NANYANG TECHNOLOGICAL UNIVERSITY

SEMESTER 2 EXAMINATION 2015-2016

EE2008 / IM1001 – DATA STRUCTURES AND ALGORITHMS

April / May 2016 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 "$y = y + 2$" is executed in each of the following algorithms.

(i)

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

(ii)

```
for i = 1 to n
    for j = i to 2i
        y = y + 2
```

(10 Marks)

Note: Question No. 1 continues on page 2.

6

(b) (i) Solve the following recurrence relation: $b_n = \dfrac{b_{n-1}}{1+b_{n-1}}$, $b_0 = 1$, where n is a non-negative integer.

(ii) Determine whether the following statement is true or false. Justify your answer.

$$\text{If } f(n) = \Theta(g(n)) \text{ then } g(n) = \Theta(f(n)).$$

(8 Marks)

(c) A pointer *start* points to the first element of a doubly-linked list L. Write an algorithm that reverses the elements of L. You are not allowed to use any additional data structure in your solution.

(7 Marks)

2. (a) (i) Let Q be a non-empty queue and S be an empty stack. Using the stack and queue ADT functions and the stack S, write an algorithm to reverse the order of the elements in Q.

(ii) Draw the 7-item hash table resulting from hashing the keys 19, 26, 13, 48, and 17 using the hash function $h(x) = x \bmod 7$. Assume that collisions are handled by double hashing using a second hash function $h'(x) = 5 - (x \bmod 5)$.

(9 Marks)

(b) Assume that the LIST ADT is implemented using a doubly linked list. Using pseudo-code, describe the implementation of the method *insertBefore(p,e)* of the LIST ADT.

(6 Marks)

(c) Suppose that the data stored at each node in a binary search tree is a positive integer. Write a recursive algorithm that finds the sum of all values, which are less than a given value x in the binary search tree.

(10 Marks)

3. (a) Show clearly what the following array looks like in each step of the siftdown algorithm when applied at the index $i=2$, assuming that we are restoring a maxheap and the index of the array starts at 1.

102	20	69	67	33	58	65	23	15

(5 Marks)

(b) Continuing from your answer obtained in part (a), show each step in the heapsort algorithm.

(12 Marks)

(c) Write an algorithm using counting sort to sort an array of integers in the range $[-k, k]$. You may make use of the functions discussed in lecture to construct your algorithm.

(8 Marks)

4. (a) Use the depth first search (dfs) algorithm starting at vertex 1 to perform topological sorting of the directed acyclic graph shown in Figure 1. Explain each step clearly by drawing the dfs trees generated and the output array at each step.

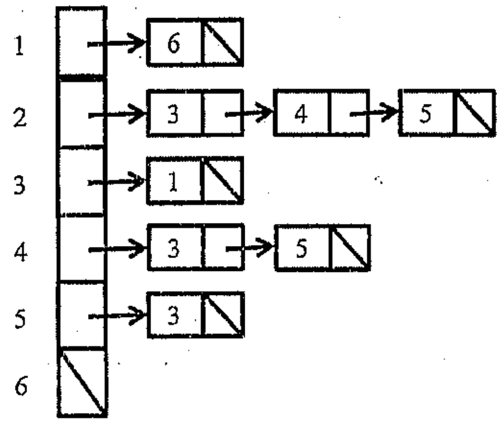


Figure 1

(10 Marks)

Note: Question No. 4 continues on page 4.

(b) Write an algorithm that finds the sum of the in-degrees of all the vertices in a directed graph. Assume that the directed graph is represented by an adjacency list.

(10 Marks)

(c) What is the time complexity of your algorithm in part (b) in terms of the number of vertices and edges? Justify your answer.

(5 Marks)

END OF PAPER

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

$= \sum_{i=1}^{n}\sum_{j=i+1}^{n} (n-j)$

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

$= n-i-1+n-i-2+n-i-3+n-i-4+\cdots\ n-i-(n-i)$

$= (n-i)^2 - (1+2+3+\cdots+n-i)$

$\sum_{i=1}^{n}(n-i)^2 = \sum_{i=1}^{n} n^2 + \sum_{i=1}^{n} i^2 - \sum_{i=1}^{n} 2ni \leq n^3 + \frac{n(n+1)(2n+1)}{6} = O(n^3)$ #

(ii) $\sum_{i=1}^{n}\sum_{j=i}^{2i} 1 = \sum_{i=1}^{n}(2i-i+1) = \sum_{i=1}^{n}(i+1) = \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1 = \frac{n(n+1)}{2} + n = O(n^2)$ #

1(b) (i) $b_n = \frac{b_{n-1}}{1+b_{n-1}}$ $b_0 = 1$ n is a non-negative integer.

