Assignment 1

Ayyappa Reddy
R11888944

Question 1

a. Question:
Determine the class $g(n)$ that the function $(n^2 + 1)^{10}$ belongs to. Use limit analysis to prove your assertion.

Hint: Consider using the limit comparison test by expressing $(n^2 + 1)^{10}$ as $n^{20} \cdot (1 + n^{-2})^{10}$. As n approaches infinity, can you make any simplifications to determine the dominant term?

Solution:
Given $f(n) = (n^2 + 1)^{10}$, we can expand and re-write it as:

$$f(n) = n^{20} \cdot (1 + n^{-2})^{10}$$

Now, let's choose $g(n) = n^{20}$. So, then we will start solving the limit of the ratio $f(n)/g(n)$ as n tends to infinity:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

$$\lim_{n\to\infty} \frac{n^{20} \cdot (1 + n^{-2})^{10}}{n^{20}}$$

Let's analyze the behaviour of "n" that is $(1 + n^{-2})^{10}$ which is completely bounded here with n tends to infinity and n^{-2} tends to 0, and $(1 + n^{-2})^{10}$ also tends to 1.

So, as per our understanding that $(1 + n^{-2})^{10}$ this is bounded by a constant.

$$\lim_{n\to\infty} \frac{n^{20} \cdot (1 + n^{-2})^{10}}{n^{20}}$$

$$\lim_{n\to\infty} 1 = 1$$

Hence the limit of the ratio $f(n)/g(n)$ is a positive number. From that we can say $f(n) = (n^2 + 1)^{10}$ belongs to the same class as $g(n) = n^{20}$. we can say that $f(n)$ is in the class of $\theta(n^{20})$.

b. Question:
Express the function $p(10n^2 + 7n + 3)$ informally and demonstrate its membership in a specific class using limit analysis.

Hint: By comparing the function $p(10n^2 + 7n + 3)$ with a well-known function class by focusing on the dominant term as n approaches infinity. You might want to express the function as a sum of individual terms and simplify.

Solution:

Given $p(10n^2 + 7n + 3)$, Now, Let's break down it into different individual terms

The given function can also be written as follows:

$$p(10n^2 + 7n + 3) = p(10n^2) + p(7n) + p(3)$$

Now, let's analyze each of the individual term that which we have break down earlier:

1. $p(10n^2)$: This term is having a degree of 2.

2. $p(7n)$: This term is having a degree of 1.

3. $p(3)$: This term is a constant.

As n tends to infinity, the dominant term in the expression will be, the one which is having an highest degree. In that case, it is $p(10n^2)$ because of (n^2) which is degree of 2.

Now, let's consider the limit analysis as n tends to infinity:

$$\lim_{n\to\infty} \frac{p(10n^2 + 7n + 3)}{p(10n^2)}$$

Let's rewrite the terms in individual as follows:

$$\lim_{n\to\infty} \frac{p(10n^2) + p(7n) + p(3)}{p(10n^2)}$$

Since $p(10n^2)$ is the dominant term, we can simplify say that the limit as follows:

$$\lim_{n\to\infty} \frac{p(10n^2) + p(7n) + p(3)}{p(10n^2)}$$

$$\lim_{n\to\infty} \frac{p(10n^2)}{p(10n^2)}$$

$$\lim_{n\to\infty} 1 = 1$$

Finally the limit n tends to infinity is a positive number. From that we can say the function $p(10n^2+7n+3)$ belongs to same class as $p(10n^2)$.we can say that $p(10n^2+7n+3)$ is in the class of $O(n^2)$.

---

c. Question:
Find the class $g(n)$ for the function $2n\log(n+2)^2 + (n+2)^2 \cdot \log(n^2)$. Provide the reasoning behind your conclusion.

Hint: Apply logarithmic properties to simplify the expression Pay attention to $2n\log(n+2)^2 + (n+2)^2 \cdot \log(n^2)$.the terms that dominate as n becomes larger. Consider isolating the highest-order terms and their factors to determine the class g(n).

