Algorithm Analysis and Design

LECTURE 3

Analyzing Recursive Algos and Solving Recurrence Relations

Joon Edward Sim

1. Motivation Example for Analysis of Recursive Algorithms

Consider the multiplicatin of two n-bit numbers, X and Y. Let

$$X = X_0 X_1 \dots X_{n-1}$$

and

$$Y = Y_0 Y_1 \dots Y_{n-1}.$$

where each X_i and Y_i represents a particular bit (actually it doesn't matter whether it's bit or digits) in the number X and Y respectively.

Figure 1 shows the typical traditional working for multiplying $n - bits$. Let's consider a single operation (could be addition or multiplication) on a pair of bits as one primitive step in this computation. How many primitive steps are there?

Firstly, we observe that there are n rows of partial products (i.e., the product of the whole X with a single bit of Y). Each partial product takes $O(n)$ primitive operations to compute. Let the function $f(n)$ represents the number of primitive operations it took to compute n rows of partial product. It is quite clear that $f(x) = O(n^2)$.

Secondly, we observe that there are $n - 1$ additions to perform to obtain the final product. Each of these additions take up to $2n$ primitive operations. Let $g(n)$ represents the number of primitive operations it take to compute the $n - 1$ additions. Clearly, $g(x) = O(n^2)$.

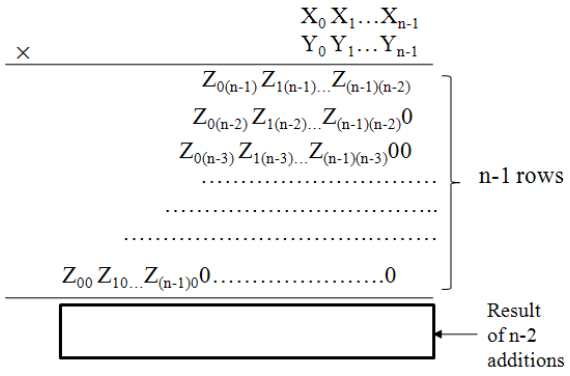

Figure 1. A typical working for n-bits multiplication. Z_{ij} represents the product of X_i and Y_j.

Finally, the total number of primitive steps is $f(n) + g(n)$. Since $f(n) = O(n^2)$ and $g(n) = O(n^2)$, $f(n)+g(n) = O(n^2)$. The traditional multiplication algorithm is in the order of $O(n^2)$.

The first section of this lecture lies in proposing an algorithm that improves the complexity of the integer multiplication algorithm. We shall discover that to do this, we require recursive functions, which in turn motivates the need to learn recurrence relations to analyze recursive functions.

1.1. Improving Integer Multiplication: a first try

Algorithm 1: RM1(X, Y)

Input: X, Y : n-bits strings
Result: returns X × Y

```
1 begin
2     if |X| = 1 then
3         Return X × Y;
4     end
5     Let X = X_A × 2^(n/2) + X_B;
6     Let Y = Y_A × 2^(n/2) + Y_B;
7     return
      RM1(X_A, Y_A) × 2^n +(RM1(X_A, Y_B)+RM1(X_B, Y_A))×2^(n/2)+
      RM1(X_B, Y_B);
8 end
```

