Chapter 3. Algorithms

Topics

- Algorithms
- The Growth of Functions
- Complexity of Algorithms

ALGORITHMS

3.1- Algorithms

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

3.1- Algorithms

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

We perform the following steps:

1. Set the temporary maximum equal to the first integer in the sequence. (The temporary maximum will be the largest integer examined at any stage of the procedure.)
2. Compare the next integer in the sequence to the temporary maximum, and 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 in the sequence.
4. Stop when there are no integers left in the sequence. The temporary maximum at this point is the largest integer in the sequence.

3.1- Algorithms

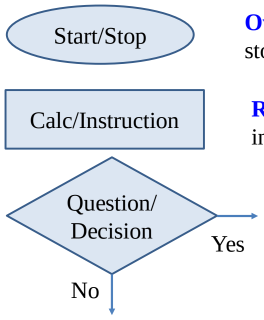
An algorithm can also be described using a computer language

→ difficult to understand and difficult to choose one particular language among so many programming languages

→ a form of **pseudocode** will be used to describe an algorithm.

Flow diagrams

A **flow diagram** is pictorial representation of an algorithm. The shape of the box indicates the type of instruction.



Oval boxes: Used for starting and stopping, inputting and outputting data

Rectangles: Used for calculations or instructions

Diamond shapes: Used for questions and decisions

Properties of an algorithm

- **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, and they are solutions 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 number of steps.
- **Effectiveness:** It must be possible to perform each step of an algorithm exactly and in a finite amount of time.
- **Generality:** Algorithm should be applicable for all problems of the desired form, not just a particular set of input values.

Some algorithms

Example 1. Find maximum element

Input: Sequence of integers $a_1, a_2, \ldots, a_n$.

Output: The maximum number of the sequence

Algorithm:

Step 1. Set the temporary maximum be the first element.

Step 2. Compare the temporary maximum to the next element, if this element is larger then set the temporary maximum to be this integer.

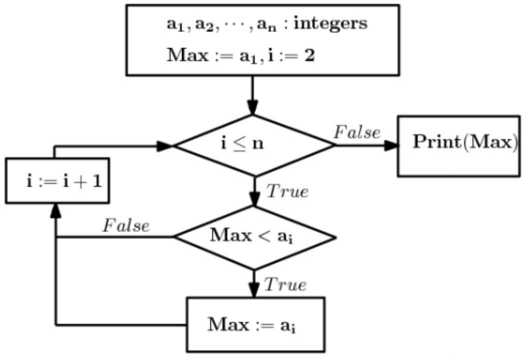
Step 3. Repeat *Step 2* if there are more integers in the sequence.

Step 4. Stop the algorithm when there are no integers left. The temporary maximum at this point is the maximum of the sequence.

Some algorithms

Example 1 (cont.). Find maximum element

Flow diagram:



Some algorithms

Example 1 (cont.).

ALGORITHM 1. Finding the Maximum Element in a Finite Sequence.

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

Some algorithms

Example 2. The linear search

Input: A sequence of distinct integers $a_1, a_2, \ldots, a_n$, and an integer x

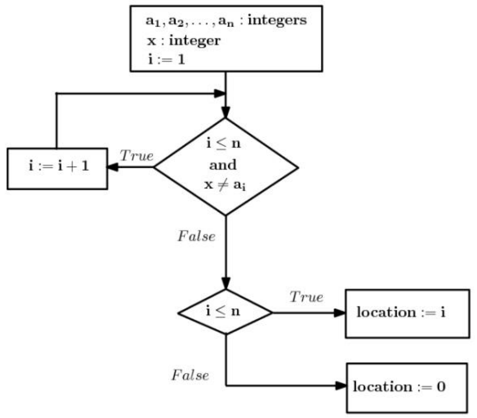
Output: The location of x in the sequence (is 0 if x is not in the sequence)

Algorithm: Compare x successively to each term of the sequence until a match is found.

