COMP 6481: Programming And Problem Solving

Winter 2024

ASSIGNMENT 01 PART 01

Date of Submission: February 16, 2024

Submitted by:

Sanjana Sayeed

ID: 40237987

Question 1

a) Write an algorithm as a pseudo code (not a program!) that would implement the encryption technique of Polyalphabetic Cipher, and returns the cipher text for a given plaintext input. Determine the input and output of your algorithm.
b) Write an algorithm as a pseudo code (not a program!) that would implement the decryption technique of Polyalphabetic Cipher, and returns the plain text for a given cipher text input. Determine the input and output of your algorithm.
c) What is the time complexity of each of your algorithms, in terms of Big-O?
d) What is the space complexity of each of your algorithms, in terms of Big-O?

Pseudo Code for encryption

```
class Encryption
        method encryption(String plaintext, int  key)
           cipherText = empty string
           for i from 0 to length of plaintext:
              charPresent = character at index i in plaintext
              encryptedCharacter = shift(charPresent, key + (i % 3))
              append  encryptedCharacter to cipherText
           end for
           return ciphertext
        end method

        method shift(character, shift):
           char[] alphabets = array of lowercase letters from 'a' to 'z'
           index = (findIndex(alphabets, character) + shift) mod 26
           if index < 0 then
              index = index + 26
           end if
           return alphabest[index]
        end method

        method findIndex(arr, targetedCharacter)
           for i from 0 to length of array:
              if array[i] == targetedCharacter then
                 return i
             end if
           return -1
           end for
        end method
        main method
          plainText = "themthenthey"
          key = 10
          ciphertext = encryption(plaintext, key)
          print "Ciphertext: " + ciphertext
        end main method
end class
```

Input: "themthenthey"
Output: ”dsqwetoyfrpk”

Big O Time Complexity

In the first function it is iterating through the length of plain text. So the time complexity of it will be 0(n). While looping the method it calls search method which in turn calls findIndex method . This method basically iterates through the 26 alphabets and searches for the targeted alphabet so the time complexity will be 0(26). We can consider 26 to be constant . So the time complexity will be O(n) where n will be the length of the plain text.

Big O Space Complexity

The input size (plaintext length) and the fixed size of the alphabet array are the key factors influencing space complexity. As a result, the space complexity is O(n), where n is the plaintext length, and the ciphertext length is also the same. Furthermore, extra space is used for variables such as cipherText, charPresent, encryptedCharacter, index but these demand constant space per iteration and contribute nothing to overall space complexity. As a result, they can be regarded as negligible.

Pseudo Code for decryption

```
class Decryption
        method decryption(String cipherText, int key):
            StringBuilder plainText;
            for i = 0 to length(cipherText) - 1:
                characterPresent = cipherText[i]
                decryptedChar = shift(characterPresent, key + (i % 3))
                append decryptedChar to plaintext
            end for
            return plaintext
        end method

        method shift(character, shift):
            alphabet = "abcdefghijklmnopqrstuvwxyz"
            index = findIndex(alphabet, character)
```

```
              index = (index - shift + 26) % 26
                    if index < 0 then
                      index = index + 26
                    end if
              return alphabet[index]
             end method

            method findIndex(arr, targetedCharacter):
              for i = 0 to length(arr) - 1:
                if arr[i] == targetedCharacter then
                  return i
                end if
              return -1
              end method

            main method
              plainText = "dsqwetoyfrpk"
              key = 10
              ciphertext = encryption(plaintext, key)
              print "Ciphertext: " + ciphertext
            end main method
end class
```

Input: "dsqwetoyfrpk”
Output: “themthenthey”

Big O Time Complexity

In the first function it is iterating through the length of plain text. So the time complexity of it will be 0(n). While looping the method it calls search method which in turn calls findIndex method . This method basically iterates through the 26 alphabets and searches for the targeted alphabet so the time complexity will be 0(26). We can consider 26 to be constant . So the time complexity will be O(n) where n will be the length of the plain text.

Big O Space Complexity

The input size (plaintext length) and the fixed size of the alphabet array are the key factors influencing space complexity. As a result, the space complexity is O(n), where n is the plaintext length, and the ciphertext length is also the same. Furthermore, extra space is used for variables such as cipherText, charPresent, encryptedCharacter, index but these demand constant space per iteration and contribute nothing to overall space complexity. As a result, they can be regarded as negligible.

Question 2

Given an unsorted array A of integers of any size, n ≥ 1, and an integer value x, write an algorithm as a pseudo code (not a program!) that would find out the sum of all elements in the array that have values bigger than x, as well as the sum of all elements in the array that have values smaller than x. For instance, assume that array A is as follows:
10 14 3 9 22 35 92 5 9 64
Given x = 9, the algorithm would find the two sums as 237 and 8. Your algorithm must handle possible special cases. Finally, the algorithm must use the smallest auxiliary/additional storage to perform what is needed.
a) What is the time complexity of your algorithm, in terms of Big-O?
b) What is the space complexity of your algorithm, in terms of Big-O?