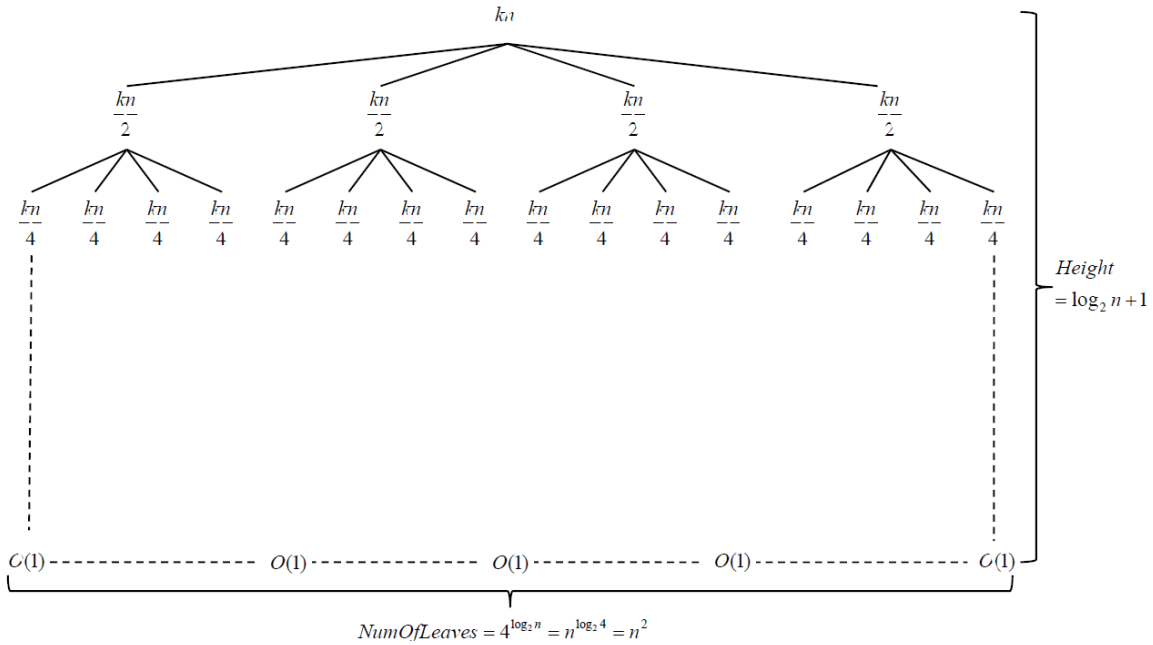

Figure 2. Recursion Tree of RM1

The correctness of the algorithm can be proven using strong mathematical induction. How this is done, is left as an exercise for the reader.

We now focus our attention on the analysis of the running time complexity of the algorithm. While for iterative algorithms, the running time complexity is bounded by the number of iterations, the running time complexity of recursive algorithms is a combination of the number of times we recurse, and also the number of primitive operations executed each time we recurse.

For algorithm 1, it is clear that a number of operations are involved each time we recurse:

- A multiplication by 2^n and 2^n. This is merely a bit shift of n and $\frac{n}{2}$ bits respectively. This can be done in $O(n)$ primitive bit operations.
- 3 additions of bit strings of at most length $2n$ (Why?). Again, these can be done in $O(n)$ primitive bit operations.

Clearly then, if $T(n)$ represents the number of primitive bits operations executed for X and Y of length $n > 1$, $T(n)$ can bounded by the following :

$$T(n) \leq 4T(\frac{n}{2}) + kn \quad \text{for some large enough constant } k.$$

This form of equation where $T(n)$ is expressed in terms of some T terms with smaller n, are known as recurrence relations. Examples of how recurrence relations can be solved are shown in section 2.

The bad news is that there's no one single method that can solve all recurrences. However, the good news is that we do know a few approaches that tend to work in practice. The methods are:

(1) Substitution Method. (Wild guess, and prove correctness by induction)
(2) Recursion Tree. (Draw the recursion tree, reason about the height of the tree, number of nodes and values of the nodes. Then add up all the node values.)
(3) Master's theorem. (A method that works for a specific class of recurrence relations.)
(4) K-degree linear homogeneous recurrences with constant coefficients (A specific class of recurrence relations where there are useful heuristic for solving them.)

Here, we try to analyze the running time of RM1 using the recursion tree method. Figure 2 shows the recursion tree as we expand out $T(n)$. Each node branches 4 times until the base case is reached, where there are no more recursions. So the height of the tree is $log_2 n + 1$. The number of leaves will be n^2.

Adding up all the values in each node and assuming $n = 2^m$, we get

$$\begin{aligned} T(n) &\leq kn + 4k(\frac{n}{2}) + 16k(\frac{n}{4}) + \ldots + n^2 k(\frac{n}{n}) \\ &\leq kn(1 + 2 + 4 + 16 + \ldots + 2^m) \quad [GeometricSeries!] \\ &\leq kn(\frac{2^{m+1} - 1}{2 - 1}) \\ &\leq kn(2n - 1) \\ &\leq 2kn^2 - nk \\ &= O(n^2), \end{aligned}$$

which is the same order of complexity as the traditional multiplication algorithm! We must do better.

1.2. Improving Integer Multiplication: try again

Let us try to improve the above. Can we reduce the number of recursions?

