Assignment 1

Data Structure and Algorithms

COMP 352

Section AA

By

Chloe Hei Yu Law

ID: 40173275

Concordia University

May 2021

Question 1

a) Pseudocode O(1)

```
even ← 0
odd ← 0
cntEven ← 0
cntOdd ← 0
for i ← 0 to n - 1 do                          O(n)
        if A[i] > 0 then
                if A[i] % 2 == 0 then
                        even ← even +  A[i]
                        increment cntEven
                else
                        odd ← odd + A[i]
                        increment cntOdd
return (even, cntEven, odd, cntOdd)            O(1)
```

b) Time Complexity

O(1) + O(n) + O(1)

= O(n)

c) Space complexity

2n bytes of memory to store array variable ‘A[]’

2 bytes of memory for integer ‘n’

8 bytes of memory for local integer variables ‘even’, ‘odd’, ‘cntEven’, ‘cntOdd’ (2 bytes each)

8 bytes return value

‘2n + 18’ bytes of memory to complete its execution

Linear space complexity of O(n)

Question 2

a) $45\ n^2 + 28\ n + 752$ is $\Omega(n)$

$f(n) = 45\ n^2 + 28\ n + 752$ $g(n) = n$

By definition of big-Ω notation, the f(n) is big-Ω of g(n) when $|f(n)| \geq c|g(n)|$ for $n > k$.

The values of c and k are positive constants

Let $c = 1$ and $n = 2$

$cg(n) = 1 \times 2 = 2$

$f(n) = 45(2)^2 + 25(2) + 752 = 982$

$982 \geq 2$

The condition is validated therefore the statement is true

Question 2

b) $256\ n + 8\ n \log n$ is $\Theta(\log n)$

$f(n) = 256\ n + 8\ n \log n$ $\qquad\qquad$ $g(n) = \log n$

The $f(n)$ is big-Θ of $g(n)$ if $f(n)$ is both $O(g(n))$ and $\Omega(g(n))$

Check if $f(n)$ is $O(g(n))$

$f(n)$ is big-O of $g(n)$ when $|f(n)| \leq c|g(n)|$ for $n \geq k$

The values of c and k are positive constants

Let $c = 2$ and $n = 2$

$cg(n) = 2\log 2$

$f(n) = 256(2) + 8(2)\log 2 = 512 + 16\log 2$

$f(n)$is not $\leq$ to $cg(n)$

Therefore, the statement is false

Question 2

c) $n^{0.8} + \log n$ is $O(\log n)$

$f(n) = n^{0.8} + \log n$ $g(n) = \log n$

$f(n)$ is big-O of $g(n)$ when $|f(n)| \le c|g(n)|$ for $n \ge k$

The values of c and k are positive constants

Let $c = 2$ and $n = 2$

$f(n) = 2^{0.8} + \log 2$

$cg(n) = 2\log 2$

$f(n)$is not $\le$ to $cg(n)$

Therefore, the statement is false

Question 2

d) $2n^2 \log n + n^3$ is $\Theta(\log n)$

$f(n) = 2n^2 \log n + n^3$ $g(n) = \log n$

f(n) is big-Θ of g(n) if f(n) is both O(g(n)) and Ω(g(n))

Check if f(n) is O(g(n))

f(n) is big-O of g(n) when $|f(n)| \leq c|g(n)|$ for $n \geq k$

The values of c and k are positive constants

Let c = 2 and n = 2

$f(n) = 2(2)^2\log 2 + 2^3 = 8\log 2 + 8$

$cg(n) = 2\log 2$

f(n)is not $\leq$ to cg(n)

Therefore, the statement is false

Question 2

e) $4\ n \log^2 n + 3\ n^2 \log n$ is $O(\log n)$

$f(n) = 4\ n \log^2 n + 3\ n^2 \log n$ $\qquad$ $g(n) = \log n$

$f(n)$ is big-O of $g(n)$ when $|f(n)| \leq c|g(n)|$ for $n \geq k$

The values of c and k are positive constants

Let $c = 2$ and $n = 2$

$f(n) = 4(2)\log^2 2 + 3(2)^2 \log 2 = 8\log^2 2 + 12\log 2$

$cg(n) = 2\log 2$

$f(n)$is not $\leq$ to $cg(n)$

Therefore, the statement is false

Question 2

f) $n^7 + 0.00000001n^6$ is $\Omega(n^6)$

$f(n) = n^7 + 0.00000001n^6$

$g(n) = n^6$

$n^7 + 0.00000001n^6 \geq 0.00000001n^6$

Therefore , f(n) is big-Ω of $\Omega(n^6)$. The statement is true.

