Recitation 11

Jie.

Page 200

17. Analyze the worst-case time complexity of the algorithm you devised in Exercise 29 of Section 3.1 for locating a mode in a list of nondecreasing integers.

Algo. for Exe 29.

29. **procedure** *find a mode*($a_1, a_2, \ldots, a_n$: nondecreasing integers)
modecount := 0
i := 1
while $i \le n$
begin
  value := a_i
  count := 1
  while $i \le n$ and a_i = *value*
  begin
    count := *count* + 1
    $i := i + 1$
  end
  if *count* > *modecount* **then**
  begin
    modecount := *count*
    mode := *value*
  end
end
{*mode* is the first value occurring most often}

Solution: $O(n)$.

Let's look at an example:

1 1 1 2 2 3 3 3 3 4

step1: $i=1$, Value $= a_1 = 1$
count $= 2$
$i = 2$
modecount $= 2$
mode $= 1$

$i=2$ (a_2 = value)
count $= 3$
$i = 3$
modecount $= 3$
mode $= 1$

$i=3$ (a_3 = value).
count $= 4$

$i=4$ $a_4 \neq 1$ then we go back to out loop.
value $= a_4 = 2$
count $= 2$
$i = 5$

$i=5$. (a_5 = value.)
Count $= 3$
$i = 6$

$i=6$ $a_6 \neq 2$. back to out loop
Value $= 3$.
count $= 2$
$i = 7$.

$i=7$ ($a_7 = 3$.)
count $= 3$
$i = 8$

$i=8$ ($a_8 = 3$)
Count $= 4$
$i = 9$

P195. Example 5:

What is the worst-case complexity of the bubble sort in terms of the number of comparisons made?

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}

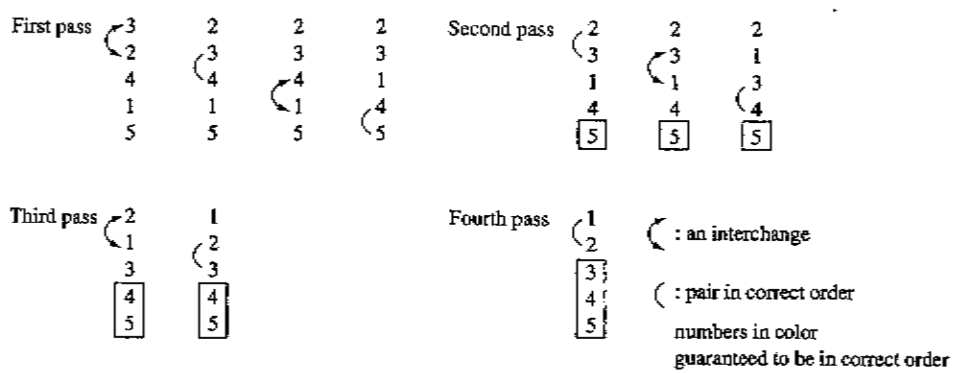

FIGURE 1 The Steps of a Bubble Sort.

Solution: The bubble sort (described in Example 4 in Section 3.1) sorts a list by performing a sequence of passes through the list. During each pass the bubble sort successively compares adjacent elements, interchanging them if necessary. When the ith pass begins, the $i - 1$ largest elements are guaranteed to be in the correct positions. During this pass, $n - i$ comparisons are used. Consequently, the total number of comparisons used by the bubble sort to order a list of n elements is

$$(n-1) + (n-2) + \cdots + 2 + 1 = \frac{(n-1)n}{2}$$

using a summation formula that we will prove in Section 4.1. Note that the bubble sort always uses this many comparisons, because it continues even if the list becomes completely sorted at some intermediate step. Consequently, the bubble sort uses $(n-1)n/2$ comparisons, so it has $\Theta(n^2)$ worst-case complexity in terms of the number of comparisons used. ◀

P199.

Ex 3: Suppose that an element is known to be among the first 4 elements in a list of elements. Would a linear search or binary search locate this element more rapidly?

Solution:

First let's look at the two algorithms:

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

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 $i < j$
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 equal to x, or 0 if x is not found}

Example: 1 2 9 10 15 17 19 ($x = 9$)

Linearch: $i=1$ $x \neq a_1$
$i=2$ $x \neq a_2$
$i=3$ $x = a_3$. location $=3$ $\therefore$ total 3 steps.

binarych: $i=1$, $j=7$, $m=4$, $\because 9 < a_4 \Rightarrow j=4$
$i=1$; $j=4$, $m=2$, $\because 9 > a_2 \Rightarrow i=3$
$i=3$, $j=4$, $m=3$, $\because 9 = a_3 \Rightarrow$ location $=2$

Then, back to our problems.

