# COMP171 Midterm Exam Fall 2004

1. Multiple Choice

   (a) (ii) $O(n)$

   (b) (ii) $O(logn)$

   (c) (i) $O(logn)$

   (d) (iii) 255

   (e) (ii) 4

2. Complexity

   (a) $i = n, i = \frac{n}{2}, i = \frac{n}{4}, \ldots \Rightarrow O(\log_2 n)$

   (b) line 6: $T(n-1)$, line 7: $T(n-1)$, line 8: $T(0)$ or $T(1)$, and $T(0) = T(1)$

$$\begin{aligned} T(n) &= T(n-1) + T(n-1) + T(1) \\ &= 2T(n-1) + T(1) \\ &= 2(2T(n-2) + T(1)) + T(1) \\ &= 2^2 T(n-2) + 2T(1) + T(1) \\ &\vdots \\ &= 2^{n-1}T(1) + 2^{n-1}T(1) + \ldots + T(1) \\ &= (2^n - 1)T(1) \\ &= O(2^n) \end{aligned}$$

   (c)

| | best case | worst case |
|---|---|---|
| unsorted array | O(1) | O(1) |
| sorted array | O(1) | O(n) |
| unsorted single-linked list | O(1) | O(1) |
| sorted single-linked list | O(1) | O(n) |
| binary search tree | O(1) | O(n) |

3. (a) "Same" as 3.34(1)

(b)
```
// return true if a cycle is found
checkCycle():
    return CheckCycle(root, root, 1, 2);

checkCycle(s, t, sStep, tStep):
    if (sStep > 0)
        if (s has left child)
            if(checkCycle(s->left, t, sStep-1, tStep))
                return true
        if (s has right child)
            if (checkCycle(s->right, t, sStep-1, tStep))
                return true
        return false

    else if (tStep > 0)
        if (t has left child)
            if(checkCycle(s, t->left, sStep, tStep-1))
                return true
        if (t has right child)
            if (checkCycle(s, t->right, sStep, tStep-1))
                return true
        return false

    else
        // compare
        if (s == t)
            return true
        else
            return checkCycle(s, t, 1, 2)
```

(c) Use a List to store all elements along the path down to the current node. Also store the information whether the right or left child has been taken, for all nodes along this path.

For every visited node, check all its ancestors (stored in the list).

4. Stacks and Queues

(a)
```
Reverse(Q):
    if !isEmpty(Q)
        temp = delete(Q),   // implicit stack
        Reverse(Q),         // recursive call
        add(Q, temp)        // "stack" pop
```

(b)
```
Reverse(Q):
        n = size(Q)
        for i = n downto 1
            // find last element in Q
            for j = 1 to i-1
                // cycle shift
                add(Q, delete(Q))
            temp = delete(Q)
            // and insert it into R
            add(R, temp)

        // copy R back into Q
        for i = 1 to n
            add(Q, delete(R))
```

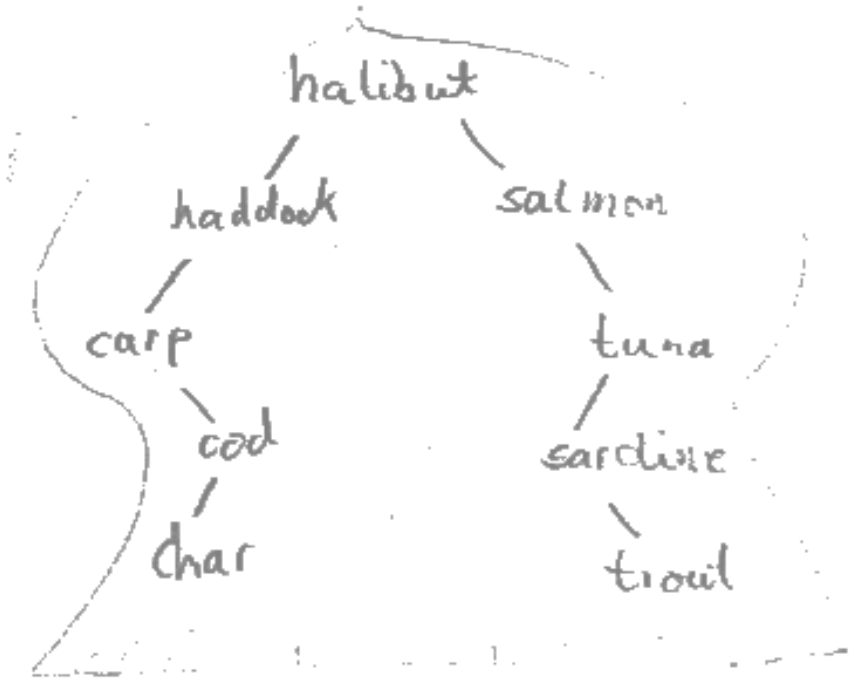
(c) BAC cannot be formed.

Because you need to push all letters so that B is the top element on $S_3$. In the first move, we cannot move A to $S_3$, so A will be at the bottom of $S_2$. In the second move, we cannot move B to $S_3$, so B will be on top of A in $S_2$. Therefore, A cannot be popped from $S_2$ earlier than B.

5. Trees

(a)


(b)




(c) Note must be in depth 3 at least. Otherwise it does not have a grand parent. Key might not be in the tree. Use stack/list to store a path from the root to the current node. Procedure as in lecture notes.

6. AVL Trees

(a)