Solution:

At first, we need to simplify the given function by using logarithmic properties:

$$2n\log(n+2)^2 + (n+2)^2 \cdot \log(n^2)$$

By using the properties of logarithms, we can rewrite these functions $log(n+2)^2$ and $\log(n^2)$ as below:

$$log(n+2)^2 \quad as \quad 2\log(n+2) \quad and \quad \log(n^2) \quad as \quad 2\log(n)$$

Let's replace the above re-written functions of $log(n+2)^2$ and $\log(n^2)$ and then we can re-write the logarithmic function as below:

$$2n \cdot 2\log(n+2) + (n+2)^2 \cdot 2\log(n)$$

Now, we will expand this $(n+2)^2$ to n^2+4n+4:

$$4n\log(n+2) + (n^2+4n+4) \cdot 2\log(n)$$

And now, let's multiply this $n^2+4n+4 \ \ with \ \ 2\log(n)$:

$$4n\log(n+2) + (2n^2+8n+8)\log(n)$$

Now, the dominant term as n becomes larger:

1. For the first we take the logarithmic functions $4n\log(n+2)$ and $2n^2\log(n)$: As n tends to infinity the term $2n^2\log(n)$ dominates, because it grows much and more faster

when compared to $4n\log(n+2)$.

2. These mentioned ($8n$ and 8) constant terms won't impact that much significantly on the growth rate of the function because these are constants.

Hence we can say that the dominant term is $2n^2\log(n)$.

And the class of $g(n)$ is $O(n^2\log(n))$, which is denoting the growth rate of given function is constrained by $n^2\log(n)$ as n tends to infinity.

---

d. Question:
Identify the class $g(n)$ that encompasses the function $2n^{-1}+3n^{-1}$. Justify your choice with reasoning.

Hint: To simplify the expression $2n^{-1}+3n^{-1}$, to find a common denominator for the terms.Consider the behaviour of the terms as n approaches infinity. Compare the simplified expression with the well-known complexity classes to determine the appropriate class of $g(n)$.

Solution:

Given function:

$$2n^{-1}+3n^{-1} \qquad (1)$$

Now, let's solve the above function of equation (1) and gets the simplified in fractions expression with same denominators as mentioned below:

$$\frac{2}{n}+\frac{3}{n}$$

After Solving:

$$\frac{2+3}{n}=\frac{5}{n}$$

As n tends to infinity.

- The term $\frac{5}{n}$ becomes smaller as n becomes larger.

- The simplified expression is $\frac{5}{n}$, which converges to zero as n tends to infinity.

- The class $g(n)$ describes this behavior is $O(\frac{1}{n})$.

Because, it signifies that the function's growth rate is bounded and approaches a constant value as n becomes larger.

where as $O(\frac{1}{n})$ is not allowed in terms of Big-O notation so we will be considering it as $O(1)$.

---

e. Question:
Convert the function $\log_2 n$ into a well-known complexity class. Briefly explain your conversion.

Hint: Express $\log_2 n$ using the base-2 logarithm property. Consider how logarithmic complexities are commonly represented in terms of big O notation, and identify the corresponding complexity class that closely matches the logarithmic behavior of $\log_2 n$.

Solution:

Given function of $\log_2 n$ represents the logarithm of n to the base 2.

which is in the form of $\log_b a$ represents that $\frac{\log a}{\log b}$.
-The base-2 logarithm property states that $\log_2 n$ can be converted into $\frac{\log n}{\log 2}$.

$$\log_2 n = \frac{\log n}{\log 2}$$

- As n increases the growth rate of $\log_2 n$ remains relatively slow compared to linear or polynomial growth, making it a logarithmic complexity class.

- Therefore, $\log_2 n$ exhibits the same logarithmic growth rate as $\log n$, and we can describe it as $O(\log n)$. Where as 2 is a constant it doesn't affect the growth rate.

-The conversion of $\log_2 n$ into $O(\log n)$ is justified by the consistent logarithmic behavior across different bases and the standard practice of representing logarithmic complexities as $O(\log n)$ in computational complexity analysis.

Question: 2

a.
Analyze the recurrence relation $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$ Use substitution to establish its complexity class, providing a step-by-step explanation.

