

DAAPY 3

Wednesday, September 20, 2023 4:50 PM

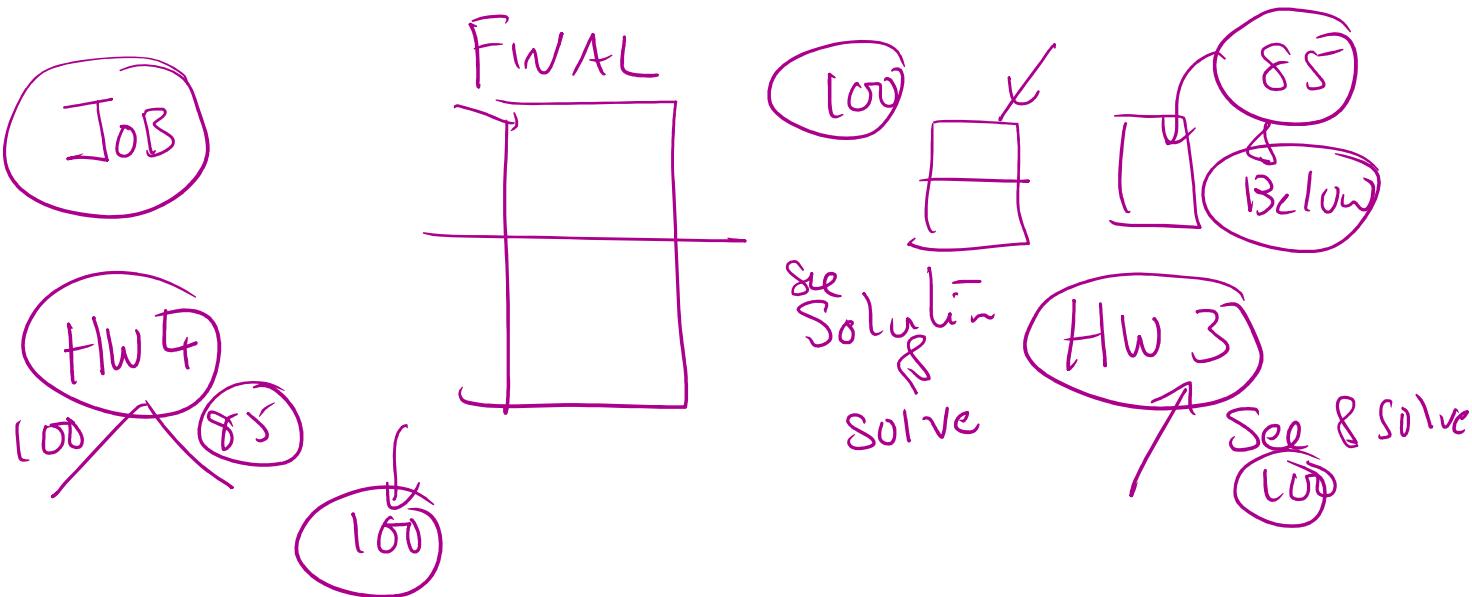
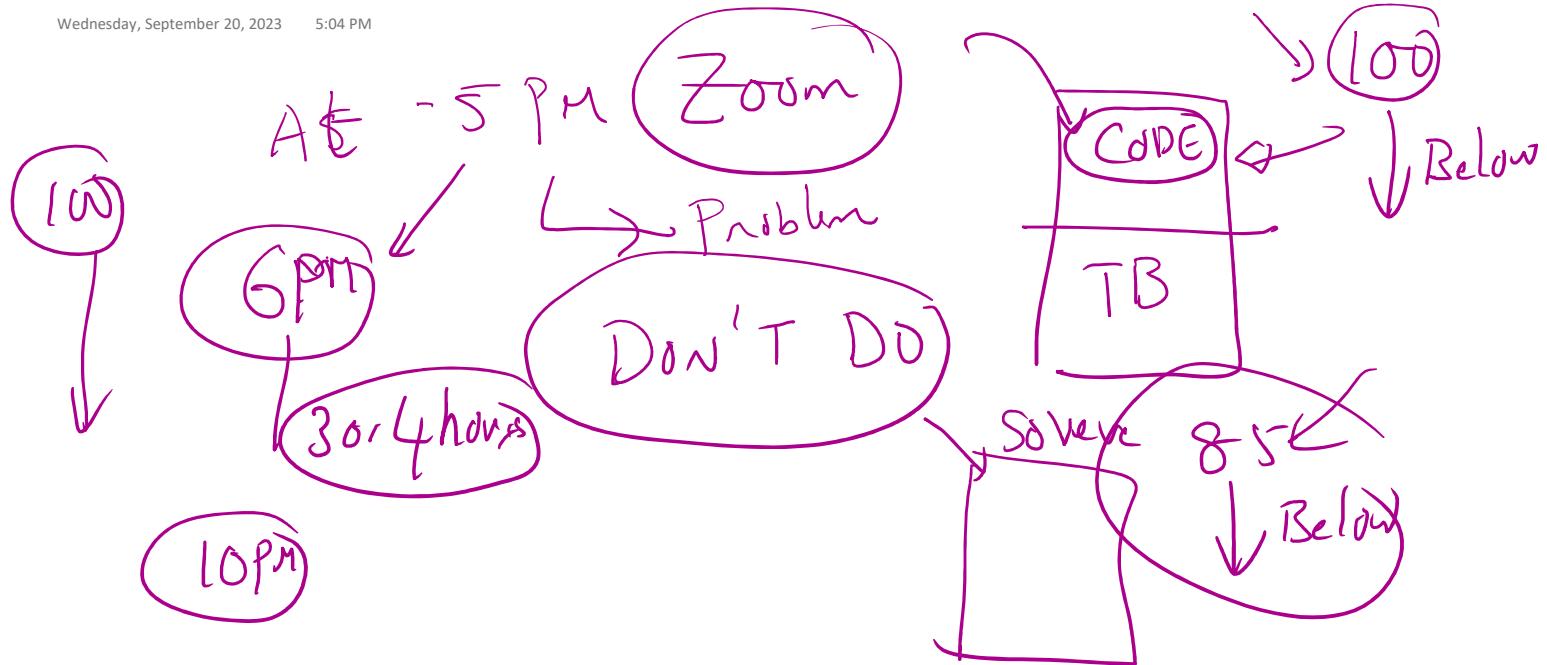
- ① Recording
- ② Screen
- ③ TALK

Final Dec
6, 2023

408.834.0284

Midterm:

OCT 25, 2023
WED 5 PM

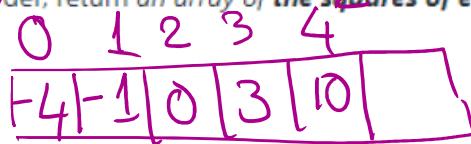


Dynamic Array

LIST

Sortin

Given an integer array `nums` sorted in **non-decreasing** order, return an array of the **squares** of each **number** sorted in **non-decreasing** order.



$n = 5$

Example 1:

Input: `nums = [-4, -1, 0, 3, 10]`

Output: `[0, 1, 9, 16, 100]`

Explanation: After squaring, the array becomes `[16, 1, 0, 9, 100]`.
After sorting, it becomes `[0, 1, 9, 16, 100]`.

[1, 3, 9]

[1, 9, 81]

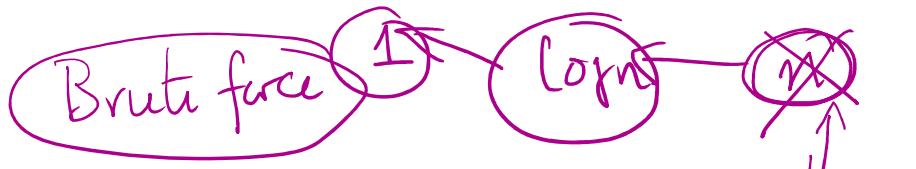
Example 2:

$n \rightarrow \infty$
Complexity

[] []
[] []

-4 -5 0 3

n



Given an integer array `nums` sorted in **non-decreasing** order, return an array of **the squares of each number** sorted in non-decreasing order.

Bubble
ma

Example 1:

Input: `nums = [-4, -1, 0, 3, 10]`

Output: `[0, 1, 9, 16, 100]`

Explanation: After squaring, the array becomes `[16, 1, 0, 9, 100]`.

After sorting, it becomes `[0, 1, 9, 16, 100]`.

-4 -1 0 3 10

16 1 0 9 [00]

Example 2:

-4 3
16 9

n^2

0 1 9 16 100

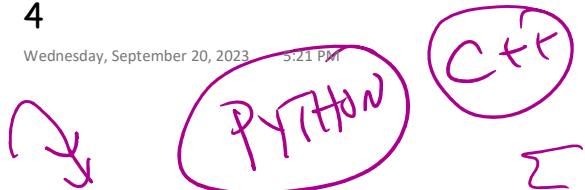
PYTHON

$n \log n$

Sort

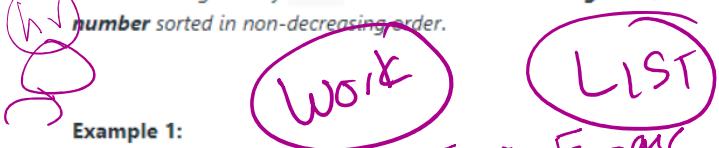
quick sort
merge

Problem Solving



$a[3]$ ③ $n = 5$

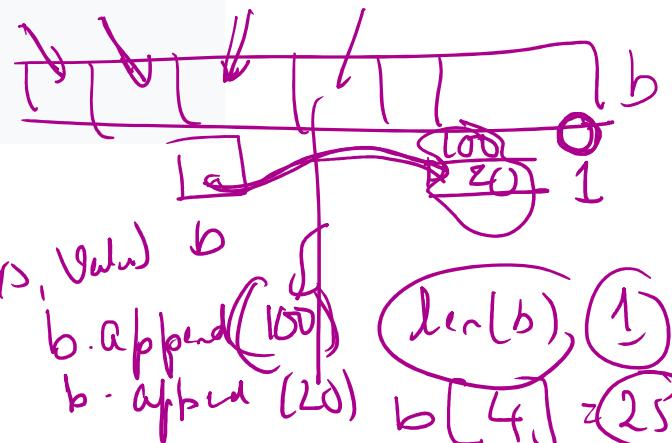
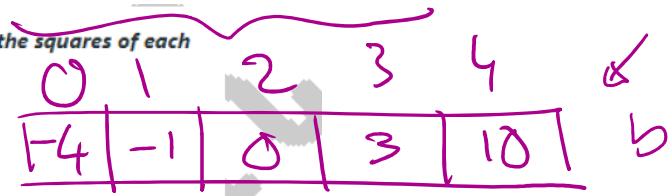
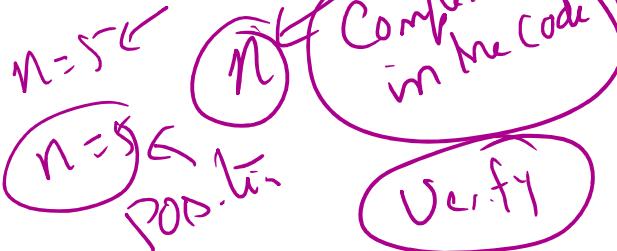
Given an integer array `nums` sorted in **non-decreasing** order, return an array of **the squares of each number** sorted in non-decreasing order.



Example 1:

Input: `nums = [-4, -1, 0, 3, 10]`
 Output: `[0, 1, 9, 16, 100]`
 Explanation: After squaring, the array becomes `[16, 1, 0, 9, 100]`.
 After sorting, it becomes `[0, 1, 9, 16, 100]`.