Let's first look at Linear search algorithm:

If X is the first element, we need 1 comparation to locate

X is the 2^{nd} element, we need 2 comparations to locate

X is the 3^{rd} element, we need 3 comparation to locate

X is the 4^{th} element, we need 4 comparations to locate

so on average, we need $\frac{1+2+3+4}{4} = 2.5$ Comparations.

Now. Let's look at Biary search algorithm:

If x is the first element: we need compare X and a_{16}.
and compare X and a_1
then, we need 2 comparations.

If x is the 2^{nd} elemet, we need to compare:

$\left\{\begin{array}{l} X \text{ and } a_{16} \\ X \text{ and } a_1 \end{array}\right.$ $\left\{\begin{array}{l} X \text{ and } a_8 \\ X \text{ and } a_1 \end{array}\right.$ $\left\{\begin{array}{l} X \text{ and } a_4 \\ X \text{ and } a_1 \end{array}\right.$ $\left\{\begin{array}{l} X \text{ and } a_2 \\ X \text{ and } a_1 \end{array}\right.$

So. 8 Comparations in total.

If X is the 3^{rd} element, we need to compare.

$\left\{\begin{array}{l} X \text{ and } a_{16} \\ X \text{ and } a_1 \end{array}\right.$ $\left\{\begin{array}{l} X \text{ and } a_8 \\ X \text{ and } a_1 \end{array}\right.$ $\left\{\begin{array}{l} X \text{ and } a_4 \\ X \text{ and } 1 \end{array}\right.$ $\left\{\begin{array}{l} X \text{ and } a_2 \\ X \text{ and } a_3 \end{array}\right.$

So. 8 Comparations in total

If X is the 4^{th} element, we need to compare.

$\left\{\begin{array}{l} X \text{ and } a_{16} \\ X \text{ and } a_1 \end{array}\right.$ $\left\{\begin{array}{l} X \text{ and } a_8 \\ X \text{ and } a_1 \end{array}\right.$ $\left\{\begin{array}{l} X \text{ and } a_4 \\ X \text{ and } a_1 \end{array}\right.$ $\left\{\begin{array}{l} X \text{ and } a_2 \\ X \text{ and } a_3 \end{array}\right.$

X and a_4

So. 9 Comparations in total.

So. on average: we need $\frac{2+8+8+9}{4} = 4.5$ comperiattons.

Thus. Linear search is more rapidly.

P79.

53. Use the greedy algorithm to make change using quarters, dimes, nickels, and pennies, for

a) 51 cents.

solution: 2 quarters.

1 penny

b) 69 cents:

Solution: 2 quarters

1 dime

1 nickel

4 pennies

c) 76 cents

solution: 3 quarters.

1 penny

d) 60 cents

Solution: 2 quarters

1 dime.

P177. 9. A Palindrome is a string that reads the same forward and backward. Decribe an algorithm for determining whether a string of n characters is a palindrome.

Solution :

procedure palindrome check ($a_1 a_2 \cdots a_n$: string)

answer := true

for $i := 1$ to $\lfloor n/2 \rfloor$

if $a_i \neq a_{n+1-i}$ then answer = false

end.

answer is true iff string is a palindrome.

P.121

35. Use the bubble sort to sort 3. 1. 5. 7. 4. showing the lists obtained at each step.

Solution: at the end of the first pass: 1. 3. 5. 4. 7

second : 1, 3, 4. 5. 7

third : 1. 3. 4, 5, 7

forth : 1. 3. 4. 5. 7

37. Adapt the bubble sort ~~to sort~~ algorithm so that it stops when no interchanges are required.

Solution:

```
procedure. better bubblesort (a_1 ... a_n : integers)
 i = 1 ; done = false
 while ( i < n and done = false )
 begin
     done = true
     for j = 1 to n - i
        if a_j > a_{j+1} then
           begin
              interchange a_j and a_{j+1}
              done = false
           end.
     i = i + 1
 end.
```

input: $(a_1 \cdots a_n)$ a set of integers

output: $a_1, \cdots a_n$ is in increasing order.