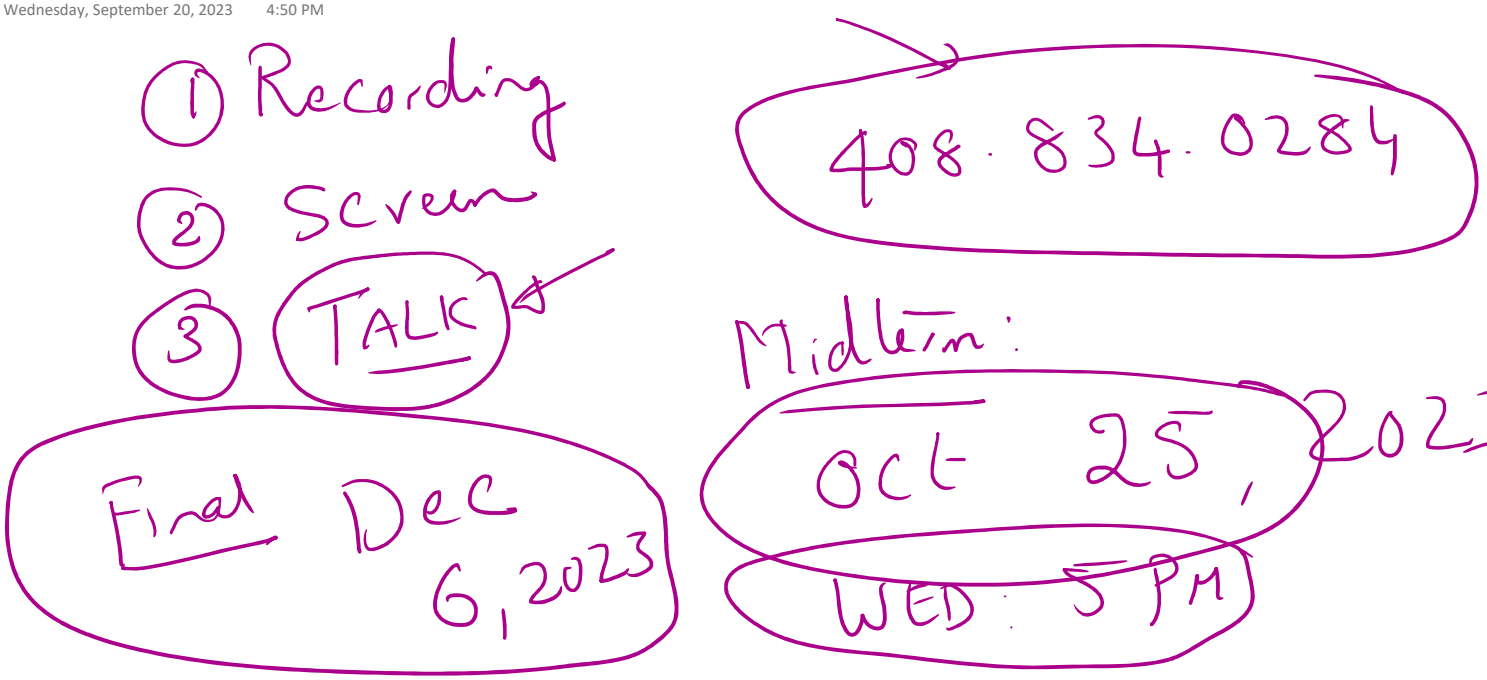
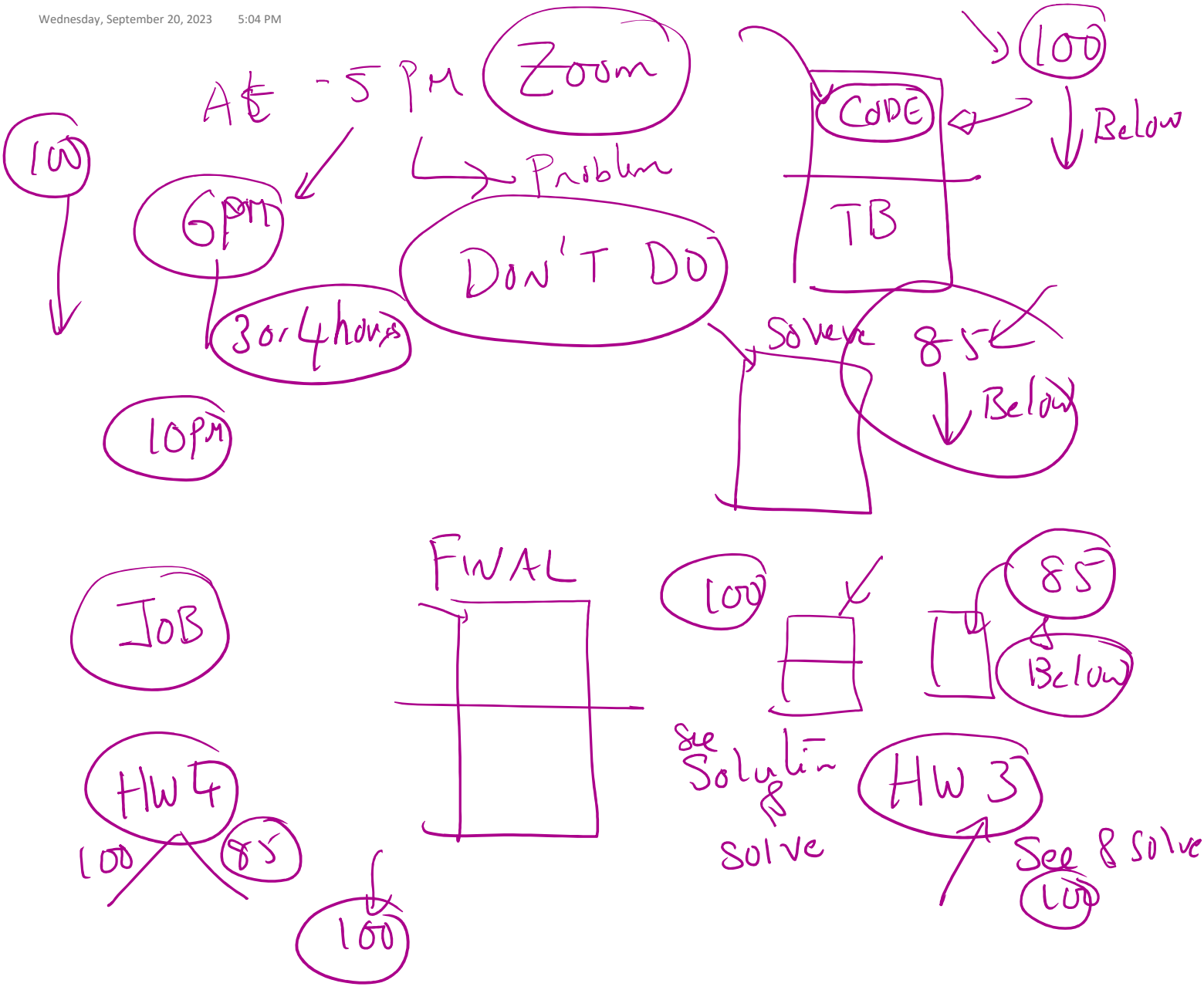
DAAPY 3