Example 2:



5

Wednesday, September 20, 2023 5:26 PM

$n=5$

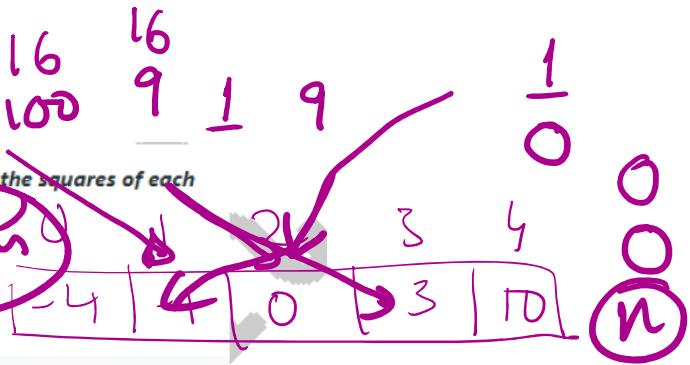
Given an integer array `nums` sorted in non-decreasing order, return an array of the squares of each number sorted in non-decreasing order.

NO

Example 1:

16 | 0 9 100
0 | 9 16 100

nums



Input: `nums = [-4, -1, 0, 3, 10]`

Output: `[0, 1, 9, 16, 100]`

Explanation: After squaring, the array becomes `[16, 1, 0, 9, 100]`.

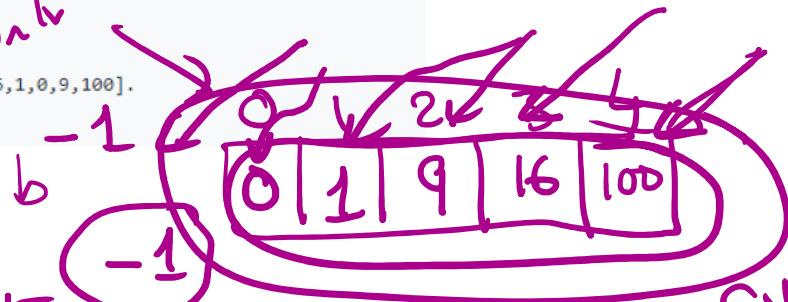
After sorting, it becomes `[0, 1, 9, 16, 100]`.

return

Example 2:

$$a[4] \text{ app } (4, 100) \\ a[4] = 100$$

$\Theta(n)$ time
 $\Theta(1)$ space

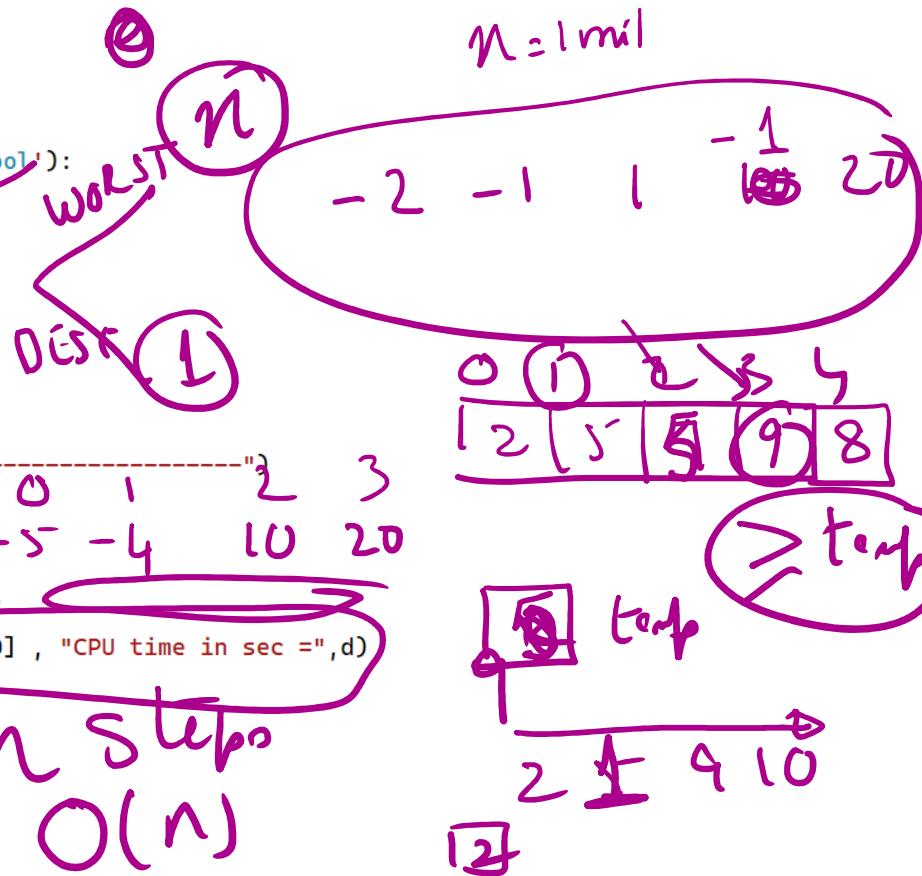


Problem in Solution

$n \log n \rightarrow n \rightarrow$

Private

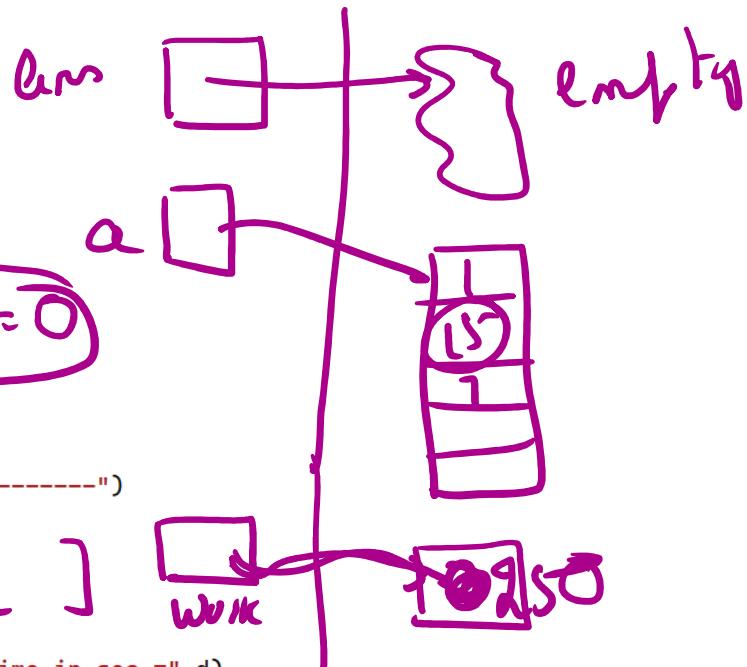
```
#PRIVATE FUNCTION
def _test1(self, a: List[int], show: bool):
    n = len(a)
    self._u.assert_ascending(a)
    work = [0]
    ans = []
    t1_start = process_time()
    b = L0977(a, ans, work, show)
    t1_stop = process_time()
    d = t1_stop - t1_start;
    if (show):
        print("-----")
        self._u.print_index(n)
        self._u.print_list(a)
        self._u.print_list(ans)
        assert(len(ans) == n)
        self._u.assert_ascending(ans)
    print(f"\n n = {n}, work = {work[0]}, CPU time in sec = {d}")
```



```

#PRIVATE sunction
def _test1(self,a>List[int],show:'bool'):
    n = len(a)
    self._u.assert_ascending(a)
    work = [0];
    ans = []
    t1_start = process_time()
    b = L0977(a,ans,work,show)
    t1_stop = process_time()
    d = t1_stop - t1_start;
    if (show):
        print("-----")
        self._u.print_index(n)
        self._u.print_list(a)
        self._u.print_list(ans)
        assert(len(ans) == n)
        self._u.assert_ascending(ans)
        print("n = ",n, "WORK = ", work[0], "CPU time in sec =",d)

```



Com'l
b]
ans

Constructors CANNOT
-- init --

Return

WORK: 28

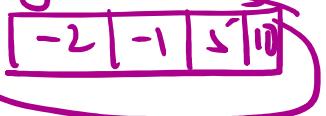
```

.6 class L0977:
.7     def __init__(self, a:List[int], ans:List[int], work:'List of size 1', show:'bool'):
.8         self._a = a ;
.9         self._ans = ans ;
.0         self._work = work ;
.1         self._show = show ;
.2         alg0 = AlgL0977(self._ans, show)
.3
.4     def size(self) -> int:
.5         return len(self._a)
.6
.7     def get(self, i:'int'):
.8         self._work[0] = self._work[0] + 1
.9         return self._a[i]
.0
.1 #####
.2 # WRITE CLASS AlgL0977
.3 # YOU CAN HAVE ANY NUMBER OF PRIVATE VARIABLES and FUNCTIONS
.4 #####
.5 class AlgL0977():
.6     def __init__(self, h:'L0977', ans:List[int], show:'bool'):
.7         self._h = h
.8         self._ans = ans
.9         self._show = show
.0         self._alg()
.1
.2 #####
.3 #   WRITE YOUR CODE BELOW
.4 #####

```

| STACK

Heads

 $n=4$ 

empty;



0

 $n=4$ 

```

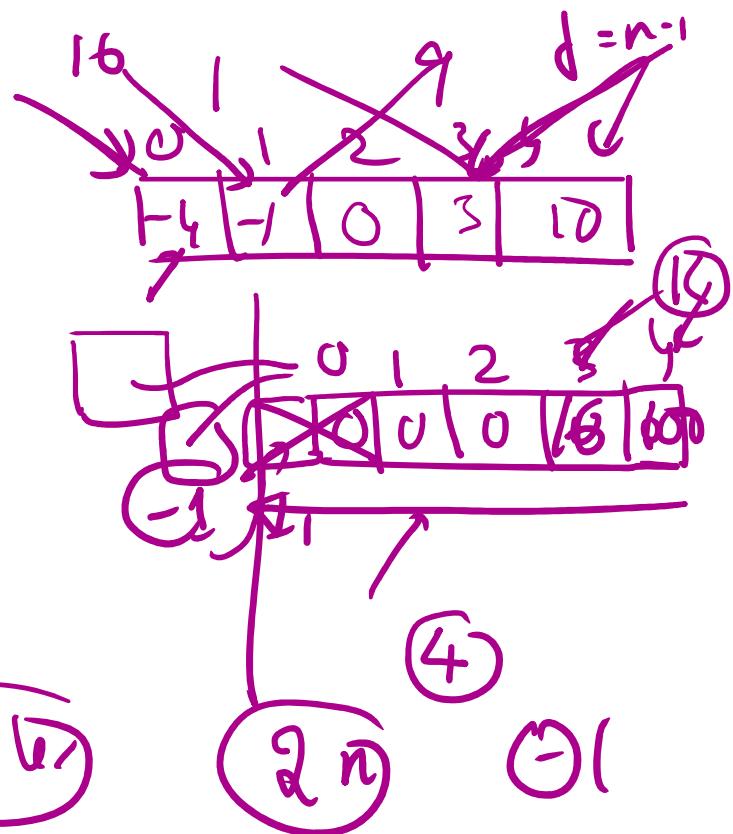
def _alg(self):
    n = self._h.size();
    if (n == 0):
        for i in range(0,n):
            self._ans.append(0) #THETA(N) Space
    i = 0 ;
    j = n - 1;
    k = n - 1;
    while (i <= j):
        x = self._h.get(i)
        i2 = x * x
        y = self._h.get(j)
        j2 = y * y ;
        if (j2 > i2):
            self._ans[k] = j2
            j = j - 1
        else:
            self._ans[k] = i2
            i = i + 1
    k = k - 1 #one item is taken care of
    assert(k == -1)