We observe that

$$\begin{aligned}(X_A + X_B)(Y_A + Y_B) &= X_AY_A + X_AY_B + X_BY_A + X_BY_B \\ X_AY_B + X_BY_A &= (X_A + X_B)(Y_A + Y_B) - X_AY_A - X_BY_B\end{aligned}$$

Also observe that from line 7 of algorithm 1, the calls to RM1(X_A, Y_B) and RM1(X_B, Y_A) costs 2 recursions and their purpose is to compute $X_AY_B + X_BY_A$. However, the equation above shows that we can change this from 2 multiplications and 1 additions into 1 multiplications and 3 additions/subtractions! This means if we use the latter expression, we have one recursion less!

Algorithm 2: RM2(X, Y)

```
   Input: X, Y : n-bits strings
   Result: returns X × Y
 1 begin
 2   if |X| = 1 then
 3     Return X × Y;
 4   end
 5   Let X = X_A × 2^(n/2) + X_B;
 6   Let Y = Y_A × 2^(n/2) + Y_B;
 7   P_1 ← RM2(X_A + X_B, Y_A + Y_B);
 8   P_2 ← RM2(X_A, Y_A);
 9   P_3 ← RM2(X_B, Y_B);
10   return P_2 × 2^n + (P_1 − P_2 − P_3) × 2^(n/2) + P_3;
11 end
```

For RM2, if $T(n)$ represents the number of primitive bits operations executed for X and Y of length $n > 1$, $T(n)$ can bounded by the following :

$$T(n) \leq 3T(\frac{n}{2}) + kn \quad \text{for some large enough constant } k.$$

Again, Figure 3 shows us the recursion tree of RM2. Adding up all the values in the nodes, we get

