자료구조 중간고사, 4/23/2018

Open book, 70분, 105점 만점

* 그런 일 없으리라 기대하지만 혹시 문제에 명시되지 않았는데 필요한 가정이 있으면 스스로 하라. 합리적인 가정이면 추가 점수를 받을 것이고, 미숙함의 결과로 도입된 가정이면 만점을 못받을 것임.

1. (20점. 각 5점) OX 문제. 맞으면 그냥 O만 쓰고, 틀리면 X를 쓰고 틀린 이유를 말하라.
   a. Quicksort의 running time은 $\Theta(n^2)$이다.
   b. Mergesort의 running time은 $\Omega(n)$이다.
   c. Selection sort의 running time은 $\Omega(n^2)$이다.
   d. Radix sort의 running time은 $O(n)$이다.

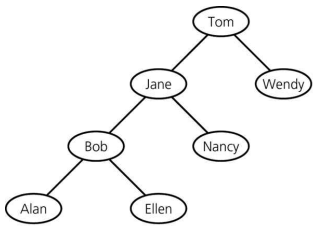
2. (10점) 노드 r을 루트로 하는 binary tree의 height를 알아내는 다음 알고리즘을 완성하라. Empty tree의 height는 0이다. 뜻만 통하면 되고 프로그래밍 언어 같은 것은 신경쓰지 않아도 됨.

```
heightBinaryTree(r)
{
        ?
}
```

3. (10점) Insertion sort에서 가장 운이 좋으면 총 몇 번의 비교로 정렬이 끝나게 되는가? 입력의 크기가 n에 대한 정확한 수를 요구함.

4. (10점) Radix sort에서 least-significant digit부터 정렬하는 대신 most-significant digit부터 정렬하면 제대로 정렬이 되는가? 대답이 Yes이면 간단한 설명을, 대답이 No이면 반례를 들고 설명하라.

5. (10점) Empty binary search tree로부터 7개의 key가 Tom, Jane, Bob, Wendy, Ellen, Alan, Nancy의 순으로 들어와서 오른쪽과 같은 모양이 되었다. 각 key를 삽입할 때마다 아래 insert( )를 수행하였다. 이 과정을 통틀어 ②에서 **return** 하는 tNode의 값이 ①에서 받은 tNode의 값과 다른 경우는 총 몇 번 발생했는가?




```
insert(Comparable newItem) {
                        root = insertItem(root, newItem);
}
TreeNode insertItem(TreeNode tNode, Comparable newItem) { ---------- ①
      if (tNode == null) { // insert after a leaf  (or into an empty tree)
                tNode = new TreeNode(newItem, null, null);
      } else  if (newItem < tNode's item) { // branch left
                tNode.setLeft( insertItem(tNode.getLeft( ), newItem) )
      } else { // branch right
                tNode.setRight( insertItem(tNode.getRight( ), newItem) );
      }
      return tNode; ------------------------------------------------------------- ②
} // end insertItem
```

6. (15점) 아래는 mergesort 알고리즘에 시험을 위해 global 변수를 하나 끼워넣은 것이다. 정렬하고자 하는 원소의 개수가 $n=2^k$일 때(k는 자연수) 정렬을 다 마친 후 totalComp의 가능한 최소값은? 점근적 수치가 아니고 정확한 수치를 요구함. (n 또는 k로 표현)

```
int totalComp ← 0; // totalComp는 global 변수로 0으로 초기화됨.
mergeSort(S) { // Input: sequence S with n elements. Output: sorted sequence S
          if (S.size( ) > 1) {
                    Let S1, S2 be the 1st half and 2nd half of S, respectively;
                    mergeSort(S1);
                    mergeSort(S2);
                    S ← merge(S1, S2);
          }
}
merge(S1, S2) {
      sorting된 두 sequence S1, S2를 합쳐 sorting된 하나의 sequence T를 만든다.
      totalComp ← totalComp + (S1, S2를 합치는 과정에서 수행된 총 비교 횟수);
      return T;
}
```

7. (15점) 크기가 작은 순으로 정렬되어 있는 배열에서 수 x가 어디 있는지 찾으려 한다. 배열에 x가 있을 수도 있고 없을 수도 있다. 아래 알고리즘은 수업 시간에 배운 binary search를 테스트용으로 살짝 바꾼 것이다.

```
anotherSearch (A[ ], x, low, high)
        // A: array. x: search key   // low, high: array bounds
{
                if (low > high) return "Not found";
                mid = ⌊ (low+high) / 3 ⌋ ;
                if (A[mid] < x) return anotherSearch(A, x, mid+1, high)
                else if (A[mid] > x) return anotherSearch(A, x, low, mid-1)
                else return mid;
}
```

a. (5점) 이렇게 해도 x를 제대로 찾는가? (간단한 설명)

b. (10점) x를 제대로 찾는가에 상관없이 위 알고리즘은 점근적 수행 시간이 배열의 크기 n에 대하여 어떻게 될까? $O(\ )$로 대답하라. 이 알고리즘의 시간을 구하는 것은 수업시간에 제대로 배웠다고 할 수 없으므로 나름대로 생각해서 결론을 도출해 보라. (여러분의 결론이 어떻게 나왔는지 채점자가 납득할 정도가 되면 된다.) 자유로운 형식으로 설명하면 되고, 생각이 맞으면 점수를 줌.

8. (15점) 아래는 수업 시간에 배운 Reference를 이용한 stack의 Java 구현 예다. 여기서는 linked list의 첫 번째 원소가 stack top이 되도록 했다. 이것을 linked list의 마지막 원소가 stack top이 되도록 바꾸어 보아라. 여러분이 전체를 다 쓸 필요는 없고 아래에서 바뀌거나 추가되는 부분만 알아볼 수 있도록 쓰라. 각 line 앞의 번호는 여러분이 답을 쓸 때 해당 line을 언급할 수 있도록 붙인 것이다. (Java 문법은 틀려도 뜻만 맞으면 괜찮음)

```
1.  public class Node {
2.          private Object item;
3.          private Node next;
4.          public Node(Object newItem) {
5.                  item = newItem;
6.                  next = null;
7.          }
8.          public Node(Object newItem, Node nextNode) {
9.                  item = newItem;
10.                 next = nextNode;
11.         }
12.         public Object getItem( ) {
13.                 return item;
14.         }
15.         // setItem, setNext, getNext
16.         …
17. } end class Node
18. public class StackReferenceBased implements StackInterface{
19.         private Node top;
20.         public StackReferenceBased( ) {
21.                 top = null;
22.         }
23.         public boolean isEmpty( ) {
24.                 return (top == null);
25.         }
26.         public void push(Object newItem) {
27.                 top = new Node(newItem, top);
28.         }
29.         public Object pop( ) {
30.                 if (!isEmpty( )) {
31.                         Node temp = top;
32.                         top = top.getNext( );
33.                         return temp.getItem( );
34.                 } else {exception 처리};
35.         }
36.         public void popAll( ) {
37.                 top = null;
38.         }
39.         public Object peek( ) {
40.                 if (!isEmpty( )) return top.getItem( );
41.                 else {exception 처리};
42.         }
```

```
43. } // end class StackReferenceBased
```