```

i

 $\Theta(n)$
 $\Theta(n)$

JL.161



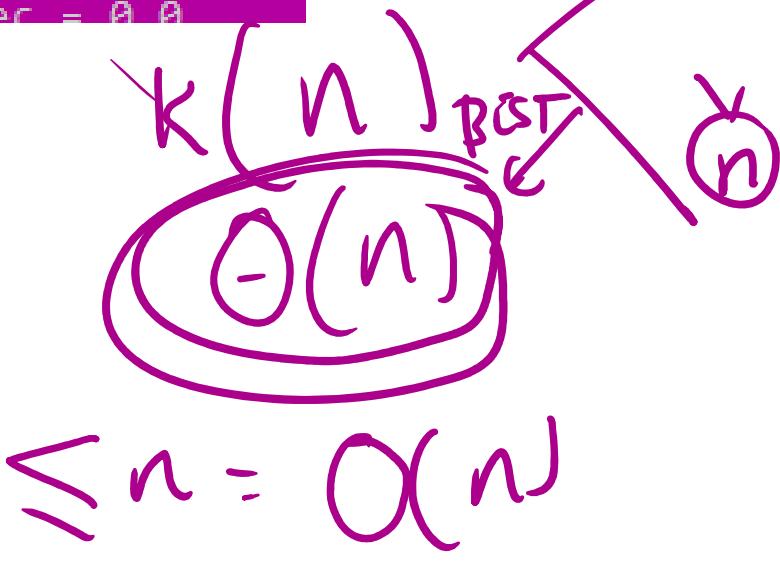
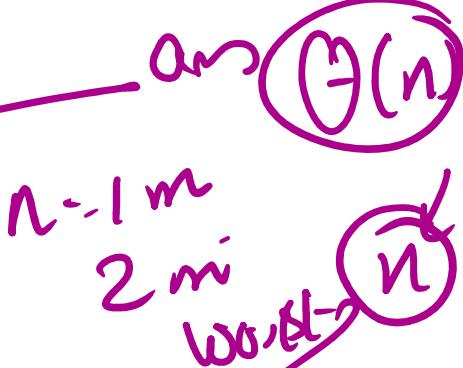
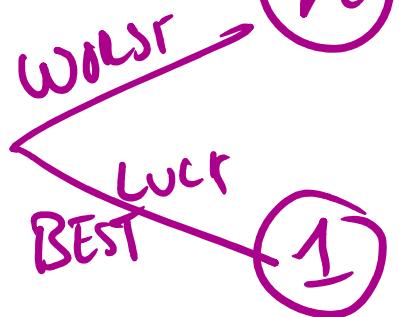
```

0   1   2   3   4
-4  -1   0   3  10
0   1   9  16 100
n = 5 work = 2.0 CPU time in sec = 0.0
Random tests on Array of size 0
n = 0 work = 0 CPU time in sec = 0.0

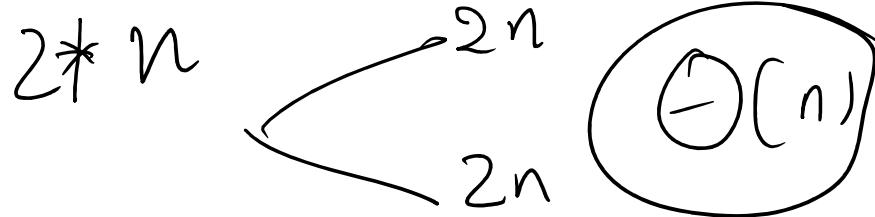
```



1



```
Random tests on Array of size 25025  
n = 25025 work = 50050 CPU time in sec = 0.03125  
Random tests on Array of size 26026  
n = 26026 work = 52052 CPU time in sec = 0.03125  
Random tests on Array of size 27027  
n = 27027 work = 54054 CPU time in sec = 0.015625  
Random tests on Array of size 28028  
n = 28028 work = 56056 CPU time in sec = 0.015625
```



```

def _alg(self):
    n = self._h.size();
    if (n == 0):
        for i in range(0, n):
            self._ans.append(0) #THETA(N) Space
    i = 0;
    j = n - 1;
    k = n - 1;
    while (i <= j):
        x = self._h.get(i)
        i2 = x * x
        y = self._h.get(j)
        j2 = y * y;
        if (j2 > i2):
            self._ans[k] = j2
            j = j - 1
        else:
            self._ans[k] = i2
            i = i + 1
        k = k - 1 #one item is taken care of
    assert(k == -1)

```

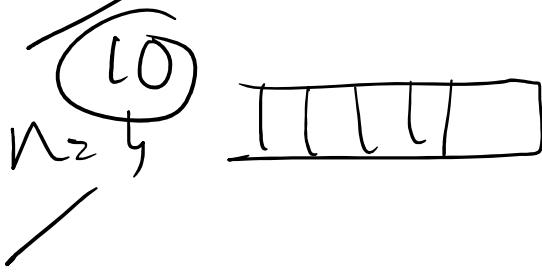
Code

2
3
4

IOV

Q

~~1 2 3 4 N=4 n=1000 n = 1 mi~~



IOVa.

IO

n

S

IO

S

(w)

IO Varah

Some Constant-
Var.a

A diagram showing a list of 10 items represented by vertical bars. To its left, the number 1 is circled with a large oval, labeled "n=1".

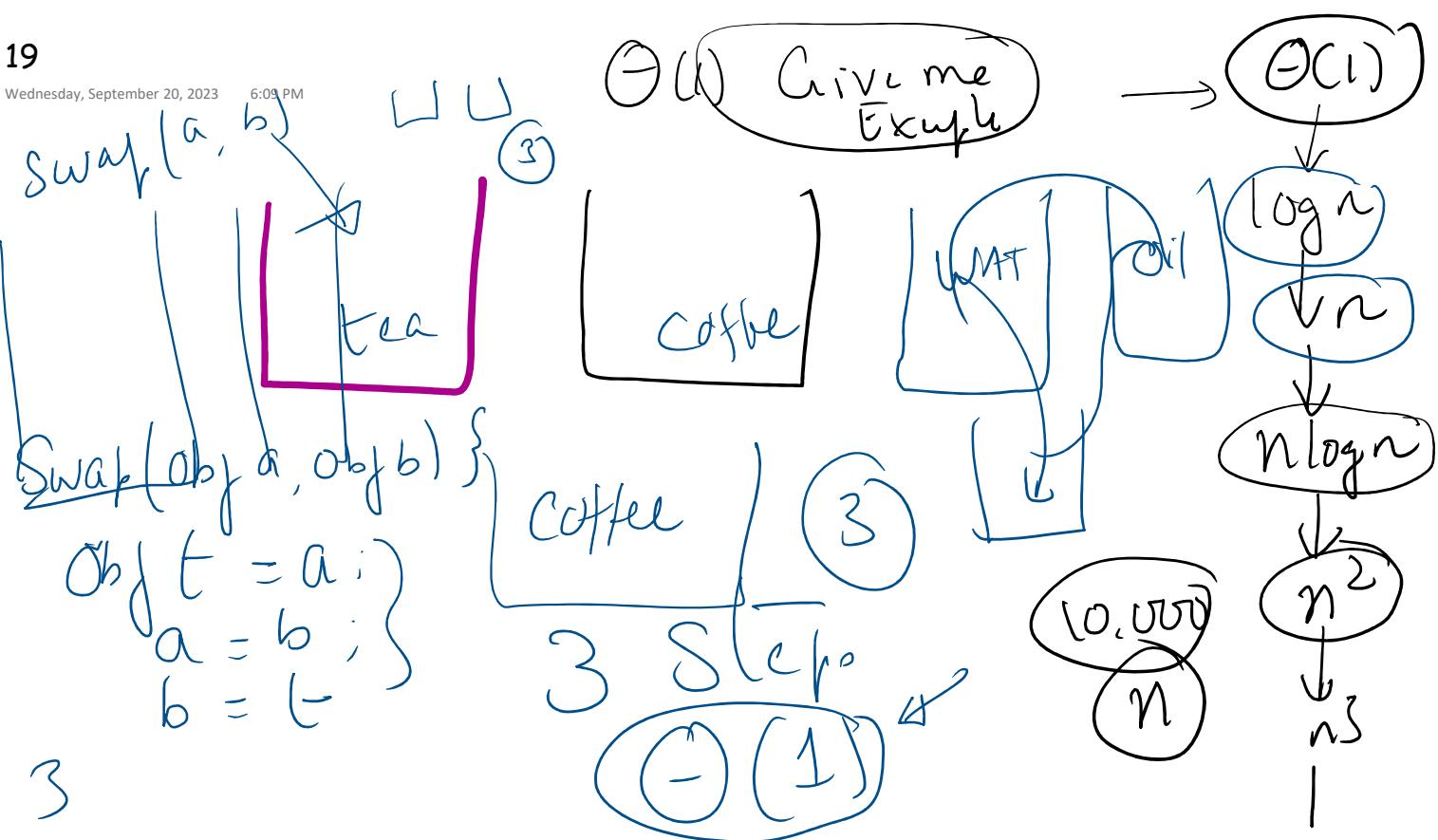
```
class Solution:  
    def sortedSquares(self, nums: List[int]) -> List[int]:  
  
        #####  
        #Nothing can be changed in class Solution  
        #####  
        class Solution:  
            #YOU CANNOT CHANGE THIS INTERFACE  
            def sortedSquares(self, nums: List[int]) -> List[int]:  
                ans = []  
                work = [0]  
                p = L0977(nums, ans, work, False)  
                return ans
```

WOK

19

Wednesday, September 20, 2023

6:09 PM



$\$5, 100, 10,000$ int $f(\text{int } n)$ int $S = 0$ ~~return $(n * (n+1)) // 2$~~

100 5 { for (int i=1; i < n; i++)

