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.

Algorithm for Exe 29.

29. procedure find mode \((a_1, a_2, \ldots, a_n; \text{nondecreasing integers})\)

\[
\text{mode} := 0 \\
i := 1 \\
\text{while } i \leq n \\
\begin{align*}
&\text{begin} \\
&\quad \text{value} := a_i \\
&\quad \text{count} := 1 \\
&\quad \text{while } i \leq n \text{ and } a_i = \text{value} \\
&\quad \begin{align*}
&\quad \quad \text{count} := \text{count} + 1 \\
&\quad \quad i := i + 1
\end{align*}
&\text{end} \\
&\quad \text{if } \text{count} > \text{mode} \text{count} \text{ then} \\
&\quad \begin{align*}
&\quad \quad \text{mode} \text{count} := \text{count} \\
&\quad \quad \text{mode} := \text{value}
\end{align*}
&\text{end}
\end{align*}
\]

\(\text{(mode is the first value occurring most often)}\)

\[\text{Solution: } O(n)\]

Let's look at an example:

1 1 2 2 3 3 3 4

Step 1: \(i = 1\), \(\text{value} = a_1 = 1\)
\[\text{count} = 2\]
\[i = 2\]
\[\text{mode} = 1\]
\[\text{mode} \text{count} = 2\]

Step 2: \(a_2 = \text{value}\)
\[\text{count} = 3\]
\[i = 3\]
\[\text{mode} \text{count} = 3\]
\[\text{mode} = 1\]

Step 3: \(a_3 = \text{value}\)
\[\text{count} = 4\]

\(i = 4\) \(a_4 = 1\) then we go back to out loop.

\[\text{Value} = a_4 = 2\]
\[\text{count} = 2\]
\[i = 5\]

\(i = 5\), \(a_5 = \text{value}\)
\[\text{count} = 3\]
\[i = 6\]

\(i = 6\) \(a_6 = 2\), back to out loop.

\[\text{Value} = a_6 = 2\]
\[\text{count} = 3\]
\[i = 7\]

\(i = 7\), \(a_7 = 3\)
\[\text{count} = 3\]
\[i = 8\]

\(i = 8\), \(a_8 = 3\)
\[\text{count} = 4\]

\(i = 9\).
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

First pass \(\begin{array}{llll}
2 & 2 & 2 & 2 \\
4 & 3 & 3 & 3 \\
1 & 1 & 4 & 4 \\
5 & 5 & 5 & 5
\end{array}\)  Second pass \(\begin{array}{llll}
2 & 2 & 2 & 2 \\
1 & 3 & 3 & 1 \\
4 & 1 & 1 & 3 \\
3 & 4 & 4 & 5
\end{array}\)

Third pass \(\begin{array}{llll}
2 & 2 & 1 & 1 \\
3 & 3 & 1 & 3 \\
4 & 4 & 4 & 4 \\
5 & 5 & 5 & 5
\end{array}\)  Fourth pass \(\begin{array}{llll}
2 & 2 & 1 & 1 \\
1 & 3 & 3 & 3 \\
4 & 1 & 4 & 4 \\
3 & 5 & 5 & 5
\end{array}\)

\(\odot\) : an interchange
\(\odot\) : pair in correct order
numbers in color guaranteed to be in correct 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 \(i\)th 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.
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:

\begin{algorithm}
\textbf{Algorithm 2} The Linear Search Algorithm.
\begin{algorithmic}
\Procedure{linear search}{x: integer, \(a_1, a_2, \ldots, a_n\): distinct integers}
\State \(i := 1\)
\While{\((i \leq n \text{ and } x \neq a_i)\)}
\State \(i := i + 1\)
\EndWhile
\If{\(i \leq n\)}
\State {location} := \(i\)
\Else
\State {location} := 0
\EndIf
\EndProcedure
\end{algorithmic}
\end{algorithm}