Some algorithms

Example 2 (cont.). The linear search

Flow diagram:



Some algorithms

Example 2 (cont.). The linear search

ALGORITHM 2. The Linear Search Algorithm.

```
procedure linearsearch(x: integer, $a_1$, $a_2$, …, $a_n$: distinct integers)
i := 1
while (i ≤ n and x ≠ $a_i$)
        i := i + 1
if i ≤ 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}
```

Some algorithms

Example 3. The binary search

Input: An increasing sequence of integers $a_1 < a_2 < \ldots < a_n$ and an integer x

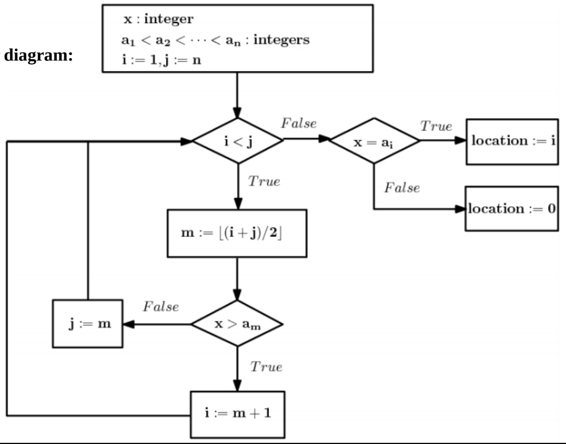
Output: The location of x in the sequence (is 0 if x is not in the sequence)

Algorithm: Compare x to the element at the middle of the list, then restrict the search to either the sublist on the left or the sublist on the right.

Some algorithms

Example 3 (cont.). The binary search

Flow diagram:



Some algorithms

Example 3 (cont.).

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 $a_i$ that equals
$x$, or is 0 if $x$ is not found}
```

Some algorithms

Example 4. Sorting – The bubble sort

Input: A sequence of integers $a_1, a_2, \ldots, a_n$

Output: The sequence in the increasing order

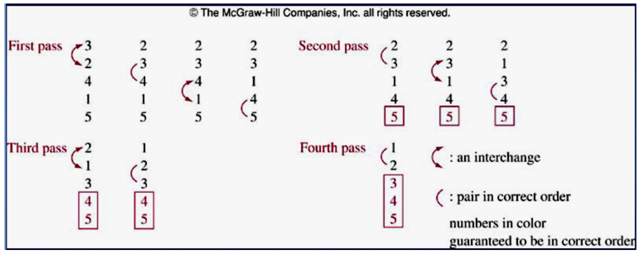
Algorithm:

1. Successively comparing two consecutive elements of the list to push the largest element to the bottom of the list.
2. Repeat the above step for the first $n - 1$ elements of the list.

Some algorithms

Example 4 (cont.). Sorting – The bubble sort

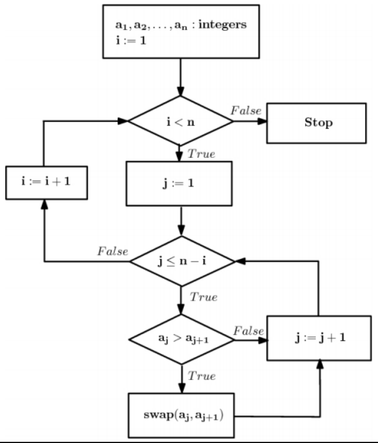
Example. Use the bubble sort algorithm to put 3, 2, 4, 1, 5 into increasing order.


© The McGraw-Hill Companies, Inc. all rights reserved.
First pass
Second pass
Third pass
Fourth pass
: an interchange
: pair in correct order
numbers in color guaranteed to be in correct order

Some algorithms

Example 4 (cont.). Sorting – The bubble sort

Flow diagram



Some algorithms

Example 4 (cont.). Sorting – The bubble sort

ALGORITHM 4. The Bubble Sort