10,000 3 return S

int $n = 5$ logⁿint $a = f(n)$ $\uparrow n$

15
845
6
n
n

SPACE = $\Theta(1)$ Time = $\Theta(n)$ Lock

$n=0 \quad 1 \quad n=5$
 $a: 0 \quad 1 \quad \text{and Unset}$

Gauss →

+ ... 25
+ 124

$\{ 1 + 2 + 3 + 4 + \dots + 25$

$1 + 2 + \dots + 124$

$\text{Contact line} \rightarrow 1 + 1 + 1 + \dots + 190$

$S = 1 + 2 + 3 + 4 + \dots + n$

$S = n + (n-1) + (n-2) + (n-3) + \dots + 1$

$2S = \underbrace{(n+1) + (n+1) + (n+1) + \dots}_{n} + (n+1)$

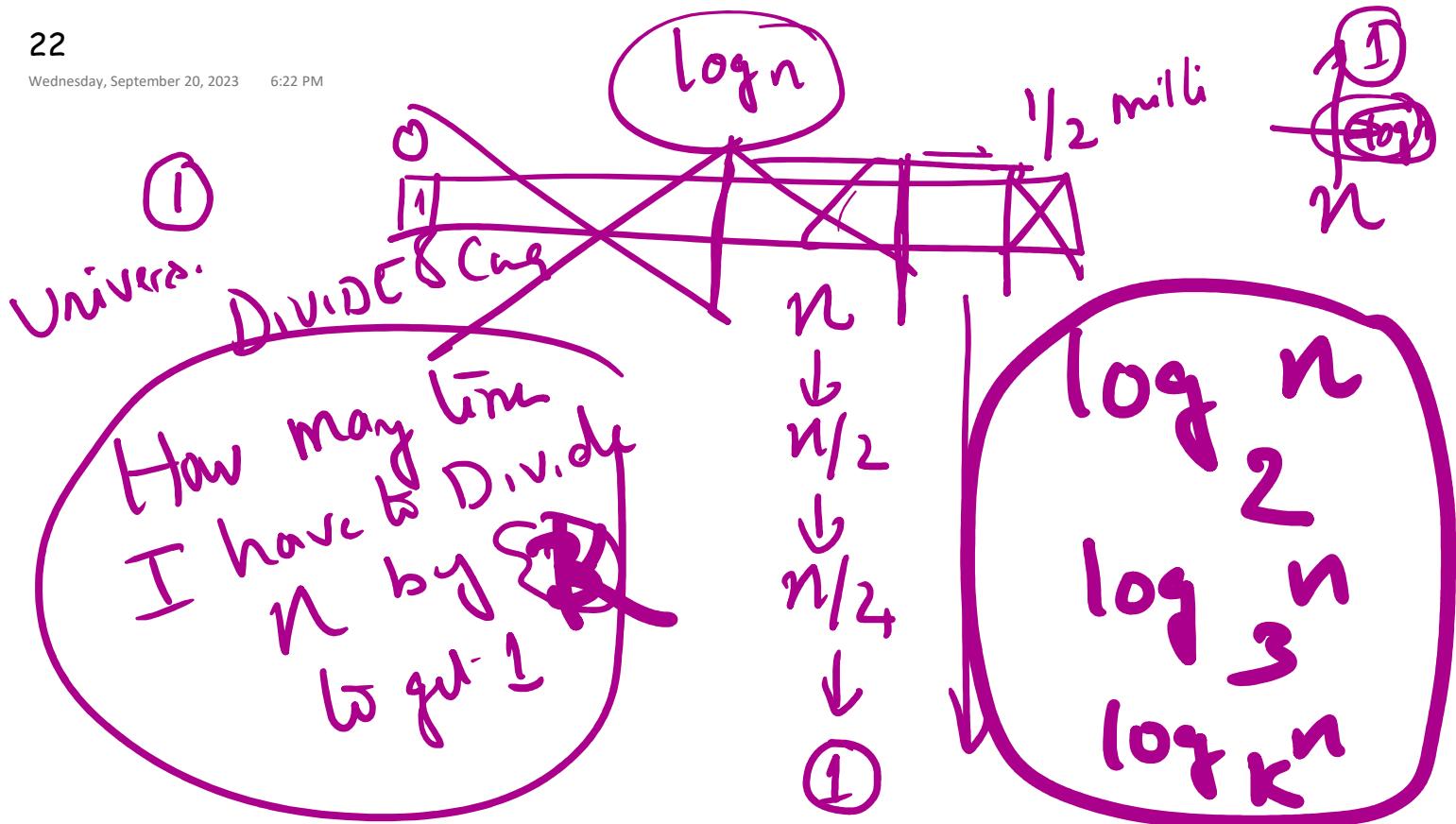
$2S = n(n+1)$

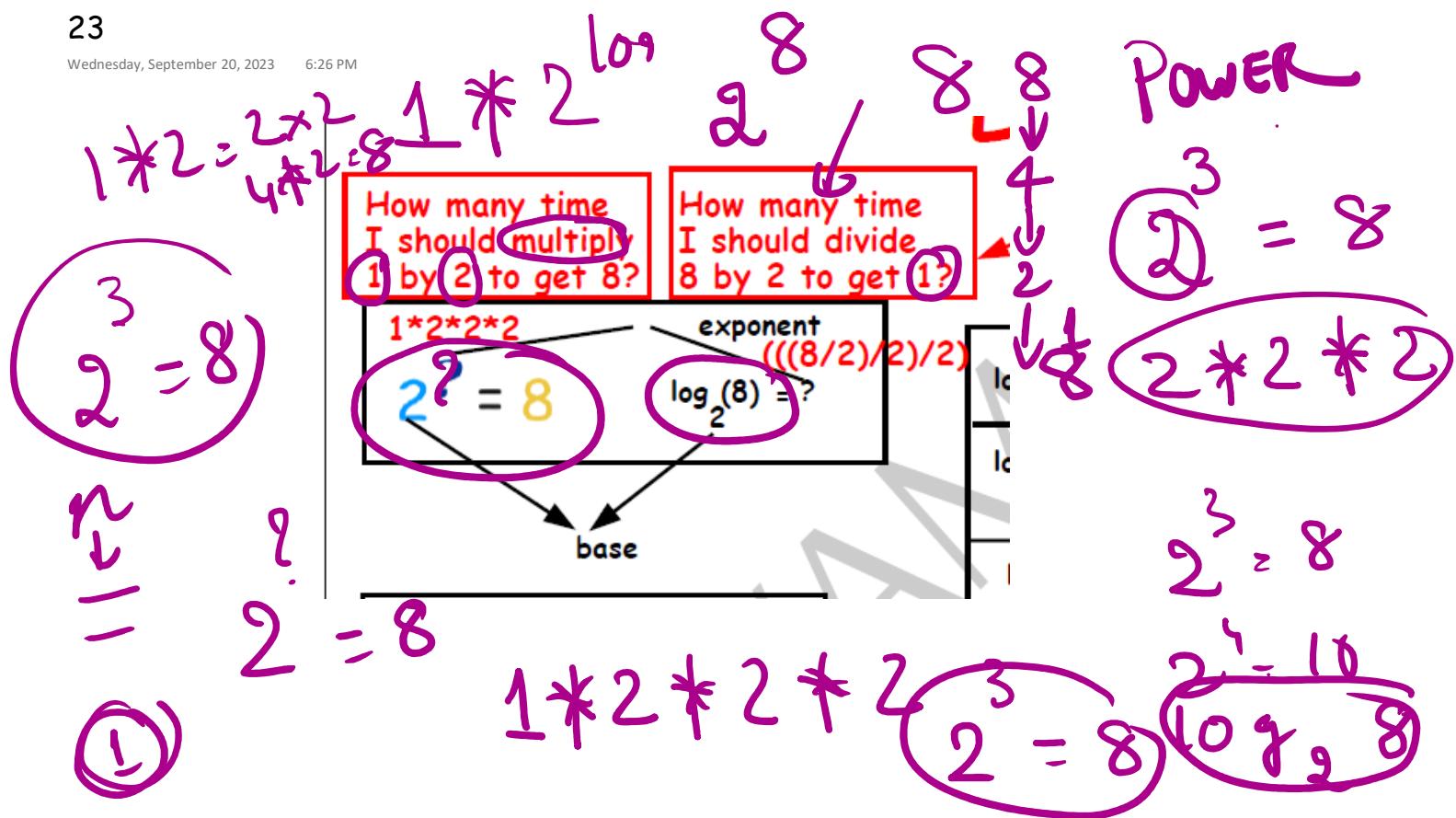
$S = \frac{n(n+1)}{2}$

3 steps ↓

$\frac{120 * (120+1)}{2} \quad \text{③}$

$\Theta(1) \text{ All}$





Binary Search Turns to Logarithm

$10 \downarrow + + \downarrow 10 \downarrow 1$

How many time
I should multiply
1 by 2 to get 8?

$$1 \cdot 2 \cdot 2 \cdot 2$$

$$2^3 = 8$$

How many time
I should divide
8 by 2 to get 1?

$$(((8/2)/2)/2)$$

$$\log_2(8) = ?$$

exponent

base

$$5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 = ?$$

$$5^5 = 625$$

$$\log_5(625) = ?$$

$$5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 = 625$$

$$\log_5(625) = 5$$

$$2^{10} = 1024$$

$$\log_2(1024) = 10$$

$$2^0 = 1$$

$$\log_2 1 = 0$$

$$2^1 = 2$$

$$\log_2 2 = 1$$

$$\text{what is } \log_2 64$$

$$2^8 = 64$$

$$\log_2 64 = 8$$

$$\log 100 = \log_{10} 100 \quad \text{Engineers}$$

$$\log 7.389 = \ln(7.389) \approx 2 \quad \text{Math}$$

$$e = \text{Euler Number} = 2.71828$$

$$\log_2 16 = \lg(16) = 4 \quad \text{CS}$$

$$\lg 2$$

$$2^5 = 32$$

$$2^6 = 64$$

$$2^7 = 128$$

$$\log_b n = \frac{\log_2 n}{\log_2 b}$$

$$2^8 = 256$$

$$2^9 = 512$$

$$2^{10} = 1024$$

$$2^{11} = 2048$$

$$2^{12} = 4096$$

$$2^{13} = 8192$$

$$2^{14} = 16384$$

$$2^{15} = 32768$$

$$2^{16} = 65536$$

$$2^{17} = 131072$$

$$2^{18} = 262144$$

$$2^{19} = 524288$$

$$2^{20} = 1048576$$

$$2^{21} = 2097152$$

$$2^{22} = 4194304$$

$$2^{23} = 8388608$$

$$2^{24} = 16777216$$

$$2^{25} = 33554432$$

$$2^{26} = 67108864$$

$$2^{27} = 134217728$$

$$2^{28} = 268435456$$

$$2^{29} = 536870912$$

$$2^{30} = 1073741824$$

$$2^{31} = 2147483648$$

$$2^{32} = 4294967296$$

$$2^{33} = 8589934592$$

$$2^{34} = 17179869184$$

$$2^{35} = 34359738368$$

$$2^{36} = 68719476736$$

$$2^{37} = 137438953472$$

$$2^{38} = 274877906944$$

$$2^{39} = 549755813888$$

$$2^{40} = 1099511627776$$

$$2^{41} = 219902325552$$

$$2^{42} = 439804651104$$

$$2^{43} = 879609302208$$

$$2^{44} = 1759218604416$$

$$2^{45} = 3518437208832$$

$$2^{46} = 7036874417664$$

$$2^{47} = 14073748835328$$

$$2^{48} = 28147497670656$$

$$2^{49} = 56294995341312$$

$$2^{50} = 112589990682624$$

$$2^{51} = 225179981365248$$

$$2^{52} = 450359962730496$$

$$2^{53} = 900719925460992$$

$$2^{54} = 1801439850921984$$

$$2^{55} = 3602879701843968$$

$$2^{56} = 7205759403687936$$

$$2^{57} = 14411518807375872$$

$$2^{58} = 28823037614751744$$

$$2^{59} = 57646075229503488$$

$$2^{60} = 115292150459006976$$

$$2^{61} = 230584300918013952$$

$$2^{62} = 461168601836027904$$

$$2^{63} = 922337203672055808$$

$$2^{64} = 1844674407344111616$$

$$2^{65} = 3689348814688223232$$

$$2^{66} = 7378697629376446464$$

$$2^{67} = 14757395258752892928$$

$$2^{68} = 29514790517505785856$$

$$2^{69} = 59029581035011571712$$

$$2^{70} = 118059162070023143424$$

$$2^{71} = 236118324140046286848$$

$$2^{72} = 472236648280092573696$$

$$2^{73} = 944473296560185147392$$

$$2^{74} = 1888946593120370294784$$

$$2^{75} = 3777893186240740589568$$

$$2^{76} = 7555786372481481179136$$

$$2^{77} = 1511157274496296235872$$

$$2^{78} = 3022314548992592471744$$

$$2^{79} = 6044629097985184943488$$

$$2^{80} = 12089258195970369886976$$

$$2^{81} = 24178516391940739773952$$

$$2^{82} = 48357032783881479547904$$

$$2^{83} = 96714065567762959095808$$

$$2^{84} = 193428131135525918191616$$

$$2^{85} = 386856262270751836383232$$

$$2^{86} = 773712524541503672766464$$

$$2^{87} = 1547425049083007345532928$$

$$2^{88} = 3094850098166014691065856$$

$$2^{89} = 6189700196332029382131712$$

$$2^{90} = 1237940039266405876426344$$

$$2^{91} = 2475880078532811752852688$$

$$2^{92} = 4951760157065623505705376$$

$$2^{93} = 9903520314131247011410752$$

$$2^{94} = 19807040628262494022821504$$

$$2^{95} = 39614081256524988045643008$$

$$2^{96} = 79228162513049976091286016$$

$$2^{97} = 158456325226099552182572032$$

$$2^{98} = 316912650452199104365144064$$

$$2^{99} = 633825300904398208730288128$$

$$2^{100} = 1267650601808796417460576256$$

$$\log 100 = \log 10^2 = 2 \log 10 = 2 \cdot 1 = 2$$

$$\log 100 = \log 10^2 = 2 \log 10 = 2 \cdot 1 = 2$$

$$\log 100 = \log 10^2 = 2 \log 10 = 2 \cdot 1 = 2$$

$$\log 100 = \log 10^2 = 2 \log 10 = 2 \cdot 1 = 2$$

$$\log 100 = \log 10^2 = 2 \log 10 = 2 \cdot 1 = 2$$

$$\log 100 = \log 10^2 = 2 \log 10 = 2 \cdot 1 = 2$$

$$\log 100 = \log 10^2 = 2 \log 10 = 2 \cdot 1 = 2$$

$$\log 100 = \log 10^2 = 2 \log 10 = 2 \cdot 1 = 2$$

$$\log 100 = \log 10^2 = 2 \log 10 = 2 \cdot 1 = 2$$

Logarithm

How many time I should multiply 1 by 2 to get 8?