Hint:
Begin by performing a substitution to simplify the recurrence relation. $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$. Let $n = 2^k$ and express the relation in terms of k.Then, solve for the recurrence in the new variable k and observe the pattern that emerges. Keep in mind that $T(2)$ can serve as the base case. As you analyze the pattern, consider well-known recurrence relations and their corresponding complexity classes.

Solution:

Given recurrence relation:

$$T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$$

From the given hint we can take $n = 2^k$:

Step 1: By Substituting $n = 2^k$:

Rewriting my expression by replacing "n" as $n = 2^k$. In the given $T(n)$ and the recurrence relation becomes:

$$T(2^k) = \sqrt{2^k} \cdot T(\sqrt{2^k}) + 2^k \tag{1}$$

Step 2: Let's expand the equation (1) as below:

$$T(2^k) = 2^{k/2} \cdot T(2^{k/2}) + 2^k \tag{2}$$

Step 3: Now let's divide the equation by 2^k:

$$\frac{T(2^k)}{2^k} = \frac{2^{\frac{k}{2}} \cdot T(2^{\frac{k}{2}})}{2^k} + 1$$

$$\frac{T(2^k)}{2^k} = \frac{T(2^{\frac{k}{2}})}{2^{k-(k/2)}} + 1$$

$$\frac{T(2^k)}{2^k} = \frac{T(2^{\frac{k}{2}})}{2^{(k/2)}} + 1 \tag{3}$$

Step 4: From the above equation (3) We have our recurrence relation in terms of k. so, let's Define the function:

$$A(k) = \frac{T(2^k)}{2^k} \tag{4}$$

Now substitute equation (4) in equation (3) then the recurrence relation becomes:

$$A(k) = A(K/2) + 1 \tag{5}$$

- Let's substitute "K" values as $K/2, K/4, ...$ till we observe the pattern.

Step 5: Now substitute that $A(K/2)$ in equation (5) to observe the pattern:

For $k = K/2$:

$$A(K) = A(K/2) + 1$$

$$A(K/2) = A(K/2^2) + 1 \tag{6}$$

Now substitute the above equation no.(6) in (5) as follows:

$$A(K) = (A(K/2^2) + 1) + 1$$

$$A(K) = A(K/2^2) + 2 \tag{7}$$

Step 6: Now substitute that $A(K/4)$ in equation (5) to observe the pattern:

For $k = K/4$:

$$A(K) = A(K/2) + 1$$

$$A(K/4) = A(K/2^3) + 1 \tag{8}$$

Now substitute the above equation no (8) in (7) as follows:

$$A(K) = (A(K/2^3) + 1) + 2$$

$$A(K) = A(K/2^3) + 3 \tag{9}$$

Now substitute that $A(K/8)$ in equation (5) to observe the pattern:

For $k = K/8$:

$$A(K) = A(K/2) + 1$$

$$A(K/8) = A(K/2^4) + 1 \tag{10}$$

Now substitute the above equation no (10) in (9) as follows:

$$A(K) = (A(K/2^4) + 1) + 3$$

$$A(K) = A(K/2^4) + 4 \tag{11}$$

Finally, we have observed a series of pattern from the equations (5) , (7) , (9) , (11) as follows:

$$A(K) = A(K/2) + 1$$

$$A(K/2) = A(K/2^2) + 2$$
$$A(K/4) = A(K/2^3) + 3$$
$$A(K/8) = A(K/2^4) + 4$$

Then we can write Sth term as $A(k) = A(k/2^S) + S$.

Now let's take a base case $A(1) = 1$

$$(k/2^S) = 1$$
$$k = 2^S$$

where as

$$S = logk$$

Now let's see A(k) from the base case as $A(1) = 1$

$$A(K) = logk$$

Where as

$$\frac{T(2^k)}{2^k} = log(K)$$

$$T(2^k) = 2^k logk$$

Finally the complexity of the class is shown below:

$$T(n) = \Theta(n \log(\log(n)))$$

Determining the Complexity of the class:

So, Finally the complexity class of the given recurrence relation $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$ is $\Theta(n \log(\log(n)))$.

