German University in Cairo
Faculty of Media Engineering and Technology
Prof. Dr. Slim Abdennadher
Dr. Aysha ElSafty

Introduction to Computer Science, Winter Semester 2017
Practice Assignment 7

Discussion: 02.12.2017 - 07.12.2017

Exercise 7-1 To be discussed

Given the following three algorithms for finding the larger number among three numbers.

- Algorithm 1:

```
a = eval(input())
b = eval(input())
c = eval(input())

if (a >= b) and (a >= c):
    print(a)
if (b >= a) and (b >= c):
    print(b)
if (c >= b) and (c >= a):
    print(c)
```

- Algorithm 2:

```
a = eval(input())
b = eval(input())
c = eval(input())

if (a >= b):
    if (a >= c):
        print(a)
    else:
        print(c)
else:
    if (b >= c):
        print(b)
    else:
        print(c)
```

- Algorithm 3:

```
a = eval(input())
b = eval(input())
c = eval(input())
max = a

if (b > max):
    max = b
if (c > max):
    max = c
print(max)
```

a) Compare the efficiency of the three algorithms. Please justify your answer.

Solution:

- Algorithm 1 will need 3 checks. The total number of executed instructions in worst case (`a,b` and `c` are equal) is 9
- Algorithm 2 will need 2 checks. The total number of executed instructions is 6.
- Algorithm 3 will need 2 checks. The total number of executed instructions in worst case (`a` is smaller than `b` and `b` is smaller than `c`) is 9.

b) Determine the order of magnitude of the three algorithms.

Solution:

Order of magnitude of the three algorithms: $O(1)$

Exercise 7-2 To be discussed

Given the following algorithms:

a) Algorithm 1 computes the sum from 1 to n:

```
n = eval(input())
result = 0
i = 1
while (i <= n):
    result = result+i
    i = i+1
print(result)
```

b) Algorithm 2 finds the smallest value in a list `A0,...,A(n-1)`.

```
list_A = eval(input())
k = len(list_A)
S = list_A[0]
i = 1
while (i < k):
    if (list_A[i] < S):
        S = list_A[i]
    i = i + 1
print(S)
```

c) Algorithm 3 prints out 64, 32, 16, 8, 4, 2.

```
i = 64
while (i > 1):
    print(i)
    i = int(i/2)
```

Find the total number of executed instructions of the algorithms and determine their order of magnitude (the big-O).

Solution:

a) Algorithm 1:

```
n = eval(input())           1 instruction --> executed once
result = 0                  1 instruction --> executed once
i = 0                       1 instruction --> executed once
while i < n:                1 instruction -->  n+1 repetitions
    result = result + i     1 instruction -->  n   repetitions
    i = i+1                 1 instruction -->  n   repetitions
print(result)               1 instruction
```

Total number of executed instructions: $1 + 1 + 1 + (n + 1) + 2n + 1 = 3n + 5$
Order of magnitude: $O(n)$

b) Algorithm 2:

```
list_A = eval(input())    1 instruction --> executed once
k = len(list_A)           1 instruction --> executed once
S = list_A[0]             1 instruction --> executed once
i = 1                     1 instruction --> executed once
while i< k:               1 instruction --> k  repetitions
  if list_A[i] < S:       1 instruction --> k-1 repetitions
      S = list_A[i]       1 instruction --> k-1 repetitions
  i = i+1                 1 instruction --> k-1 repetitions
print(S)                  1 instruction
```

Total number of executed instructions: $1 + 1 + 1 + 1 + k + 3(k - 1) + 1 = 4 + 4k - 2 = 4k + 2$
Order of magnitude: $O(k)$

c) Algorithm 3

```
i = 64                   1 instruction  --> executed once
while (i > 1):           1 instruction  --> 7 repetitions
    print(i)             1 instruction  --> 6 repetitions
    i = int(i/2)       1 instruction  --> 6 repetitions
```

Total number of executed instructions: $1 + 7 + 6 + 6 = 20$
Order of magnitude: $O(1)$

Exercise 7-3

Find the total number of instructions and the order of magnitude of the following algorithms

a)
```
import math

m, n = eval(input()), eval(input())
a = ((m * m) - (n * n))
b = (2 * m * n)
c = (math.sqrt((a * a) + (b * b)))

print(" The Pythagorean Triple consists of the following sides: ")
print(a, b, c)
```