```
procedure bubblesort($a_1, a_2, \ldots, a_n$: 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 $a_{j+1}$
{$a_1, a_2, \ldots, a_n$ is in increasing order}
```

Some algorithms

Example 5. Sorting – The insertion sort

Input: A sequence of integers $a_1, a_2, \ldots, a_n$

Output: The sequence in the increasing order

Algorithm:

1. Sort the first two elements of the list
2. Insert the third element to the list of the first two elements to get a
list of 3 elements of increasing order.
3. Insert the fourth element to the list of the first three elements to get a list of 4 elements of increasing order.

...

n Insert the nth element to the list of the first n − 1 elements to get a list of increasing order.

Some algorithms

Example 5 (cont.). Sorting – The insertion sort

Example. Use the insertion sort to put the elements of the list 3, 2, 4, 1, 5 in increasing order.

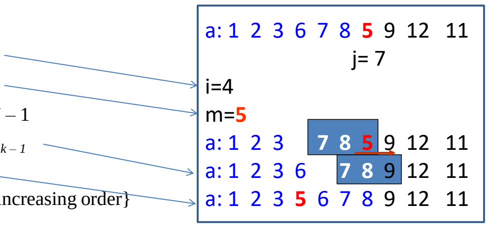
Solution: The insertion sort first compares 2 and 3. Because 3 > 2, it places 2 in the first position, producing the list 2, 3, 4, 1, 5 (the sorted part of the list is shown in color). At this point, 2 and 3 are in the correct order. Next, it inserts the third element, 4, into the already sorted part of the list by making the comparisons 4 > 2 and 4 > 3. Because 4 > 3, 4 remains in the third position. At this point, the list is 2, 3, 4, 1, 5 and we know that the ordering of the first three elements is correct. Next, we find the correct place for the fourth element, 1, among the already sorted elements, 2, 3, 4. Because 1 < 2, we obtain the list 1, 2, 3, 4, 5. Finally, we insert 5 into the correct position by successively comparing it to 1, 2, 3, and 4. Because 5 > 4, it stays at the end of the list, producing the correct order for the entire list.

Some algorithms

Example 5 (cont.). Sorting – The insertion sort

ALGORITHM 5. The Insertion Sort.

```
procedure insertionsort($a_1, a_2, \ldots, a_n$: 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 $j - i - 1$
      $a_{j-k} := a_{j-k-1}$
   $a_i := m$
{$a_1, a_2, \ldots, a_n$ is in increasing order}
```



Greedy Algorithm

- They are usually used to solve optimization problems: Finding out a solution to the given problem that either minimizes or maximizes the value of some parameter.
- Selecting the best choice at each step, instead of considering all sequences of steps that may lead to an optimal solution.
- Some problems:
  - Finding a route between two cities with smallest total mileage (number of miles that a person passed).
  - Determining a way to encode messages using the fewest bits possible.
  - Finding a set of fiber links between network nodes using the least amount of fiber.

Greedy Algorithm

Example 6. Consider the problem of making n cents change with quarters, dimes, nickels, and pennies, and using the least total number of coins.

Input: n cents

Output: The least number of coins using quarters (= 25 cents), dimes (= 10 cents), nickles (= 5 cents) and pennies (= 1 cent).

Greedy Algorithm

Example 6 (cont.).

ALGORITHM 7. Cashier's Algorithm

```
procedure change($c_1$, $c_2, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > \cdots > c_r$; n: a positive integer)
for i := 1 to r
    $d_i := 0$ {$d_i$ counts the coins of denomination $c_i$ used}
    while $n \geq c_i$
        $d_i := d_i + 1${add a coin of denomination $c_i$}
        $n := n - c_i$
{$d_i$ is the number of coins of denomination $c_i$ in the change for $i$ = 1, 2, … , r}
```

Exercises

1. List all the steps used by Algorithm 1 to find the maximum of the list 1, 8, 12, 9, 11, 2, 14, 5, 10, 4.

2. List all the steps used to search for 9 in the sequence 1, 3, 4, 5, 6, 8, 9, 11 using

a) a linear search. b) a binary search.

3. Use the bubble sort to sort 6, 2, 3, 1, 5, 4, showing the lists obtained at each step.

4. Show all the steps used by the binary insertion sort to sort the list 3, 2, 4, 5, 1, 6.

Exercises

5. Consider the linear search algorithm