---

b. Question:
Investigate the recurrence relation $T(n) = 2T\left(\frac{n}{2}\right) + n^2 \log(n)$. Express it as summation of sub problems and determine its complexity class. Include the details of your analysis.

Solution:

Given recurrence relation:

$$T(n) = 2T(n/2) + n^2 \log n \qquad (1)$$

The above recurrence relation tells that the problem n is divided into two sub problems of size $2T(n/2)$ and $n^2 \log n$.

Now, let's use Summation of sub problems to determine its complexity class.

So, Let's first express the recurrence for smaller instances by dividing the given problem in to two halves:

Let's substitute T(N/2) in equation (1)

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2 \log\left(\frac{n}{2}\right) \qquad (2)$$

Let's substitute T(N/4) in equation (2) to follow the iteration and to find out the occurrence as follows:

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^2 \log\left(\frac{n}{4}\right) \qquad (3)$$

The recurrence will move on till the Kth term and after K steps we'll be having an equation as:

$$T\left(\frac{n}{2^k}\right) = 2T\left(\frac{n}{2^{k+1}}\right) + \left(\frac{n}{2^k}\right)^2 \log\left(\frac{n}{2^k}\right) \qquad (3)$$

Then after this the iteration needs to keep on repeating till the $n/2^k$ becomes 1.

$$n = 2^k$$

$$k = \log_2(n)$$

Now , we have all the sub problems and we need to sum up them:

$$T(n) = n^2 \log n + 2\left(\frac{n^2}{4}\right)\log\left(\frac{n}{2}\right) + 2^2\left(\frac{n^2}{4^2}\right)\log\left(\frac{n}{4}\right) + \ldots + 2^k\left(\frac{n^2}{4^k}\right)\log\left(\frac{n}{2^k}\right) \qquad (4)$$

From the above all the equations we notice that each term in the sum is related to the previous term by a factor of 4 in both the numerator and denominator. We can simplify this geometric sum as follows:

$$T(n) = n^2 \log n + \sum_{i=1}^{k}\left(\frac{n^2}{4^i}\right)\log\left(\frac{n}{2^i}\right) \qquad (5)$$

Let's evaluate the second term as follows:

$$\sum_{i=1}^{k}\frac{n^2}{4^i} log \frac{n}{2^i}$$

$$n^2 \sum_{i=1}^{k} \frac{1}{4^i} \left[\log n - log 2^i\right]$$

$$n^2 \sum_{i=1}^{k} \frac{1}{4^i} \left[\log n - i\right]$$

$$n^2 \sum_{i=1}^{k} \frac{1}{4^i} \left[\log n - i\right]$$

$$n^2 \log n \sum_{i=1}^{k} \frac{1}{4^i} - \sum_{i=1}^{k} \frac{i}{4^i}$$

$$n^2 \log n \cdot \left(\frac{1-(1/4)^k}{1-(1/4)} - \log 2 \sum_{i=1}^{k} i \left(\frac{1}{4}\right)^i \right)$$

And then the sum can be calculated as follows:

$$\sum_{i=1}^{k} i \left(\frac{1}{4}\right)^i = \frac{1}{4} + 2\left(\frac{1}{4}\right)^2 + 3\left(\frac{1}{4}\right)^3 + ...$$

$$T(n) = n^2 \log n + \sum_{i=1}^{k} \left(\frac{1}{4}\right)^i \log\left(\frac{n}{2^i}\right) \tag{5}$$

Where as let simplify the sum as follows:

$$T(n) = n^2 \log n + n^2 \left(\frac{4}{3}\right)^k \sum_{i=1}^{k} \left(\frac{3}{4}\right)^i \log\left(\frac{n}{2^i}\right)$$

The sum is bounded and we can denote this as O(1)

$$\sum_{i=1}^{k} \left(\frac{3}{4}\right)^{i} \log\left(\frac{n}{2^{i}}\right)$$

Where as $(3/4)^i$.

Becomes constant so it wouldn't matter that much to change the complexity of the given function.

Therefore the final complexity of the recurrence relation is $T(n) = (n^2 \log n) + \theta(n^2)$.