Wednesday, September 20, 2023 4:50 PM



1

Wednesday, September 20, 2023 5:04 PM



2

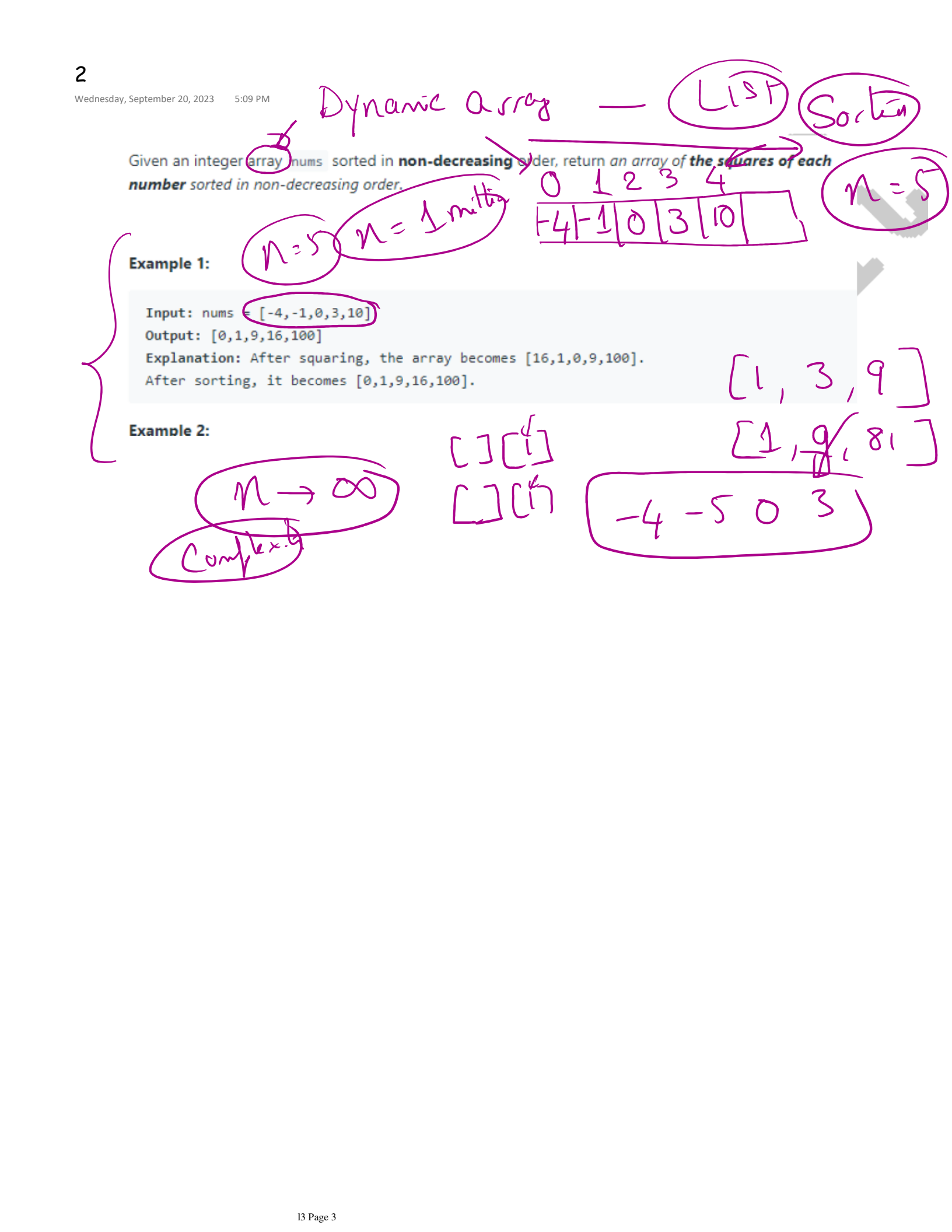
Wednesday, September 20, 2023 5:09 PM

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:



