Section 7.3
Analysis of Algorithms

Time Complexity

- The time complexity of an algorithm is a function $f: \boldsymbol{Z}^+ \rightarrow \boldsymbol{Z}^+$ that takes the size of an algorithm's input and returns the number of atomic operations that the algorithm executes when processing an input of that size

- An atomic operation is a basic operation such as an assignment, arithmetic operation, comparison, or return statement.

Time Complexity

- Example: Summing a sequence of numbers

  Input: $a_1, a_2, a_3, \ldots, a_n$

```
sum := 0
for i:= 1 to n
   sum := sum + a_i
end-for
return sum
```

Time Complexity

- Example: Summing a sequence of numbers

  Input: $a_1, a_2, a_3, \ldots, a_n$

```
sum := 0                          1 operation
for i:= 1 to n                    2 operations: increment and test
  sum := sum + a_i                2 operations: addition and assignment   } done n times
end-for
return sum                        1 operation
```

Time Complexity

- Example: Summing a sequence of n numbers

  Input: $a_1, a_2, a_3, \ldots, a_n$

```
sum := 0
for i:= 1 to n
   sum := sum + a_i
end-for
return sum
```

1 operation

2 operations: increment and test

2 operations: addition and assignment

} done n times

1 operation

Time complexity: $f(n) = 4n + 2$ is $\mathcal{O}(n)$

Time Complexity

- Another example: Finding the minimum of a sequence of numbers

  Input: $a_1, a_2, a_3, \ldots, a_n$

```
min := a_1
for i:= 2 to n
  if a_i< min
    min := a_i
  end-if
end-for
return min
```

Time Complexity

- Another example: Finding the minimum of a sequence of numbers

  Input: $a_1, a_2, a_3, \ldots, a_n$

```
min := a1                    1 operation
for i:= 2 to n               2 operations: increment and test
  if ai< min                 1 operation
    min := ai                1 operation
  end-if
end-for
return min                   1 operation
```

done n-1 times

Time Complexity

- Another example: Finding the minimum of a sequence of numbers

  Input: $a_1, a_2, a_3, \ldots, a_n$

```
min := a1                          1 operation
for i:= 2 to n                     2 operations: increment and test
  if ai< min                       1 operation
    min := ai                      1 operation
  end-if
end-for
return min                         1 operation
```

done n-1 times

Worst case complexity: $f(n) = 4(n-1) + 2 = 4n - 2$ is $\mathcal{O}(n)$

Time Complexity

- Yet another example: Counting duplicate pairs in a sequence

  Input: $a_1, a_2, a_3, \ldots, a_n$

```
count := 0
for i:= 1 to n
  for j:= i+1 to n
    if a_i = a_j
      count := count + 1
    end-if
  end-for
end-for
return couunt
```

Time Complexity

- Yet another example: Counting duplicate pairs in a sequence

  Input: $a_1, a_2, a_3, \ldots, a_n$

```
count := 0                          1 operation
for i:= 1 to n                        2 operations: increment and test
  for j:= i+1 to n                      2 operations: increment and test
    if a_i = a_j                        1 operation
      count := count + 1                2 operations
    end-if
  end-for
end-for
return min                          1 operation
```

1st time: n-1 times
2nd time: n-2 times
nth time: n-n times

Time Complexity

- Yet another example: Counting duplicate pairs in a sequence

  Input: $a_1, a_2, a_3, \ldots, a_n$

```
count := 0                          1 operation
for i:= 1 to n                        2 operations: increment and test
  for j:= i+1 to n                      2 operations: increment and test
    if a_i = a_j                        1 operation
      count := count + 1                2 operations
    end-if
  end-for
end-for
return min                          1 operation
```

1st time: n-1 times
2nd time: n-2 times
nth time: n-n times

Worst case time complexity: $f(n) = 2n + 2 + 5(n - 1 + n - 2 + \ \ldots + 2 + 1 + 0)$

Time Complexity

- Yet another example: Counting duplicate pairs in a sequence

  Input: $a_1, a_2, a_3, \ldots, a_n$

```
count := 0                          1 operation
for i:= 1 to n                        2 operations: increment and test
  for j:= i+1 to n                      2 operations: increment and test
    if a_i = a_j                        1 operation
      count := count + 1                2 operations
    end-if
  end-for
end-for
return min                          1 operation
```

1st time: n-1 times
2nd time: n-2 times
nth time: n-n times

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

Worst case time complexity: $f(n) = 2n + 2 + 5(n - 1 + n - 2 + \ \ldots + 2 + 1 + 0)$

Time Complexity

- Yet another example: Counting duplicate pairs in a sequence

  Input: $a_1, a_2, a_3, \ldots, a_n$

```
count := 0                          1 operation
for i:= 1 to n                        2 operations: increment and test
  for j:= i+1 to n                      2 operations: increment and test
    if a_i = a_j                        1 operation
      count := count + 1                2 operations
    end-if
  end-for
end-for
return min                          1 operation
```

1st time: n-1 times
2nd time: n-2 times
nth time: n-n times

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

Worst case time complexity: $f(n) = 2n + 2 + 5(n - 1 + n - 2 + \ \ldots + 2 + 1 + 0)$ is $\mathcal{O}(n^2)$