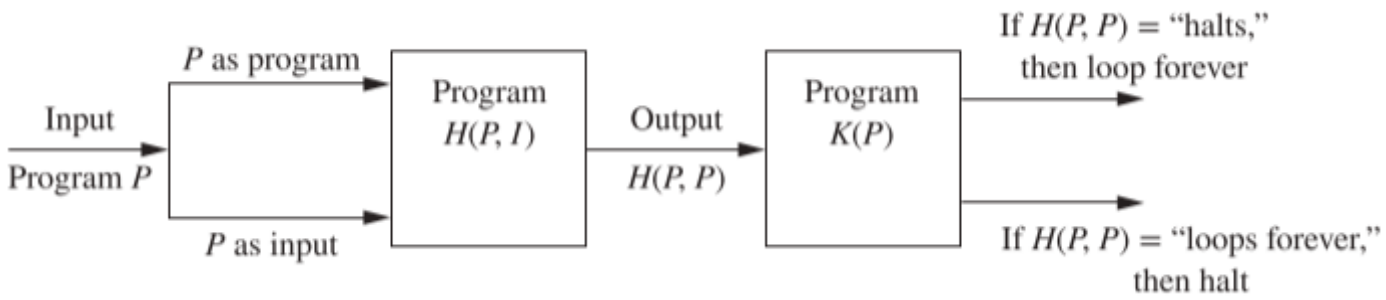
Greedy Algorithms

→ solve optimization problems

→ Algorithms that make what seems to be the "best" choice at each step

→ We call the algorithm "greedy" whether or not it finds an optimal solution

The Halting Problem



Can't just give an input to test, because

if it halts, we have our answers

but if it is still running after any fixed length of time, we do not know whether it will never halt or we just did not wait long enough for it to terminate.

Appendix 3 Pseudocode

```
# precedure statements
procedure maximum(L: list of integers)

# assignments
variable := expression
max := a
x := largest integer in the list L
interchange a and b

# comments
{ x is the largest element in L }

# conditional
if condition then statement

if condition then
  block of statements

if condition then
  statement 1
  statement 2
  statement 3
  .
  .
  .
  statement n

if condition then statement 1
else statement 2

if condition 1 then statement 1
else if condition 2 then statement 2
else if condition 3 then statement 3
  .
  .
  .
else if condition n then statement n
else statement n + 1

# loop
for variable := initla value to final value
  (block of) statement(s)

sum := 0
for i := 1 to n
  sum := sum + i

for all elements with a certain property

while condition
  (block of) statement(s)

sum := 0
while n > 0
  sum := sum + n
  n := n - 1

# precedures in other precedures
max(L)

# return statements
return x
return f(n-1)
```