EL9343 Homework 1

(Due September 25th, 2020)

No late assignments accepted

All problem/exercise numbers are for the third edition of CLRS text book

---

1. Prove the *Symmetry* property of $\Theta(\cdot)$, i.e. $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$.

2. Problem 3-2 in CLRS Text book.

3. You have three algorithms to a problem and you do not know their efficiency, but fortunately, you find the recurrence formulas for each solution, which are shown as follows:

A: $T(n) = 5T\left(\frac{n}{2}\right) + \Theta(n)$

B: $T(n) = 2T\left(\frac{9n}{10}\right) + \Theta(n)$

C: $T(n) = T\left(\frac{n}{3}\right) + \Theta(n^2)$

Please give the running time of each algorithm (in Θ notation), and which of your algorithms is the fastest (You probably can do this without a calculator)?

4. Use the substitution method to prove that $T(n) = 2T\left(\frac{n}{2}\right) + cn\log_2 n$ is $O(n\,(\log_2 n)^2)$.

5. First use the iteration method to solve the recurrence

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{3}\right) + n$$

Then use the substitution method to verify your solution.

6. Solving the recurrence: (Base is 2)

$$T(n) = 2T\left(\sqrt{n}\right) + \log n$$

(*Hint: Making change of variable*)

7. You have 5 algorithms, A1 took $O(n)$ steps, A2 took $\Theta(n\log n)$ steps, and A3 took $\Omega(n)$ steps, A4 took $O(n^3)$ steps, A5 took $o(n^2)$ steps. You had been given the exact running time of each algorithm, but unfortunately you lost the record. In your messy desk you found the following formulas:

(a) $7n\log_2 n + 12\log_2\log_2 n$

(b) $7\left(2^{2\log_2 n}\right) + \frac{n}{2} + 7$

(c) $\frac{2^{\log_4 n}}{3} + 120n + 7$

(d) $(\log_2 n)^2 + 75$

(e) $7n!$

(f) $2^{3\log_2 n}$

(g) $2^{2\log_2 n}$

For each algorithm write down all the possible formulas that could be associated with it.

8. Determine the time complexity of following code pieces (Figure 1), using big-O notation (every single statement has O(1) time complexity).

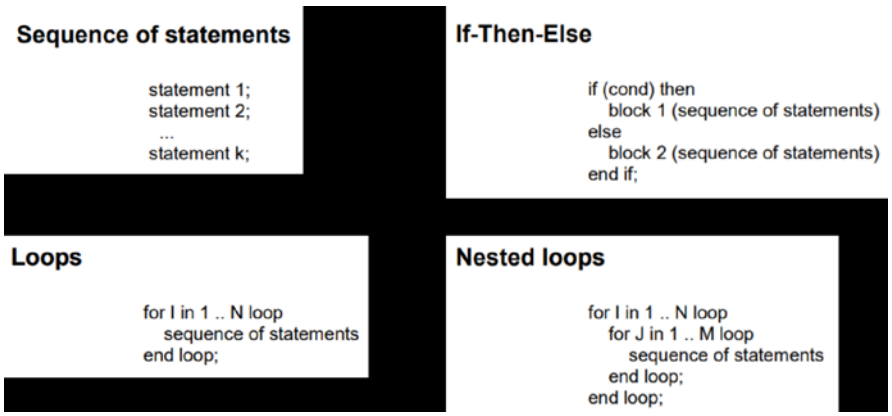

Figure 1

9. Analyze the time complexity of following code pieces (Figure 2), using big-O notation (assume 'A' in the following code is an integer array).

```
low = 0
high = N - 1
while (low <= high) {
    // invariants: value > A[i] for all i < low
                   value < A[i] for all i > high
    mid = (low + high) / 2
    if (A[mid] > value)
        high = mid - 1
    else if (A[mid] < value)
        low = mid + 1
    else
        return mid
```

Figure2: Code