자료구조 중간고사

4/29, 2015. Open book, 75분, 110점 만점

1. (20점. 각 5점) O, X로 대답하라. 당신이 그 답을 제대로 알고 있다는 것을 알 수 있을 정도의 간단한 설명을 곁들여야 한다.
   1.1 Quicksort의 running time은 $\Theta(n\log n)$이다.
   1.2 Heapsort는 in-place sorting이 아니다.
   1.3 이미 sorting되어 있는 실수 배열이 입력으로 들어올 때, 수업 시간에 배운 sorting 알고리즘들 중 가장 매력적인 것은 quicksort이다.
   1.4 정렬되어 있는 array[1...n]에서 원소 x를 찾기 위해 binary search를 사용할 때, 최악의 경우 running time은 $\Omega(\log n)$이다.

2. (20점, 각 10점)
   2.1 Selection sort에서 최대값이 두 개 이상 있을 때는 가장 뒤에 있는 것을 최대로 삼는다면 selection sort는 stable sorting이 되는가?
   2.2 Array[1...10]이 아래와 같이 주어진 상태에서 min-heap을 사용하여 heapsort를 한다면 최초로 일어나는 원소의 자리 이동은 무엇인가?

5	10	3	9	7	12	2	11	3	8

3. (10점) n개의 서로 다른 key를 차례대로 받아 binary search tree를 만들었다. 이 binary search tree를 만드는 과정에서 소요된 총 시간의 합은 어떻게 되는가? Asymptotic average time을 O( ) notation으로 답하라. n개의 key가 배열될 수 있는 모든 경우의 수는 같은 확률을 갖는다고 가정한다. 수업 시간에 제공된 결론은 증명할 필요없이 그대로 인용하면서 설명하면 된다.

4. (20점) 아래는 binary search tree에서의 삭제 작업을 위한 pseudo code이다. 여기서 밑줄 친 부분은 후속함수의 return 값을 기다리는 방식으로 코딩된 것이다. 이 4개의 setLeft와 setRight를 쓰지 않고 같은 의미를 갖는 코딩으로 만들어 보라. 즉, 후속함수의 return 값을 기다리지 않는 방식으로 바꾸어 보라. 아래 코드 전체를 다 다시 쓰되, 바뀌지 않는 부분은 굳이 다 쓸 필요는 없고 바뀌는 부분이 어딘지 분간할 수 있는 수준에서 쓰면 됨. Java code가 아니므로 Java 문법은 신경쓰지 말 것.

```
TreeNode deleteItem (TreeNode tNode, Comparable searchKey) {
        if (tNode == null) {exception 처리 };
        else {
              if (searchKey == tNode's key) {
                        tNode = deleteNode(tNode);
              } else if (searchKey < tNode's key) {
                        tNode.setLeft(deleteItem(tNode.getLeft( ), searchKey));
              } else {
                        tNode.setRight(deleteItem(tNode.getRight( ), searchKey) );
              }
        }
        return tNode; // tNode: parent에 매달리는 노드
}
TreeNode deleteNode (TreeNode tNode) {
        if ( (tNode.getLeft( ) == null)  && (tNode.getRight( ) == null)) {
```

```
            return null;
        } else if (tNode.getLeft( ) == null) {
            return tNode.getRight( );
        } else if (tNode.getRight( ) == null) {
            return tNode.getLeft( );
        } else {
            tNode.setItem(minimum item of tNode's right subtree);
            tNode.setRight(deleteMin(tNode.getRight( ));
            return tNode;
        }
}
TreeNode deleteMin (TreeNode tNode) {
    if (tNode.getLeft( ) == null) {
            return tNode.getRight( );
    } else {
            tNode.setLeft(deleteMin(tNode.getLeft( ));
            return tNode;
    }
}
```

5. (20점) Double-ended queue(줄여서 deque)는 queue의 enqueue와 dequeue가 back 또는 front로 제한되는 것이 아니고 back과 front에서 다 가능한 변형 queue이다. 따라서 deque에서는 작업이 enqueue_front, enqueue_back, dequeue_front, dequeue_back으로 구분된다.(이 설명만으로 의미를 파악하기 바람. 이에 대한 질문은 불허.) 아래는 queue를 reference-based로 구현한 Java code이다. 이를 deque에서의 위 네 작업으로 바꾸어 보라. 내용 중 아래에서 그대로 가져다 쓸 수 있는 부분은 굳이 다시 쓰지 않고 어떤 부분과 동일하다고 명시하면 된다.(예: “enqueue( )의 3번째줄부터 8번째줄과 동일”, “enqueue( )의 전체 내용과 동일”) Java code의 문법도 지키면서 만들 것.

```
public class QueueReferenceBased implements QueueInterface{
    private Node lastNode;

    public QueueReferenceBased( ) {
            lastNode = null;
    }
    ...
    public void enqueue(Object newItem) {
            Node newNode = new Node(newItem);
            if (isEmpty( )) newNode.setNext(newNode);
            else {
                    newNode.setNext(lastNode.getNext( ));
                    lastNode.setNext(newNode);
            }
            lastNode = newNode;
    }
    public Object dequeue( ) {
            if (!isEmpty( )) {
                    Node firstNode = lastNode.getNext( );
                    if (firstNode == lastNode) { // special case?
                            lastNode = null;  // only one node in queue
                    } else {  // more than one item
                            lastNode.setNext(firstNode.getNext( ));
                    }
                    return  firstNode.getItem( );
            } else {exception 처리 }
    }
```

...

6. (20점. 각 10점) 아래는 n개의 원반을 옮기는 Hanoi Tower 알고리즘이다.

```
move(n, A, B, C)
{
        if (n=1) then move the disk from A to B;
        else {
                move(n-1, A, C, B);
                move(1, A, B, C);
                move(n-1, C, B, A);
        }
}
```

6.1 이제 이것을 좀 변형해본다. 한 번에 3개까지의 원반을 옮길 수 있다고 하자. 즉, 옮겨야 할 원반이 3개 이상 있으면 그 중 3개를 한 번에 옮기고, 옮겨야 할 원반이 3개 미만이면 한 번에 그들을 다 옮긴다. 위 알고리즘을 이렇게 변형한 Hanoi Tower 알고리즘으로 바꾸어 보라.

6.2 이렇게 하면 n개의 원반을 A에서 B로 옮기기 위해 총 몇 번의 원반 이동이 필요한가? 최대 3개까지 한 번에 옮기는 행위를 한 번의 원반 이동으로 센다. 예를 들어, A에 원반이 3개 있다면 단 한 번의 원반 이동으로 작업이 끝난다. 참고로 표준적인 Hanoi Tower 문제에서는 총 $2^n - 1$번의 원반 이동이 필요하다. 당신의 답에 대한 증명도 필요하다.