Where as this can be simplified as an

$$T(n) = \theta(n^2 \log n).$$

Determining the complexity of the class:

Hence, the solution for this recurrence relation $T(n) = 2T(n/2) + n^2 \log n$ is $T(n) = \theta(n^2 \log n)$.

---

c. Question:
Given the recurrence relation $T(n) = 2T(n/2) + n^3$, apply the Master Theorem, to determine the complexity class $T(n)$.Explain your steps and reasoning.

Solution:

Given recurrence relation:

$$T(n) = 2T(n/2) + n^3 \qquad (1)$$

Step 1: At first, we have to Identify the form of the recurrence relation. whether the given recurrence relation is in Master Theorem or not.

Where as the master theorem is having it's own form of recurrence relations as $T(n) = aT(n/b) + f(n)$.

After comparing the normal form of master theorem to the above given recurrence relation of equation (1):

- $a = 2$ (Number of sub problems)

- $b = 2$ (Factor by which the problem size is reduced)

- $f(n) = n^3$ (Non-recursive part)

Step 2: Now, we have to substitute the values of a and b in this relation $n^{\log_b(a)}$:

$$n^{\log_2(2)} = n^1 = n$$

Step 3: Then Compare $f(n)$ to $n^{\log_b(a)}$:

$$f(n) = n^3 \qquad (2)$$

$$n^{\log_2(2)} = n \qquad (3)$$

After comparing above equations (2) and (3), since $f(n)$ is Asymptotically larger than $n^{\log_b(a)}$.

Case 1: If $f(n) = O\left(n^{\log_b(a)-\epsilon}\right)$ for some $\epsilon > 0$, then the time complexity is $T(n) = \Theta\left(n^{\log_b(a)}\right)$.
Case 2: If $f(n) = \Theta\left(n^{\log_b(a)}\right)$, then the time complexity is $T(n) = \Theta\left(n^{\log_b(a)} \log n\right)$.
Case 3: If $f(n) = \Omega\left(n^{\log_b(a)+\epsilon}\right)$ for some $\epsilon > 0$, and $af\left(\frac{n}{b}\right) \leq kf(n)$ for some $k < 1$ and sufficiently large n, then the time complexity is $T(n) = \Theta(f(n))$.

So, from the above master theorem cases we fall under the case 3 where as

$$f(n) > n^{\log_b(a)}$$

Then the C value is $C < 1$. Substituting the values of

$$2T(n/2)^3 < \ C * n^3 \qquad (4)$$

From the above equation (4) we can re-write it as below:

$$(1/4) \leq C \qquad (5)$$

$$where\ C < 1$$

Where as C = 1/4 is the solution, from this we can finally say that $T(n)$ is bounded by $O(n^3)$.

Determining the complexity of the class:

Hence, $T(n)$ is bounded by $\theta(n^3)$, the solution for the recurrence relation $T(n) = 2T(n/2) + n^3$ is $T(n) = \theta(n^3)$.

---

d. Question:
Examine the recurrence relation $T(n) = 3T\left(\frac{n}{2}\right) + nk \log n$. Apply the Master Theorem to deduce the complexity class $T(n)$ falls into. Explain your reasoning clearly.

Solution:

Given recurrence relation $T(n) = 3T(n/2) + nk \log n$.

Step 1: At first, Have to Identify the form of the recurrence relation. The Master Theorem is only applied to recurrence relations in the form of $T(n) = aT(n/b) + f(n)$.

From the above recurrence relation as follows:

- $a = 3$ (Number of sub problems)

- $b = 2$ (Factor by which the problem size is reduced)

- $f(n) = nk \log n$ (Non-recursive part)

Step 2: Now, Let's Calculate $n^{\log_b(a)}$:

$$n^{\log_2(3)}$$

Step 3: Now, Compare $f(n)$ to $n^{\log_b(a)}$:

$$f(n) = n.k \log n \qquad (1)$$

$$n^{\log_2(3)} \sim n^{1.5} \qquad (2)$$

So, From the above results of equation (1) and (2). we can say that $f(n)$ is smaller than $n^{\log_b(a)}$.