$b_0 = 1$

$b_1 = \frac{b_{1-1}}{1+b_{1-1}} = \frac{b_0}{1+b_0} = \frac{1}{1+1} = \frac{1}{2}$

$b_2 = \frac{b_1}{1+b_1} = \frac{1}{3}$

~~$b_3 =$~~

$b_n = \frac{b_{n-1}}{1+b_{n-1}}$ $\quad b_{n-1} = \frac{b_{n-2}}{1+b_{n-2}}$

$= \frac{\frac{b_{n-2}}{1+b_{n-2}}}{1+\frac{b_{n-2}}{1+b_{n-2}}} = \frac{b_{n-2}}{2b_{n-2}+1}$ $\quad b_{n-2} = \frac{b_{n-3}}{1+b_{n-3}}$

$= \frac{b_{n-3}}{3b_{n-3}+1}$

$= \frac{b_{n-4}}{4b_{n-4}+1}$

$\vdots$

$= \frac{1}{n+1}$ #

(ii) If $f(n) = \Theta(g(n))$

$\therefore f(n) \leq k_1 g(n)$ for all $n \geq n_0$ and $k_1 > 0$

$f(n) \geq k_2 g(n)$ for all $n \geq n_0$ and $k_2 > 0$

prove $g(n) \geq k_3 f(n)$ & $g(n) \leq k_4 f(n)$, ...

for $f(n) \leq k_1 g(n)$ for all $n \geq n_0$ and $k_1 > 0$

$\frac{1}{k_1} f(n) \leq g(n)$

shows that $g(n) \geq k_3 f(n)$

Date: No.

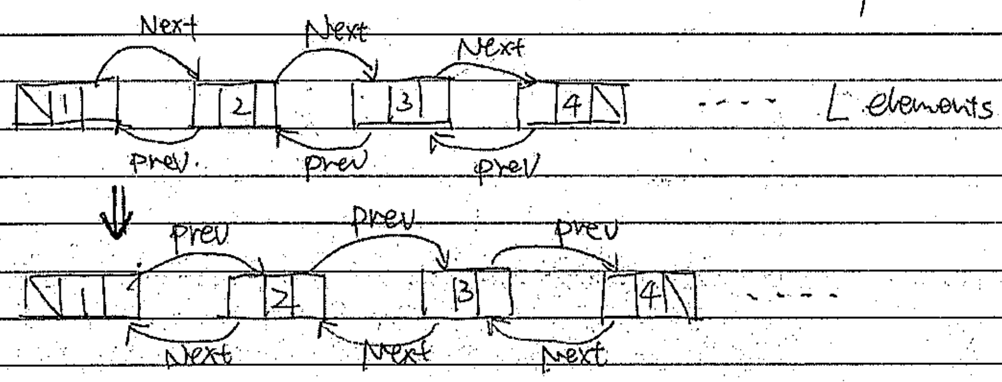
for $f(n) \geq k_2 g(n)$ for all $n > n_0$ and $k_2 > 0$

$$g(n) \leq \frac{1}{k_2} f(n)$$

Shows that $g(n) \leq k_4 f(n)$

Thus, if $f(n) = \Theta g(n)$ then $g(n) = \Theta f(n)$. #

1(C)




```
While (start.next != null) {
    swap (start.next.data, start.prev.data)
    start = start.next
}
return ;
```

#

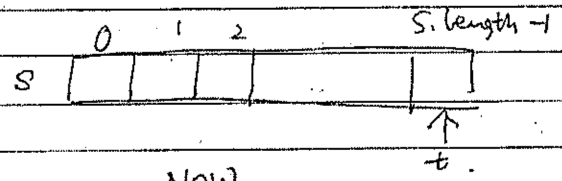
The idea is let all values in Q insert into S firstly. (Q is FIFO & S is FILO)
Secondly, put all values back into Q but will be reversed.

3/9

Date: No.

