University of Colombo School of Computing

SCS 1308 – Foundations of Algorithms

Take – Home 03

Instructions

- Try the following questions and upload your answer script as a zip file to the given link in the UGVLE on/before 21st December at 10 PM.
- Your answers should be typed and uploaded as a pdf file.
- It's an individual task. The deadline is 10:00 PM. Submissions will be accepted until midnight with a late penalty of 5% deducted for each hour past the deadline.
- Note: Rename your zip file as indexNo_1_last_namefname.zip.

01. Running time T(n) of processing n data items with a given algorithm is described by the recurrence:

$$\boxed{T(n) = k \cdot T\left(\frac{n}{k}\right) + c \cdot n; \quad T(1) = 0.}$$

Derive a closed form formula for T(n) in terms of c, n, and k. What is the computational complexity of this algorithm in a "Big-Oh" sense?

Hint: To have the well-defined recurrence, assume that $n = k^m$ with the integer $m = \log_k n$ and k. The closed-form formula can be derived by guessing from a few values and using then math induction.

02. Running time T(n) of processing n data items with another, slightly different algorithm is described by the recurrence:

$$\boxed{T(n) = k \cdot T\left(\frac{n}{k}\right) + c \cdot k \cdot n; \quad T(1) = 0.}$$

Derive a closed form formula for T(n) in terms of c, n, and k and determine computational complexity of this algorithm in a "Big-Oh" sense.

Hint: To have the well-defined recurrence, assume that $n = k^m$ with the integer $m = \log_k n$ and k. The closed-form formula can be derived either by "telescoping"[1] or by guessing from a few values and using then math induction.

03. What value of k = 2, 3, or 4 results in the fastest processing with the above algorithm?

Hint: You may need a relation $\log_k n = (\ln n)/(\ln k)$ where ln denotes the natural logarithm with the base $e = 2.71828...$).

04. Convert the following foo() functions to suitable recurrence relations and express the time complexity of each function in big O notation by using iterative substitution methods. Assume n>=1

A.

```
int32_t foo(int32_t n) {
  if (n == 1) {
    return 1;
  } // if
  return foo(n - 1) + foo(n - 1) + 1;
} // foo()
```

B.

```
void foo(int32_t n) {
  if (n == 0) {
    return;
  } // if
  for (int32_t i = 1; i < n; i *= 2) {
    std::cout << "281" << std::endl;
  } // for i
  foo(n - 1);
} // foo()
```

C.

```
void foo(int32_t n) {
  if (n == 1) {
    return;
  } // if
  foo(n / 2);
  for (int32_t i = 0; i < n; ++i) {
    std::cout << "281" << std::endl;
  } // for i
  foo(n / 2);
} // foo()
```

05. For each of the following recurrences, give an expression for the runtime T(n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply

1. $T(n) = 2T(n/2)+n\log n$
2. $T(n) = 2T(n/2)+n/\log n$
3. $T(n) = 2T(n/4)+n^{0.51}$
4. $T(n) = 0.5T(n/2)+1/n$
5. $T(n) = 16T(n/4)+n!$
6. $T(n) = 3T(n/2)+n^2$
7. $T(n) = 4T(n/2)+n^2$
8. $T(n) = T(n/2)+2n$
9. $T(n) = 2nT(n/2)+n^n$
10. $T(n) = 16T(n/4)+n$