3

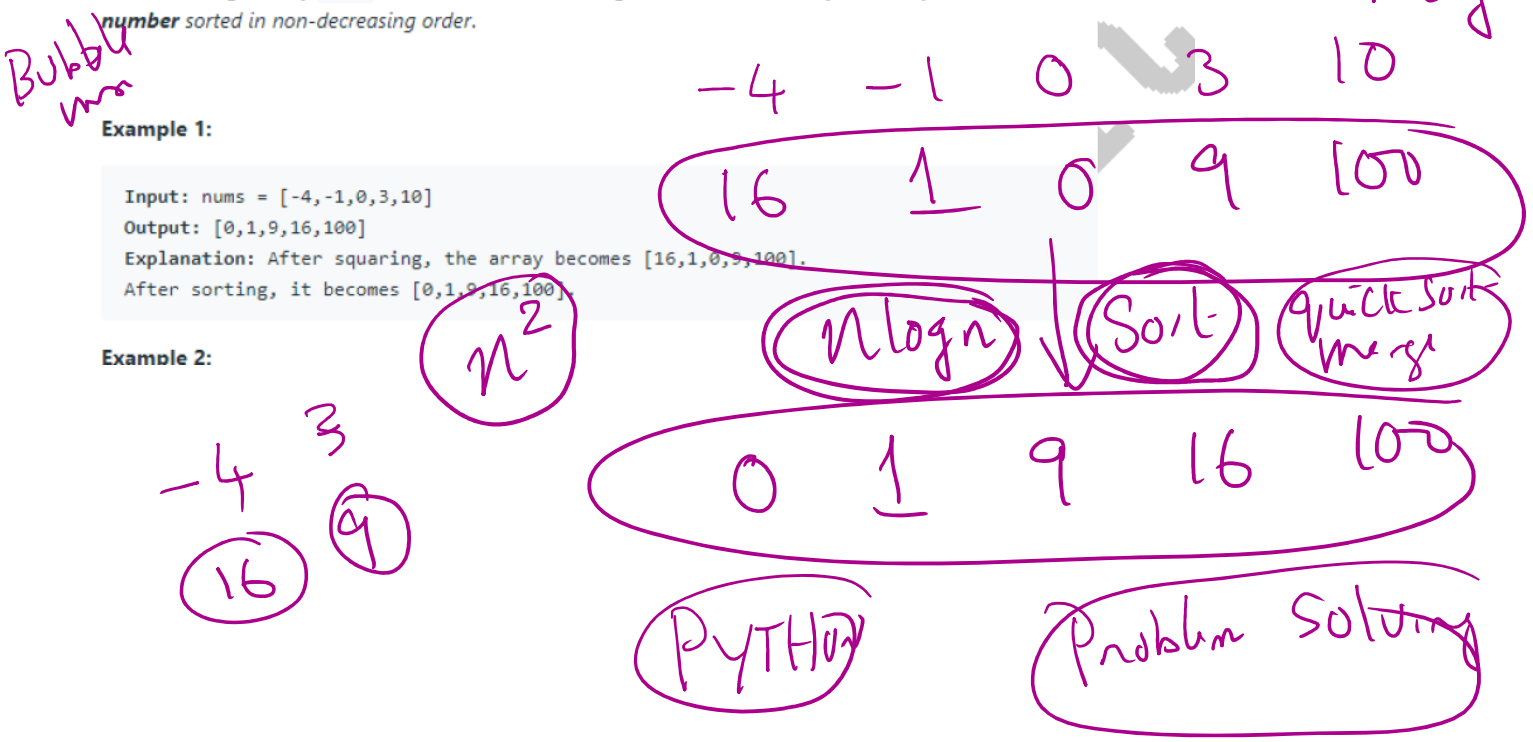
Wednesday, September 20, 2023 5:17 PM

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:



4

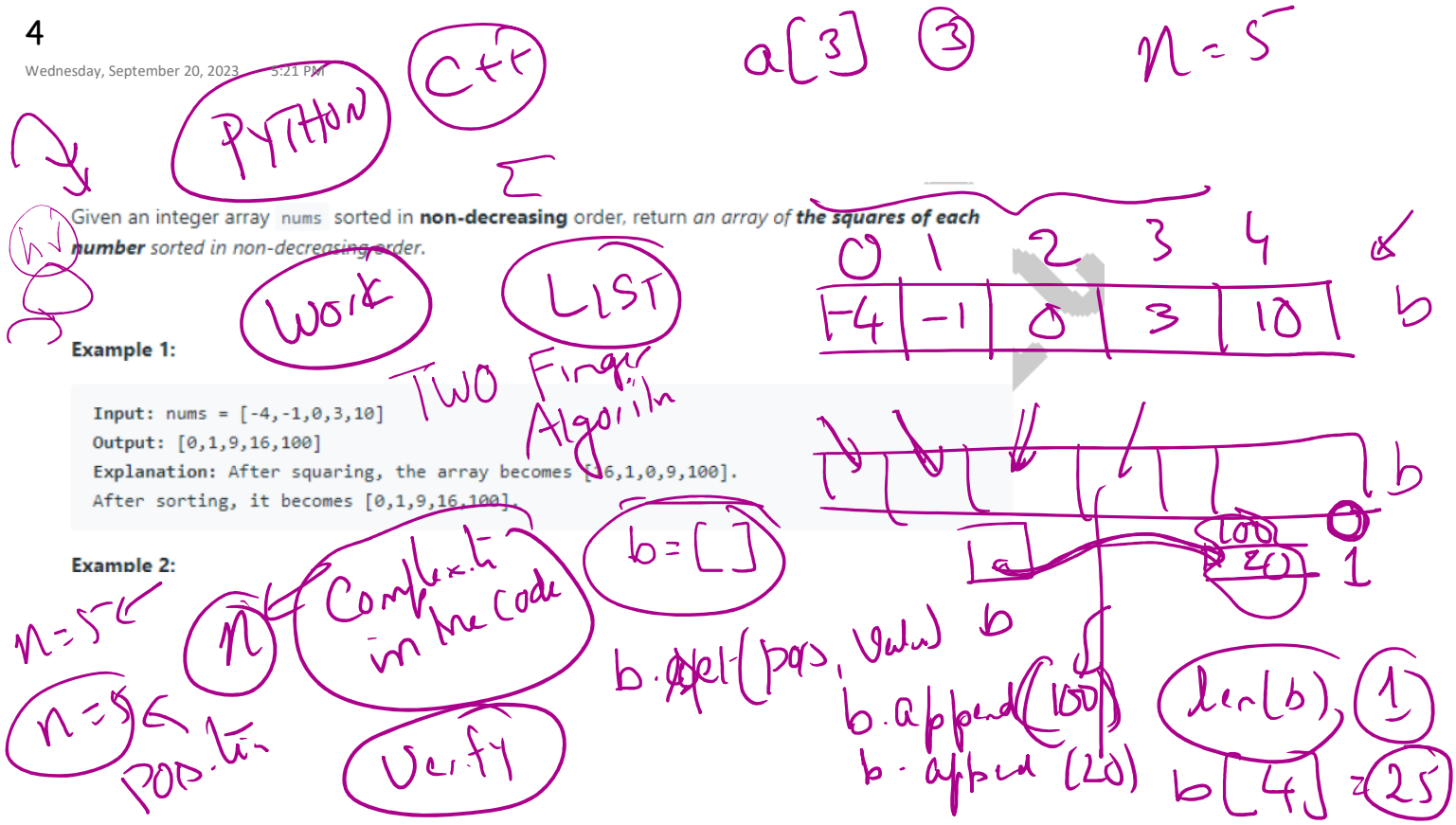
Wednesday, September 20, 2023 5:21 PM

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

