DEPARTMENT OF CYBER SECURITY

DESIGN AND ANALYSIS OF ALGORITHMS	
SUBMITTED BY	211042 Noman Masood Khan A
SEMESTER	Fifth
SUBJECT	Design and Analysis of Algorithms
SUBMITTED TO	Dr. Ammar Masood

1. Is 2^{n+1} = O(2^n)? Is 2^{2n} = O(2 n)

- Part I : 2^{n+1} = O(2^n)

For the proving the big O condition, we must know the **value** of **c** and **no**.

For the f (x) to be the O of g (x), then

- f (x) ≤ c . g (x)

for the 2^{n+1} = O(2^n), then

- $2^{n+1} \leq c\,(2^n)$

Dividing both the sides with 2^n

- $\dfrac{2^{n+1}}{2^n} \leq c\,\dfrac{2^n}{2^n}$
- $2^{n+1-n} \leq c$
- $2^1 \leq c$
- $2 \leq c$

The results show us that for any value of c greater than or equals to 2,

- the condition $2^{n+1} \leq c\,(2^n)$ is going to satisfy
- **Hence 2^{n+1} = O (2^n)**

- Part II : Is 2^{2n} = O (2^n)

For the proving the big O condition, we must know the **value** of **c** and **no**.

Finding the value of c.

- 2^{2n} = O (2^n)
- $2^{2n} \le c\,(2^n)$

Dividing both the sides with 2^n

- $$\frac{2^{2n}}{2^n} \le c\,\frac{2^n}{2^n}$$
- $2^{2n-n} \le c$
- $2^n \le c$

The result shows us that:

- The constant c will have value which is greater than or equals to 2^n.
- But 2^n has variable in power. Variable values are always changing and increasing.
- Thus, c can never have value which is greater than or equals to 2^n.
- **This proves us that the $2^{2n} \neq$ O (2^n)**

2. Prove Theorem 3: For any two functions f (n) and g(n), we have f (n) = Θ(g(n)) if and only if f (n) = O(g(n)) and f (n) = Ω(g(n)).

- For f(n) = Θ g(), the condition is c1g(n) ≤ f(n) ≤ g(n)

We have two functions f(n) and g(n)

- f(n) = $4n^2$
- g(n) = $7n^2$

- <u>For Big-O</u>

$4n^2$ = O $(7n^2)$ when c =1. This means that for all c ≥ 1, the f (n) ≤ g(n).

- <u>For Big- Ω</u>

$7n^2$ = Ω $4n^2$ when c = 1. The f(n) ≥ c g(n) for all c ≥1.

- <u>For Big- Θ</u>

For the Θ g(n) to hold, the f(n) should be O g(n) and g(n) be Ω f(n). Since $4n^2$ = O $(7n^2)$ and $7n^2$ = Ω $4n^2$ when c = 1. Therefore, the $4n^2$ = Θ $7n^2$.

- This proves the Theorem

3. Indicate for each pair of expressions (A, B) in the table below whether A is O, o, Ω, ω, or Θ of B.

A	B	O	o	Ω	ω	Θ
lg^k**n**	n^2	NO	YES	NO	NO	NO
n^k	c^n	Yes	NO	NO	NO	NO
$\sqrt{n}$	$n^{\sin n}$	NO	NO	NO	NO	NO
2^n	$2^{n/2}$	NO	NO	YES	NO	NO
n^{logc}	c^{logn}	Yes	NO	YES	NO	YES
lg (n!)	lg(n^n)	YES	NO	YES	NO	YES

4. Use the substitution method to show that each of the following recurrences has the asymptotic solution

- a.) T (n) = T (n – 1) + n has solution T (n) = O(n^2).

Since the Guess state is given in the question so

T (n) = O (n) ^ 2

T (n) ≤ c (n) ^2

T (n-1) ≤ c (n -1) ^2 **(I)**

Given that:

T (n) = T (n-1) + n

Using the equation, I in the value of T (n-1)

≤ c (n-1)^2 +n

= c (n^2 – 2n +1) + n

= c n^2 - 2nc + 1c + n

= cn^2 - (2n - 1)n + c for some value of c

≤ cn^2

- Showed that the T (n) = O(n^2)

- b. $T(n) = T(n/2) + \Theta(1)$ has solution $T(n) = O(\lg n)$.

