Algorithms

1 Review on Tree Method

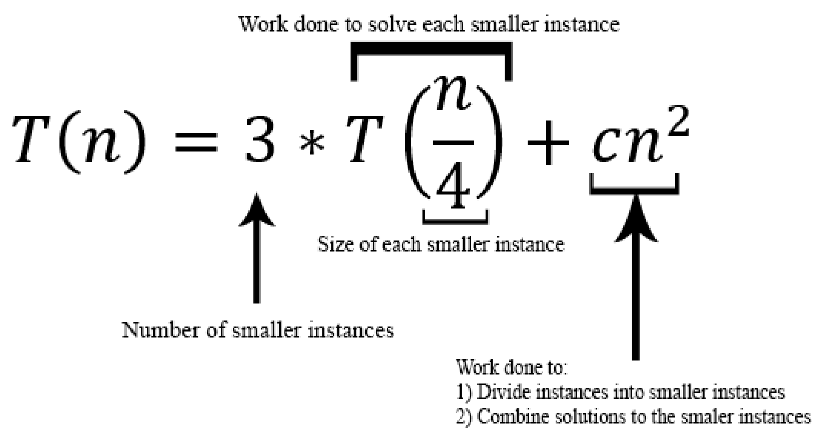

Figure 1: The meaning of different parts inside a complexity equation

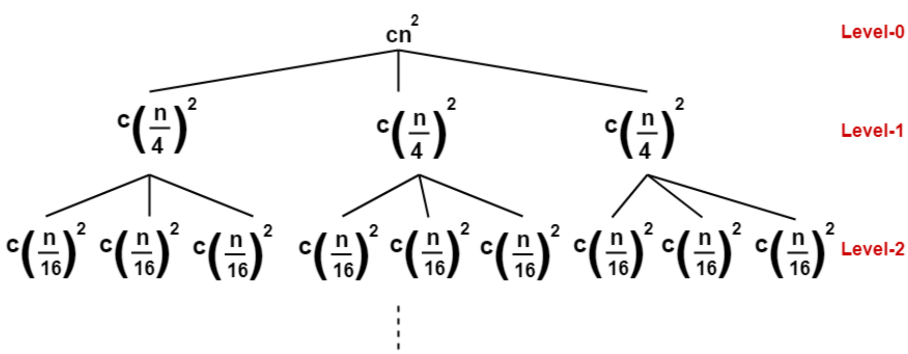

Figure 2: Representation of given T(n) by Akshay Singhal

1)Continue on last lecture, for this problem, what is the size at depth i?

$$\frac{n}{4^i} \tag{1}$$

2)At the bottom of the tree, $n/4^i$ is equal to 1, then we are able to solve for the depth of the tree.

```
4^i = n
i = log base 4(n)
```

So the depth of the tree is $log_4 n$.
As we known, the number of nodes at each level is 3^i and the cost at depth i for each nodes is $C * [instancesize]^2$.

3)How to derive the cost of each node at depth i?
We first look at the instance size of depth i.

$$n, \frac{n}{4}, \frac{n}{4^2}, \frac{n}{4^3}, \frac{n}{4^4}... \tag{2}$$

So, for this problem cost of each node at depth i is

$$C * (\frac{n}{4^i}) \tag{3}$$

Then we combine these to each other,
The cost of each node at level i is equal to:

$$number of nodes * cost of each node = 3^i * C * (\frac{n}{4^i}) = 3^i * C(\frac{n^2}{4^{2i}}) = \frac{(3^i * Cn^2)}{4^{2i}} = (\frac{3}{16})^i Cn^2 \tag{4}$$

As we already calculated the cost of each node at level i, to calculate the total cost, we need to know the number of nodes
The depth at the last level is $log_4 n$.
Therefore the number of nodes on it is $3^{log_4 n}$, which is equivalent to $n^{log_4 3}$.

Therefore the total cost at the last level is

$$n^{log_4 3} * (T) \in \Theta(n^{log_4 3}) \; which \; T \; is \; a \; constant. \tag{5}$$

The time complexity

$$\begin{aligned} T(n) = & \sum_{i=0}^{log_4(n-1)} (\frac{3}{16})^i Cn^2 + \Theta(n^{log_4 3}) \\ & < \sum_{i=0}^{\infty} (\frac{3}{16})^i Cn^2 + \Theta(n^{log_4 3}) >>>>> The \; upper \; bound \\ & = \frac{1}{/} 1 - \frac{3}{16}) Cn^2 + \Theta(n^{log_4 3}) \\ & = \frac{16}{13} Cn^2 + \Theta(n^{log_4 3}) \\ & \in O(n^2) \end{aligned} \tag{6}$$

2 Master Method (Theorem)

T(n) = aT(n/b) + f(n) $>>>>>$**General Divide and Conqure Recurrence**

If f(n) $\in \Theta(n^d)$**:**

$$\begin{aligned} T(n) \in \Theta(n^d) >>>>>> if\ a < b^d \\ T(n) \in \Theta(n^d log_n) >>>>> if\ a = b^d \\ T(n) \in \Theta(n^{log_b a}) >>>>> if\ a > b^d \end{aligned} \tag{7}$$

Example1 T(n) = 2T($\frac{n}{3}$)+ n^2 + 7

```
in this case, a = 2, b = 3, d = 2
b^d = 9,
So a < b^d,
```

According to the Master Theorem,
T(n)$\in \Theta(n^d)$
T(n)$\in \Theta(n^2)$

Example2 T(n) = 2T($\frac{n}{2}$)+ n

```
in this case, a = 2, b = 2, d = 1
b^d = 2,
So a = b^d,
```

According to the Master Theorem,
T(n)$\in \Theta(n^d logn)$
T(n)$\in \Theta(nlogn)$

3 Example of Binary Search

```
10   13   19   27   54   62   68   75
```

1) Divide - smaller instance, size = $\frac{n}{2}$

2) Conquer - find the key X in the smaller instance

3) Combine - not needed here

Worst case Time complexity of Binary Search:

1)Divide and Conquer Method:

basic operation: comparison of key and L[mid]
input size: n = the number of the list elements
We assume n is a power of 2
Worst case occur when key < L[0] or > L[len-1];
To say it clearly it happens in two situation: the key is greater than all list elements or the key is less than all list elements.
If key > all list elements:

```
W(n) = W(n/2) + 1
```

Then we list initial terms:

```
W(1) = 1

W(2) = W(2/2) + 1 = W(1) + 1 = 2

W(4) = W(4/2) + 1 = W(2) + 1 = 3

W(8) = W(8/2) + 1 = W(4) + 1 = 4
```

Therefore, we can guess W(n)= lg(n) + 1, and conform this guess with a proof by induction.

2)Master Method

```
W(n) = W(n/2) + 1
```

In this formula, we can observe that a = 1, b = 2,
Since f(n)=1, which is n^0, therefore d = 0,
So b^d = 1 = a,
According to the Master Theorem,
W(n) $\in \Theta lg(n)$