\begin{algorithm}
\textbf{Algorithm 3} The Binary Search Algorithm.
\begin{algorithmic}
\Procedure{binary search}{x: integer, \(a_1, a_2, \ldots, a_n\): increasing integers}
\State \(i := 1\) \quad \{i is left endpoint of search interval\}
\State \(j := n\) \quad \{j is right endpoint of search interval\}
\While{\(i < j\)}
\State \(m := \lfloor (i + j)/2 \rfloor\)
\If{\(x > a_m\)}
\State \(i := m + 1\)
\Else
\State \(j := m\)
\EndIf
\EndWhile
\If{\(x = a_i\)}
\State {location} := \(i\)
\Else
\State {location} := 0
\EndIf
\EndProcedure
\end{algorithmic}
\end{algorithm}

\textbf{Example:} \(1 \ \ 2 \ \ 9 \ \ 10 \ \ 15 \ \ 17 \ \ 19 \ \ (x=9)\)

\begin{itemize}
\item \textbf{Linear Search:} \(i=1\) \quad \(x \neq a_1\)
\item \(i=2\) \quad \(x \neq a_2\)
\item \(i=3\) \quad \(x = a_3, \ \text{location} = 3\) \quad \ldots \text{total 3 steps.}
\item \textbf{Binary Search:} \(i=1\), \(j=6\), \(m = 4\), \quad 9 < a_4 \Rightarrow j=4\)
\item \(i=1\), \(j=4\), \(m = 3\), \quad 9 > a_3 \Rightarrow i=3\)
\item \(i=3, \ j=4, \ m = 3, \quad 9 = a_3 \Rightarrow \text{location} = 2\)
\end{itemize}
Then, back to our problems.

Let's first look at linear search algorithm:

If \( x \) is the first element, we need 1 comparison to locate it.
\( x \) is the 2nd element, we need 2 comparisons to locate it.
\( x \) is the 3rd element, we need 3 comparisons to locate it.
\( x \) is the 4th element, we need 4 comparisons to locate it.

So, on average, we need \( \frac{1+2+3+4}{4} = 2.5 \) comparisons.

Now, let's look at binary search algorithm:

If \( x \) is the first element, we need compare \( x \) and \( a_1 \)
and compare \( x \) and \( a_2 \)
then, we need 2 comparisons.

If \( x \) is the 2nd element, we need to compare:
- \( x \) and \( a_1 \)
- \( x \) and \( a_2 \)
- \( x \) and \( a_3 \)

So, 3 comparisons in total.

If \( x \) is the 3rd element, we need to compare:
- \( x \) and \( a_2 \)
- \( x \) and \( a_3 \)
- \( x \) and \( a_4 \)
- \( x \) and \( a_1 \)

So, 4 comparisons in total.

If \( x \) is the 4th element, we need to compare:
- \( x \) and \( a_3 \)
- \( x \) and \( a_4 \)
- \( x \) and \( a_1 \)
- \( x \) and \( a_2 \)
- \( x \) and \( a_4 \)

So, 5 comparisons in total.

So, on average, we need \( \frac{2+3+4+5}{4} = 4.5 \) comparisons.

Thus, linear search is more rapidly.
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.
**Problem:** A Palindrome is a string that reads the same forward and backward. Describe an algorithm for determining whether a string of $n$ characters is a palindrome.

**Solution:**

procedure palindrome check (a, a2, ... an : string)

answer := true

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

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

end.

answer is true iff string is a palindrome.
35. Use the bubble sort to sort 3, 1, 5, 7, 4, showing the list 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

Fourth: 1, 3, 4, 5, 7

81. Adapt the bubble sort algorithm so that it stops when no interchanges are required.

Solution:

procedure better_bubblesort (a, ... a_n : integers)
        i := 1; done = false
        while (i < n and done = false)
            begin
                    done = true
                    for j := 1 to n-i
                    if aj > aj+1 then
                        begin
                            interchange aj and aj+1
                            done = false
                        end;
                    i := i + 1
                end;
        input: (a1... a_n) a set of integers
        output: a1... a_n is in increasing order.