Solution to CSE 250 Midterm Exam 2

Fall 2013
Time: 50 minutes.

Total points: 100   There are 4 questions.

Please use the space provided for each question, and the back of the page if you need to. Please do not use any extra paper. The space given per question is **a lot** more than sufficient to answer the question. Please be brief. Longer answers do not get more points!

- **No electronic devices of any kind. You can open your textbook and notes**
- **Please leave your UB ID card on the table**
- **This booklet must not be torn or mutilated in any way and must not be taken from the exam room**
- **Please stop writing when you are told to do so. We will not accept your submission otherwise.**
- **If you wanted to, you can answer the extra credit question without answering all of the other questions**

Your name:	
UBID:	

The rest of this page is for official use only. Do not write on the page beyond this point.

Problem Number	Score obtained
name and id (5 max)	
Problem 1 (20 max)	
Problem 2 (15 max)	
Problem 3 (30 max)	
Problem 4 (30 max)	
Total Score: (100 max)	

Problem 1 (20 points)**.** Order the following functions in increasing order of asymptotic growth rate. You don't have to explain how you get the order.

$$n^3,\ \frac{3^n}{n^3},\ \frac{n^2}{(\log n)^2},\ n^{1.5},\ n(\log n)^4,\ (\log n)^2,\ n^2\sqrt{n}\log n$$

Answer.

$$(\log n)^2,\ n(\log n)^4,\ n^{1.5},\ \frac{n^2}{(\log n)^2},\ n^2\sqrt{n}\log n,\ n^3,\ \frac{3^n}{n^3},$$

□

Problem 2 (15 points)**.** Use the recurrence tree method to solve the following recurrence relation

$$T(n) = 2T(n/2) + n^2$$

As usual, you can assume $T(k) = O(1)$ for $k \leq 10$. Show your work: draw the tree and do the summation.

Solution. Drawing trees on the computer takes too much time, so I didn't draw the trees and put figures here; but you should have. Briefly, the solutions are as follows.

$$T(n) = 2T(n/2)+n^2 = 2^kT(n/2^k)+\sum_{i=0}^{k-1} 2^i\cdot(n/2^i)^2 = 2^kT(n/2^k)+n^2\sum_{i=0}^{k-1} 1/2^i = 2^kT(n/2^k)+n^2\cdot\frac{1-(1/2)^k}{1-1/2}$$

For $k \geq 1$, it is clear that

$$1 \leq \frac{1-(1/2)^k}{1-1/2} \leq 2.$$

Hence, by setting $k = \log_2 n$ we have

$$T(n) == 2^kT(n/2^k) + \Theta(n^2) = n\Theta(1) + \Theta(n^2) = \Theta(n^2).$$

□

Problem 3 (30 points)**.** Let `Node` be the following structure

```
struct Node {
    int payload;
    Node* next;
};
```

Write **a recursive version** and an **iterative version** of a C++ function that takes a pointer to `Node` which is the head of a `NULL`-terminated singly linked list and modifies all of the nodes' payloads so that the *new* payload of each node is the sum of all the old payloads from that node to the end of the list. For example, if the list has four elements $a.4 \to b.2 \to c.5 \to d.2$, then after running the new payloads are $a.13 \to b.9 \to c.7 \to d.2$.

```
void iterative_add(Node* head) {
    Node* tmp;
    for (; head != NULL; head = head->next) {
        for (tmp = head->next; tmp != NULL; tmp = tmp->next)
            head->payload += tmp->payload;
    }
}

void recursive_add(Node* head) {
    if (head == NULL || head->next == NULL) return;
    recursive_add(head->next);
    head->payload += head->next->payload;
}
```

Problem 4 (30 points)**.** Let `Node` be the following structure

```
struct Node { int payload; Node* next; }
```

Write **a recursive version** and an **iterative version** of a C++ function that takes a pointer to `Node` which is the head of a `NULL`-terminated singly linked list. The function swaps the first and the second element, the third and the fourth, the fifth and the sixth, etc., and then returns a pointer to the head of the new list. **Only pointer manipulation is allowed. No point will be given if any of the payloads is modified.** If there are an odd number of elements, then the last one is left alone.
For example, if the input list is $a \to b \to c \to d \to e \to f \to$ NULL
then the output list is $b \to a \to d \to c \to f \to e \to$ NULL, and a pointer to b is returned
And, if the input list is $a \to b \to c \to d \to e \to$ NULL
then the output list is $b \to a \to d \to c \to e \to$ NULL, and a pointer to b is returned.

```
Node* iterative_swap(Node* head) {
    if (head == NULL || head->next == NULL) return head;
    Node* ret    = head->next;
    Node* prev   = NULL;
    Node* first  = head;
    Node* second;
    while (first != NULL && first->next != NULL) {
        second = first->next;
        if (prev != NULL) prev->next = second;
        first->next  = second->next;
        second->next = first;
        prev  = first;
        first = prev->next;
    }
    return ret;
}

Node* recursive_swap(Node* head) {
    if (head == NULL || head->next == NULL) return head;
    Node* first  = head;
    Node* second = head->next;
    first->next  = recursive_swap(second->next);
    second->next = first;
    return second;
}
```