CSE100 Algorithm Design and Analysis — Homework set #1
Fall semester 2018

Miguel Á. Carreira-Perpiñán

Total possible marks: 100. You have to submit this homework to a TA due Sep. 26 at 8 pm or at your own lab session (also at TA's office hours if it is before the deadline). Homeworks must be solved individually. This set covers chapters 1–4 of the textbook *Introduction to Algorithms*, 3rd. ed., by Cormen et al.

Exercise 1: asymptotic behavior (20 points).

1. (10 points) Assume you have two computers, C_A and C_B, capable of performing 10^6 and 10^8 operations per second, respectively. Both computers run a set of algorithms whose precise complexities $f(n)$ are given below. Determine the size n^* of the biggest input that can be processed in 1 second for each computer, as in the example.

$f(n)$	n^* for C_A	n^* for C_B
$\lg \lg n$		
$\sqrt{n}$	10^{12}	10^{16}
$14n + 4$		
$n \log_3 n + n$		
$n^2 + 7n$		
n^{10}		
2^n		
3^n		
$n!$		
n^n		

   The precise complexity tells you how many operations are performed to solve an instance of size n. Assume each operation takes the same time and that the input sizes are natural numbers $1, 2, 3, \ldots$

2. (10 points) Prove formally that $f(n) = an^2 + bn + c$ where $a > 0$ is $\Theta(n^2)$. *Hint*: find values for the constants c_1, c_2, n_0 in the definition of $\Theta(\cdot)$ and show it holds.

Exercise 2: sorting algorithms, correctness and runtime (44 points). Consider an algorithm Insertion-Merge-Sort with the following pseudocode:

```
INSERTION-MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    INSERTION-MERGE-SORT(A, p, q)
    INSERTION-MERGE-SORT(A, q + 1, r)
    INSERTIONMERGE(A, p, q, r)
```

where the InsertionMerge algorithm has the following pseudocode:

```
INSERTIONMERGE(A, p, q, r)
for j = q + 1 to r
    key = A[j]
    i = j − 1
    while i ≥ p and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

1. (9 points) Prove that InsertionMerge solves the same problem as the Merge algorithm of p. 30–31 in the textbook but works in place. *Hint*: use a loop invariant.
2. (3 points) Prove that Insertion-Merge-Sort sorts the input array A. *Hint*: use induction.
3. (6 points) Identify the best and worst case of InsertionMerge and compute the runtime $T(n)$ in asymptotic notation for each. Be explicit about summing the number of iterations in the loops.
4. (8 points) Identify the best and worst case of Insertion-Merge-Sort and compute the runtime $T(n)$ in asymptotic notation for each. Give the recurrence explicitly for each.
5. (5 points) Imagine that the partition in Insertion-Merge-Sort in two subarrays is $(\frac{9}{10}n, \frac{1}{10}n)$ instead of $(\frac{1}{2}n, \frac{1}{2}n)$. Give the recurrence and its runtime again (consider only the best case).
6. (4 points) Is Insertion-Merge better than Merge? Is Insertion-Merge-Sort better than Insertion-Sort or Merge-Sort? Consider the complexity in time and memory for each case.
7. (9 points) Give the pseudocode for an algorithm Selection-Merge-Sort(A, p, r) that modifies Selection-Sort(A) (exercise 2.2-2) in the same way as Insertion-Merge-Sort modified Insertion-Sort, so that it can merge two sorted arrays. Identify its best and worst case and compute the runtime $T(n)$ in asymptotic notation for each. Based on this, would this improve over Insertion-Merge-Sort? Explain.

Exercise 3: recurrent equations (36 points). (This is based on textbook problems 4.1 and 4.3.) Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$.

1. (6 points) $T(n) = T(9n/10) + n\sqrt{n}$
2. (6 points) $T(n) = T(\sqrt{n}) + 1$
3. (6 points) $T(n) = 2T(2n) + n^2$
4. (6 points) $T(n) = 7T(n/2) + n^2 + 3n + 1$
5. (6 points) $T(n) = T(n-1) + n$, base case: $T(0) = 0$
6. (6 points) $T(n) = 2T(n/4) + \sqrt{n}$

Justify your answers. If you use the master theorem, specify which case and show that its hypotheses are satisfied. If you use induction then you should prove induction hypothesis.

Bonus exercise: recurrent equations (10 points). Consider the recurrence

$$T(n) = aT(n/b) + f(n).$$

Case 3 of the master theorem requires that $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$ and that $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n. Prove that the regularity condition is always satisfied if $f(n) = n^k$. (This means we don't need to check it when f is a polynomial.)

Bonus exercise: recurrent equations (30 points). Consider the following particular type of recurrence:

$$T(n) = \begin{cases} 1, & n = 1 \\ aT(n/b) + n^c, & n > 1 \end{cases}$$

where $a \geq 1$ and $b > 1$ are integers, and $k = \log_b n$ is integer (that is, we can only pick sizes n of the form $n = b^k$ where $k \geq 0$ is an integer). Use mathematical induction to prove that

$$T(n) = \begin{cases} n^c(1 + \log_b n), & \text{if } c = \log_b a \\ \frac{\frac{a}{b^c} n^{\log_b a} - n^c}{\frac{a}{b^c} - 1}, & \text{if } c \neq \log_b a. \end{cases}$$

Hint: as a simpler example, see exercise 2.3-3.

Consequently, prove the master theorem for this recurrence, that is, prove that

$$T(n) = \begin{cases} \Theta(n^{\log_b a}), & \text{if } c < \log_b a \\ \Theta(n^c \lg n), & \text{if } c = \log_b a \\ \Theta(n^c), & \text{if } c > \log_b a. \end{cases}$$