2(a) (i) Q → Non-empty queue f Ⓐ Ⓑ Ⓒ Ⓓ ... r

Initial { S → Empty Stack

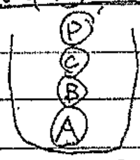



```
Step 1 While (r != -1) {
  1.1 front() {
        if (empty())
           throw QEmptyException
        else return Q[f] }
        val = Q[f]  // Set val equals to the
                       front value of Queue.
  1.2 push(val) {
        if t == S.length - 1 then
           throw FullStackException
        else
           t = t+1
           S[t] = val
      }  // Insert the val into Stack

  1.3 dequeue() {
        if (empty())
           throw Q Empty Exception
        else {
           if (r == f)
              r = f = -1
           else {
              f = f+1
              if (f == Q.size) f = 0
           }
        }
      }  // Delete the value in Q
    }  // While Q is not empty, repeat
         the steps until all elements
         are inserted into S.
```

Now Queue f empty r

Stack



```
Step 2.
  While (t != -1) {
  2.1 top() {
        if S.empty() then
           throw EmptyStackException
        else
           return S[t] }
        val2 = S[t]  // Last item
                        value in the Stack

  2.2 enqueue (val2) {
        if (empty())
           r = f = 0
        else {
           r = r+1
           if (r == Q.size) r = 0
           if r == f
              throw FullQException
        }
        Q[r] = val 2
      }  // Insert the val to Q
  2.3 pop() {
        if S.empty() then
           throw EmptyStackException
        else t = t-1 }
        // Remove the value in S
  }  // Transfer all values in S to Q.
Step 3. return Q  #
```

$h(x) = x \bmod 7$ $h'(x) = 5 - (x \bmod 5)$

2(a) (ii)

k	h(k)	d(k)	Probes	
19	5	1	5	
26	5	4	5	2
13	6	2	6	
48	6	2	6	1
17	3	3	3	

19: $h(k) = 19 \bmod 7 = 5$ $d(k) = 5 - (19 \bmod 5) = 1$
26: $h(k) = 26 \bmod 7 = 5$ $d(k) = 5 - (26 \bmod 5) = 4$
13: $h(k) = 13 \bmod 7 = 6$ $d(k) = 5 - (13 \bmod 5) = 2$
48: $h(k) = 48 \bmod 7 = 6$ $d(k) = 5 - (48 \bmod 5) = 2$
17: $h(k) = 17 \bmod 7 = 3$ $d(k) = 5 - (17 \bmod 5) = 3$

7-item hash table:

0	1	2	3	4	5	6
	48	26	17		19	13

#

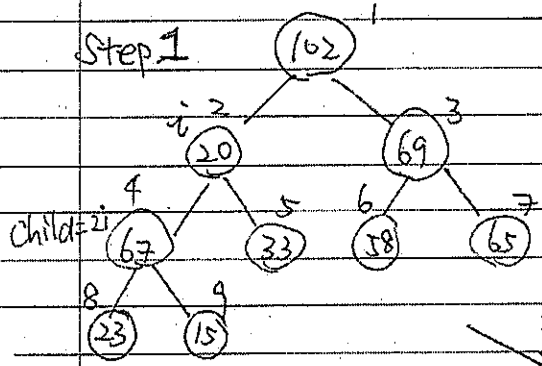
2(b)
```
insertBefore(p,e){
    v = new node
    v.data = e
    v.prev = p.prev
    v.next = p
    p.prev.next = v
    p.prev = v
}
```
#

2(c)
```
Sum(root,x){
    if (root == null)
        return 0
    else if (root.data < x)
        return root.data + sum(root.left,x) + sum(root.right,x)
    else return sum(root.left,x) + sum(root.right,x)
}
```
#

3(a)

A	102	20	69	67	33	58	65	23	15
i	1	2	3	4	5	6	7	8	9

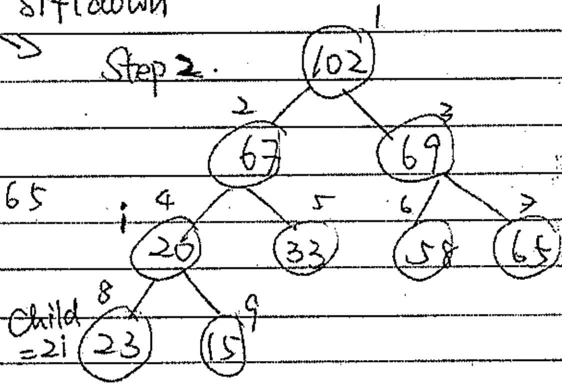
Step 1



1. Find largen child 2. Swap

siftdown

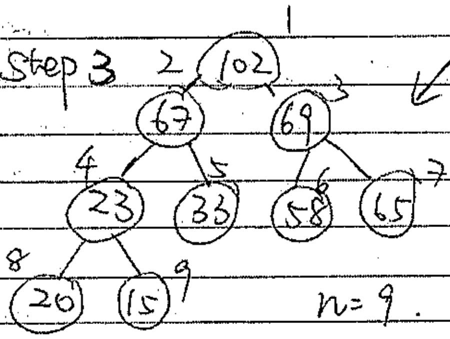
Step 2.



