NANYANG TECHNOLOGICAL UNIVERSITY

SEMESTER 2 EXAMINATION 2018-2019

EE2008 / IM1001 – DATA STRUCTURES AND ALGORITHMS

April / May 2019 Time Allowed: 2½ hours

INSTRUCTIONS

1. This paper contains 4 questions and comprises 4 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.

```
$i = n$
while ( $i \geq n/2$ ) {
    for $j = 1$ to $n$
        $r = r + 1$
    $i = i - 2$
}
```

(5 Marks)

(b) Use mathematical induction to prove that the following equation is true.

$$\sum_{i=1}^{n-1} i(i+1) = \frac{n(n-1)(n+1)}{3}, \quad \text{where } n \geq 2.$$

(8 Marks)

Note: Question No. 1 continues on page 2.

21

(c) Determine the asymptotic upper bound of the following functions using the $O(g(n))$ notation with the simplest function $g(n)$ possible.

(i) $\binom{n}{2}$

(ii) $1+3+5+7+\cdots+(2n-1)$

(12 Marks)

2. (a) A pointer *start* points to the first element of a singly-linked list S of sorted integers. Write an algorithm to insert a new integer x into S so that the resultant list is still sorted. Assume that all integers in S and the new integer x are unique.

(8 Marks)

(b) Draw the 11 item hash table resulting from hashing the keys 35, 336, 112, 69, 333, 403, 165 using the hash function $h(x) = (3x \bmod 11)$. Assume that collisions are handled by linear probing.

(7 Marks)

(c) For the AVL tree shown in Figure 1, show the steps taken to balance the tree when a node with value 44 is inserted, by

(i) drawing the tree after the rebalancing; and

(ii) writing down the sequence of pointer assignments needed to achieve the rebalancing. Assume that you are given a pointer called *ref* that points to the first node where the imbalance occurs.

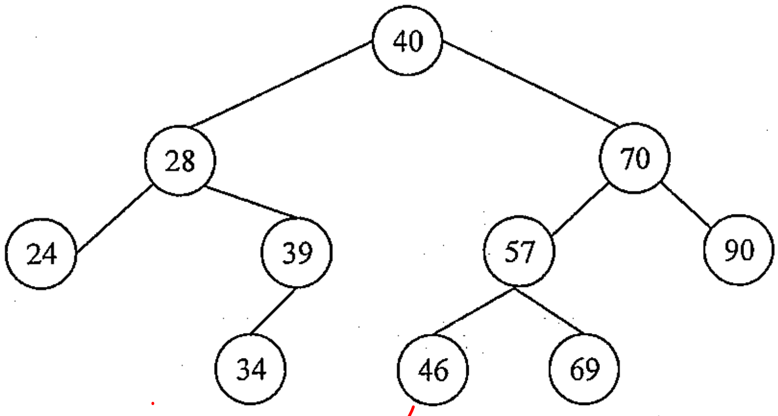


Figure 1

(10 Marks)

44

3. (a) Consider the following array. Explain and show clearly what the array looks like in each step if you are required to perform the following operations:

   (i) heapify the array so that it becomes a maxheap;

   (ii) then delete the second element of the array while maintaining the maxheap structure.

5	8	4	7	10	9	6

(10 Marks)

(b) Explain and show how to perform counting sort on the array shown below, assuming that all values in the array are in the range 1 to 5.

4	3	5	2	3	3	5

(10 Marks)

(c) Write in pseudocode a new partition algorithm that enables quicksort to sort in *non-increasing* order.

(5 Marks)

4. (a) Using depth-first search (dfs), perform topological sorting on the directed graph whose adjacency list is shown in Figure 2. Explain each step clearly and show the dfs trees constructed.

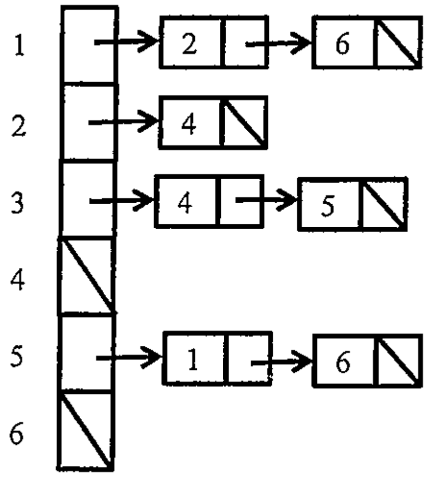


Figure 2

(10 Marks)

Note: Question No. 4 continues on page 4.

