NANYANG TECHNOLOGICAL UNIVERSITY

SEMESTER 1 EXAMINATION 2016-2017

EE2008 / IM1001 – DATA STRUCTURES AND ALGORITHMS

November / December 2016 Time Allowed: 2½ hours

INSTRUCTIONS

1. This paper contains 4 questions and comprises 3 pages.
2. Answer ALL questions.
3. All questions carry equal marks.
4. This is a closed-book examination.
5. Unless specifically stated, all symbols have their usual meanings.

---

1. (a) Determine the asymptotic upper bound for the number of times the statement "$r = r + 1$" is executed in each of the following algorithms.

(i)

```
for i = 1 to n
    for j = 1 to i
        for k = 1 to i
            r = r + 1
```

(ii)

```
i = n
while (i ≥ 1) {
    r = r + 1
    i = i / 2
}
```

(9 Marks)

Note: Question No. 1 continues on page 2.

(b) Use Mathematical Induction to prove that the following formula is true.

$$\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1} \quad \text{, where } n \geq 1.$$

(8 Marks)

(c) For each of the following sums, determine its order of growth using the $O(g(n))$ notation with the simplest possible function $g(n)$.

(i) $\sum_{i=1}^{n} \left(i + k^2\right)$, where k is a constant

(ii) $1 + 2 + 2^2 + 2^3 + \cdots + 2^n$

(8 Marks)

2. (a) Suppose that a stack which contains a set of integers is implemented using an array. Write an algorithm that returns the number of positive integers in the stack. Ensure that the stack holds the original set of data upon completion of the algorithm. You may use additional stacks in your algorithm.

(8 Marks)

(b) An unique integer is stored in each node of a doubly-linked list which has a reference, *start*, that points to the first node of the list. Write an algorithm to delete the node with the largest integer from the list.

(7 Marks)

(c) Design a recursive algorithm which finds the node with the largest value in a singly-linked list.

(10 Marks)

3. (a) Write an algorithm which finds a node with value x in a binary search tree. Given a binary search tree of height L in which level i $(0 \leq i < L)$ of the binary tree has 2^i nodes, analyze the worst case time complexity of the designed algorithm for such a binary search tree.

(8 Marks)

Note: Question No. 3 continues on page 3.

(b) (i) Write an algorithm which computes the sum of values of all the leaves in a binary tree.

(ii) Analyze the worst case and best case time complexities of the designed algorithm.

(10 Marks)

(c) Write a recursive algorithm which counts the number of elements larger than value x but smaller than y $(x < y)$ in an array.

(7 Marks)

4. (a) Write an algorithm which counts the number of vertices having the same value x of a connected graph. You may directly use / call the depth-first-search or breadth-first-search method without giving details of these two graph search methods.

(8 Marks)

(b) Write an algorithm which sorts the array so that for a given value x, all the elements smaller than or equal to x will be put in the left part of the array and all the elements greater than x will be put in the right part of the array although the resultant array may not be in non-decreasing order.

(10 Marks)

(c) Can Prim's algorithm find a minimal spanning tree for a given weighted connected graph of which weights may not all be positive? Justify your answer.

(7 Marks)

END OF PAPER

EE2008

1 (a) (i) $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{i} 1$

$= \sum_{i=1}^{n} \sum_{j=1}^{i} i$

$= \sum_{i=1}^{n} i^2$

$= 1^2 + 2^2 + \cdots + n^2$

$< n^2 + n^2 + \cdots + n^2$

$= O(n^3)$

(ii) Assume r=r+1 execute T(n) times for size n.

$T(n) = T(n/2) + 1$

$= T(n/4) + 2$

...

$= T(1) + k$ where $2^k = n$, $k = \log_2 n$

$= 1 + \log_2 k$

$= O(\log_2 k)$

(b) For n = 1

$LHS = \sum_{i=1}^{1} \frac{1}{i(i+1)} = \frac{1}{1(1+1)} = \frac{1}{2}$ $RHS = \frac{1}{1+1} = \frac{1}{2}$

Assume n=k

$\sum_{i=1}^{k} \frac{1}{i(i+1)} = \frac{k}{k+1}$

Then for n=k+1

$LHS = \sum_{i=1}^{k+1} \frac{1}{i(i+1)} = \sum_{i=1}^{k} \frac{1}{i(i+1)} + \frac{1}{(k+1)(k+2)}$

$= \frac{k}{k+1} + \frac{1}{(k+1)(k+2)}$

$= \frac{(k+1)^2}{(k+1)(k+2)}$

$= \frac{k+1}{k+2}$

$= RHS$

Thus the formula holds for $n \geq 1$

(c) (i) $\sum_{i=1}^{n} (i+k^2) = nk^2 + \frac{(1+n)n}{2} \leq n^2 = O(n^2)$

(ii) $1 + 2 + \cdots + 2^n = \frac{1(1-2^{n+1})}{1-2} = 2^{n+1} - 1 < 2 \cdot 2^n = O(2^n)$

2(a) Input: Stack S, Stack P
Output: number of positive integers n

```
count_positive_number {
    n=0;
    while (! s. empty())
    {   if s.top() > 0 {
            n = n+1 }
        p.push(s.top())
    while (! p.empty())
    {  s.push(p.top())
       p.pop()
    }
    return n
```

(b) Input: start

```
Delete_Largest {
    p = start
    q = start
    q.data = max
    while (p.next) {
        p = p.next
        if p.data > q.data
            q = p
    }
    if q == start
        q.next.prev = null
    if q.next == null
        q.prev.next = null
    else
        q.prev.next = q.next
        q.next.prev = q.prev
}
```

(c)

```
Find_max (start) {
    if start.next == null
        return start.data
    else {
        max = Find_max(start.next)
        if max > start.data
            return max
        } else return start.data
}
```

3(a)

```
Find_x (root) {
    if (root != NULL) {
        if (root.data == x)
            return root;
        else {
            if root.data < x
                return Find_x (root.right);
            else return Find_x (root.left);
        }
    }
    else
        return NULL.
}
```

(b) i)

```
sum_of_tree (root) {
    if root == null
        return 0;
    else
        return sum_of_tree (root.left)
            + sum_of_tree (root.right)
            + root.data
}
```

ii) No matter the best or the worst case case, the complexities is O(lgn)

3(c)
```
count(A, n) {
  if n < 0
    return 0
  else
    if A[n] > x and A[n] < y
      return 1 + count(A, n-1)
    else return count(A, n-1)
}
```

4(a)
```
bfs(Graph, root) {
  num = 0
  enqueue(root);
  while(Q is not empty) {
    k = Q.dequeue
    if(k.data == x)
      num = num + 1
    for each node n that is adjacent to k {
      if (n is not in S)
        Q.enqueue(n);
    }
  }
  return num
}
```

(b)
```
algorithm( array a[n]) {
  right = n - 1;
  for(i=0; i<right; i++) {
    if a[i] < x {
      temp = a[i]
      a[i] = a[right]
      a[right] = temp
      right--;
    }
  }
}
```

(c) Yes, Prim's algorithm find a minimal spanning tree for a given weighted graph with negative weigh. Since each step of Prim's algorithm is to find the minimum of the weight no matter it is positive or negative.