102 67 69 20 33 58 65 23 15

siftdown

Step 3



$n=9$.

⇓

A	102	67	69	23	33	58	65	20	15
i	1	2	3	4	5	6	7	8	9

#.

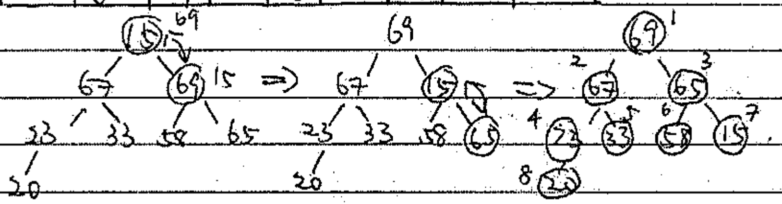
Date: No.

3(b)

	1	2	3	4	5	6	7	8	9
A	102	67	69	23	33	58	65	20	15

① Exchange A[1] with A[n] (A[1] with A[9])
Treat A[1, …. n-1] As a new heap. (A[1, 2, … 8] As a new heap)

A	15	67	69	23	33	58	65	20	102

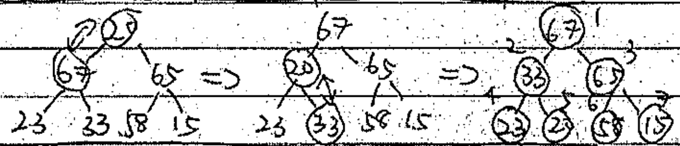


siftdown

A	69	67	65	23	33	58	15	20	102

② Swap A[1] & A[8]. Treat A[1, 2, … 7] as a new heap.

A	20	67	65	23	33	58	15	69	102

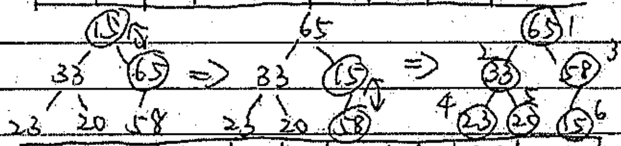


Siftdown

A	67	33	65	23	20	58	15	69	102

③ Swap A[1] & A[7]. Treat A[1, 2, …, 6] As a new heap.

A	15	33	65	23	20	58	67	69	102

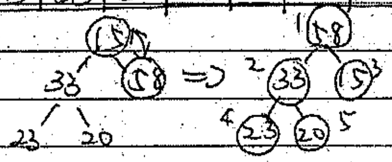


siftdown

A	65	33	58	23	20	15	67	69	102

④ Swap A[1] with A[6]. Treat A[1, …, 5] as a new heap

A	15	33	58	23	20	65	67	69	102



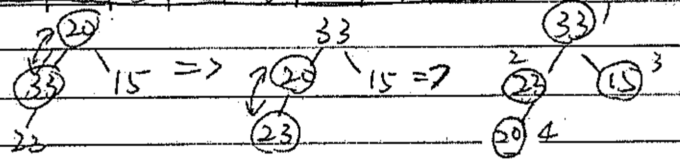
siftdown

A	58	33	15	23	20	65	67	69	102

Date: No.