22

(b) A graph signal on the vertices of a graph is a function *f* mapping each vertex of the graph to a real value. The average signal value of a vertex is the average of its own *f* value and its neighbours' *f* values. Given the adjacency list *adj* of a graph and an array representing the function *f*, write an algorithm in pseudocode to return an array containing the average signal value for every vertex. What is the time complexity of your algorithm?

(9 Marks)

(c) The following divide-and-conquer algorithm for finding a minimum spanning tree (MST) in a connected weighted graph has been proposed:

(i) Randomly divide the vertices in the graph into two subsets to form two connected subgraphs with equal number of vertices or differing by at most one. Each subgraph contains only the edges whose end vertices both belong to the subgraph's vertex set.

(ii) Find a MST for each subgraph.

(iii) Connect the 2 MSTs by choosing an edge with minimum weight amongst those edges connecting them.

Is the final spanning tree found a MST? Justify your answer.

(6 Marks)

END OF PAPER

EE 2008

April / May 2019

1a) * Base:

- $S_{n/2} = n$

* general

- $S_n = n + S_{n-2}$

$S_n = n + n + S_{n-4}$

$\vdots$

$S_n = in + S_{n-2i}$

- $n - 2i = \frac{1}{2}n$

$\frac{1}{2}n = 2i$

$\frac{1}{4}n = i$

- $S_n = \frac{n^2}{4} + n \approx n^2 = O(n^2)$

b) • base

$n = 2$

$$\sum_{i=1}^{1} 1(1+1) = \frac{2(2-1)(2+1)}{3}$$

$$\sum_{i=1}^{1} 2 = \frac{2\cdot 1\cdot \cancel{3}}{\cancel{3}}$$

$$2 = 2$$

• let $n = n+1$

$$\sum_{i=1}^{n} i(i+1) = \frac{(n+1)(n)(n+2)}{3}$$

$$\sum_{i=1}^{n-1} i(i+1) + n(n+1) = \frac{(n+1)n(n+2)}{3}$$

$$\frac{n(n-1)(n+1)}{3} + n(n+1) = \frac{(n+1)n(n+2)}{3}$$

$$\frac{n(n-1)(n+1)}{3} + \frac{3}{3}n(n+1) = \frac{(n+1)n(n+2)}{3}$$

$$\frac{n(n+1)}{3}(n-1+3) = \frac{(n+1)n(n+2)}{3}$$

$$\frac{n(n+1)(n+2)}{3} = \frac{(n+1)n(n+2)}{3}$$

proven

c) i) $$\frac{n!}{2!\,(n-2)!} = \frac{n(n-1)\cancel{(n-2)!}}{2!\,\cancel{(n-2)!}}$$

$$= \frac{n^2-n}{2} \approx n^2 = O(n^2)$$

ii) $$S_n = 1+3+5+\ldots+(2n-1)$$

$$S_n = 2n-1 + 2n-3 + 2n-5 + \ldots + 1 \quad +$$

$$2S_n = 2n + \underbrace{2n + 2n + \ldots + 2n}_{n \text{ times}}$$

$$2S_n = 2n^2$$

$$S_n = n^2 = O(n^2)$$

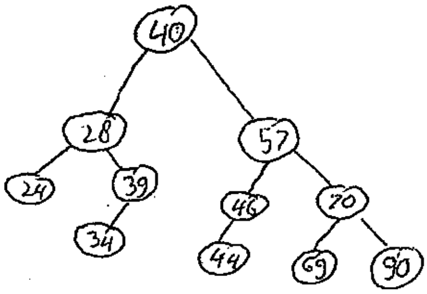
2a)

```
Sort_Single_link(S, x)
{
    new node = A
    A.data = x
    if start.data > x
    {
        A.next = start
    }
    else
    {
        While start.next.data < x
        {
            Start = start.next
        }
        A.next = start.next
        Start.next = A
    }
}
```