How many time I should divide 8 by 2 to get 1?

$$1 \cdot 2 \cdot 2 \cdot 2 = 8$$

$$2^? = 8$$

exponent $((8/2)/2)/2$

$$\log_2(8) = ?$$

$\textcircled{1} = 625$ base

$$5^? = 625$$

$$\log_5(625) = ?$$

$$5 \cdot 5 \cdot 5 \cdot 5 = 625$$

$$\log_5(625) = 4$$

$\textcircled{4}$

$$\log_{10} 100 = \log_{10} 100 \quad \text{Engineers}$$

$$\log_e 7.389 = \ln(7.389) \approx 2 \quad \text{Math}$$

$e = \text{Euler Number} = 2.71828$

$$\log_2 16 = \lg(16) = 4 \quad \text{CS}$$

$$2^{10} = 1024 \quad \log_2(1024) = 10$$

$\textcircled{0} = 1$ $\log_2 1 = 0$

$\textcircled{1} = 2$ $\log_2 2 = 1$

what is $\log_2 64$ $\frac{8}{2} = 64$

$$\log_2 64 = 6$$

$$\log_b n = \frac{\log_2 n}{\log_2 b}$$

$\log 10$

$$\log_b(xy) = \log_b(x) + \log_b(y).$$

$$\log_b(b^x) = x \log_b(b) = x.$$

$$b^{\log_b(y)} = y$$

$n = 1024$

$\sqrt[10]{1024} = 2$

$$\begin{array}{r}
 0 = 1 \\
 2^1 = 2 \\
 2^2 = 4 \\
 2^3 = 8 \\
 2^4 = 16 \\
 2^5 = 32 \\
 2^6 = 64 \\
 2^7 = 128 \\
 2^8 = 256 \\
 2^9 = 512 \\
 2^{10} = 1024
 \end{array}$$

Binary Search Tree

 n

$$\log_2 n = 1$$

$$\log_K n = \underline{\log_2 n}$$

$$\log_K^{n-1} (\log_3 n = 1)$$

$$\log_3 n = \underline{K \log_2 n}$$

$$\underline{\log_5 n} = 1$$

$$\underline{\log_3 2}$$

\log_{10}	x	$\log_{10}x$	$\log_2 x$	$\log_e x$
1	1	0.000000	0.000000	0.000000
2	2	0.301030	1.000000	0.693147
3	3	0.477121	1.584963	1.098612
4	4	0.602060	2.000000	1.386294
5	5	0.698970	2.321928	1.609438
6	6	0.778151	2.584963	1.791759
7	7	0.845098	2.807355	1.945910
8	8	0.903090	3.000000	2.079442
9	9	0.954243	3.169925	2.197225
10	10	1.000000	3.321928	2.302585
20	20	1.301030	4.321928	2.995732
30	30	1.477121	4.906891	3.401197
40	40	1.602060	5.321928	3.688879
50	50	1.698970	5.643856	3.912023
60	60	1.778151	5.906991	4.094345
70	70	1.845098	6.129283	4.248495
80	80	1.903090	6.321928	4.382027
90	90	1.954243	6.491853	4.499810
100	100	2.000000	6.643856	4.605170
200	200	2.301030	7.643856	5.298317
300	300	2.477121	8.228819	5.703782
400	400	2.602060	8.643856	5.991465
500	500	2.698970	8.965784	6.214608
600	600	2.778151	9.228819	6.396930
700	700	2.845098	9.451211	6.551080
800	800	2.903090	9.643856	6.684612
900	900	2.954243	9.813781	6.802305
1000	1000	3.000000	10.000000	6.907755
10000	10000	4.000000	13.287712	9.210340

2-

$$\log_b n = \frac{\log_2 n}{\log_2 b}$$

~~$\log_2 b$~~

Independent
 $O(\log n)$

Base does not matter

increases by $\log n$ constant amount independent of n

$$\begin{aligned} 2^3 &= 8 \\ 2^4 &= 16 \\ 2^5 &= 32 \\ 2^6 &= 64 \\ 2^7 &= 128 \end{aligned}$$

$$\begin{aligned} 2^{12} &= 4096 \\ 2^{10} &= 1024 \\ 2^{11} &= 2048 \\ 2^{12} &= 4096 \end{aligned}$$

(Constant)

log n Behavior

A bartender offers 10000\$ bet. If you win, you get 10000\$ or you should pay him 10000\$

You choose a number between 1 to 1,000,000. Bartender will tell that number in 20 guesses.

After each guess, you should tell bar attendant:

1. TOO HIGH
2. TOO LOW
3. YOU(bar attendant) are right.

If bartender cannot get your chosen number in 20 guesses, you will win 10000\$. Otherwise, you have to pay 10000\$ to the bartender.

Will you take that bet ?

How many MAXIMUM guesses are required to answer a number between 1-10 ?

$4 = \log_2 10$

In general $\log_2 N$ guesses

$\log_2 1000000 = 19.93 = 20$ guesses

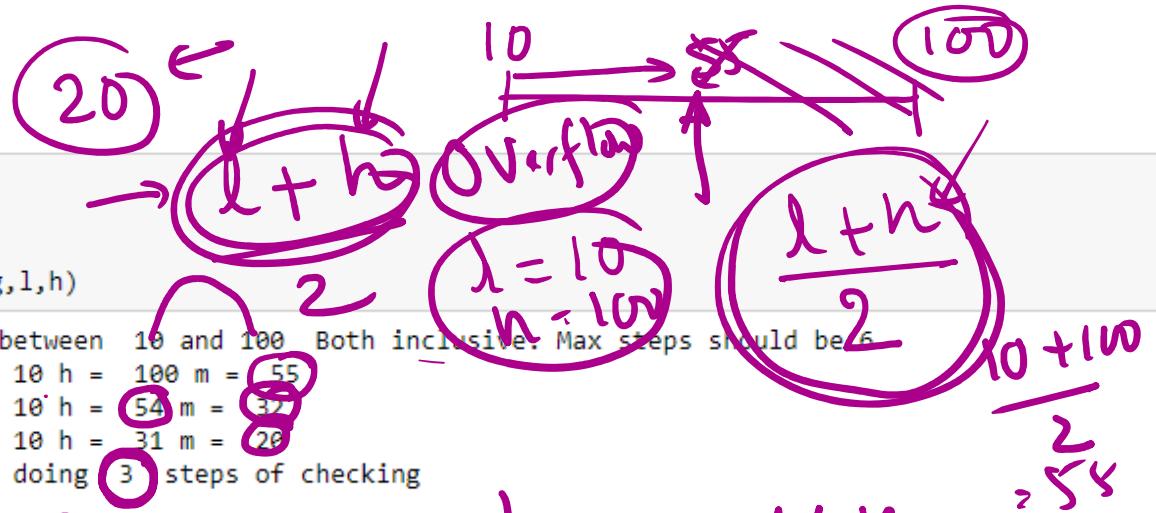
$2^{20} = 1048576$

In [7]:

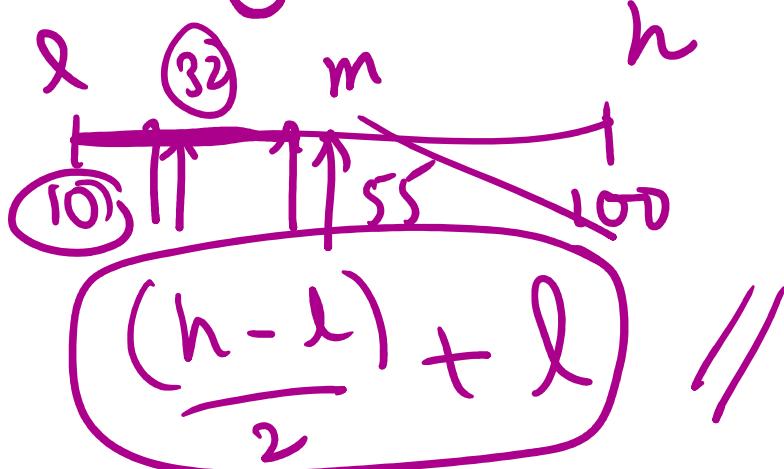
```

1 g = 20
2 l = 10
3 h = 100
4 guesstop(g,l,h)

```



Python



[27]:

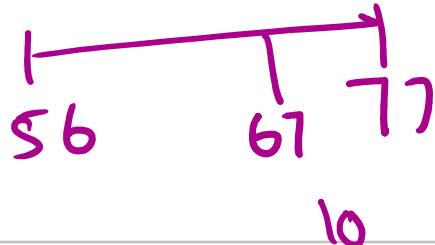
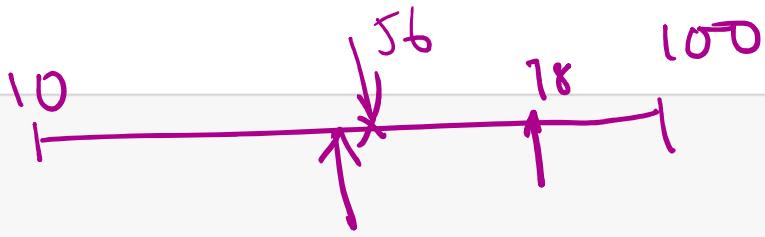
```

1 g = 76
2 l = 10
3 h = 100
4 guesstop(g,l,h)

```

is 76 there between 10 and 100 Both inclusive. Max steps should be 6
 step = 1 l = 10 h = 100 m = 55
 step = 2 l = 56 h = 100 m = 78
 step = 3 l = 56 h = 77 m = 66
 step = 4 l = 67 h = 77 m = 72
 step = 5 l = 73 h = 77 m = 75
 step = 6 l = 76 h = 77 m = 76
 76 Found after doing 6 steps of checking

(7b)



$$\begin{array}{r}
 6 \\
 2^6 = 64 \\
 2^7 = 128
 \end{array}
 \quad 7 \quad 20 \quad 2$$

In [28]:

```
1 g = 500178
2 l = 1
3 h = 1000000
4 guesstop(g,l,h)

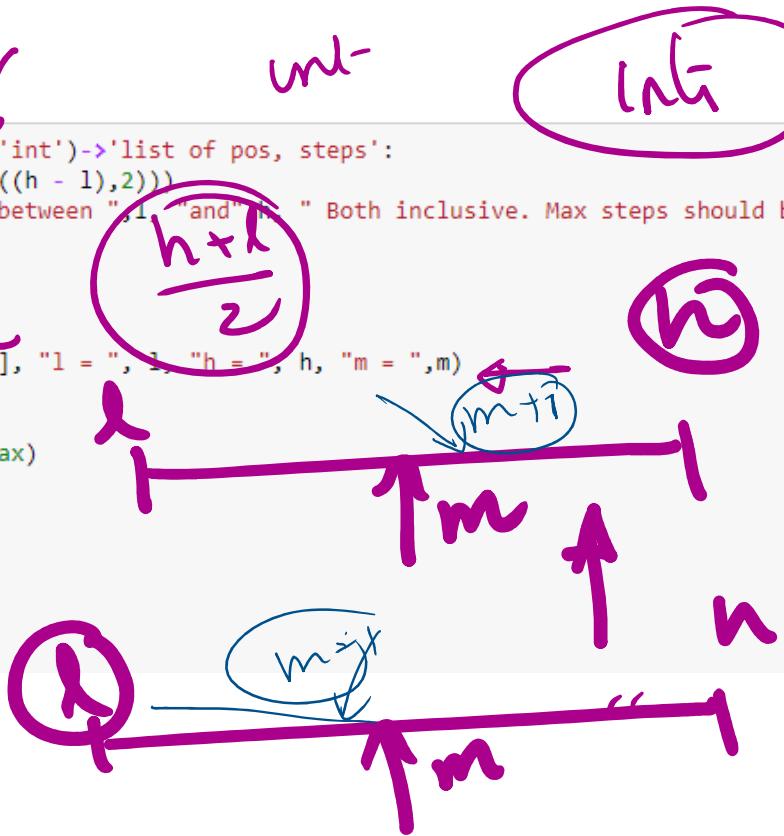
is 500178 there between 1 and 1000000 Both inclusive. Max steps should be 20
step = 1 l = 1 h = 1000000 m = 500000
step = 2 l = 500001 h = 1000000 m = 750000
step = 3 l = 500001 h = 749999 m = 625000
step = 4 l = 500001 h = 624999 m = 625000
step = 5 l = 500001 h = 562499 m = 531250
step = 6 l = 500001 h = 531249 m = 515625
step = 7 l = 500001 h = 515624 m = 507812
step = 8 l = 500001 h = 507811 m = 503906
step = 9 l = 500001 h = 503905 m = 501953
step = 10 l = 500001 h = 501952 m = 500976
step = 11 l = 500001 h = 500975 m = 500488
step = 12 l = 500001 h = 500487 m = 500244
step = 13 l = 500001 h = 500243 m = 500122
step = 14 l = 500123 h = 500243 m = 500183
step = 15 l = 500123 h = 500182 m = 500152
step = 16 l = 500153 h = 500182 m = 500167
step = 17 l = 500168 h = 500182 m = 500175
step = 18 l = 500176 h = 500182 m = 500179
step = 19 l = 500176 h = 500178 m = 500177
step = 20 l = 500178 h = 500178 m = 500178
500178 Found after doing 20 steps of checking
```