⑤ Swap A[1] with A[5]. Treat A[1, ⋯, 4] as a new heap

A | 20 | 33 | 15 | 23 | 58 | 65 | 67 | 69 | 102 |

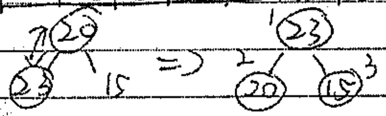


siftdown

A | 33 | 23 | 15 | 20 | 58 | 65 | 67 | 69 | 102 |

⑥ Swap A[1] with A[4]. Treat A[1, ⋯, 3] as a new heap

A | 20 | 23 | 15 | 33 | 58 | 65 | 67 | 69 | 102 |



siftdown

A | 23 | 20 | 15 | 33 | 58 | 65 | 67 | 69 | 102 |

⑦ Swap A[1] & A[3]

A | 15 | 20 | 23 | 33 | 58 | 65 | 67 | 69 | 102 | #

```
for(i=1 to k){
3(C) Counting-sort(A,B,K){
        n=A.last
        for i=1 to k{
            count[i]=0
        }
        for j=1 to n{
            count[A[j]]=count[A[j]]+1
        for i=1 to k{
            count[i]=count[i-1]+count[i]
        for j=n downto 1{
            B[count A[j]] = A[j]
            count[A[j]]=count[A[j]]-1
        }
      }
   }
```

```
for(i=-k to 1){
    Counting_sort(A,B,k);
}
return;
```

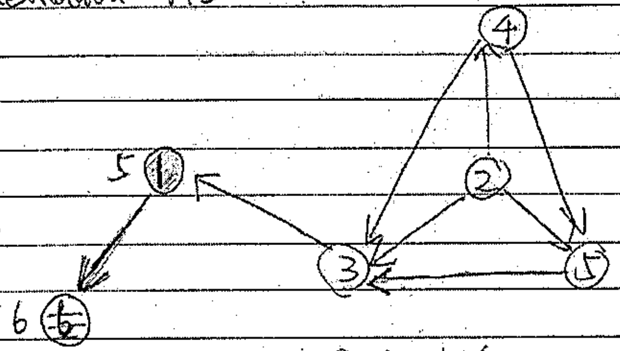
#

(8/9)

4(a) Implementation : DFS

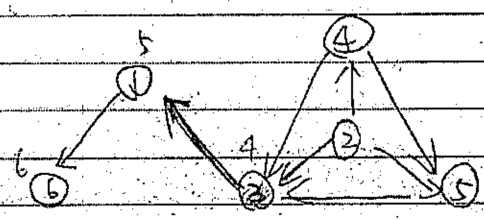
①

dfs(1)
dfs(6)



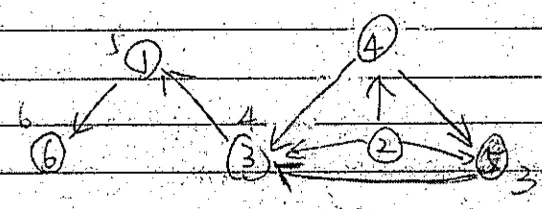
Sort : 1, 6

② dfs(3)



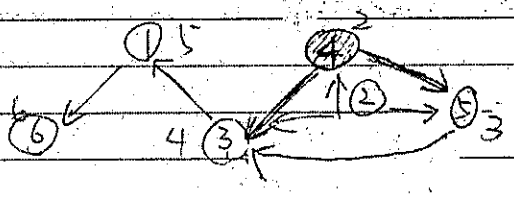
Sort : 3, 1, 6

③ dfs(5)



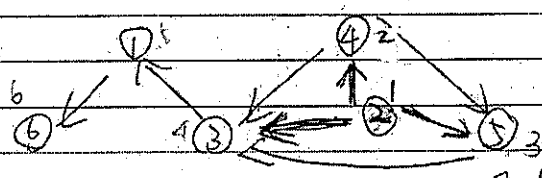
Sort : 5, 3, 1, 6

④ dfs(4)



Sort : 4, 5, 3, 1, 6.

⑤ dfs(2)



Sort : 2, 4, 5, 3, 1, 6

⑥ Sort will be 2, 4, 5, 3, 1, 6

#

Date: No.

4(b)

```
Sum_indegrees (adj) {
    n = adj.last  <  sum = 0
    for i=1 to n
        indegree[i] = 0
    for i=1 to n {
        ref = adj[i]
        while (ref != null) {
            indegree[ref.data] = indegree[ref.data] + 1
            ref = ref.next
        }
    }
    for (i=1 to n) {
        Sum = Sum + sum[i]
    }
    Return Sum     #
```

4(c) Time complexity is O(n)

Total Need n times

Happened once for each i

```
for i=1 to n {
    ref = adj[i]
    while (ref != null) {
        indegree[ref.data] = indegree[ref.data] + 1
        ref = ref.next
    }
```

$\sum_{i=1}^{n} 1 = n$. $\therefore$ Time complexity is $O(n)$

#

AZONE