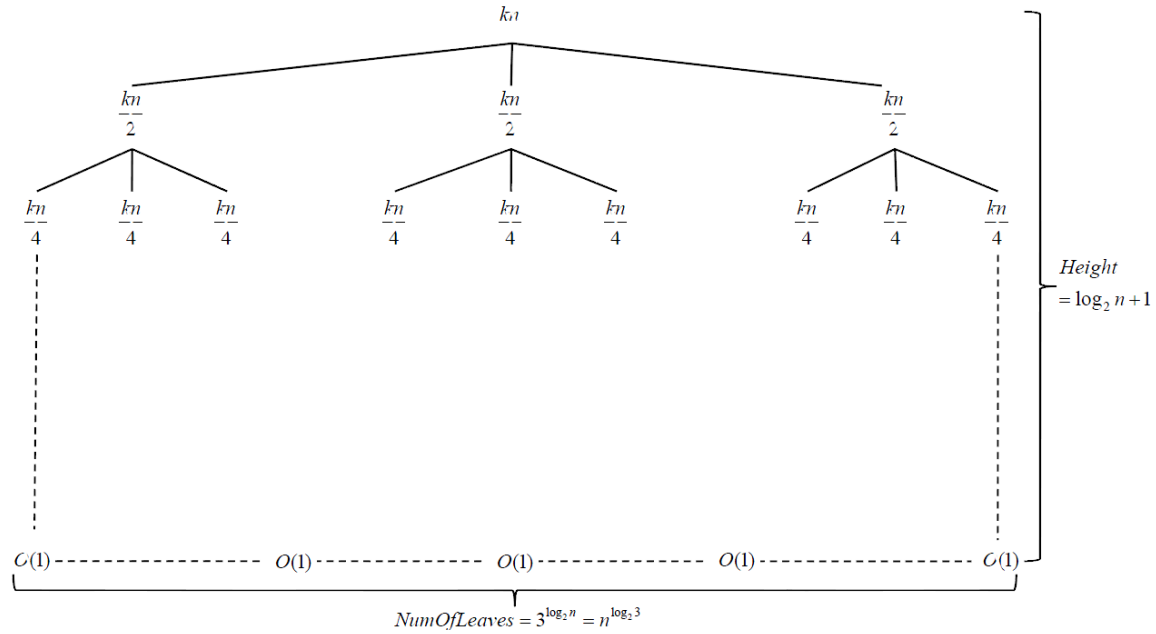

Figure 3. Recursion Tree of RM2

$$
\begin{aligned}
T(n) &\leq kn + 3k(\frac{n}{2}) + 9k(\frac{n}{4}) + \ldots + 3^{log_2 n}(\frac{kn}{n}) \\
&\leq kn(1 + \frac{3}{2} + \frac{9}{4} + \frac{27}{8} + \ldots + (\frac{3}{2})^m) \quad [\text{Since } n = 2^m \text{ and } m = log_2 n] \\
&\leq kn((\frac{3}{2})^0 + (\frac{3}{2})^1 + (\frac{3}{2})^2 + (\frac{3}{2})^3 + \ldots + (\frac{3}{2})^m) \quad [GeometricSeries!] \\
&\leq kn(\frac{(\frac{3}{2})^{m+1} - 1}{2 - \frac{3}{2}}) \\
&\leq 2kn((\frac{3}{2})^{m+1} - 1) \\
&\leq 3k3^m - 2kn \\
&\leq 3k3^{log_2 n} - 2kn \\
&\leq 3kn^{log_2 3} - 2kn \\
&\leq 2kn^{1.59} - 2kn \quad [\text{Since } log_2 3 \approx 1.59.] \\
&= O(n^{1.59}),
\end{aligned}
$$

which is better than the traditional multiplication algorithm.

2. Solving Recurrences

Example 1. $T(n) = 2T(n-1)$, initial condition $T(1) = 1$.

Wild guess:$T(n) = 2^{n-1}$

Prove it by mathematical induction:

- **Base case**: $T(1) = 1 == 2^0 = 2^{1-1}$ done.
- **Induction Hypothesis**: assume that $T(k) = 2^{k-1}$
- **Induction Step:** Need to show that $T(k+1) = 2^k$ if induction hypothesis is true.

$$\begin{aligned} T(k+1) &= 2T(k) \quad [\text{ according to the original recurrence }] \\ &= 2 \times 2^{k-1} \quad [\text{ by the assumption }] \\ &= 2^{(k+1)-1} \quad [\text{ simplification }] \end{aligned}$$

Example 2. $T(n) = T(n-1) + 2n - 1$, induction condition $T(1) = 1$.

Wild guess:$T(n) = n^2$

Prove it by mathematical induction:

- **Base case:** $T(1) = 1 = 1^2$ done.
- **Induction Hypothesis:** assume that $T(k) = k^2$.
- **Induction Step:** Need to show that $T(k+1) = (k+1)^2$ if induction hypothesis is true.

$$\begin{aligned} T(k+1) &= T(k+1-1) + 2(k+1) - 1 \quad [\text{ according to the original recurrence }] \\ &= k^2 + 2(k+1) - 1 \quad [\text{ by the assumption }] \\ &= k^2 + 2k + 1 = (k+1)^2 \quad [\text{ simplification }] \end{aligned}$$

Example 3. $T(n) = 3T(n-1) - 2T(n-2)$, initial conditions $T(0) = 5, T(1) = 8$.

This is an example of a second-order linear homogeneous recurrence with constant coefficients.

Characteristic equation:

$$r^2 - 3r + 2 = 0$$

Solving the characteristic equation:

$$r_1 = 1, r_2 = 2$$

Applying the rule that when the roots of the characteristic equation are real and distinct, the general form term of the sequence is expressed in:

$$\begin{aligned} T(n) &= \alpha_1 (r_1)^n + \alpha_2 (r_2)^n. \\ &= \alpha_1 + \alpha_2 (2)^n. \quad [\text{Applying } r_1 = 1,\ r_2 = 2.] \end{aligned}$$

Using the initial conditions:

$$T(0) = \alpha_1 + \alpha_2 = 5$$

$$T(1) = \alpha_1 + 2\alpha_2 = 8$$

Solving the simultaneous equations:

$$\alpha_1 = 2, \alpha_2 = 3$$

So solution of recurrence is

$$T(n) = 2 + 3 \times 2^n.$$

Example 4. $T(n) = 2T(n-1) - T(n-2)$, initial conditions $T(0) = 1, T(1) = 2$. This is an example of a second-order linear homogeneous recurrence with constant coefficients.

