Problem 1

(a) In the lecture notes, the algorithm employs four recursive calls, where the number of digits in each recursive call is set to m. This approach introduces many unnecessary recursions, where the only varying factor is the number of digits. For instance, given an input $(1, 0, 2)$, the algorithm makes four calls, but only one of them is not redundant. A more efficient choice for the algorithm, that is presented on slide 18, is to use the following:

$$\begin{aligned} e &\leftarrow \textbf{Multiply}(a, c, n-m), \\ f &\leftarrow \textbf{Multiply}(b, d, m), \\ g &\leftarrow \textbf{Multiply}(b, c, m), \\ h &\leftarrow \textbf{Multiply}(a, d, m). \end{aligned}$$

If n is even, then $n - m = m$; otherwise, $n - m = m - 1$. Nevertheless, the following implementation corresponds to the original algorithm as is presented on slide 18.

	Input	Output
1st Call	$(101, 011, 3)$	1111
2nd Call	$(1, 0, 2)$	0
3rd Call	$(0, 0, 1)$	0
4th Call	$(1, 0, 1)$	0
5th Call	$(1, 0, 1)$	0
6th Call	$(0, 0, 1)$	0
7th Call	$(1, 11, 2)$	11
8th Call	$(0, 1, 1)$	0
9th Call	$(1, 1, 1)$	1
10th Call	$(1, 1, 1)$	1
11th Call	$(0, 1, 1)$	0
12th Call	$(1, 0, 2)$	0
13th Call	$(0, 0, 1)$	0
14th Call	$(1, 0, 1)$	0
15th Call	$(1, 0, 1)$	0
16th Call	$(0, 0, 1)$	0
17th Call	$(1, 11, 2)$	11
18th Call	$(0, 1, 1)$	0
19th Call	$(1, 1, 1)$	1
20th Call	$(1, 1, 1)$	1
21st Call	$(0, 1, 1)$	0

(b) Similar to the previous case, a more efficient choice for the algorithm, that is presented on slide 21, is to use the following:

$$\begin{aligned} e &\leftarrow \textsc{Karatsuba-Multiply}(a, c, n-m), \\ f &\leftarrow \textsc{Karatsuba-Multiply}(b, d, m), \\ g &\leftarrow \textsc{Karatsuba-Multiply}(|a-b|, |c-d|, m). \end{aligned}$$

The following implementation corresponds to the original algorithm as is presented on slide 21.

	Input	Output
1st Call	$(111, 101, 3)$	100011
2nd Call	$(1, 1, 2)$	1
3rd Call	$(0, 0, 1)$	0
4th Call	$(1, 1, 1)$	1
5th Call	$(1, 1, 1)$	1
6th Call	$(11, 1, 2)$	11
7th Call	$(1, 0, 1)$	0
8th Call	$(1, 1, 1)$	1
9th Call	$(0, 1, 1)$	0
10th Call	$(10, 0, 2)$	0
11th Call	$(1, 0, 1)$	0
12th Call	$(0, 0, 1)$	0
13th Call	$(1, 0, 1)$	0

(c)

	Input	Output
1st Call	$\begin{pmatrix} 1 & 3 \\ 5 & 2 \end{pmatrix}, \begin{pmatrix} -1 & 4 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 7 \\ -5 & 22 \end{pmatrix}$
2nd Call	$(1), (3)$	(3)
3rd Call	$(4), (1)$	(4)
4th Call	$(7), (-1)$	(-7)
5th Call	$(2), (1)$	(2)
6th Call	$(3), (0)$	(0)
7th Call	$(1), (1)$	(1)
8th Call	$(-4), (3)$	(-12)

Problem 2

(a) To verify, we need to pick a function $f \in O(n^3)$ and show that $T(n) \leq f(n)$ for all n. The easiest choice is to pick the function f as $f(n) = Cn^3$ for some $C > 0$. We need to find a universal constant $C > 0$ such that $T(n) \leq Cn^3$.

- **Base case:** For $n = 1$, we have $T(1) = T(0)+1^2$. We can pick $C > T(0)+1$ so that $T(1) \leq C \times 1^3$.
- **Inductive hypothesis:** Assume $T(k) \leq Ck^3$ for some $k \geq 1$.
- **Inductive step:** We need to show that $T(k+1) \leq C(k+1)^3$. Expanding this:

$$\begin{aligned} T(k+1) &\overset{?}{\leq} C(k+1)^3, \\ \implies T(k) + (k+1)^2 &\overset{?}{\leq} C(k+1)^3, \\ \implies T(k) &\overset{?}{\leq} Ck^3 + 3Ck^2 + 3Ck + C - k^2 - 2k - 1, \\ T(k) &\overset{\checkmark}{\leq} Ck^3 + (3C-1)k^2 + (3C-2)k + C - 1, \quad \text{(Holds for } C \geq 1\text{)}. \end{aligned}$$

  Hence, if we choose $C = T(0) + 2$, then $T(k+1) \leq C(k+1)^3$ holds.

Therefore, $T(n) = O(n^3)$.

(b) If you try to prove this using induction, it won't work. Hence, you should suspect that that $T(n) \neq O(n^5)$. Notice that induction not working won't imply that the solution is incorrect. To prove $T(n) \neq O(n^5)$, you need to provide a separate argument.