In [28]:

```

1 def guess(g:'int', l:'int', h:'int')->'list of pos, steps':
2     max = int(round(math.log((h - l), 2)))
3     print("is ", g, " there between ", l, " and ", h, " Both inclusive. Max steps should be", max)
4     a = [-1, 0]
5     while (l <= h):
6         a[1] = a[1] + 1
7         m = (h - l) // 2 + l
8         print("step = ", a[1], " l = ", l, " h = ", h, " m = ", m)
9         if (g == m):
10            a[0] = m
11            assert(a[1] <= max)
12            return a
13         if (m < g):
14             l = m + 1
15         else:
16             h = m - 1
17     return a
18

```



[31]:

```
1 g = 101
2 l = 1
3 h = 100
4 guess(g,l,h)
```

is 101 there between 1 and 100 Both inclusive. Max steps should be 7

step = 1 l = 1 h = 100 m = 50
step = 2 l = 51 h = 100 m = 75
step = 3 l = 76 h = 100 m = 88
step = 4 l = 89 h = 100 m = 94
step = 5 l = 95 h = 100 m = 97
step = 6 l = 98 h = 100 m = 99
step = 7 l = 100 h = 100 m = 100

101 is not there after doing 7 steps of checking

7 steps

7:22 PM

log n Behavior

How many **MAXIMUM** guesses are required to answer a number between 1-10 ?

$$4 = \log_2 10$$

In general $\log_2 N$ guesses

Number to be guessed 1

$$\begin{array}{ll} 1: l = 1 & r = 10 \text{ m} = 5 \\ 2: l = 1 & r = 4 \text{ m} = 2 \\ 3: l = 1 & r = 1 \text{ m} = 1 \end{array}$$

Number to be guessed 2

1: l= 1 r = 10 m = 5
2: l= 1 r = 4 m = 2

Number to be guessed 3

$$\begin{array}{ll} 1: l = 1 & r = 10 \text{ m} = 5 \\ 2: l = 1 & r = 4 \text{ m} = 2 \\ 3: l = 3 & r = 4 \text{ m} = 3 \end{array}$$

Number to be guessed 4

1: l = 1 r = 10 m = 5
2: l = 1 r = 4 m = 2
3: l = 3 r = 4 m = 3
4: l = 4 r = 4 m = 4

Number to be guessed 5

$$1: l = 1 \quad r = 10 \quad m = 5$$

Number to be guessed 6

- 1: $I = 1 \text{ r} = 10 \text{ m} = 5$
 2: $I = 6 \text{ r} = 10 \text{ m} = 8$
 3: $I = 6 \text{ r} = 7 \text{ m} = 6$

Number to be guessed 7

1: l = 1 r = 10 m = 5
2: l = 6 r = 10 m = 8
3: l = 6 r = 7 m = 6
4: l = 7 r = 7 m = 7

Number to be guessed 8

1: l = 1 r = 10 m = 5
2: l = 6 r = 10 m = 8

$$\frac{3}{2} = 8$$
$$\frac{4}{2} = 16$$

4 steps

Luck

1

A hand-drawn diagram illustrating an algorithm's execution flow. A large bracket labeled $O(n)$ covers the top path. A smaller bracket labeled $O(n)$ covers the bottom path. An arrow labeled n steps points from the bottom path to the right.

Logout

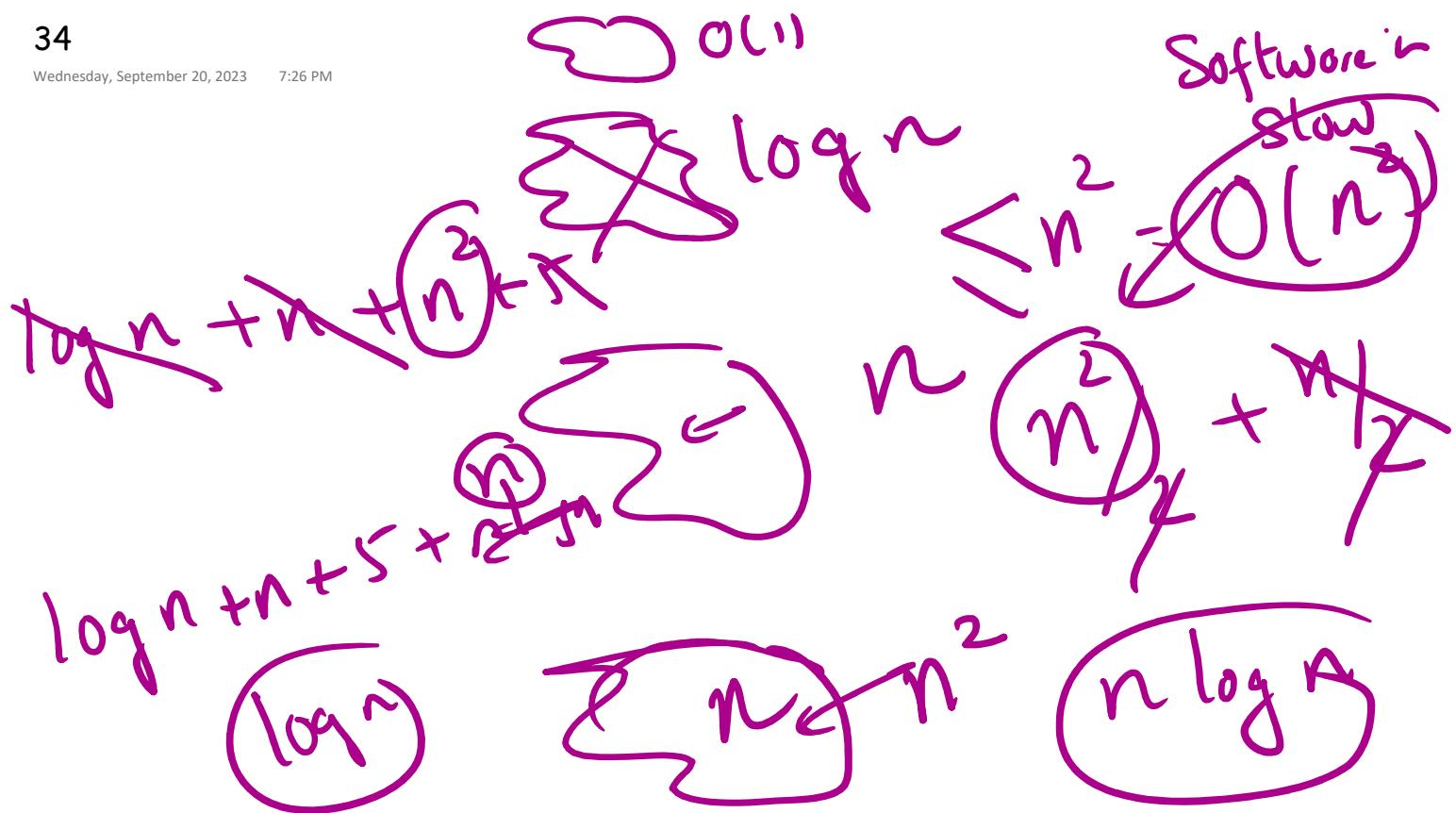
A hand-drawn diagram consisting of a horizontal rectangle divided into five equal vertical segments by four internal lines. Below the first segment, there is a double-headed arrow indicating its width.

