ABV- Indian Institute of Information Technology & Management Gwalior
Department of Computer Science and Engineering
Data Structures

Assignment # 1 — Due date: Monday, 1-May-2023

1. Consider the pseudo of the following algorithms:

Algorithm 1: to find the maximum element in an array of integers, $A[n]$

```
1  max = A[0];
2  for i = 1 to n − 1 do
3  |   if A[i] > max then
4  |   |   max = A[i];
5  |   end
6  end
7  return max;
```

Algorithm 2: to find the index of the first occurrence of a given element in an array of integers, $A[n]$

```
1  found = false;
2  i = 0;
3  while i < n and not found do
4  |   if A[i] == x then
5  |   |   found = true;
6  |   else
7  |   |   i = i + 1;
8  |   end
9  |   if found then
10 |   |   return i;
11 |   else
12 |   |   return −1;
13 |   end
14 end
15 return max;
```

a) What is the loop invariant condition for both the algorithms

b) Prove the correctness of both the algorithms.

2. What value is returned by the following functions? Express your answer as a function of n . Give the worst-case running time using Big Oh notation.

Algorithm 3: prestiferous(n)

```
1  r = 0;
2  for i = 1 to n do
3  |   for j = 1 to i do
4  |   |   for k = j to i + j do
5  |   |   |   for l = 1 to i + j − k do
6  |   |   |   |   r = r + 1;
7  |   |   |   end
8  |   |   end
9  |   end
10 end
11 return r;
```

Algorithm 4: conundrum(n)

```
1  r = 0;
2  for i = 1 to n do
3  |   for j = i + 1 to n do
4  |   |   for k = i + j − 1 to n do
5  |   |   |   r = r + 1;
6  |   |   end
7  |   end
8  end
9  return r;
```

3. a) Give asymptotic upper bound for the following recurrences:

$$\text{T(n)} = \begin{cases} O(1) & \text{if } n \le 2 \\ T(7n/10) + n & \text{otherwise} \end{cases} \qquad \text{T(n)} = \begin{cases} O(1) & \text{if } n \le 2 \\ T(n-2) + n^2 & \text{otherwise} \end{cases}$$

$$\text{T(n)} = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(n/3) + nlgn & \text{otherwise} \end{cases} \qquad \text{T(n)} = \begin{cases} O(1) & \text{if } n = 1 \\ T(n/2) + T(n/4) + TT(n/8) + n & \text{otherwise} \end{cases}$$

b) Consider the following recurrence relation:

$$\text{T(n)} = \begin{cases} c & \text{if } n = 1 \\ aT(n/b) + O(n^d) & \text{otherwise} \end{cases}$$

where $a \ge 1, b > 1, d \ge 0$. If $a > b^d$, then show that $T(n) = O(n^{log_b^a})$.

4. Consider the following recursive procedures:

a) Write down the recurrence formulae for time complexity for both the procedures.

b) Give the asymptotic notations for time complexity.

Algorithm 5: power(x, n)

```
1 if n == 0 then
2 |   return 1;
3 if ( n%2 = 0 ) then
4 |   y = power(x, n/2);
5 |   return y * y;
6 else
7 |   y = power(x, (n − 1)/2);
8 |   return x * y * y;
9 end
```

Algorithm 6: tower_of_hanoi(n, source, destination, auxiliary)

```
1 if n == 1 then
2 |   move disk from source to destination;
3 else
4 |   tower_of_hanoi(n, source, destination, auxiliary);
5 |   move disk from source to destination;
6 |   tower_of_hanoi(n, source, destination, auxiliary);
7 end
```

5. **Array:**

   a) Given two sorted arrays of integers, write a function to merge them into a single sorted array. What is the time complexity of your algorithm?

   b) Consider a 2D array arr of dimensions $M \times N$, where each element in the array occupies k bytes of memory. Assume that the array is stored in row-major order, and the address of the first element in the array is $addr\_0$. Using this information, answer the following questions:

      b.1) What is the address of the element at row i and column j in the array?

      b.1) What is the formula for calculating the address of the element at row i and column j in the array, in terms of i, j, N, k, and $addr\_0$?

      b.1) Suppose that the array is stored in column-major order instead. How would the formula for calculating the address of the element at row i and column j change in this case?

   c) Write a function to insert an element into a sorted array of integers while maintaining the sorted order of the array. What is the worst-case time complexity of your algorithm?

6. **Stack:**

   a) Write a function to reverse a string using a stack.

   b) A common problem for compilers and text editors is determining whether the parentheses in a string are balanced and properly nested. For example, the string(())())() contains properly nested pairs of parentheses, which the strings)()(and()) do not. Give an algorithm that returns true if a string contains properly nestedand balanced parentheses, and false if otherwise. For full credit, identify the positionof the first offending parenthesis if the string is not properly nested and balanced.

   c) Convert the infix expression "$3 + 4 * 2/(1 - 5) \uparrow 2 \uparrow 3$" to postfix notation using a stack.

   d) Convert the postfix expression "$532 + *82 - +$" to infix notation using a stack.

   e) Convert the infix expression "$5 * (4 + 7) - 2$" to prefix notation using a stack.

7. **Linked List:**

   a) Write an algorithm to reverse the direction of a given singly-linked list. In other words, after the reversal all pointers should now point backwards. Your algorithm should take linear time.

   b) Write a function to detect if a linked list has a cycle in it. If the list contains a cycle, the function should return true; otherwise, it should return false.

   c) Write a function to reverse a linked list in groups of size k. The function should take a linked list and an integer k as input, and should return a new linked list where each group of k nodes has been reversed. For example, if the input list is $1- > 2- > 3- > 4- > 5- > 6- > 7$ and k is 3, the output list should be $3- > 2- > 1- > 6- > 5- > 4- > 7$. Your function should have a time complexity of $O(n)$.