Characteristic equation:

$$r^2 - 2r + 1 = 0$$

Solving the characteristic equation:

$$r_1 = r_2 = 1 \quad [\text{ two equal roots of 1 }]$$

Applying the rule that when the roots of the characteristic equation real and equal, the general form term of the sequence is expressed in:

$$\begin{aligned} T(n) &= \alpha_1(r_1)^n + \alpha_2 n(r_1)^n. \\ &= \alpha_1 + \alpha_2 n. \quad [\text{Applying } r_1 = r_2 = 1.] \end{aligned}$$

Using the initial conditions:

$$T(0) = \alpha_1 = 1$$

$$T(1) = \alpha_1 + \alpha_2 = 2$$

Solving the simultaneous equations:

$$\alpha_1 = 1, \alpha_2 = 1$$

So solution of recurrence is

$$T(n) = 1 + n.$$

Example 5. $T(n) = T(n-1) + 1$, initial condition $T(1) = 1$.

This is a linear and one-degree recurrence with constant coefficients. But it is **non-homogeneous**.

Convert it into a homogeneous recurrence via a trick:

$$T(n) = T(n-1) + 1 \quad [\text{recurrence for n}]$$

$$T(n+1) = T(n) + 1 \quad [\text{recurrence for n+1}]$$

Perform a subtraction:

$$\begin{aligned} T(n+1) - T(n) &= (T(n)+1) - (T(n-1)+1) \\ &= T(n) - T(n-1) \\ T(n+1) &= 2T(n) - T(n-1) \text{ [homogeneous!]} \end{aligned}$$

The characteristic equation will be:

$$r^2 - 2r + 1 = 0$$

which looks a lot like Example 4. So

$$T(n) = \alpha_1 + \alpha_2 n$$

Using the initial conditions:

$$T(1) = \alpha_1 + \alpha_2 = 1$$
$$T(2) = \alpha_1 + 2\alpha_2 = 2$$

Solving the equations, we get $\alpha_2 = 1$ and $\alpha_1 = 0$. So $T(n) = n$.

Example 6. $T(n+1) = T(n) + 2n + 1$, initial condition $T(1) = 1$.

Again – non-homogeneous,

Try the trick

$$T(n+2) = T(n+1) + 2(n+1) + 1$$

Subtracting $T(n+1)$ from $T(n+2)$,

$$\begin{aligned} T(n+2) - T(n+1) &= (T(n+1) + 2(n+1) + 1) - (T(n) + 2n + 1) \\ T(n+2) &= 2T(n+1) + 2n + 2 + 1 - T(n) - 2n - 1 \\ T(n+2) &= 2T(n+1) - T(n) + 2 \quad [\text{ not homogeneous! }] \end{aligned}$$

Reapply the trick:

$$T(n+3) = 2T(n+2) - T(n+1) + 2$$

Subtracting $T(n+2)$ from $T(n+3)$,

$$\begin{aligned} T(n+3) - T(n+2) &= (2T(n+2) - T(n+1) + 2) - (2T(n+1) - T(n) + 2) \\ T(n+3) &= 3T(n+2) - 3T(n+1) + T(n) \text{ [homogeneous! finally!]} \end{aligned}$$

The characteristic equation is

$$r^3 - 3r^2 + 3r - 1 = 0$$

and the roots will be $r_1 = r_2 = r_3 = 1$, since

$$(r-1)^3 = r^3 - 3r^2 + 3r - 1.$$

The general form of the solution for cubic characteristic equations with equal roots is:

$$\begin{aligned} T(n) &= \alpha_1 r_1^n + \alpha_2 n r_2^n + \alpha_3 n^2 r_3^n \\ &= \alpha_1 + \alpha_2 n + \alpha_3 n^2 \quad [\text{Applying } r_1 = r_2 = r_3 = 1] \end{aligned}$$

Using the initial conditions, we derive the following:

$$\begin{aligned} \alpha_1 + \alpha_2 + \alpha_3 &= 1 \\ \alpha_1 + 2\alpha_2 + 4\alpha_3 &= 4 \\ \alpha_1 + 4\alpha_2 + 9\alpha_3 &= 9 \end{aligned}$$

Solving the equations, $\alpha_3 = 1$, $\alpha_2 = \alpha_1 = 0$. Therefore,

$$T(n) = n^2$$

Example 7. Fibonacci numbers: $F(n) = F(n-1) + F(n-2)$, initial conditions $F(0) = 0, F(1) = 1$.