The guess value is given that the $T(n) = O(\lg n)$

$T(n) = O(\lg n)$

$T(n) \le c(\lg n)$

$T(n/2) \le c \lg(n/2)$ ------ (I)

Proving by Induction

$T(n) = T(n/2) + \Theta(1)$

Substituting the value of T (n/2)

$\le c \lg(n/2) + 1$

$= c \lg n - c \lg 2 + 1$

For value of some value of c

$T(n) \le c \lg(n)$

- Showed that $T(n) = O(\lg n)$

5. Show that a substitution proof with the assumption T (n) ≤ c*n^2 fails. Then show how to subtract a lower-order term to make a substitution proof work

Given

T (n) = 4T (n/2) + n ---- (I)

We made assumption that T(n) ≤ cn^2 . Now, we are going to show that.

T(n) ≤ cn^2

T(n/2) = c$(n/2)^2$ ------ (ii)

Substituting I in II

T(n) ≤ 4c$(n/2)^2$ +n

= 2cn^2 +n ------- (iii)

The equation iii shows that our assumption fails, which was that the T (n) ≤ c n^2 . The equation iii is never going to be smaller than the c n^2 for any n.

Now showing that subtracting the lower order term to make the substitution proof works.

Let's again guess

T(n) ≤ c n^2 -d*n ------ (I)

Putting T(n/2) in the T(n) which is the recurrence function.

T(n) $\leq 4\left(c\left(\frac{n}{2}\right)^2 - d\left(\frac{n}{2}\right)\right) + n$

$= cn^2 - 2dn + n$ ------------ (II)

Finding the value of d by setting the Eq I and II equal

-2dn+n = -d n

d = n

Then

$= cn^2 - 2dn + n$ = c n^2 - n^2

This means setting d equal to n our assumption will become true.

6. The recurrence T (n) = 2T (n – 1) + 1 has the solution T (n) = O(2 n). Show that a substitution proof fails with the assumption T (n) ≤ c 2 n Then show how to subtract a lower-order term to make a substitution proof work.

Suppose we have function T

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

We suppose that

- $T(n) \leq c2^n$
- T(n-1) = c 2^{n-1}

Substituting this into the recurrence relation gives us

$$T(n) \leq 2\,(\text{c}\,2^{n-1}) + 1$$

$$= c2^n + 1$$

Because of the +1 with the $c2^n$, the assumption prove fails.

Since the assumption has not proved, so I am going to show that subtracting lower order term to make the substitution work.

Now I am taking the function

T(n) $\leq$ $c2^n$ – n

Following the same steps

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

$\leq \ 2(c2^{n-1} - (n-1)) + 1$

$= \ c2^{n} - 2n + 3$

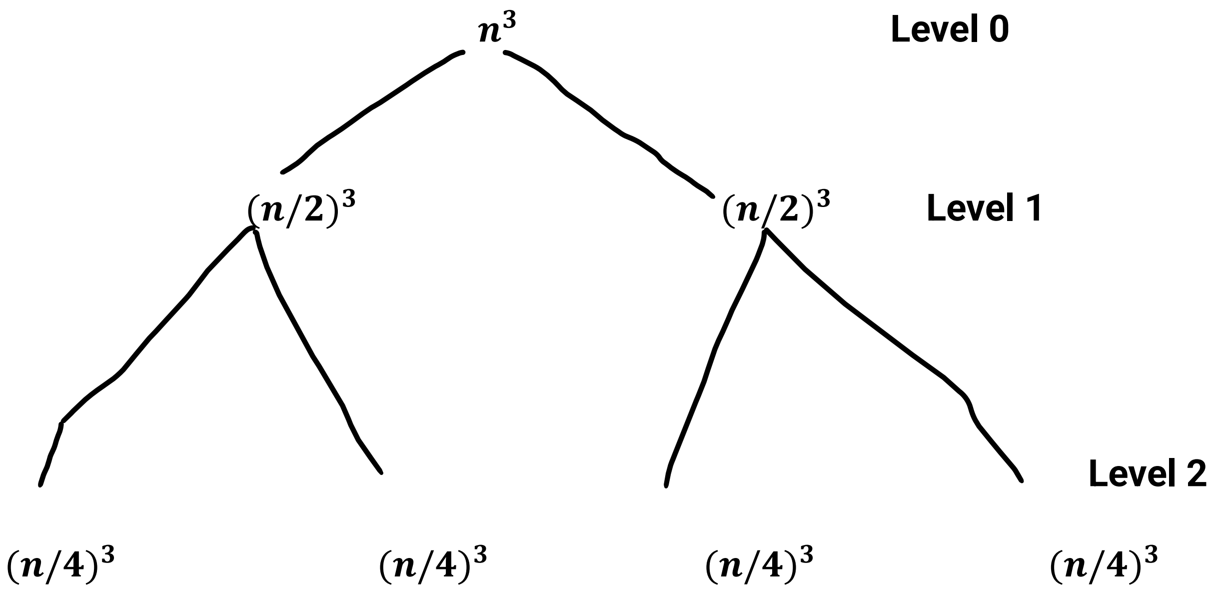
Now Clearly seen that n and c are large values then

$$\boldsymbol{c2^{n} - 2n + 3 \leq c2^{n} - n}$$

- This proved the requirements

> Sketch its recursion tree, and guess a good asymptotic upper bound on its solution

- A. Sketching the Recursion Tree for T (n) = T (n/2) + n^3



- A. Good Asymptotic upper bound

  A good upper bound for this algorithm will $O(n)^3$.

