Homework Assignment 02

Name: **Asrar Syed**
Due by: **October 06, 2025**
Course Section: **CSC 4520-006**

PROBLEM 1 [20 points]

The running times T(n) of algorithms are defined by the following recurrent equations.
Evaluate the running times using Θ-notation.

a) $T(n) = 4T\left(\frac{n}{4}\right) + 12$

Identify: $a = 4, b = 4, f(n) = 12$
Notice: $n^{log_4 4} = n^1 = n$
Notice: $f(n) = 12 = \Theta(1),$ so c = 0

n^1 grows faster meaning the recursion dominates: $T(n) = \Theta(n^{log_4 4}) = \Theta(\mathrm{n})$

b) $T(n) = 25T\left(\frac{n}{5}\right) + n^2$

Identify: $a = 24, b = 4, f(n) = n^2$
Notice: $n^{log_5 25} = n^2$
Notice: $f(n) = n^2 = \Theta(\mathrm{n}^2),$ so c = 2

n^2 grows at the same rate meaning they both dominate: $T(n) = \Theta(n^{log_4 25}) = \Theta(\mathrm{n}^2\mathrm{logn})$

c) $T(n) = 12T\left(\frac{n}{3}\right) + n^2$

Identify: $a = 12, b = 3, f(n) = n^2$
Notice: $n^{log_3 12} = n^{2.26}$
Notice: $f(n) = n^2 = \Theta(\mathrm{n}^2),$ so c = 2

$n^{2.26}$ grows faster meaning the recursion dominates: $T(n) = \Theta(n^{log_3 12}) = \Theta(\mathrm{n}^{2.26})$

d) $T(n) = 16T\left(\frac{n}{8}\right) + 2logn + \frac{n}{3}$

Identify: $a = 16, b = 8, f(n) = 2logn + \frac{n}{3}$
Notice: $n^{log_8 16} = n^{4/3}$
Notice: $f(n) = 2logn + \frac{n}{3} = \Theta(n),$ so c = 1 # constants 2 and 1/3 don't matter, logn is slower

$n^{4/3}$ grows faster meaning the recursion dominates: $T(n) = \Theta(n^{log_8 16}) = \Theta(\mathrm{n}^{4/3})$

PROBLEM 2 [10 points]

Suppose there are three options for dividing a problem of size n into subproblems:

✓ Algorithm A solves the problem by recursively solving eight instances of size n/2 and then combining their solutions in time Θ(n^3).

Recurrence form: $a = 8, b = 2$
Comput form: $log_2 8 = 3$, the combine cost is $\Theta(\mathrm{n}^3)$, so $f(n) = \Theta(\mathrm{n}^3)$, so c = 3

Since $log_b a = c$, then this is a tie: Master Theorem middle case is...

$$T(n) = \Theta(\mathrm{n}^3 \mathrm{logn})$$

✓ Algorithm B solves the problem by recursively solving twenty instances of size n/3 and then combining their solutions in time Θ(n^2).

Recurrence form: $a = 20, b = 3$
Comput form: $log_3 20 = 2.727$, the combine cost is $\Theta(\mathrm{n}^2)$, so $f(n) = \Theta(\mathrm{n}^2)$, so c = 2

Since $log_b a > c$, then the recursion term dominates: Master Theorem first case is...

$$T(n) = \Theta(n^{log_3 20}) = \Theta(\mathrm{n}^{2.727})$$

✓ Algorithm C solves the problem by recursively solving two instances of size 2n, and then combining their solutions in time Θ(n).

Algorithm C does not make sense for a terminating divide-and-conquer strategy as it expands to an ever-larger problem i.e. it doesn't terminate.

Which one is preferable, and why?

Algorithm B is preferable because it grows slower than Algorithm A.
For large n, any polynomial $n^{2.727}$ grows much slower than $n^3 \log n$. So B is asymptotically faster/preferable.

PROBLEM 3 [10 points]

Describe how the following integers would be sorted by a base-10 radix sort.

$$(329, 457, 657, 839, 436, 720, 355) \rightarrow (329, 355, 436, 457, 657, 720, 839)$$

There are two main types of radix sort i.e. Least Significant Digit (LSD) radix sort and Most Significant Digit (MSD) radix sort.

We will use Least Significant Digit (LSD) radix sort because all the numbers are 3-digits:

This sorts the digits, starting with the ones place (the least significant digit). It then sorts by the tens place, then the hundreds place, and so on, until the most significant digit is reached.

Start with: (329, 457, 657, 839, 436, 720, 355)

Order after one's sort: (720, 355, 436, 457, 657, 329, 839)

After tens place sort: (720, 329, 436, 839, 355, 457, 657)

After hundreds place sort: (329, 355, 436, 457, 657, 720, 839)

Final Sorted Order: **(329, 355, 436, 457, 657, 720, 839)**

PROBLEM 4 [10 points]

Describe a linear time algorithm to sort a set of n strings, each having k English characters.

A linear-time algorithm to sort n fixed-length (k) strings over a constant alphabet (like English letters) is to perform a **radix sort** using **counting sort** as the stable subroutine, starting from the last character and moving to the first.

The total running time is **O(n·k)**.

An example sort: ["dog", "cat", "car", "cot"]; k = 3

Pass by-pass (using stable counting sort):

3rd char: g, t, r, t → order: [car, cat, cot, dog]

2nd char: a, a, o, o → order: [car, cat, cot, dog] (still same)

1st char: c, c, c, d → final: [car, cat, cot, dog]

Result: sorted lexicographically.