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:

   a) What is the loop invariant condition for algorithm 1 and 2?

   b) Prove the correctness of the algorithm 1 and 2.

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;
```

Solution: Algorithm 1:

The loop invariant condition is that max contains the maximum value seen so far in the array A[0:i-1] for each iteration of the loop.

To prove the correctness of the algorithm, we need to show that the loop invariant holds initially, is maintained through each iteration of the loop, and implies the correctness of the algorithm when the loop terminates.

Initialization: Before the loop, max is initialized to A[0], which is the maximum value seen in A[0:0-1], so the loop invariant holds initially.

Maintenance: Assume that the loop invariant holds for some i. Then, for i+1, we compare A[i] to max and update max if A[i] is greater. Therefore, max will contain the maximum value seen so far in A[0:i], which satisfies the loop invariant for the next iteration.

Termination: When the loop terminates, max contains the maximum value in A[0:n−1], since it contains the maximum value seen so far in each iteration of the loop. Therefore, the algorithm is correct.

Algorithm 2: ‘

The loop invariant condition is that the element x has not been found in the array A[0:i−1] for each iteration of the loop.

To prove the correctness of the algorithm, we need to show that the loop invariant holds initially, is maintained through each iteration of the loop, and implies the correctness of the algorithm when the loop terminates.

Initialization: Before the loop, i is initialized to 0, and found is initialized to false. Therefore, the loop invariant holds initially, since x has not been found in A[0:0−1].

Maintenance: Assume that the loop invariant holds for some i. If A[i] is not equal to x, then x has not been found in A[0:i], so the loop invariant holds for i+1. If A[i] is equal to x, then x has been found in A[0:i], and found is set to true, so the loop invariant holds for i+1 as well.

Termination: When the loop terminates, found is true if and only if x was found in the array. If found is true, then the loop invariant implies that i is the index of the first occurrence of x. If found is false, then the loop invariant implies that x is not present in the array

2. Give asymptotic upper bound for the following recurrences:

a) T(n) $= \begin{cases} O(1) & \text{if } n \leq 2 \\ T(7n/10) + n & \text{otherwise} \end{cases}$

b) T(n)$= \begin{cases} O(1) & \text{if } n \leq 2 \\ T(n-2) + n^2 & \text{otherwise} \end{cases}$

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

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

Solution: a) $T(n) = \theta(n^4)$, b) $T(n) = \theta(n)$, c) $T(n) = \theta(n^{log_3^4})$, and d) $T(n) = \theta(n)$

3. 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 3: 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 4: 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-1, source, destination,
  |  auxiliary);
5 |  move disk from source to destination;
6 |  tower_of_hanoi(n-1, source, destination,
  |  auxiliary);
7 end
```

Solution: For Algo: 1. $T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T(n/2) + 1 & \text{otherwise} \end{cases}$

For Algo: 2. $T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n-1) + 1 & \text{otherwise} \end{cases}$

4. 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 5: 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 6: 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;
```

Solution: Algo: 5
$r = \frac{n^2(n+1)^2}{8} + \frac{n(n+1)(2n+1)}{12}$
complexity: $O(n^4)$

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) Given an array of n doubles, where each double occupies 8 bytes, what is the memory address of the ith element of the array, assuming the array is stored in row-major order and the first element is stored at address 1000?

c) Given a 2D array of size m x n, where each integer occupies 2 bytes, what is the memory address of the element at position (i, j), assuming the array is stored in column-major order and the first element is stored at address 200?

d) 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 a program 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 $O(n)$.