From this recurrence relation, $f(n)$ is smaller than $nk \log n$, and $n^{\log_2(3)}$ is a power of n with a non-integer exponent.

As per master's theorem we know that

$$a \geq 1, b > 1, f(n) = \Theta(n^k \log^p n), K \geq 0$$

As of that above mentioned a, b, f(n) values, we can compare now with

$$f(n) = \theta(n^k \log^p n) \qquad (3)$$

From the above equation (3) we can get the values of K and P as K=1 and P=1.

Now, Finally we have to compare the values of equation (1), (2) and (3) as

$$log_b a > K$$

As per, Master's theorem we know that $log_b a > K$ satisfies the case 1 of Masters theorem.

Case 1: If $f(n) = O\left(n^{\log_b(a)-\epsilon}\right)$ for some $\epsilon > 0$, then the time complexity is $T(n) = \Theta\left(n^{\log_b(a)}\right)$.

From the defined masters theorem case 1 we can apply that T(n): as:

$$T(n) = \Theta\left(n^{\log_b(a)}\right)$$

$$T(n) = \Theta\left(n^{\log_2(3)}\right)$$

$$T(n) = \Theta\left(n^{1.5}\right)$$

So Finally, we found that $T(n)$ is bounded by $\Theta\left(n^{1.58}\right)$.

Determining the complexity of the class:

Hence $T(n)$ is bounded by $\Theta\left(n^{1.58}\right)$. The solution to the recurrence relation $T(n) = 3T(n/2) + nk \log n$ is $T(n) = \Theta\left(n^{1.58}\right)$

Question 3

Provide pseudocode implementation for binary search algorithm:

Algorithm 1: Binary Search:

Definition of Binary Search:

Binary Search is defined as a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduces the time complexity.

$$
\begin{aligned}
&int \;\; bin\_search(arr[], x, start, end)\\
&\{\\
&\quad int\ mid\ \tfrac{(start+end)}{2};\\
&\quad if\ (arr[mid] = x)\\
&\quad \{\\
&\quad\quad return\ mid;\\
&\quad \}\\
&\quad if\ \ (start = end)\\
&\quad \{\\
&\quad\quad return\ -1;\\
&\quad \}\\
&\quad if\ arr[mid] > x\\
&\quad \{\\
&return \;\; bin\_search(arr[], x, start, mid-1);\\
&\quad \}\\
&\quad if\ arr[mid] < x\\
&\quad \{\\
&return \;\; bin\_search\ (arr[], x, mid+1, end);\\
&\quad \}\\
&\}
\end{aligned}
$$

Algorithm 2: Ternary Search:

Definition of Ternary search:

Ternary search is a divide-and-conquer search algorithm that works on a sorted array. It repeatedly divides the search interval into three equal parts and determines which part of the interval contains the target element.

$$
\begin{array}{l}
int\ \ ternary\_search\ (int\ \ l,\ \ int\ \ r\ ,\ int\ \ x) \\
\quad \{ \\
\qquad if\ \ (r >= l) \\
\qquad\ \ \{ \\
\qquad int\ mid1 = l + \frac{(r-l)}{3}; \\
\qquad int\ mid2 = r - \frac{(r-l)}{3}; \\
\qquad if\ \ (ar[mid1] == x) \\
\qquad\quad return\ mid1; \\
\qquad if\ \ (ar[mid2] == x) \\
\qquad\quad return\ mid2; \\
\qquad\ \ if\ \ (x < ar[mid1]) \\
return\ \ ternary\_search(l, mid1 - 1, x); \\
\qquad else\quad if\ (x > ar[mid2]) \\
return\ \ ternary\_search(mid2 + 1, r, x); \\
\qquad\ \ else \\
return\ \ ternary_search(mid1 + 1, mid2 - 1, x); \\
\qquad\quad \} \\
\qquad\quad return\ -1; \\
\quad \}
\end{array}
$$

Calculate the time complexity of each algorithm in terms of the number of elements in the list and represent it using a mathematical formula: (10 points).

Hint: To calculate the time complexity of each algorithm in terms of the number of elements in the list, analyze the provided.