Solution:

```
import math
m, n = eval(input()), eval(input())          ---> 2 instructions
a = ((m * m) - (n * n))                      ---> 1 instruction
b = (2 * m * n)                              ---> 1 instruction
c = (math.sqrt((a * a) + (b * b)))           ---> 1 instruction
print(" The Pythagorean Triple consists of") ---> 1 instruction
print(a, b, c)                               ---> 1 instruction
```

Total number of executed instructions : 7

Order of Magnitude = $O(1)$

b)
```
x, y, z = eval(input()), eval(input()), eval(input())
if (x > 0):
  average = (x + y + z)/3
   print(average)
else
   print("Bad data")
endif
```

Solution:

```
x, y, z = eval(input()), eval(input()), eval(input())   ---> 3 instructions
if (x > 0):                                             ---> 1 instruction
  average = (x + y + z)/3                               ---> 1 instruction
  print(average)                                        ---> 1 instruction
else:
   print("Bad data")
```

Total number of executed instructions : 6

Order of Magnitude = $O(1)$

c)
```
n = eval(input())
F = [1, 1]
i = 2
while i < n:
  F = F + F[i-1] + F[i-2]
  print(F[i])
  i = i + 1
```

Solution:

```
n = eval(input())                  ---> 1 instruction ---> executed once
F = [1, 1]                         ---> 1 instruction ---> executed once
i = 2                              ---> 1 instruction ---> executed once
while i < n:                       ---> 1 instruction ---> n-1 repetitions
   F = F + F[i-1] + F[i-2]         ---> 1 instruction ---> n-2 repetitions
   print(F[i])                     ---> 1 instruction ---> n-2 repetitions
   i = i + 1                       ---> 1 instruction ---> n-2 repetitions
```

Total number of executed instructions : $3 + (n-1) + 3 * (n-2) = 4n - 4$

Order of Magnitude = $O(n)$

Exercise 7-4 To be discussed

Consider the following algorithm:

```
n = eval(input())
i = 1
sum = 0
while i <= n:
  sum = sum + (1/i - 1/(i+2))
  i = i + 4
print(sum)
```

a) What is the output of the algorithm for `n = 10`? You do not need to calculate the final result.

Solution:

$\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11}$

b) Calculate the total number of executed instructions of the algorithm and give its order of magnitude.

Solution:

```
n = eval(input())              ----> 1 instruction --> executed once
i = 1                          ----> 1 instruction --> executed once
sum = 0                        ----> 1 instruction --> executed once
while i <= n:                  ----> 1 instruction --> ceiling(n/4) + 1 repetitions
  sum = sum + (1/i - 1/(i+2)) ----> 1 instruction --> ceiling(n/4) repetitions
  i = i + 4                    ----> 1 instruction --> ceiling(n/4) repetitions
print(sum)                     ----> 1 instruction --> executed once
```

Total number of executed operations = $3 \times ceiling(n/4) + 5$
The order of magnitude of the algorithm: O(n)

Exercise 7-5 To be discussed
Mystery

Consider the following algorithm:

```
n = eval(input())
m = eval(input())
y = 0

while(n>0):
  y += 1
  n -= 1

while(m>0):
  y += 1
  m -= 1
print(y)
```

a) What is the output of the algorithm for `n = 5` and `m = 3`?

Solution:

$1+1+1+1+1+1+1+1 = 8$

b) What is the functionality of the algorithm?

Solution:

The algorithms sums up the value of n and m.

c) Calculate the total number of executed instructions of the algorithm and give its order of magnitude.

Solution:

```
n = eval(input())  ----> 1 instruction --> executed once
m = eval(input())  ----> 1 instruction --> executed once
y = 0              ----> 1 instruction --> executed once

while(n>0):    ----> 1 instruction --> executed n + 1 times
  y += 1       ----> 1 instruction --> executed n times
  n -= 1       ----> 1 instruction --> executed n times

while(m>0):    ----> 1 instruction --> executed m + 1 times
  y += 1       ----> 1 instruction --> executed m times
```

```
  m -= 1       ----> 1 instruction --> executed m times
print(y)       ----> 1 instruction --> executed 1 times
```

Total number of executed operations $= 3 \times n + 3 \times m + 6$
The order of magnitude of the algorithm: O(n+m)

Exercise 7-6 To be discussed

Find the total number of instructions and the order of magnitude of the following algorithm:

```
list_A = eval(input())
n = len(list_A)
i = 0
while (i < int(n/2)):
  tmp = list_A[i]
  list_A[i] = list_A[n-(i+1)]
  list_A[n-(i+1)] = tmp
  i = i + 1

print(list_A)
```