Pseudo Code

```
class MaxMinUnsortedArray
   arr[n];
   sumMax ;
   sumMin ;
   target ;
   for i = 0 to arr.length - 1
      If arr[i] > target then
         sumMax = sumMax + arr[i]
      else If arr[i] < target then
         sumMin = sumMin + arr[i]
      end If
   end for

   Output sumMin
   Output sumMax
end class
```

Big O Time Complexity

The provided code consists of a single loop that iterates over each element of the input array arr. Within the loop, constant time operations include comparison (arr[i] > target and arr[i] < target) and addition (sumMax = sumMax + arr[i] and sumMin = sumMin + arr[i]). As a result, the code's time complexity is O(n), where n is the number of entries in the input array arr.

Big O Space Complexity

The code's space complexity is O(1) since it requires the same amount of extra space regardless of the size of the input array. There are no data structures in use that scale with the size of the input array. The variables sumMax, sumMin, target, and arr all have defined sizes. Thus, the space complexity is constant, denoted as O(1).

Question 3

Given the same problem as in Question 2; however, the array is sorted. write an algorithm as a pseudo code (not a program!) that would find out the sum of all elements in the array that have values bigger than x, as well as the sum of all elements in the array that have values smaller than x.
a) Is the time complexity of your algorithm different than the one of Question 1? If so, what is it in terms of Big-O, and why is it different?
b) What is the space complexity of your algorithm, in terms of Big-O? Is there any change from the space complexity in Question 1? Explain your answer.

Pseudo Code

```
method findSumSortedArray(arr, x):
   n = length of arr
   xIndex = binarySearch(arr, x)

   sumMin = 0
   sumMax = 0

   // Calculate sum of elements smaller than x
   for i = 0 to xIndex - 1
      sumMin += arr[i]
  end for

   // Calculate sum of elements larger than x
   for i = xIndex + 1 to n - 1
      sumMax += arr[i]
   end for

   print("Sum of elements smaller than", x, ":", sumMin)
   print("Sum of elements larger than", x, ":", sumMax)

method binarySearch(arr, x):
   low = 0
   high = length of arr - 1

   while low <= high:
      mid = low + (high - low) / 2
```

```
        if arr[mid] == x
            return mid
    end if
        else if arr[mid] < x
            low = mid + 1
         End else if
        else
            high = mid - 1
        end else
    return low
end method

main method
    int[] arr
    int target
    findSumSortedArray(arr, target)
end main method
```

Big O Time Complexity

The binary search to identify the index closest to x runs in O(log n) time. Calculating the sum of items less than and greater than x requires O(n/2) time, which is equivalent to O(n) time. Thus, the overall time complexity of this optimised technique is O(log n), which is faster than the prior O(n) time complexity. So the time complexity of a sorted array is less than the unsorted array if binary search is implemented.

Big O Space Complexity

The space complexity remains O(1) since there is no major change in the use of additional space.

Question 4

a) Given an array of integers of any size, $n \geq 1$, write an algorithm as a pseudo code (not a program!) that would reverse every two consecutive elements of the left half of the array (i.e. reverse elements at index 0 &1, at index 2 & 3, etc.). Additionally, for the right half of the array, the algorithm must change the second element of every two consecutive elements to have the sum of the two elements. For instance, if the right half starts at index 14, and the values at index 14 & 15 are 72 and 11, then the value at index 15 is changed to 83. If the given array is of an odd size, then the exact middle element should not be manipulated by the algorithm in any way. Finally, your algorithm must use the smallest auxiliary/additional storage to perform what is needed.

b) What is the time complexity of your algorithm, in terms of Big-O?
c) What is the space complexity of your algorithm, in terms of Big-O?
Hint: Please read the question carefully!

Pseudo Code

```
class ReverseSum
            int[] arr;
            int low;
            int high;
           if arr.length even then
              midPosition to ((low + high) / 2) + 1
           else
              (low + high) / 2
          initialise i to 0.
         while i < midPosition then
                Swap arr[i] and arr[i+1]
                Increment i by 2
           end while
         initialise k to midPosition.
        while k < length of arr - 2:
               sum = arr[k] + arr[k+1]
               arr[k+1] = sum
               Increment k by 2
        Print the elements of arr
class end
```

Big O Time Complexity

Swapping adjacent members to the middle position takes O(n/2) time, where n is the array's length. Calculating the sum of adjacent pairs from midPosition to the second last element takes O(n/2) time.Printing the array requires O(n) time. Thus, the total time complexity is O(n).

Big O Space Complexity

The space complexity is O(1) because the algorithm allocates a fixed amount of extra space regardless of the size of the input array.