Assignment - 3

Question 2:

i) def find_complexity (n):

```
    i = 1 , s = 1
    while s <= n:
        i += 1
        s += i
    return.
```

Consider n = 5

i	s
1	1
2	3
3	(6) ← Here the value of s is greater than n.

∴ the statement will break.

Consider n = 10

i	s
1	1
2	3
3	6
4	10
5	(15) ← Here the value of s is greater than n.

∴ The statement will break.

Here, we can see that the value of s is increasing in ~~an~~ a squared of ~~number~~ i manner approximately.

For

$i = 1 \quad \sqrt{s} = \sqrt{1} = 1$

$i = 2 \quad \sqrt{s} = \sqrt{3} \approx 2$

and so on.

∴ we can say that the Time complexity of the function is $O(\sqrt{n})$.

---

ii)
```
def find_complexity_2(n):
    for i in range(n//2, n+1):
        for j in range(1, n-(n//2)+1):
            k=1
            while k<=n:
                k*=2
    return
```

We will have to calculate the loops from inner while to outermost for.

The value of k increase exponentially in the while loop.

```
k=1
while k <= n
    k*= 2
```

consider $n = 12$

$\therefore$ Values for k will be 2, 4, 8, (16) ← value will break.

$\therefore$ We can say that, time complexity of while is $O(\log n)$.

For the second for loop. let $n=12$

j goes from 1 to 7

$\therefore$ we can say that the time complexity is $O(n/2) \approx O(n)$

as $7 = \left(\frac{n}{2} + 1\right)$ where $n=12$.

For the first for loop. $n=12$

i goes from 7 $\longrightarrow$ 12

i.e $\frac{n}{2}+1 \rightarrow n$

$\therefore$ There are only half of n iterations

$\therefore$ Time complexity $= O(n/2) \approx O(n)$.

$\therefore$ Total complexity $=$

$n \times n \times \log n = \boxed{O(n^2 \log n)}$

iii)
```
def find_complexity_3(n):
    if (n <= 2):
        return
    else:
        return find_complexity_3(
            math.floor(math.sqrt(n)) + 1)
```

The time complexity of this function is proportional to the depth of recursion

Let k = depth of recursion.
consider $n = 2^{\log n}$ as we have taken square root of n.

$2^{\log n/2^k}$ → After taking square root k times.

$$2^{\log n/2^k} = 2$$

$$\log_{n/2} k = 1$$

$$\therefore \log n = 2^k$$

$$\log(\log n) = k$$

$$\therefore O(n) = \log(\log(n))$$

iv) def find_complexity_4(n, epsilon):

```
    s,e = 1, n
    m = (s+e) / 2

    while abs(n - m*m) > epsilon:
        if m*m > n
            e = m
        else
            s = m
        m = (s+e) / 2
    return.
```

The complexity is counted on how many times the while loop runs.

Assuming $e = 10^{-8}$, i.e really small.

After calculating the graph for the plotted line using the given code, the observed line looks to be of type $n\log(n)$

∴ ~~$O(n) = n\log n$~~ Time complexity $= O(n\log(n))$.

Assignment - 3

Question 5.

i) Prove $f(n) = 5n^3 + 2n^2 + 7n + 1$ is $O(n^3)$, $\Theta(n^3)$

→ Definition : $O(n)$

$O(g(n))$ defines a set of functions such that there exists positive constants $c > 0$ and $n_0 > 0$, where $f(n)$ is always less than or equal to $c \cdot g(n)$ for all $n > n_0$.

$$O(g(n)) = \{ f(n) \mid 0 \leq f(n) < c \cdot g(n), \forall n \geq n_0, \exists c > 0, \exists n_0 > 0 \}$$

All $f(n)$ functions are upper bounded by $g(n)$.

* Proof: $f(n) = 5n^3 + 2n^2 + 7n + 1$ is $O(n^3)$

Here $g(n) = n^3$

Assume: $f(n) = O(n^3)$ for $n = n_0$ & $c = c_0$

$\therefore$ $5n_0^3 + 2n_0^2 + 7n_0 + 1 \leq c_0 \times n_0^3$

If we prove that $f(n) = O(n^3)$ $\forall n > n_0$ & $c > c_0$ and $f(n) \leq c \cdot n^3$, we can say that $f(n)$ is upper bounded by $g(n)$.