Solution:

```
list_A = eval(input())       ----> 1 instruction --> executed once
n = len(list_A)              ----> 1 instruction --> executed once
i = 0                        ----> 1 instruction --> executed once
while (i < int(n/2)):        ---->1 instruction --> int(n/2)+1 repetitions
  tmp = list_A[i]            ---->1 instruction --> int(n/2) repetitions
  list_A[i] = list_A[n-(i+1)]  ---->1 instruction --> int(n/2) repetitions
  list_A[n-(i+1)] = tmp        ---->1 instruction --> int(n/2) repetitions
  i = i + 1                  ---->1 instruction --> int(n/2) repetitions

print(list_A)                ----> 1 instruction --> executed once
```

Total number of executed instructions $= 5 + 5 \times int(n/2)$
The order of magnitude of the algorithm: O(n)

Exercise 7-7 To be discussed

Find the total number of instructions and the order of magnitude of the following algorithm and determine the best and worst case scenarios:

```
list_A = eval(input("Enter List"))
n = len(list_A)
x = eval(input("Enter Number"))
i = n - 1
c = 0
while(i>=0):
  if(list_A[i] < x):
    list_A[i] = 0
  else:
    list_A[i] = 1
    c +=1
  i-=1
print(list_A,", ",c)
```

Solution:

```
list_A = eval(input("Enter List"))          ----> 1 instruction --> executed once
n = len(list_A)                             ----> 1 instruction --> executed once
x = eval(input("Enter Number"))             ----> 1 instruction --> executed once
i = n - 1                                   ----> 1 instruction --> executed once
c = 0                                       ----> 1 instruction --> executed once
while(i>=0):                                ---> 1 instruction ---> n+1 repetitions
  if(list_A[i] < x):                        ---> 1 instruction ---> n repetitions
    list_A[i] = 0
  else:
    list_A[i] = 1                           ---> 1 instruction ---> n repetitions
    c +=1                                   ---> 1 instruction ---> n repetitions
  i-=1                                      ---> 1 instruction ---> n repetitions
print(list_A,", ",c)                        ----> 1 instruction --> executed once
```

Best case scenario: all the elements in the list are less than x.

Total number of executed instructions $= 7 + 4 \times n$
Order of magnitude: $O(n)$

Worst case scenario: all the elements in the list are greater or equal to x.

Total number of executed instructions $= 7 + 5 \times n$
Order of magnitude: $O(n)$

Exercise 7-8

Find the total number of instructions and the order of magnitude of the following algorithm and determine the best and worst case scenarios:

```
a = eval(input())
b = eval(input())
m = len(a)
k = len(b)
c = []
i = 0
if(m <= k):
    n = m
else:
    n = k
while(i < n):
    c = c + a[i]
    c = c + b[i]
    i += 1
if(i < m):
    while(i < m):
      c = c + a[i]
      i += 1
elif(i < k):
    while(i < k):
      c = c + b[i]
      i += 1
print(c)
```

Solution:

```
a = eval(input())        ----> 1 instruction --> executed once
b = eval(input())        ----> 1 instruction --> executed once
m = len(a)               ----> 1 instruction --> executed once
```

```
k = len(b)                ----> 1 instruction --> executed once
c = []                    ----> 1 instruction --> executed once
i = 0                     ----> 1 instruction --> executed once
if(m <= k):               ----> 1 instruction --> executed once
    n = m                 ----> 1 instruction --> executed once
else:
    n = k
while(i < n):             ----> 1 instruction -->  n+1 repetitions
    c = c + a[i]      ----> 1 instruction -->  n repetitions
    c = c + b[i]      ----> 1 instruction -->  n repetitions
    i += 1            ----> 1 instruction -->  n repetitions
if(i < m):                ----> 1 instruction --> executed once
    while(i < m):
      c = c + a[i]
      i += 1
elif(i < k):            ----> 1 instruction --> executed once
    while(i < k):       ----> 1 instruction -->  (k-n)+1 repetitions
      c = c + b[i]      ----> 1 instruction -->  (k-n) repetitions
      i += 1            ----> 1 instruction -->  (k-n) repetitions
print(c)                ----> 1 instruction --> executed once
```

Best case scenario: both lists have the same length.

Total number of executed instructions $= 12 + 4 \times n$
Order of magnitude: $O(n)$

Worst case scenario: the second list has a larger length.

Total number of executed instructions $= 13 + 3 \times k + 1 \times n$
Order of magnitude: $O(x)$, where $x = n + k$