Characteristic equation:

$$r^2 - r - 1 = 0$$

Solving the equation:

$$r_1, r_2 = \frac{1 \pm \sqrt{5}}{2}$$

The roots are real and distinct, so the form of the solution would be

$$\begin{aligned} F(n) &= \alpha_1 r_1^n + \alpha_2 r_2^n \\ &= \alpha_1 (\frac{1+\sqrt{5}}{2})^n + \alpha_2 (\frac{1-\sqrt{5}}{2})^n \end{aligned}$$

Using the initial conditions:

$$\begin{aligned} \alpha_1 + \alpha_2 &= 0 \\ \alpha_1 (\frac{1+\sqrt{5}}{2}) + \alpha_2 (\frac{1-\sqrt{5}}{2}) &= 1 \end{aligned}$$

Solving the equations, we get $\alpha_1 = \frac{1}{\sqrt{5}}$ and $\alpha_2 = -\frac{1}{\sqrt{5}}$, and

$$F(n) = \frac{1}{\sqrt{5}} (\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}} (\frac{1-\sqrt{5}}{2})^n$$

Example 8. $T(n) = T(\frac{n}{2}) + 1$, initial condition $T(1) = 1$.

There is a problems here, homogeneous recurrence only work when the RHS has T with arguments $n-1$, $n-2$ etc, but not $\frac{n}{2}$.

Another trick: let $n = 2^m$, then recurrence looks like

$$T(2^m) = T(2^{(m-1)}) + 1$$

Let $T(2^m) = S(m)$ and so,

$$S(m) = S(m-1) + 1.$$

Being careful,

$$S(0) = T(2^0) = T(1) = 1.$$

$S(m)$ is right now non-homogeneous, but using the subtraction trick:

$$S(m) = m + 1$$

The working for this is left for the reader.

Now change the variable back to T and n:

$$T(2^m) = m + 1$$

$$T(n) = m + 1 \quad \text{(Applying } n = 2^m\text{)}$$

$$T(n) = log_2 n + 1 \quad \text{(Applying } n = 2^m\text{)}$$

Master's Theorem

The Master's method can be applied to particular family of recurrences of the form

$$T(n) = aT(\frac{n}{b}) + f(n)$$

where $a \geq 1$, $b > 1$, $f(n)$ asymptotically positive for large enough n i.e., $\forall n \geq n_0, f(n) \geq 0$.

There are 3 cases that can be applied for the master's method. Each of the cases depend on comparing $f(n)$ with various forms of $n^{\log_b a}$.

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$ **for some** $\epsilon > 0$

In this case, $T(n) = \Theta(n^{\log_b a})$.

An example that falls into this case would when $T(n) = 4T(\frac{n}{2}) + n$. Here, $f(n) = n, a = 4, b = 2$. Clearly, there would be some $\epsilon > 0$ where $n = O(n^{\log_2 4 - \epsilon}) = O(n^{2-\epsilon})$ (An example ϵ could be 0.1). Therefore, $T(n) = \Theta(n^2)$.

Case 2: $f(n) = \Theta(n^{\log_b a}(\lg n)^k)$ **for some** $k \geq 0$

Note that due to the nature of Θ (tight bound), for each case, only one particular k value is going to satisfy the condition. In this case, $T(n) = \Theta(n^{\log_b a}(\lg n)^{k+1})$.

An example that falls into this case would when $T(n) = 4T(\frac{n}{2}) + n^2$. Here, $f(n) = n^2, a = 4, b = 2$. Clearly, when k=0, where $n^2 = \Theta(n^{\log_2 4})(\lg n)^0) = O(n^2)$ Therefore, $T(n) = \Theta(n^2 \lg n)$.

Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ **for some** $\epsilon > 0$**, and** $af(\frac{n}{b}) \leq (1-\epsilon')f(n)$ **for some** $\epsilon' > 0$**.**

Note that we need 2 non-zero positive constants ϵ and ϵ' to satisfy the conditions. In this case, $T(n) = \Theta(f(n))$.

An example that falls into this case would when $T(n) = 4T(\frac{n}{2}) + n^3$. Here, $f(n) = n^3, a = 4, b = 2$. Clearly, $n^3 = \Omega(n^{\log_2 4 + \epsilon}) = \Omega(n^{2+\epsilon})$ for some $\epsilon > 0$ (An example ϵ could be 0.5). Also, $4(\frac{n}{2})^3 = \frac{n^3}{2} \leq (1 - \epsilon')n^3$ for some $\epsilon' > 0$ (An example ϵ' could be 0.1). Therefore, $T(n) = \Theta(n^3)$.