```
procedure linearsearch(x: integer, $a_1, a_2, \ldots, a_n$: 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
return location
```

Given the sequence 3, 1, 5, 7, 4, 6. How many comparisons required for searching $x = 7$?

Exercises

6. Given the algorithm

```
procedure tt($a_1, a_2, \ldots, a_n$: integers)
i := 1
while $i \le n$ and $a_i \ne a_{i+1}$ do
        $i := i + 1$
if $i > n$ then print 0
else print i
```

If the input is the list {1, 3, 5, 5, 4, 4, 4, 6, 9}, what is the output?

A. 5 B. 1 C. 3

D. 4 E. 0 F. None of the other choices is correct

Exercises

7. Use the greedy algorithm to make change using quarters, dime, nickels, pennies for 87 cents. The total number of coins obtained is

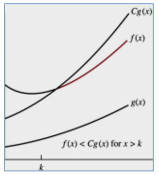
A. 5 B. 6 C. 9 D. 7 E. None of the other choices is correct

THE GROWTH OF FUNCTIONS

3.2- The Growth of Functions

- The complexity of an algorithm that acts on a sequence depends on the number of elements of sequence.
- The growth of a function is an approach that help selecting the right algorithm to solve a problem among some of them.
- Big-O notation is a mathematical representation of the growth of a function.

3.2.1-Big-O Notation



Definition: 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)) (read f(x) is big-oh of g(x)) if there are constants C and k such that |f(x)| ≤ C|g(x)| whenever x > k

Example*:* Show that f(x) = x^2 + 2x +1 is O(x^2)

Examine with x > 1 ➔ x^2 > x

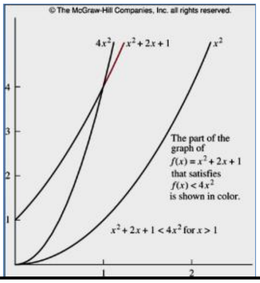
➔ f(x) = x^2 + 2x +1 < x^2 + 2x^2 + x^2

➔ f(x) < 4x^2

➔ Let g(x)= x^2

➔ C=4, k=1, |f(x)| ≤ C|g(x)|

➔f(x) is O(x^2)



Big-O Estimates for Some Important Functions

Theorem 1. Let $f(x) = a_nx^n + a_{n-1}x^{n-1} + a_1x + a_0$, where $a_0, a_1, \dots, an$ are real numbers, then $f(x)$ is $O(x^n)$.

Proof.

If $x > 1$

$|f(x)| \ = |a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0|$

$\leq |a_nx^n| + |a_{n-1}x^{n-1}| + \cdots + |a_1x| + |a_0|$ triangle inequality

$\leq x^n(|a_n| + |a_{n-1}x^{n-1}/x^n| + \cdots + |a_1x/x^n| + |a_0/x^n|)$

$\leq x^n(|a_n| + |a_{n-1}/x| + \cdots + |a_1/x^n| + |a_0/x^n|)$

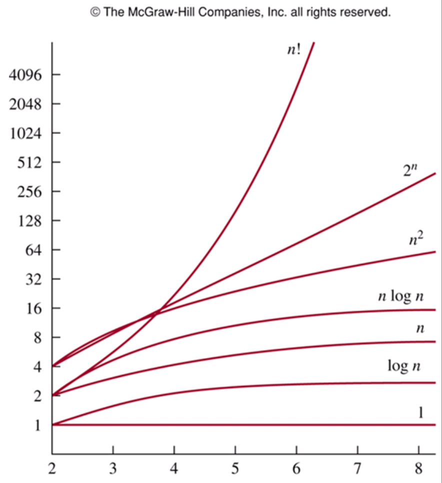
$\leq x^n(|a_n| + |a_{n-1}| + \cdots + |a_1| + |a_0|)$

Let $C = |a_n| + |a_{n-1}| + \cdots + |a_1| + |a_0|$, we have

$$|f(x)| \ \leq \ Cx^n$$

so f(x) is O(x^n)

The Growth of Combinations of Functions



The Growth of Combinations of Functions

Theorem 2. Suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$. Then $(f_1 + f_2)(x)$ is $O(\max(|g_1(x)|, |g_2(x)|))$.

Corollary 1. Suppose that $f_1(x)$ and $f_2(x)$ are both $O(g(x))$. Then $(f_1 + f_2)(x)$ is $O(g(x))$.

Theorem 3. Suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$. Then $(f_1 f_2)(x)$ is $O(g_1(x) g_2(x))$.

3.2.2- Big-Omega and Big-Theta Notation

- Big-O does not provide the lower bound for the size of f(x)
- Big-Ω, Big-Θ were introduced by Donald Knuth in the 1970s
- Big-Ω provides the lower bound for the size of f(x)
- Big-Θ provides the upper bound and lower bound on the size of f(x)

Big-Omega

Definition. 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 $\Omega\big(g(x)\big)$ if there are positive constants C and k such that

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

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

Example. $f(x) = 8x^3 + 5x^2 + 7$, $g(x) = x^3$

It is easy to see that $8x^3 + 5x^2 + 7 \geq 8x^3$ for $\forall x > 0$