Expanding the recursive function, we have

$$\begin{aligned} T(n) &= 2T(n-1) + n \\ &= 2^2T(n-2) + 2(n-1) + n \\ &= 2^3T(n-3) + 2^2(n-2) + 2(n-1) + n \\ &\vdots \\ &= 2^{n-1}T(1) + \sum_{i=0}^{n-2} 2^i(n-i) \end{aligned}$$

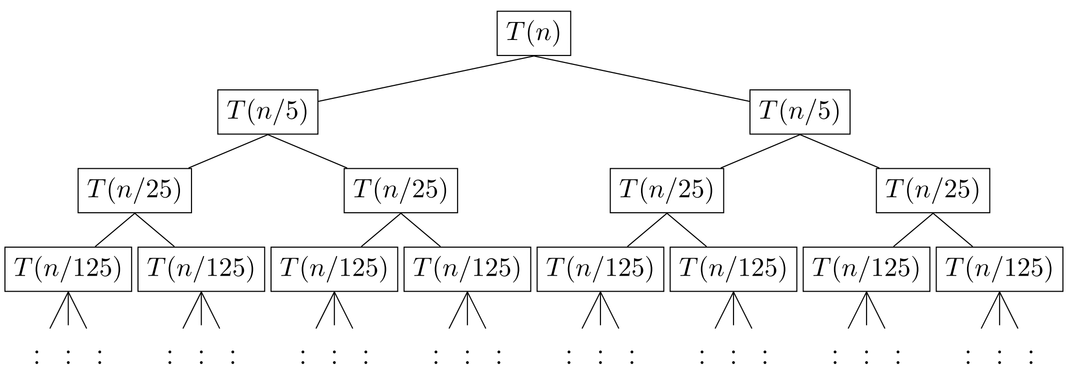
Clearly, $T(n) = \theta(2^n)$, hence $T(n) \neq O(n^5)$.

Problem 3

(a) Notice that the additional work on the level $i \geq 0$ of the tree is

$$\text{\# of nodes} \times \left(\frac{n}{5^i}\right)^3 = 2^i \times \left(\frac{n}{5^i}\right)^3.$$

Also, the depth of the tree is going to be $\log_5(n)$.



Assuming $T(1) = 1$, the total complexity is given by

$$\text{Total Work} = \sum_{i=0}^{\log_5(n)} 2^i \times \left(\frac{n}{5^i}\right)^3 = n^3 \times \sum_{i=0}^{\log_5(n)} \left(\frac{2}{125}\right)^i = \theta(n^3).$$

Notice that $\sum_{i=0}^{\infty}(2/125)^i < \infty$.

Problem 4

(a) The Master Method applies to recurrences of the form $T(n) = aT(\frac{n}{b}) + f(n)$. In this case $a = 5, b = 4$, and $f(n) = n^3$:

$$n^{\log_b a} = n^{\log_4 5} \approx n^{1.16}$$

$\rightarrow$ Here, since $f(n) = n^3$ grows faster than $n^{\log_b a}$, we fall into case 3 of the Master Method.

$$\rightarrow T(n) = \Theta(n^3)$$

(b) This time $a = 3, b = 8$, and $f(n) = n^2$:

$$n^{\log_b a} = n^{\log_8 3} \approx n^{0.52}$$

$$\to \text{Since } f(n) = n^2 \text{ grows faster than } n^{\log_b a}\text{, again we fall into case 3 of the Master Method.}$$

$$\to T(n) = \Theta(n^2)$$

Problem 5

(a) We cannot directly apply the Master Theorem, as it does not follow the format $T(n) = aT\left(\frac{n}{b}\right) + n^c$. The $\log\log n$ term must be dealt with to make the recurrence usable.

(b) Notice that $n^2\sqrt{n}\log(\log(n)) = n^{2.5}\log(\log(n))$. Therefore, the highest degree that can act as a lower bound for the formula is $\alpha = 2.5$. We now identify the proper case of the Master Theorem:

Case 1: If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Case 2: If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3: If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Since $2.5 > \log_3 2$, we use **Case 1**, which gives $\Omega(n^{2.5})$.

(c) Notice that n^β should be an upper bound for $n^{2.5}\log(\log(n))$. Hence, we have $\beta = 2.5 + \epsilon$ for any $\epsilon > 0$. In this case, since $\beta > \log_3 2$, we again use **Case 1**, which gives $\Omega(n^{2.5+\epsilon})$.

Problem 6

(a) The divide-and-conquer approach splits an $n \times n$ matrix multiplication into smaller $m \times m$ matrix multiplications. Since $n = m^l$, this means each dimension of the matrices is scaled down by a factor of m at each level of recursion. Using k multiplications for each $m \times m$ matrix multiplication, the recurrence for the time complexity is:

$$T(n) = kT(\tfrac{n}{m}) + O(n^2)$$

Here, $O(n^2)$ represents the time taken to combine the results of the recursive calls. By applying the Master Method with $a = k, b = m$, and $f(n) = n^2$:

$$n^{\log_b a} = n^{\log_m k}$$

So

$$T(n) = \begin{cases} \Theta(n^{\log_m k}) & \text{if } \log_m k > 2, \\ \Theta(n^2 \log n) & \text{if } \log_m k = 2, \\ \Theta(n^2) & \text{if } \log_m k < 2. \end{cases}$$

(b) Given that $m = 48$ and we want the complexity to be $O(n^{2.79})$:

$$\log_{48} k \leq 2.79 \to k = 49052$$

(c) We can utilize team Y's algorithm for multiplication. Suppose we want to multiply matrices A and B. By applying the squaring algorithm to the matrix $\left(\begin{smallmatrix} 0 & A \\ B & 0 \end{smallmatrix}\right)$, we obtain the matrix $\left(\begin{smallmatrix} AB & 0 \\ 0 & BA \end{smallmatrix}\right)$. This way, the complexity remains the same.