A. Substitution method to verify answer

T(n) $\leq$ O $(n)^3$

T(n/2) $\leq$ c $(n/2)^3$

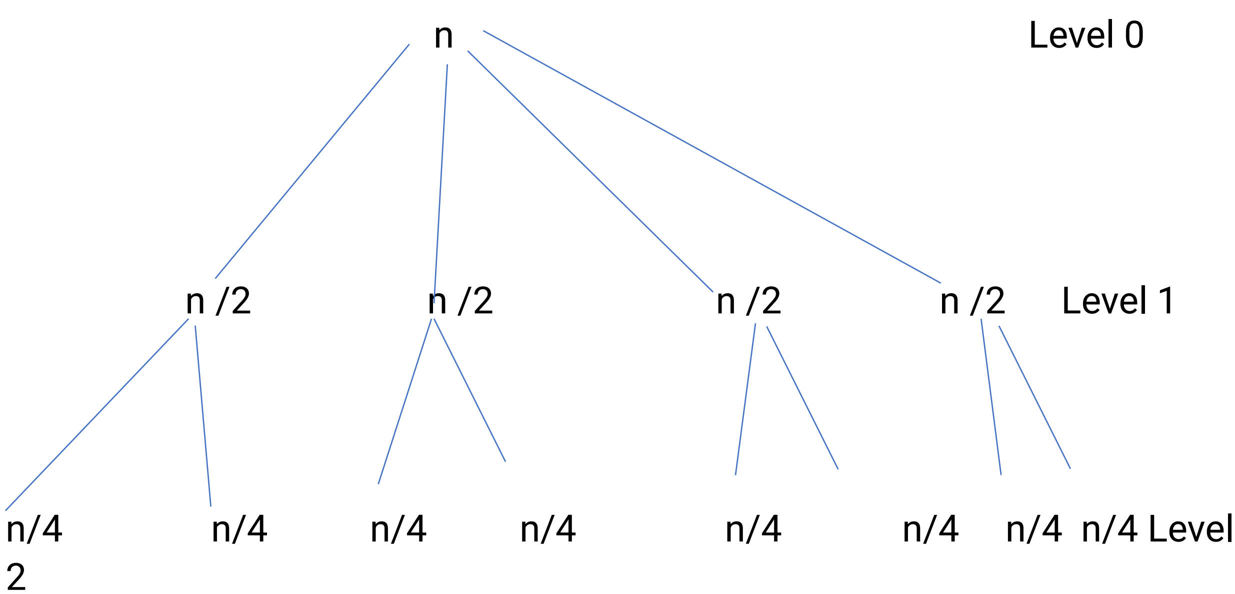
Now substituting this value in the recursion algorithm.

T (n) = T (n/2) **+** $\boldsymbol{n^3}$

T (n) = $(\boldsymbol{n/2})^3$ **+** $(\boldsymbol{n})^3$

= n^3

B. **Recursion Tree for the T (n) = 4T (n/2) + n**



B. Upper Bound for the function

The function T (n) = 4T (n/2) + n upper bound will n.

T (n) = O (n)

B. Substitution Method to prove the answer

We guessed that

$$T(n) = O(n)$$

$$T(n/2) \leq c(n/2)$$

Now substituting values in the recursion algorithm

$T(n) \leq 2cn + n$

$T(n) \leq n$

Use the master method to give tight asymptotic bounds for the following recurrences

- a. T (n) = 2T (n/4) + 1

Here a =2 , b=4, and f(n) =1

So f (n) = $O(n^{log_b a-\epsilon})$

$= O(n^{0.5-\epsilon})$) when $\epsilon > 0$

From here using the first case of master theorem

It is proved that *T*(*n*)=Θ *n*

- b. T (n) = 2T (n/4) + $\sqrt{n}$

Here, a=2, b=4, and f(n)=n.

We have that f(n)=Θ(n log of a to base b)=Θ(n). Therefore, by case 2 of the Master Theorem, we have T(n)=Θ((nlogba)logn)=Θ((n)logn).

- C. T (n) = 2T (n/4) + $\sqrt{n}$ lg ^2 n.