NANYANG TECHNOLOGICAL UNIVERSITY

SEMESTER 1 EXAMINATION 2017-2018

EE2008 / IM1001 – DATA STRUCTURES AND ALGORITHMS

November / December 2017 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 the following algorithm.

```
for i = 1 to n
    for j = 2i to n
        for k = n to 2n
            r = r + 1
```

(5 Marks)

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

$$\sum_{i=1}^{n} i2^{i} = (n-1)2^{(n+1)} + 2 \qquad \text{where} \quad n \geq 1$$

(8 Marks)

Note: Question No. 1 continues on page 2.

17

(c) Determine whether the following statement is true or false. If the statement is true, prove it. If the statement is false, give a counterexample.

If $h(n) = \max\{ f(n),\ g(n) \}$, then $f(n) + g(n) = O(h(n))$.

(5 Marks)

(d) Consider a singly-linked list S with the pointer *start* pointing to the first node of the list. Write an algorithm to add a new node, whose data value is *val*, at the beginning of the linked list and return *start*.

(7 Marks)

2. (a) The running time of a recursive algorithm is given as $T(n) = 2nT(n-1)$, where $T(0) = 1$. Solve the equation to derive the running time of the algorithm.

(5 Marks)

(b) Write an algorithm that appends a doubly-linked list P onto the end of another doubly-linked list Q. Pointers *StartP* and *StartQ* point to the first elements of P and Q, respectively.

(5 Marks)

(c) Draw the 13-item hash table resulting from hashing the keys 16, 54, 41, 63, 76, 81, 20, 25, 39, 46, using the hash function $h(x) = (x) \bmod 13$ and assume that collisions are handled by **separate chaining**.

(5 Marks)

(d) Write an algorithm which computes the product of absolute values stored in all leaves in a binary tree.

(10 Marks)

3. (a) Given an array with only odd or even numbers in its elements, write an algorithm with **linear time complexity** and **minimum space complexity** which sorts the array with all odd elements in the left hand of the array and all even elements in the right hand of the array.

(10 Marks)

(b) Write a **recursive algorithm** which computes the difference between maximum value and minimum value of an array.

(10 Marks)

(c) Write an algorithm which can make any heap become maxheap and binary search tree concurrently; in other words, the resultant maxheap is also a binary search tree.

(5 Marks)

4. (a) Write an algorithm which detects whether a directed graph is a directed acyclic graph (DAG).

(10 Marks)

(b) Write an algorithm which sorts text words based on their alphabetical order in dictionaries. Analyze the worst case time complexity of the designed algorithm.

(10 Marks)

(c) Write an algorithm which can find a minimum spanning tree among all possible shortest path trees for a given source vertex in a connected weighted graph.

(5 Marks)

END OF PAPER

18

EE 2008 Nov/Dec, 2017

1 (a) $T(n) = \sum_{i=1}^{n} (n-2i+1)(2n-n+1)$

$= \sum_{i=1}^{n} [n^2 + 2n + 1 - 2(n+1)i]$ *

$< \sum_{i=1}^{n} (n^2 + 2n + 1)$

$= n^3 + 2n^2 + n$

$= O(n^3)$

(b) $n=1$, $1 \cdot 2^1 = 0 \cdot 2^2 + 2$

When $n=k$, let $\sum_{i=1}^{k} i 2^i = (k-1)2^{k+1} + 2$.

Thus, $\sum_{i=1}^{k+1} i 2^i = \sum_{i=1}^{k} i 2^i + (k+1)2^{k+1} = (k-1)2^{k+1} + 2 + (k+1)2^{k+1}$

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

(c) Since $h(n) = \max\{f(n), g(n)\}$.

$f(n) \le h(n)$ and $g(n) \le h(n)$ for $n \ge 0$.

Thus, $f(n) + g(n) \le 2h(n)$ for $n \ge 0$

Thus, $f(n) + g(n) = O[h(n)]$

(d) Algo AddNode (start, val) {

```
temp = new node;
temp.data = val;
temp.next = null;
if (start == null)
    start = temp;
else { temp.next = start.next;
       start = temp; }
return start; }
```

2. (a) $T(n) = 2nT(n-1)$

$= 2n \cdot 2(n-1)\, T(n-2)$

$= \cdots$

$= 2n \cdot 2(n-1) \cdot 2(n-2) \cdot \cdots \cdot 2\times2 \cdot 2\times1 \cdot T(0)$

$= 2^n [n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1]$

$= 2^n n!$

(b) Algo Append (StartP , StartQ) {

While (StartQ.next != null)

StartQ = StartQ.next;

StartQ.next = StartP;

StartP.prev = StartQ; }

(c) keys 16 54 41 63 76 81 20 25 39 46

h(x) 3 2 2 11 11 3 7 12 0 7

0 → [39]

1

2 → [54] → [41]

3 → [16] → [81]

4

5

6

7 → [20] → [46]

8

9

10

11 → [63] → [76]

12 → [25]

(d) Algo Product (root) {

if (root == null) return 1;

if (root.left != null && root.right != null)

return root.data;

else

return Product(root.left) * Product(root.right)

}

3. (a) Algo sort(A, n) {
index_odd = 1;
Index_even = n;
i = 1;
while (i <= index_even) {
if(A[i] mod 2 == 1) {
index_odd = i;
i = i+1; }
else {
swap(A[i], A[index_even]);
index_even = index_even - 1; } } }.

(b) Algo MaxMin(A, i, j, min, max) {
if (i = j) return 0;
k = (i+j)/2;
MaxMin(A, i, k, min_L, max_L);
MaxMin(A, k+1, j, min_R, max_R);
if(min_L <= min_R) min = min_L;
else min = min_R;
if(max_L >= max_R) max = max_L;
else max = max_R;
return max - min; }.

(c) Algo Maxheap Sort(V, n) {
heapify(V, n);
L = lg(n);
for p = 1 to L-1
mergesort(V, 2^p, $2^{p+1}-1$);
if($2^L - 1 < n$)
mergesort(V, 2^L, n); }.

P.S. The question has been changed to "Write a algorithm which can make any heap become maxheap, and each horizontal level of the heap is in non-decreasing order." during the exam.

4. (a) Algo ISDAG(s, n) {
for i = 1 to n
visit[i] = False;
while (s != null) {
visit[s] = true;
Q.enqueue(s);
while (!Q.empty()) {
v = Q.front();
u = adj[v];
while (u != null) {
if (visit[u] == false) {
Q.enqueue(u);
visit[u] = true; }
else return False;
u = u.next; }
Q.dequeue(v); }
s = s.next; }
return True; }

(b) Algo SortWords {
① Divide words into groups according to the total numbers of letters in the words;
② For words with same no. of letters:
Sort the words according to their alphabetical order of the first letter;
If have same first letter,
Repeat sorting for the next letter, until all words are sorted.
③ Repeat ① and ②, until all groups of words are sorted.
④ Sort all groups of words together. }

Let n be the total no. of words, m be the no. of letters in the longest word, then worst case time complexity
$T(n) = n \cdot m \cdot T(sort)$
where $T(sort)$ is the time complexity of the sorting method used.

(c) didn't answer.