Fundamentals of Data Structures

Mid-Term Exam (Full mark is 15 points) October 17th,2013

1. Which one of the following statements is true as N grows? (0.5 point).

   A. For any x, x^N grows faster than $N!$ B. $(\log N)^2$ grows faster than $\sqrt{N}$

   C. $\log N^2$ grows faster than $(\log N)^2$ D. N grows faster than $\sqrt{N}(\log N)^2$

2. What is the major difference among lists, stacks, and queues? (0.5 point)
   A. Lists use pointers, and stacks and queues use arrays
   B. Stacks and queues are lists with insertion/deletion constraints
   C. Lists and queues can be implemented using circularly linked lists, but stacks cannot
   D. Stacks and queues are linear structures while lists are not

3. Push and pop 1, 2, 3, 4, 5, 6 sequentially into then out of a stack. Each number is pushed into a queue right after it gets out of the stack, and the dequeue sequence is 2, 5, 6, 4, 3, 1. If both the stack and the queue are initially empty, then the minimum size of the stack must be _____. (0.5 point)

   A. 1 B. 2 C. 3 D. 4

4. For a binary tree, given the preorder traversal sequence 12345 and the postorder traversal sequence 24531, the corresponding inorder traversal sequence must be ____. (0.5 point)

   A. 23145 B. 32154 C. 21435 D. Cannot be determined

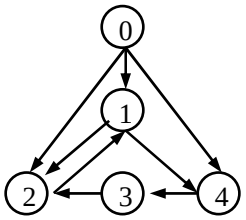
5. If on the 6th level of a complete binary tree (assume that the root is on the 1st level) there are 8 leaf nodes, then the maximum number of nodes of this tree must be _____. (0.5 point).

   A. Cannot be determined B. 52 C. 111 D. 119

6. Given a list of integers { 5, 2, 7, 3, 4, 1, 6 }. Please draw the results of the following:
   A. insert the numbers into an initially empty binary search tree (1 point);
   B. insert the numbers into an initially empty min-heap (1 point); and
   C. adjust the min-heap obtained in B into a max-heap (1 point).

7. If we represent the tree in problem 6(A) by an in-order threaded tree, please draw the threads of the nodes. (1 point)

8. For the digraph given by the figure, obtain:
(1) the in-degree and out-degree of each vertex; (1 point)
(2) its adjacency matrix; (1 point)
(3) its adjacency list representation; (1 point) and
(4) its strongly connected components. (0.5 point)



9. Please fill in the blanks in the program which performs Find as a Union/Find operation with path compression. (1 point)

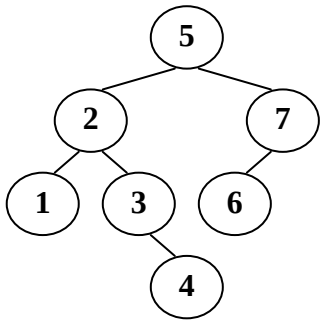
```
SetType  Find ( ElementType  X, DisjSet  S )
{
    ElementType  root,  trail,  lead;
    for ( root = X; S[ root ] > 0; ①_____ )
        ;
    for ( trail = X; trail != root; trail = lead ) {
        lead = S[ trail ] ;
        ②_____;
    }
    return  root ;
}
```

10. Please write a pseudo-code algorithm to delete a given element at position **p** from a min-heap **H**. (4 points)

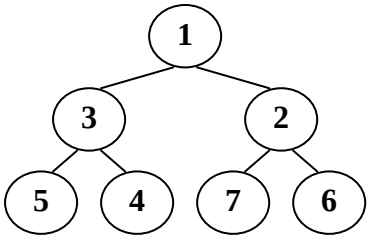
Answer Sheet

1. D	2. B	3. D	4. C	5. C

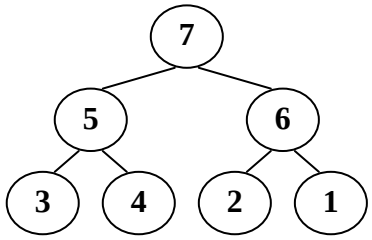
6A.



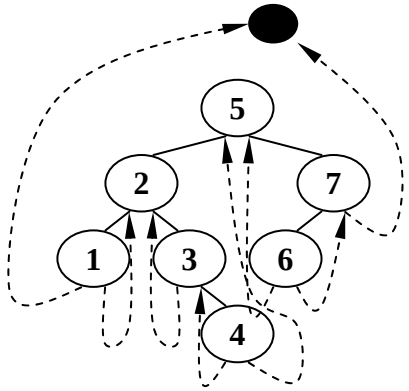
6B.



6C.



7.

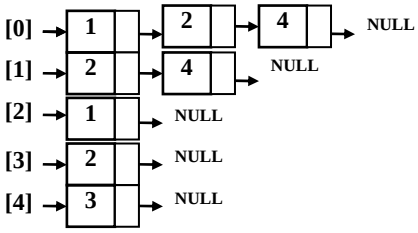


8. (1)

i	0	1	2	3	4
in	0	2	3	1	2
out	3	2	1	1	1

(2)
$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

(3)



(4) `{1,2,3,4}, {0}`

9.

(1) root = S[root]

(2) S[trail] = root

10.

```
Deletion ( PriorityQueue H,   int p )   /* delete the element   H->Elements[p] */
{
      temp = H-> Elements[ H->Size-- ]; /* delete the last element after saving its value */
     if ( temp < H->Elements[p] ) {   /* if the last element is smaller */
        while ( (p != 1) && (temp < H->Elements[p/2]) ) {
           /* start percolate-up from H->Elements[p] */
           H->Elements[p] = H->Elements[p/2];
           p /= 2;
        }   /* end - while */
     }   /* end - if */
    else {   /* if the last element is larger */
       while( (child = 2*p) <= H->Size) {   /* start percolate-down from H->Elements[p] */
            if ( child != H->Size && H->Elements[child+1] < H->Elements[child])
                  child ++;   /* take the smaller child */
            if ( temp > H->Elements[child]) {   /* if the child is smaller, percolate down */
                  H->Elements[p] = H->Elements[child];
                  p = child;
             }   /* end - if the child is smaller */
             else
                  break;
       }   /* end - while */
     }   /* end - else */
     H->Elements[p] = temp;   /* save the last element at the proper position */
}
```