Lets take $n = n_0 + 1$ & $c = c_0 + 1$

$\therefore$

Solving for LHS

$$5(n_0+1)^3 + 2(n_0+1)^2 + 7(n_0+1) + 1$$

$$= 5(n_0^3 + 3n_0^2 + 3n_0 + 1) + 2(n_0^2 + 2n_0 + 1) + 7n_0 + 7 + 1$$

$$= 5n_0^3 + 15n_0^2 + 15n_0 + 5 + 2n_0^2 + 4n_0 + 2 + 7n_0 + 7 + 1$$

$$= \cancel{5n_0^3 + 2n_0^2 + 7n_0 + 7 + 15n_0^2 + \ldots}$$

$$= (5n_0^3 + 2n_0^2 + 7n_0 + 1) + 15n_0^2 + 19n_0 + 12$$

RHS: $c \times g(n)$

$$= (c_0+1) \times (n_0+1)^3$$

$$= (c_0+1) \times (n_0^3 + 3n_0^2 + 3n_0 + 1)$$

$$= (c_0 n_0^3) + c_0(3n_0^2 + 3n_0 + 1) + (n_0^3 + 3n_0^2 + 3n_0 + 1)$$

$$= (c_0+1)(n_0^3) + c_0(3n_0^2 + 3n_0 + 1) + 3n_0^2 + 3n_0 + 1$$

Since we are ~~a~~have proved that $f(n) = O(g(n)) \therefore g(n) = n^3$ for $c \geq c_0$ & $n = n_0$, If we add the quadratics to LHS & RHS we end up Upper bound $f(n)$.

Prove $f(n) = \Theta(n^3)$

Definition: $\Theta(g(n))$ defines a set of functions such that there exists positive constant $c_1 > 0$ & $c_2 > 0$ and $n_0 > 0$, where $f(n)$ is always less than or equal to $c_1 \cdot g(n)$ & greater than equal to $c_2 \cdot g(n)$.

$$\Theta(g(n)) = \{ f(n) \mid 0 \leq c_2 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n), \forall\, n \geq n_0, \exists\, (c_1 > 0, c_2 > 0, n_0 > 0) \}$$

By Induction:

$$0 \leq c_2 \cdot n^3 \leq 5n^3 + 2n^2 + 7n + 1 \leq c_1 n^3$$

Divide by n^3

$$0 \leq c_2 \leq 5 + \frac{2}{n} + \frac{7}{n^2} + \frac{1}{n^3} \leq c_1$$

Here choose $n = n_0$ lets say $= 10$

$\therefore$ & $c_1 = 10$ & $c_2 = 5$

$$5 + \frac{2}{10} + \frac{7}{100} + \frac{1}{1000}$$

$\therefore \quad 5 \leq 5 + (0.2 + 0.07 + 0.001) \leq 10$

which is true.

If we prove for $n = 20$, we can surely say that $f(n) = \Theta(g(n))$ $\because g(n) = n^2$

$$\therefore 5 \leq 5 + \frac{2}{20} + \frac{7}{400} + \frac{1}{8000} \leq 10$$

which is true.

Hence proved.

ii) Prove $5n^3+2n^2+7n+1$ is $\omega(n^2)$, $\Omega(n^2)$

* Little omega $\omega(g(n))$:

Definition:

$\omega(g(n))$ defines set of functions such that for any $c>0$ there is $n_0>0$ where $f(n)$ is always greater than $c \cdot g(n)$ for all $n \geq n_0$.

Here $f(n) = 5n^3+2n^2+7n+1$

By defi^n

$$0 \leq c \cdot g(n) < f(n) \quad \forall\, n > n_0$$

= Consider $c = 10$

$$0 \leq 10 \cdot n^2 < 5n^3+2n^2+7n+1$$

Let's take $n = 1$ $\therefore n_0 = 1$

$\therefore 0 \leq 10 < 5+2+7+1$ $\therefore 0 \leq 10 < 15$ which is true

If we prove the equation is true for $n > 1$, then we have proven $f(n) = \omega(n^2)$

Lets take $n = 2$

$\therefore$ ~~$0 \leq 10 \times 4 \leq 40 + 8 +$~~

$0 \leq 10 \times 4 \leq 5 \times 8 + 2 \times 4 + 7 \times 2 + 1$

$0 \leq 40 \leq 63$

which is True, Hence proved.

* Prove $5n^3 + 2n^2 + 7n + 1$ is $\Omega(n^2)$

$\Omega(g(n))$ defines a set of functions such that there exists positive constants $c > 0$ & $n_0 > 0$, where $f(n)$ is always greater than or equal to $c \cdot g(n)$ for all $n > n_0$

By defn:

$0 \leq c \cdot g(n) \leq f(n)$

Consider c is 10 & $n = n_0 = 1$

$\therefore 0 \leq 10 \times 1^2 \leq 5 + 2 + 7 + 1$

$\therefore 0 \leq 10 \leq 15$ which is true.

If we prove, the equation is true for $n > 1$, we have proven $f(n) = \Omega(n^2)$

lets take $n = 2$

$\therefore 0 \leq 40 \leq 63$

which is True.

Assignment - 3

Question - 6.

i) Find number of steps required for Erlich to take to reach the Treasure.