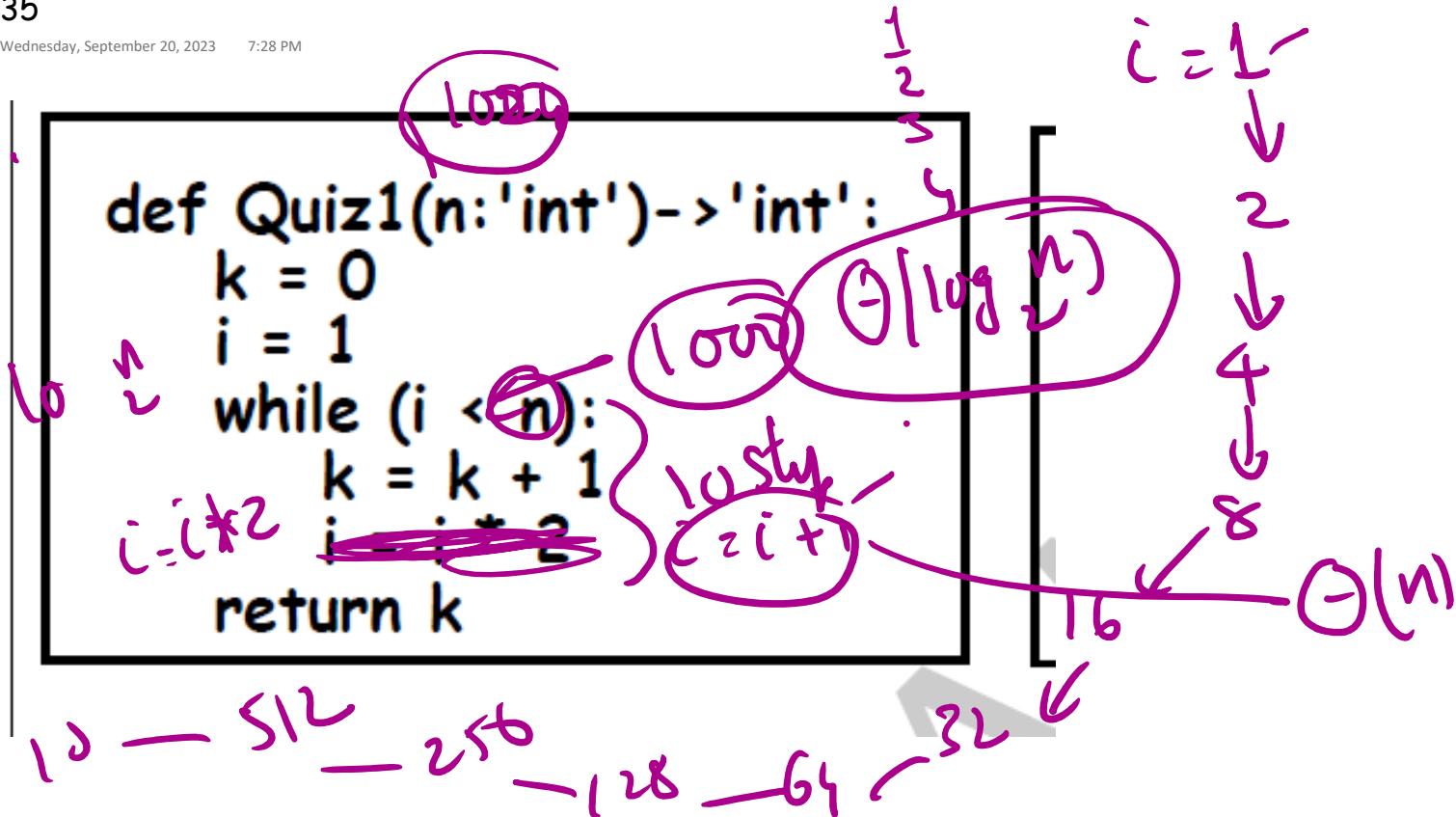
UCL & N
Value

$\Theta(n)$

1

~~COLK~~ n stop





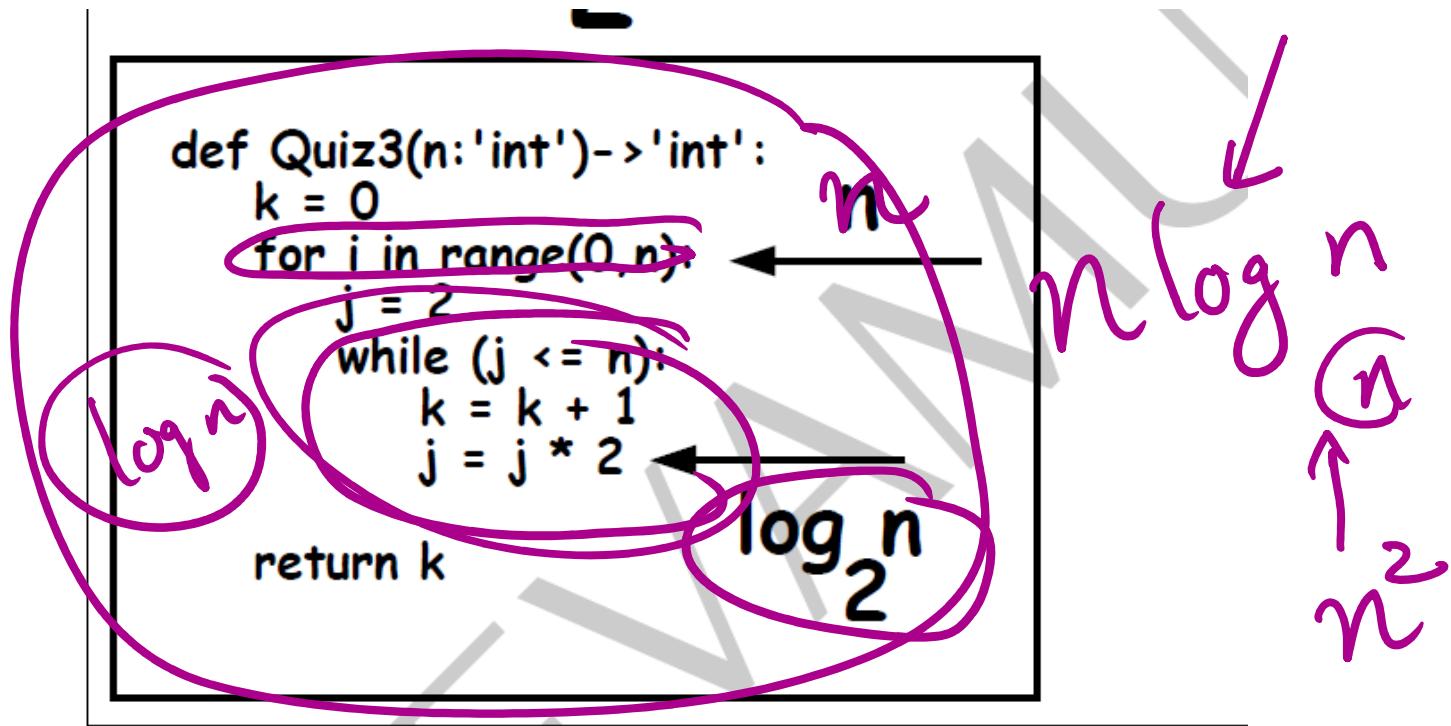
$n = 1024$

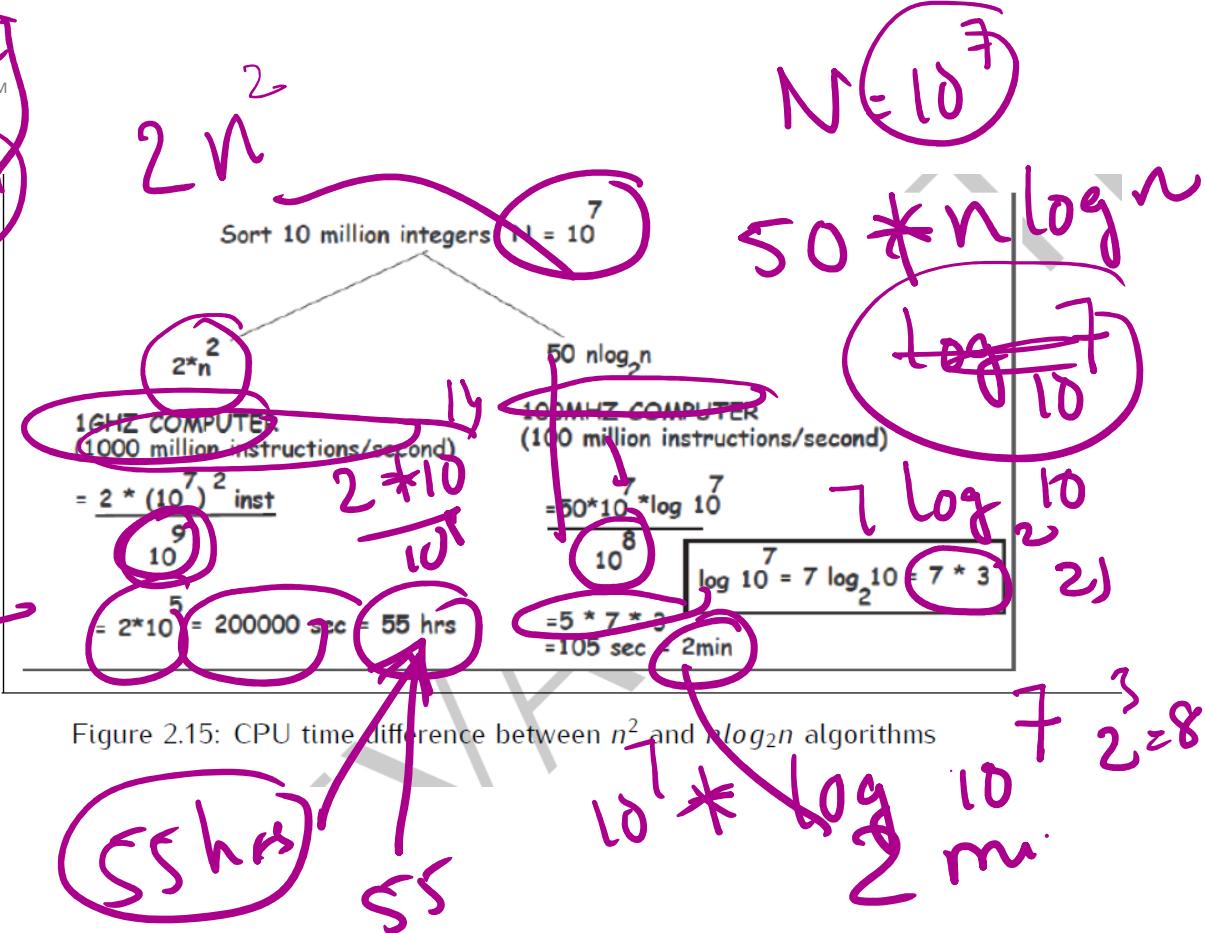
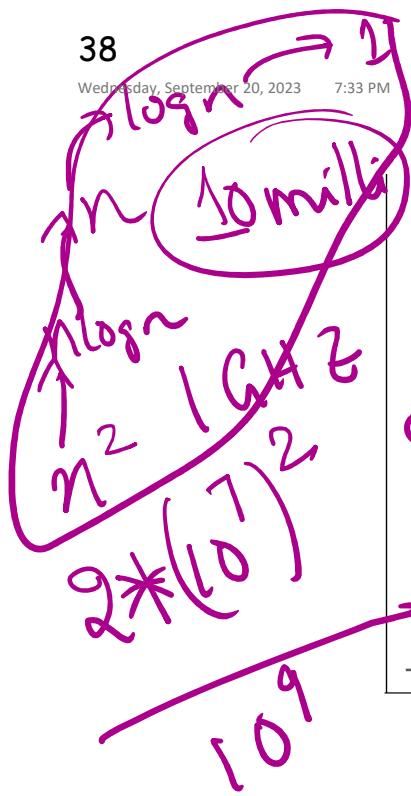
```
def Quiz2(n: 'int') -> 'int':  
    k = 0  
    while (n > 1):  
        k = k + 1  
        n = n // 2  
    return k
```