$\Rightarrow g(x) = x^3$ is $O(8x^3 + 5x^2 + 7)$

Remark. $f(x)$ is $\Omega\big(g(x)\big)$ if and only if $g(x)$ is $O\big(f(x)\big)$

Big-Theta

Definition. 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 $\boldsymbol{\Theta\big(g(x)\big)}$ if $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega\big(g(x)\big)$. When $f(x)$ is $\Theta\big(g(x)\big)$ we say that f is big-Theta of $g(x)$, that $f(x)$ is of **order** $g(x)$, and that $f(x)$ and $g(x)$ are of the *same order*.

In other word, $f(x)$ is $\Theta\big(g(x)\big)$ if and only if there are real numbers C_1 and C_2 and a positive real number k such that

$C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$, whenever $x > k$

Example. $f(x) = 3x^2 + 8x\log x$, $g(x) = x^2$

$0 \leq 8x\log x \leq 8x^2 \Rightarrow 3x^2 + 8x\log x \leq 11x^2$ for $x > 1$

$\Rightarrow f(x)$ is $O\big(g(x)\big)$

Otherwise, $g(x)$ is $O\big(f(x)\big)$

Therefore, $f(x)$ is $\Theta\big(g(x)\big)$

Big-Theta

Theorem. Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, where $a_0, a_1, \ldots, a_n$ are real numbers with $a_n \neq 0$. Then $f(x)$ is of order x^n.

Exercises

1. Determine whether each of these function is $O(x)$

a) $f(x) = 10$ **b)** $f(x) = 3x + 7$
c) $f(x) = x^2 + x + 1$ **d)** $f(x) = 5\log x$
e) $f(x) = \lfloor x \rfloor$ **f)** $f(x) = \lceil x/2 \rceil$

2. Determine whether each of these function is $O(x^2)$

a) $f(x) = 17x + 11$ b) $f(x) = x\log x$ c) $f(x) = 2^x$

3. Find the least integer n such that $f(x)$ is $O(x^n)$ for each of these functions

a) $f(x) = 2x^3 + x^2\log x$ c) $f(x) = (x^4 + x^2 + 1)/(x^3 + 1)$
b) $f(x) = 3x^3 + (\log x)^4$ d) $f(x) = (x^4 + 5\log x)/(x^4 + 1)$

4. Show that $x^2 + 4x + 17$ is $O(x^3)$ but that x^4 is not $O(x^3)$
5. Show that $3x^4 + 1$ is $O(x^4/2)$ and $x^4/2$ is $O(3x^4 + 1)$

Exercises

6. Determine whether x^3 is $O(g(x))$ for each of these functions $g(x)$

a) $g(x) = x^2$ d) $g(x) = x^2 + x^4$
b) $g(x) = x^3$ e) $g(x) = 3^x$
c) $g(x) = x^2 + x^3$ f) $g(x) = x^3/2$

7. Arrange the functions $\sqrt{n}$, $1000\log n$, $n\log n$, $2n!$, 2^n, 3^n, $n^2/1{,}000{,}000$ in a list so that each function is big-O of the next function.
8. Show that each of these pairs of functions are of the same order.

a) $3x + 7, x$ c) $\log(x^2 + 1), \log_2 x$
b) $2x^2 + x - 7, x^2$ d) $\log_{10} x, \log_2 x$

Exercises

9. Find the least integer n such that $f(x) = O(x^n)$

$$f(x) = \frac{(x^3 + 7x^2 + 3)^2}{(2x + 1)^2}$$

A. 2
B. 4
C. 3
D. 1
E. None of the other choices is correct

10. Prove that $2n^3 - 7n + 1 = \Omega(n^3)$
11. Prove that $7x^2 + 1 = \Theta(x^2)$

Exercises

12. Give as good a big-O estimate as possible for each of these functions.

a) $(n^2 + 8)(n + 1)$ b) $(n\log n + n^2)(n^3 + 2)$

c) $(n! + 2^n)(n^3 + \log(n^2 + 1))$

COMPLEXITY OF ALGORITHMS

3.3- Complexity of Algorithms

- Computational complexity = Time complexity + space complexity.
- Time complexity can be expressed in terms of the number of operations used by the algorithm.
  - Average-case complexity (the average number of operations used to solve the problem over all possible inputs of a given size).
  - Worst-case complexity (how many operations an algorithm requires to guarantee that it will produce a solution).
- Space complexity will not be considered.

Worst-case

Example 1. Describe the time complexity (worst-case) of the algorithm for finding the largest element in a set:

```
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
```

i	Number of comparisons
2	2
3	2
…	2
n	2
n+1	1, max < a_i is omitted

2(n-1) +1 = 2n-1 comparisions

Time Complexity is $\Theta(n)$

Average-case

Example 2. Describe the average-case time complexity of the linear-search algorithm:

```
Procedure linearsearch(x: integer, $a_1, a_2, \ldots, a_n$ :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
return location
```

Average-case

Example 2 (cont.). Describe the average-case time complexity of the linear-search algorithm:

If x is the first term a_1 of the list, three comparisons are needed, one $i \le n$ to determine whether the end of the list has been reached, one $x \ne a_i$ to compare x and the first term, and one $i \le n$ outside the loop. If x is the second term a_2 of the list, two more comparisons are needed, so that a total of five comparisons are used. In general, if x is the ith term of the list ai, two comparisons will be used at each of the i steps of the loop, and one outside the loop, so that a total of 2i + 1 comparisons are needed. The average number of comparisons used equals

$$\frac{3+5+7+\cdots+(2n+1)}{n} = \frac{2(1+2+3+\cdots+n)+n}{n}$$
$$= \frac{2[n(n+1)/2]}{n} + 1 = n+2$$

which is $\Theta(n)$.

Some terminologies used for the Complexity of Algorithms

Complexity	Terminology
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	$n \log n$ complexity
$\Theta(n^b)$, $b \ge 1$, integer	Polynominal complexity
$\Theta(b^n)$, $b>1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

Exercises

1. Give a big-O estimate for the number of operations (where an operation is an addition or a multiplication) used in this segment of an algorithm.

```
t := 0
for i := 1 to 3
    for j := 1 to 4
        t := t + ij
```

Exercises

2. Give a big-O estimate for the number of operations, where an operation is an addition or a multiplication, used in this segment of an algorithm (ignoring comparisons used to test the conditions in the while loop)

```
i := 1
t := 0
while i ≤ n
    for j := 1 to 4
        t := t + i
        i := 2i
```

Exercises

3. The conventional algorithm for evaluating a polynomial $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ at $x = c$ can be expressed in pseudocode by

```
procedure polynomial(c, a_0, a_1, ..., a_n: real numbers)
    power := 1
    y := a_0
    for i := 1 to n
        power := power * c
        y := y + a_i * power
    return y {y = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0}
```

where the final value of y is the value of the polynomial at x = c.

a) Evaluate $3x^2 + x + 1$ at $x = 2$ by working through each step of the algorithm showing the values assigned at each assignment step.

b) Exactly how many multiplications and additions are used to evaluate a polynomial of degree n at x = c? (Do not count additions used to increment the loop variable.)

Thanks