Solution Midterm Exam EDA 05/11/2018

Proposed solution to problem 1

(a) The answers:

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
TRUE	X		X	X		X			X	X
FALSE		X			X		X	X		

Proposed solution to problem 2

(a) The cost in the worst case is $\Theta(n^3)$. The worst case always happens.

(b) The cost in the best case is $\Theta(n^2)$. The best case happens, for example, when the two matrices only have *true* in their coefficients.

The cost in the worst case is $\Theta(n^3)$. The worst case happens, for example, when the two matrices only have *false* in their coefficients.

(c) The function considers the Boolean matrices as matrices of integers, in which *false* is interpreted as 0 and *true* as 1. Then we apply Strassen's algorithm for the product of matrices. Let M be the resulting matrix. The coefficient of the i-th row and j-th column of M is

$$m_{ij} = \sum_{k=0}^{n-1} (a_{ik} \cdot b_{kj}).$$

As the coefficients of the input matrices are 0 or 1, the product as integer numbers is the same as the logical $\wedge$ operation. So m_{ij} counts the number of pairs (a_{ik}, b_{kj}) where both a_{ik} and b_{kj} are true at the same time. Therefore, to obtain the logical product we only have to define p_{ij} as 1 if $m_{ij} > 0$, and 0 otherwise. The cost is dominated by Strassen's algorithm, which has cost $\Theta(n^{\log_2 7}) \approx \Theta(n^{2.8})$.

Proposed solution to problem 3

(a) Given an integer $n \geq 0$, function *mystery* computes $\lfloor \sqrt{n} \rfloor$.

(b) First of all we find the cost $C(N)$ of the function *mystery_rec* in terms of the size $N = r - l + 1$ of the interval $[l, r]$. This cost follows the recurrence $C(N) = C(N/2) + \Theta(1)$, as at each call a recursive call is made on an interval with half the elements, and there is additional non-recursive work with constant cost. According to the master theorem of divisive recurrences, the solution to this recurrence is $\Theta(\log N)$. As *mystery*(n) consists in calling *mystery_rec* (n, 0, n+1), we can conclude that the cost of *mystery*(n) is $\Theta(\log n)$.

Proposed solution to problem 4

(a) $\Theta(n)$

(b) $\Theta(n^2)$

(c) $\Theta(n \log n)$

(d) $\Theta(n \log n)$

(e) A possible solution:

```
void my_sort(vector<int>& v) {
  int n = v.size();
  double lim = n * log(n);
  int c = 0;
  for (int i = 1; i < n; ++i) {
    int x = v[i];
    int j;
    for (j = i; j > 0 and v[j − 1] > x; −−j) {
      v[j] = v[j − 1];
      ++c;
    }
    v[j] = x;
    if (c > lim) {
      merge_sort(v);
      return;
    }
  }
}
```

(f) First of all we observe that, as the outermost **for** loop makes $n - 1$ iterations, each of which has cost $\Omega(1)$, the cost of the overall execution is $\Omega(n)$.

If for example the vector is already sorted in increasing order, then *my_sort* behaves as insertion sort: the innermost **for** loop is never entered, c is always 0, and *merge_sort* is not called. As each iteration of the outermost **for** loop has constant cost, the cost in this case is $\Theta(n)$. So the cost of *my_sort* in the best case is $\Theta(n)$.

To see the cost in the worst case, we distinguish two cases:

- Suppose that *merge_sort* is not called. Then the cost is proportional to the final value of variable c. As *merge_sort* is not called, we have that $c \le n \ln n$, and the cost is $O(n \ln n) = O(n \log n)$.
- Suppose that *merge_sort* is eventually called. If it is called at the end of the first iteration of the outermost **for** loop, then the cost is $O(n)$ from the innermost **for** loop plus $\Theta(n \log n)$ of the *merge_sort*. Altogether, the cost is $\Theta(n \log n)$.

  If *merge_sort* is called at the end of the second, or third, etc. iteration of the outermost **for** loop, then *merge_sort* was not called at the previous iteration of the outermost **for** loop. Moreover, in the last iteration c may have increased in i at most, so when *merge_sort* is called we have that $n \ln n < c \leq i + n \ln n \leq n + n \ln n \leq n \ln n + n \ln n = 2n \ln n$ (for n big enough), and hence $c = \Theta(n \log n)$. Since the cost of *merge_sort* is $\Theta(n \log n)$, altogether the cost is $\Theta(n \log n)$.

The worst case happens for example when the vector is sorted backwards, that is, in decreasing order. As in this case insertion sort has cost $\Theta(n^2)$, at some point of the execution of *my_sort* the subprocedure *merge_sort* will be called, and by the previous reasoning the cost will be $\Theta(n \log n)$.