1024
↓
512
↓
256
↓
128
↓
64

loss
20 steps

2 - 4 - 8 - 16 - 32 ← 64



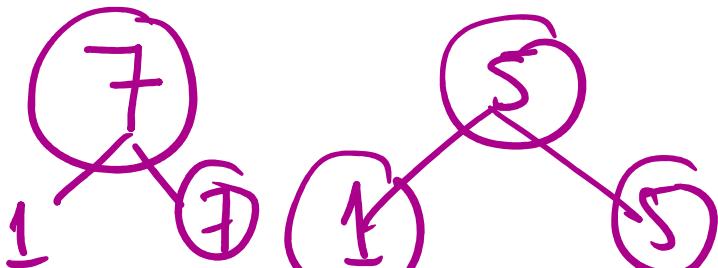


1 is NOT A

The prime numbers until 25 are
 2, 3, 5, 7, 11, 13, 17, 19, 23

$$\#(P_{25}) = 9$$

25



$$n = 1000$$

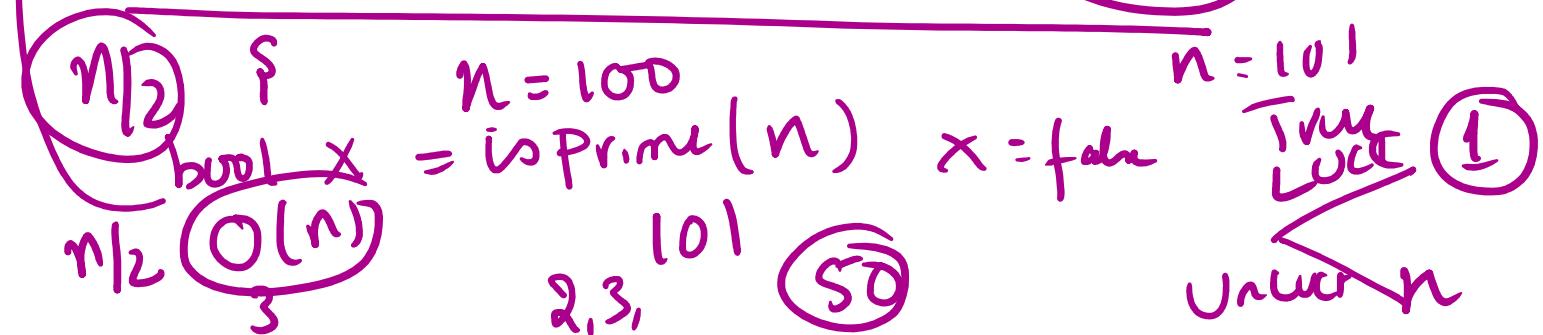
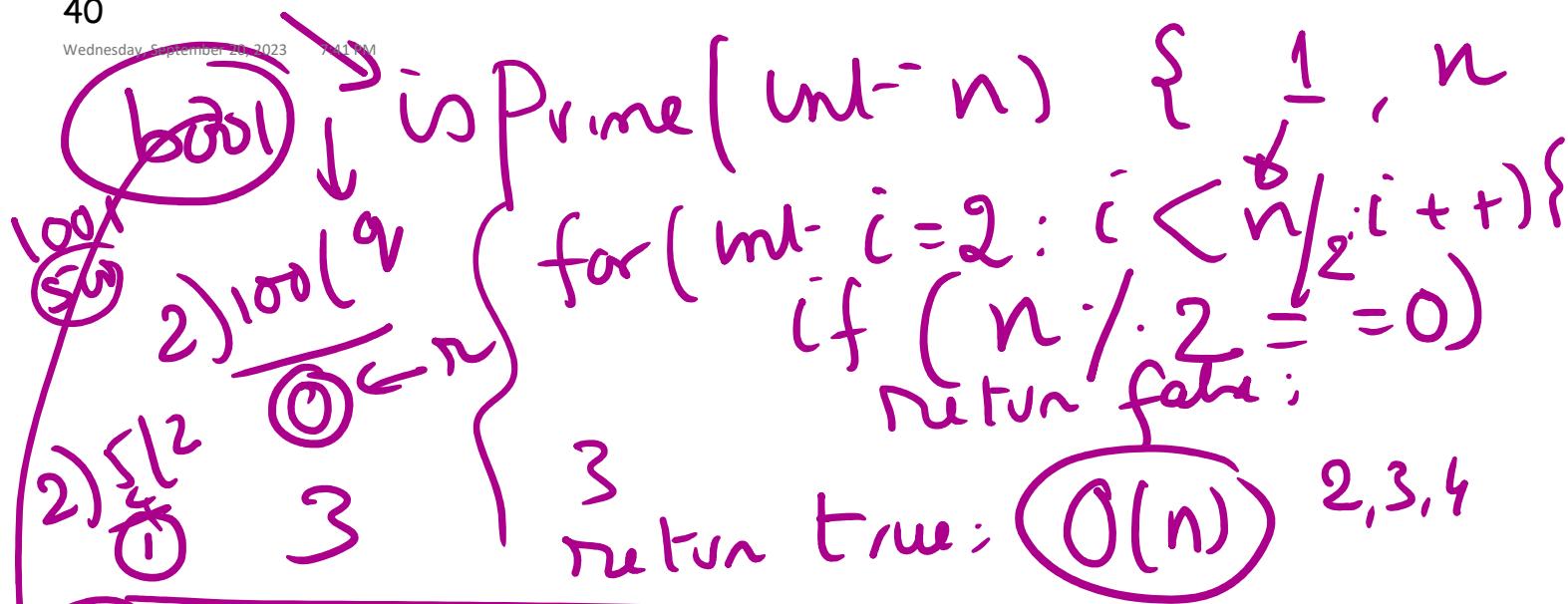
$$n = 1 \text{ m}$$

2

1, 5
 2, 3, 5, 7, 11, 13, 17

$$n = 25$$

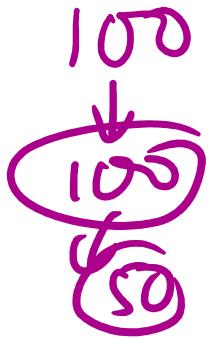
19 23



100

n

n/2



$$n = \sqrt{n} * \sqrt{n}$$

$$25 = \cancel{5} + \cancel{5}$$

for ($i=2 : i < \sqrt{n}$, $+ri$)

10 * 10

10

$\log n$

1 and itself (2, 3, 5, 7, 11, 13, 17, 19, 23)

boolean isPrimeUptoSquareRoot(int n) {

If n is factorisable

$$n = r * q$$

r or q must be $\leq \text{SQRT}(n)$

$$n \quad \text{SQRT}(n) \quad (r * q)$$

25	5	$(5 * 5)$
18	4.2	$(3 * 6)$
24	4.8	$(2 * 12)$

}

void uptoSquareRoot() {

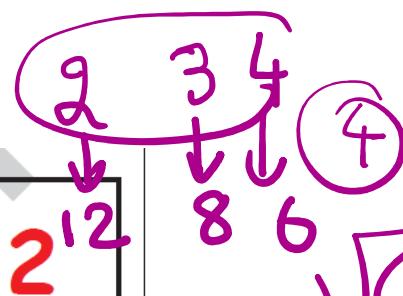
for (int i = 2; i $\leq \text{max}$; ++i) {

if (isPrimeUptoSquareRoot(i) == true) {

p[pkount++] = i;

}

,



$$O(n\sqrt{n})$$

$$\sqrt{24} = 4.8$$

$$= 4.8$$

2, 3, 4

3, 8

```
boolean isPrimeUptoSquareRoot(int n) {
    If n is factorisable
        n = r * q
        r or q must be <= SQRT(n)
```

$n = \text{SQRT}(n)$ ($r * q$)

25	5	(5 * 5)
18	4.2	(3 * 6)
24	4.8	(2 * 12)

```
}
```

```
void uptoSquareRoot() {
    for (int i = 2; i <= max; ++i) {
        if (isPrimeUptoSquareRoot(i) == true) {
            p[pkount++] = i;
        }
    }
}
```

2

$O(n\sqrt{n})$

MAX = 25

$n\sqrt{n}$

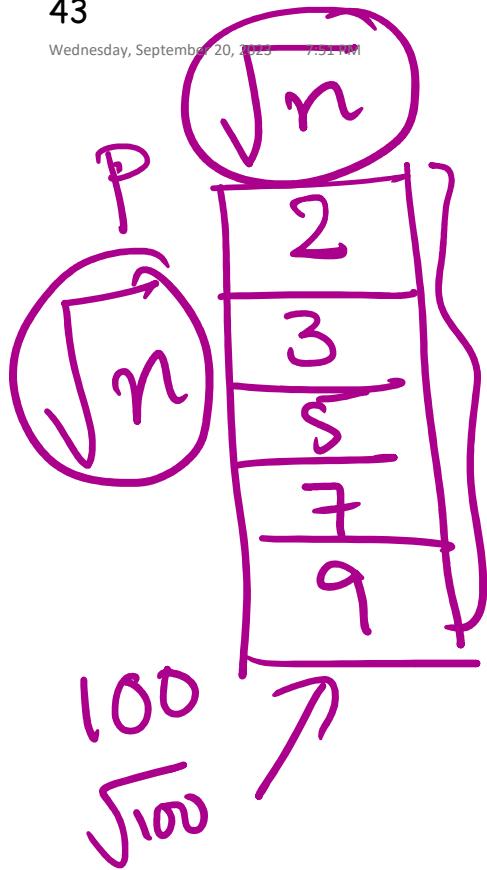
for (2 to \sqrt{n})

25

1 2 3 4 5
2 3 5 7 11 13 17 19 23
6 18

loop
con 50
10

$n = 9$



UP TO PRIME Number
 $n = 25$

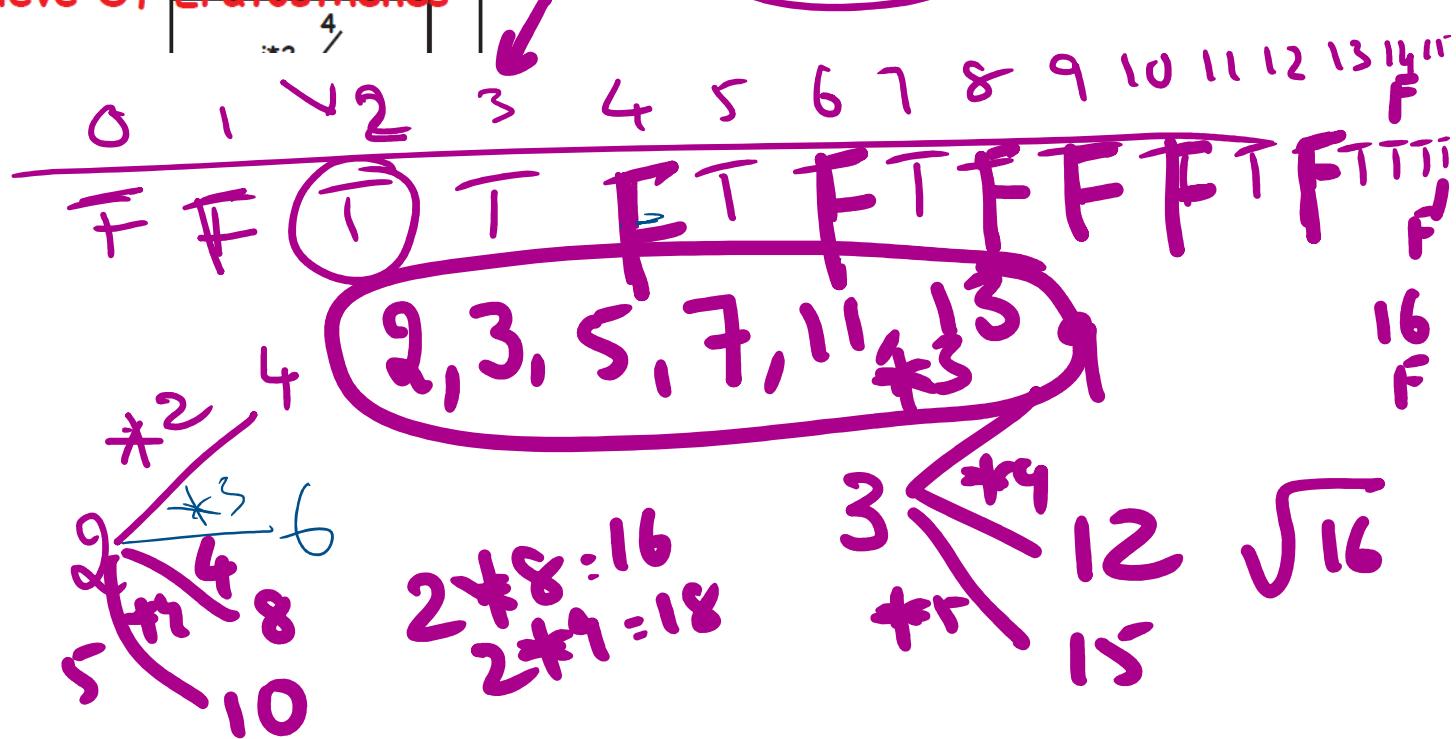
23 5
 Is n is divisible by PRIM

\hat{n} $\boxed{1 \ 2 \ 3 \ 5 \ 7 \ 9}$ \sqrt{n}

\sqrt{n} $n = 16$

Memory

Sieve Of Eratosthenes



25

9

10000

```
5 #####  
6  
7 class Solution:  
8     def __init__(self,n:'int',prime_number_list:'list')->None:  
9         self._n = n  
10        self._prime_number_list = prime_number_list #You need to append this using _append_prime_number()  
11        self._work_done = 0 #You need to increment using _increment_work_done()  
12        ##YOU CANNOT have anything here  
13  
14 #####  
15 # All work done goes through this routine.  
16 # Nothing can be changed.  
17 #####  
18 def _increment_work_done(self):  
19     self._work_done = self._work_done + 1  
20  
21 #####  
22 # All prime numbers are appended to the list.  
23 # Nothing can be changed.  
24 #####  
25 def _append_prime_number(self,p):  
26     self._prime_number_list.append(p)
```

fr[i=0];
S

```

27     ## Required function to implement
28     def nsquare(self) -> 'int':
29         ## NOTHING CAN BE CHANGED HERE
30         ## YOU CANNOT CALL math.log from Python Library
31         self._nsquare()
32         return self._work_done
33
34
35     ## Required function to implement
36     def up_to_prime(self) -> 'int':
37         ## NOTHING CAN BE CHANGED HERE
38         ## you cannot call log from Python Library
39         self._up_to_primes()
40         return self._work_done
41
42     ## Required function to implement
43     def sieve_of_eratosthenes(self) -> 'int':
44         ## NOTHING CAN BE CHANGED HERE
45         ## YOU CANNOT CALL math.log from Python Library
46         self._sieve_of_eratosthenes()
47         return self._work_done
48
49 ######
50 # Implement your code BELOW
51 # You can have any number of private variables and functions
52 # YOU CANNOT CALL math.log from Python Library
53 #####
54
55

```

\sqrt{n}

$\mathcal{O}(n)$

\sqrt{n}

$2 \quad 4 \quad 6$

$n\sqrt{n}$