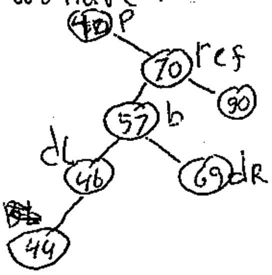
b)

403	0
165	1
	2
	3
	4
	5
35	6
336	7
112	8
69	9
333	10

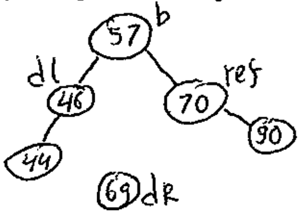
2) i)



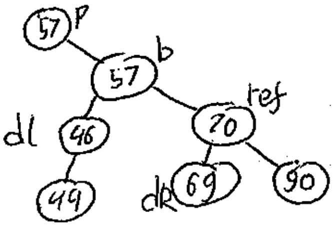
ii) *we have :



P = ref. parent
b = ref.left
dR = reft.right
dL = ref. left
* b.right = ref

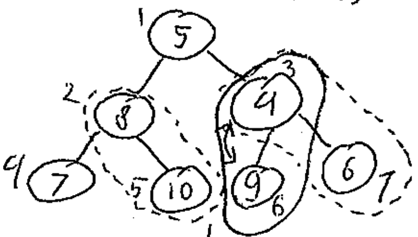


* ~~P.left ref~~
P. left = b
reft.left = dR
ref. parent = b



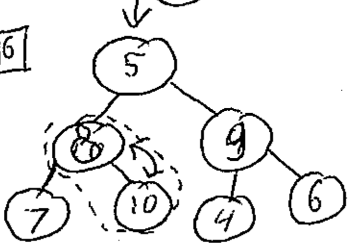
3.a) idea : apply siftdown starting near the leaves :

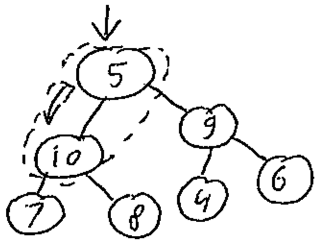
i) - for i = n/2 down to 1
sift down (A,i)



A

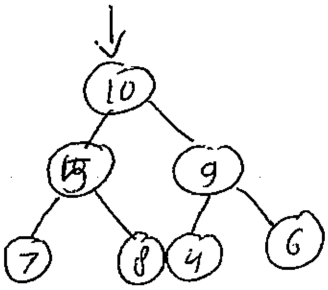
5	8	9	7	10	4	6

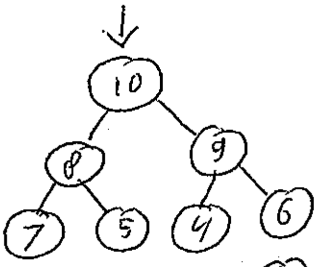




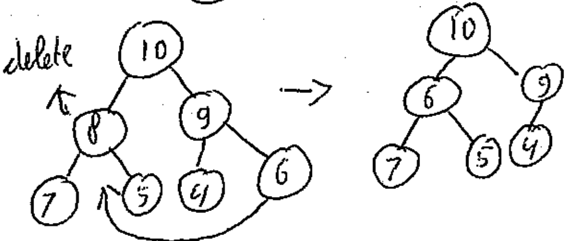
A

5	10	9	7	8	4	6

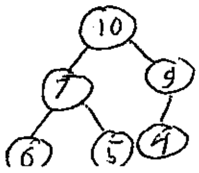




ii) * delete



If A[i/2] ≥ A[i]
sift down (A,i)
else
sift up (A,i)



3 b) 1) initialize another array count [1... k]

2) go through A [1... n]

- increase Count [A [i]] by 1

3) count [i] now contain the number of elements = i

4) To get the number ~~equal to i~~ of elements ≤ i : Cumulative sum

5) Use Count array to sort A [1.....n] by placing each A [i] the right ~~[illegible]~~ location given by Count [A [i]]

c) We perform counting sort:

* A:

5	8	4	7	10	9	6

Count : ~~a~~

0	0	0	0	0	0	0	0	0	0
1	2	3	4	5	6	7	8	9	10

↓

Count

0	0	0	0	1	0	0	0	0	0
1	2	3	4	5	6	7	8	9	10

Count

				1	0		1		
1	2	3	4	5	6	7	8	9	10

↓

⋮

Count

0	0	0	1	1	1	1	1	1	1
1	2	3	4	5	6	7	8	9	10

perform

```
for j to n {
    Count [A [j]] = Count [A [j]]+1
}
```

* perform: Count [i] = count [i-1]+ Count [i] for i = 1 to k

Count

0	0	0	1	2	1	1	1	1	1
1	2	3	4	5	6	7	8	9	10

↓

Count

0	0	0	1	2	3	1	1	1	1
1	2	3	4	5	6	7	8	9	10

↓

⋮

↓

Count

0	0	0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8	9	10

* Perform:

```
for j = n down to 1
{
    B [Count [A [j]]] = A [j]
    Count [A [j]] = Count [A [j]] - 1
}
```

B :

		6				

Count

0	0	0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8	9	10

↓

⋮

↓

B :

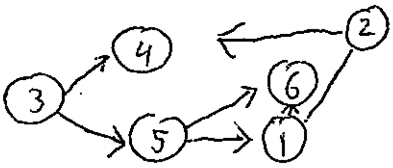
4	5	6	7	8	9	10

Count

0	0	0	0	1	2	3	4	5	6
1	2	3	4	5	6	7	8	9	10

```
3 c)  quick sort_ non increase ( A, P, r)
       {
           ~~if (p~~ quick sort (A, P r)
              reverse (A)
            return A
       }
     quick sort(A, P, r) {
      if (p< r){
         h= partition (A,P,r)
          quick sort (A, P, h-1)
          quick sort ( A, h+1, r)
         }
       }
```

1) a) 1) We must start with vertex with in-degree of zero:
3



2) at this point, we may consider those edges which ~~conect~~ connect vertex 3 to other vertices, and thus we may chose vertex 4:

3, 4

3) there is no other connected vertex on vertex 4. Thus we chose another connected vertex from ~~the~~ vertex 3 :

3, 4, 5

4) we ~~may~~ may repeat the previous step by choosing vertex 6 or 1 :

~~3, 4, 5, 6~~ 3, 4, 5, 1

5) adding the edges extending from vertex 5

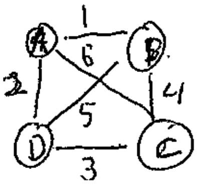
3, 4, 5, 1, 2

6) we can't extend from vertex 2 because we have chosen vertex 4. thus, we have to chose the last vertex

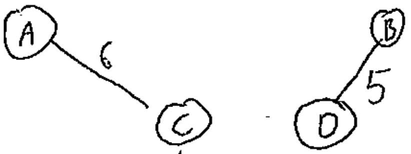
3, 4, 5, 1, 2, 6

b)
```
bfs(adj, s){
  n = adj.last
  for i = 1 to n
    { visit [i] = false
      average [i] = ~~adj.data~~ value [i]
    }
  visit [s] = true
  q.enqueue(s)
  While (! q.empty())
   {
     V = q.front()
     ref = adj [V]
     count = 1
     While (ref ! null) {
        if (!visit [ref.data]) {   visit [ref.data] = true
          q.enqueue (ref.data)
          average [i] = average [i] + value [ref.data]
           count = count + 1
          }
        ref = ref.next
        average [q.front] = average [q.front] / count
   }
   q.dequeue()
   }
```

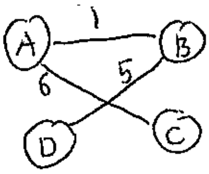
4) c) -We have spanning tree



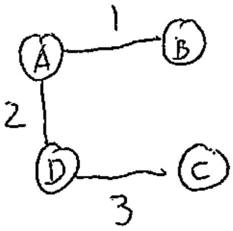
- we randomly choose



- then connect the 2MST by choosing smallest edge



contradict, because the spanning tree should be



hence, the final spanningtree isn't the MST