Question 3

a) Time complexity function

```
Algorithm arraySpecialSum(A, n)
        currentMax ← A[0]                        ← n
        for i ← 1 to n – 1 do                     ← n^2 - 1
                if A[i] > currentMax then
                        currentMax ← A[i]
                { increment counter i }
        currentMaxOccurence = 0                  ← 1
        for i ← 0 to n – 1 do                     ← n^2 + 2n - 2
                if A[i] == currentMax then
                        { increment currentMaxOccurence}
                { increment counter i }
        specialSum = 0                           ← 1
        for i ← 0  to n – 1 do                    ← n^2 + 4n - 2
                for j ← 1 to currentMaxOccurence do
                        { specialSum = specialSum + A[i]}
                        { increment counter j }
                { incement counter i}
        return specialSum                        ← 1
```

$f(n) = 3n^2 + 7n - 2$

b) Time complexity algorithm is $O(n^2)$

c) Space complexity algorithm is $O(n^2)$

Question 3

d) There is a way to improve this algorithm to achieve a better time complexity and still return the same specialSum value. The time complexity of the modified algorithm will be O(log n) because a binary search is used to cut down the complexity by half every time an index is searched.

```
Algorithm arraySpecialSum(A, n)
        maxValue ← findMax(A, 0, A.length() – 1)
        occurenceMaxValue ← findMaxOccurence(A, maxValue, A.length())
        specialSum ← findSpecialSum(A, n, occurenceMaxValue, i, j)
        return specialSum
```

Algorithm : findMax(A, start, end)

Inputs: array A of integers, integer value start for the starting index of the array to search, integer value end for the ending index of the array to search

Output: return integer of the maximum value

```
        if (end >= start) then
                mid ← (start + end) / 2
                if (mid < high and A[mid + 1] < A[mid]) then
                        return A[mid]
                if (mid > start and A[mid] < A[mid – 1] then
                        return A[mid – 1]
                if (A[start] > A[end]) then
                        return findMax(A, start, mid – 1)
                else
                        return findMax(A, mid+ 1, end)
```

Question 3

Algorithm : findMaxOccurence(A, x, n)

Inputs: array A of integers, integer value x for the current max value, integer value n for array length

Output: return integer of number of occurrence of the maximum value

```
i ← first(A, 0, A.length() – 1, x, n)
if (i == -1) then
        return i
j ← last(A, i, A.length() – 1, x, n)
return j – i + 1
```

Algorithm : first(A, start, end, x, n)

Inputs: array A of integers, integer value start for the starting index of the array to search, integer value end for the ending index of the array to search, integer value x for the max value, integer value n for array length

Output: return integer of the first occurrence of x in the array

```
if ( end >= start) then
        mid ← (start + end) / 2
        if ((mid == 0 or x > A[mid – 1]) and A[mid] == x) then
                return mid
        else if (x > A[mid])
                return first(A, mid + 1, end, x, n)
        else
                return first(A, start, mid – 1, x , n)
return -1
```

Question 3

Algorithm : last(A, start, end, x, n)

Inputs: array A of integers, integer value start for the starting index of the array to search, integer value end for the ending index of the array to search, integer value x for the max value, integer value n for array length

Output: return integer of the last occurrence of x in the array

```
    if ( end >= start) then
        mid ← (start + end) / 2
        if ((mid == n - 1 or x < A[mid + 1] and A[mid] == x) then
            return mid
        else if (x < A[mid])
            return last(A, start, mid – 1, x, n)
        else
            return last(A, mid + 1, end, x , n)
    return -1
```

Algorithm: findSpecialSum(A, n, x, i, j)

Inputs: array A of integers, integer value n of the array, integer value x for the max value, integer value i for counter, integer value j for counter

Output: return integer of the specialSum

```
    if (i < n) then
        if (j < x) then
            specialSum = specialSum + A[i]
            findSpecialSum(A, n, x, i, ++j)
        else
            findSpecialSum(A, n, x, ++i, 0)
    return specialSum
```