We can say the $f(n) \rightarrow$ number of steps to reach the treasure from starting position.

Let us assume the treasure is 10 steps away to the left.

In one iteration Erlich will take 4 rounds to reach its initial position. i.e, Let us assume, He will take the right step first.

So those 4 rounds are right → left → left → right.

To improve efficiency, we can ~~take~~ multiply an exponential to the number of step in each round.

Let us take 2^i, $\because$ i is the iterative.

For first iteration $i = 0 \therefore 2^i = 1$

$\therefore$ The steps are:

right×1 → left×1 → left×1 → right×1.

Total steps = 4

For second iteration $i = 1 \therefore 2^i = 2$

$\therefore$ The steps are:

right×2 → left×2 → left×2 → right×2.

Total steps = 4×2 = 8

So on, he will take 4 → 8 → 16 → 32... steps

We can see that the number of steps are increasing exponentially.

We will run this loop until the current position of Erlich is the treasure position

To calc. the number of steps taken to reach the treasure.

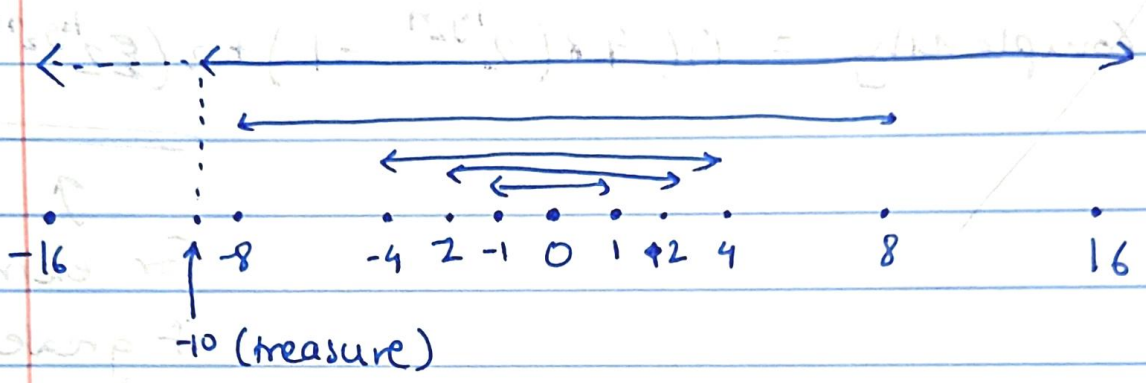
We can take log of total steps to the base.

$\therefore$ no. of iterations = $(\log_2 10)$ integer. -1

$= 3 + 1 = 4$

But total iterations will be

We can prove it using geometry.



Here we can see that

1st iteration = $-1 \longrightarrow 1$

2nd iteration = ~~$-4 \longrightarrow 4$~~ $-2 \rightarrow 2$

3rd iteration = $-4 \longrightarrow 4$

4th iteration = $-8 \longrightarrow 8$

5th iteration = $-16 \longrightarrow 16$

2^4

here i = 4 if started from 0

In 5th iteration you will find that the treasure lies within.

We can say that generalized equation for above will be

$$\log_r^n + 1$$

r = exponential increase.

Sum of geometric progression.

$= \frac{a(r^{k-1} - 1)}{r - 1} \Rightarrow$ for $(k-1)^{th}$ steps.

if $a = 1$ $r = 2$ & iterations $= 5$

$\boxed{k = 4}$

∴ Number of steps:-

$4 \times 2^0 + 4 \times 2^1 + 4 \times 2^2 + 4 \times 2^3 + 4 \times 2^4$

$= 4 \times (2^0 + 2^1 + 2^2 + 2^3 + 2^4)$

$= 4 \times (1 + 2 + 4 + 8 + 16)$

$= 4 \times (31) =$ 124 steps

We can generalize the value of k as

$$k = \log_r^n$$

r_2 Exponential steps

$k =$ floor $(\log_r^n + 1)$

Upper bound Analysis:-

Complexity =

$$O\left(4\left(2^{\log_2 n} - 1\right) + 2\left(2^{\log_2 n} + 2\right) + 2\right)$$

Here $\log_2 n \approx \text{floor}\left(\log_n \frac{2}{3}\right)$

$$\frac{O\left(4(n-1) + 2(n+2) + n\right)}{4n - 1}$$

∴ Time complexity is $O(n)$.

$$f(n) = (4(n-1) + 2(n+2) + n)/3$$
$$= (4n - 4 + 2n + 2 + n)/3$$
$$= (7n - 2)/3$$

Let's take $n = 1$ & $c = 3$

∴ $\frac{7n-2}{3} \leq \frac{5}{3}$ & $c \cdot g(n) = 3 \times 1 = 3$

∴ $f(n) \leq c \cdot g(n)$

∴ It is upper bounded by $g(n)$.