Solution: Time Complexity for Binary Search:

$$T(N) = C + T(\tfrac{N}{2}) \qquad (1)$$

Here from above the binary search recurrence relation ”N” is time taken for N elements where as ”N/2” is time taken for N/2 elements.

Let’s Substitute: N=N/2 in equation (1)

$$T(\frac{N}{2}) = C + (\frac{N}{4}) \qquad (2)$$

Substitute equation (2) in (1)

$$T(N) = 2C + T(\frac{N}{4}) \qquad (3)$$

Let's Substitute: N=N/4 in equation (1)

$$T(\frac{N}{4}) = C + T(\frac{N}{8}) \qquad (4)$$

Substitute equation (4) in (3)

$$T(N) = 3C + T(\frac{N}{8}) \qquad (5)$$

So, Finally we identified the pattern which is

$$T(N) = T(\frac{N}{2^i}) + iC$$

At some point of time, as N/2 i diminishes we reach only element in the array T(N/2^i) = $T(1)$.

$$T(N) = T(\frac{N}{2^i}) + iC \qquad (6)$$

So,from the above equation (6) we can re-write it as

$$T(\frac{N}{2^i}) = T(1)$$

$$(\frac{N}{2^i}) = 1$$

Now , Apply log on both sides

$$\log_2 N = \log_2 2^i$$

As per above equation we know that $\log_2 2^i$ is "i".

$$\log_2 N = i \qquad (7)$$

Finally Substitute equation no.(7) in (6)

$$T(N) = T(\frac{N}{2^{log_2 N}}) + \log_2 N * C$$

So, as per the above equation 2^{log_2} is "1"

$$T(N) = T(\frac{N}{N}) + \log_2 N * C$$

$$T(N) = T(1) + C.\log_2 N$$

T(1) is One element in range of search, we can make it as a constant K.

$$T(N) = K + C.\log_2 N \qquad (8)$$

Ignore the Constants from the above equation (8)

$$T(N) = \log_2 N$$

Hence, Finally the time complexity of the binary search algorithm is $O(\log N)$.

Solution:

Time Complexity for Ternary Search:

$$T(N) = C + T(\tfrac{N}{3}) \qquad (1)$$

Here from above the binary search recurrence relation ”N” is time taken for N elements where as ”N/3” is time taken for N/3 elements.

Let's Substitute: N=N/3 in equation (1)

$$T(\tfrac{N}{3}) = C + (\tfrac{N}{9}) \qquad (2)$$

Substitute equation (2) in (1)

$$T(N) = 2C + T(\tfrac{N}{9}) \qquad (3)$$

Let's Substitute: N=N/9 in equation (1)

$$T(\tfrac{N}{9}) = C + T(\tfrac{N}{27}) \qquad (4)$$

Substitute equation (4) in (3)

$$T(N) = 3C + T(\tfrac{N}{27}) \qquad (5)$$

So, Finally we identified the pattern which is

$$T(N) = T(\tfrac{N}{3^i}) + iC$$

At some point of time, $N/3^i$ diminishes we reach only element in the array $T(N/3^i) = T(1)$.

$$T(N) = T(\tfrac{N}{3^i}) + iC \qquad (6)$$

So,from the above equation (6) we can re-write it as

$$T(\tfrac{N}{3^i}) = T(1)$$

$$(\tfrac{N}{3^i}) = 1$$

Now , Apply log on both sides

$$\log_3 N = \log_3 3^i$$

As per above equation we know that $\log_3 3^i$ is ”i”.

$$\log_3 N = i \qquad (7)$$

Finally Substitute equation no.(7) in (6)

$$T(N) = T(\frac{N}{3^{log_3 N}}) + \log_3 N * C$$

So, as per the above equation 3^{log_3} is "1"

$$T(N) = T(\frac{N}{N}) + \log_3 N * C$$

$$T(N) = T(1) + C.\log_3 N$$

T(1) is One element in range of search, we can make it as a constant K.

$$T(N) = K + C.\log_3 N \qquad (8)$$

Ignore the Constants from the above equation (8)

$$T(N) = \log_3 N$$

Hence, Finally the time complexity of the Ternary search algorithm is $O(\log N)$.