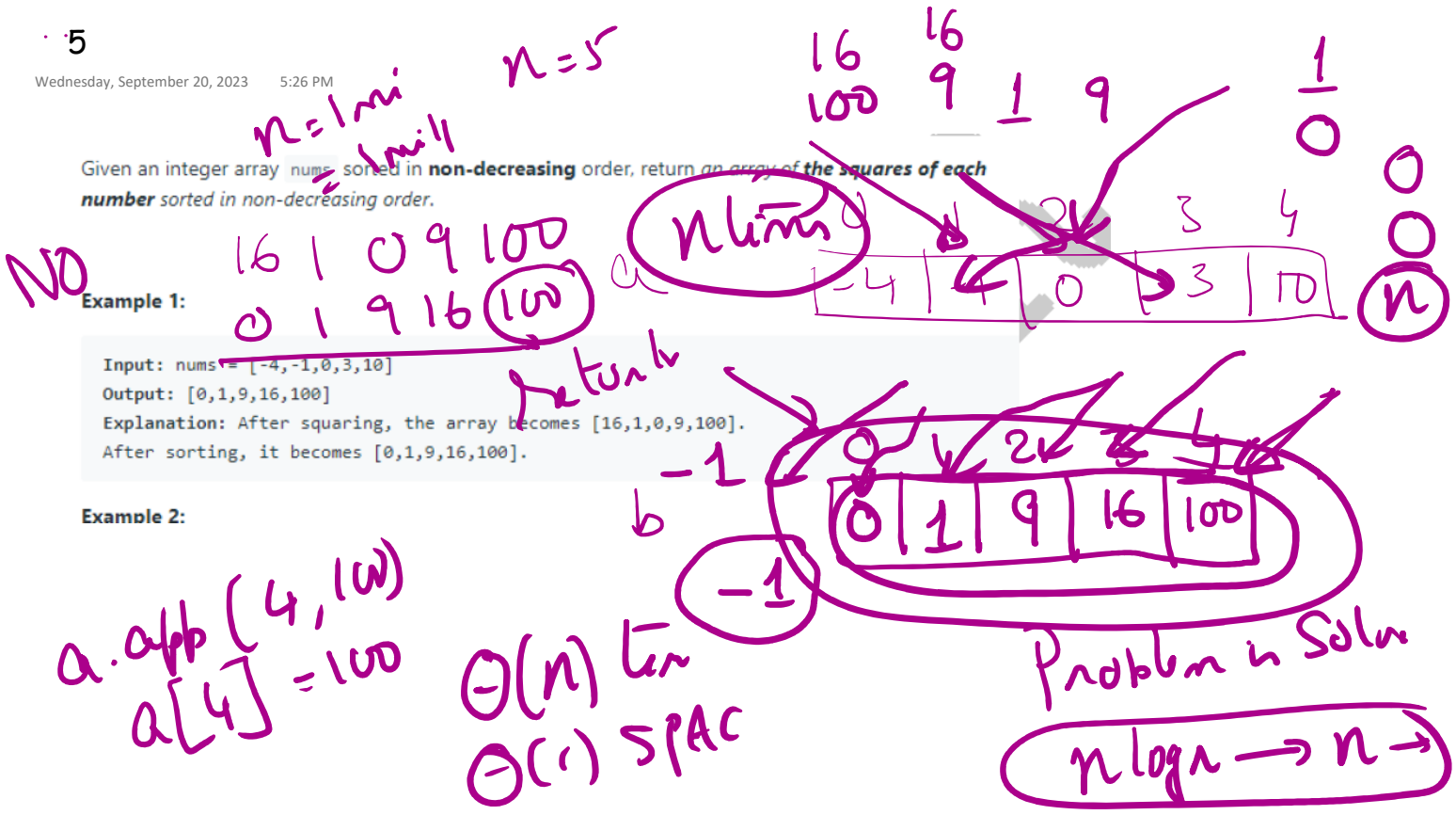
Wednesday, September 20, 2023 5:26 PM

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:

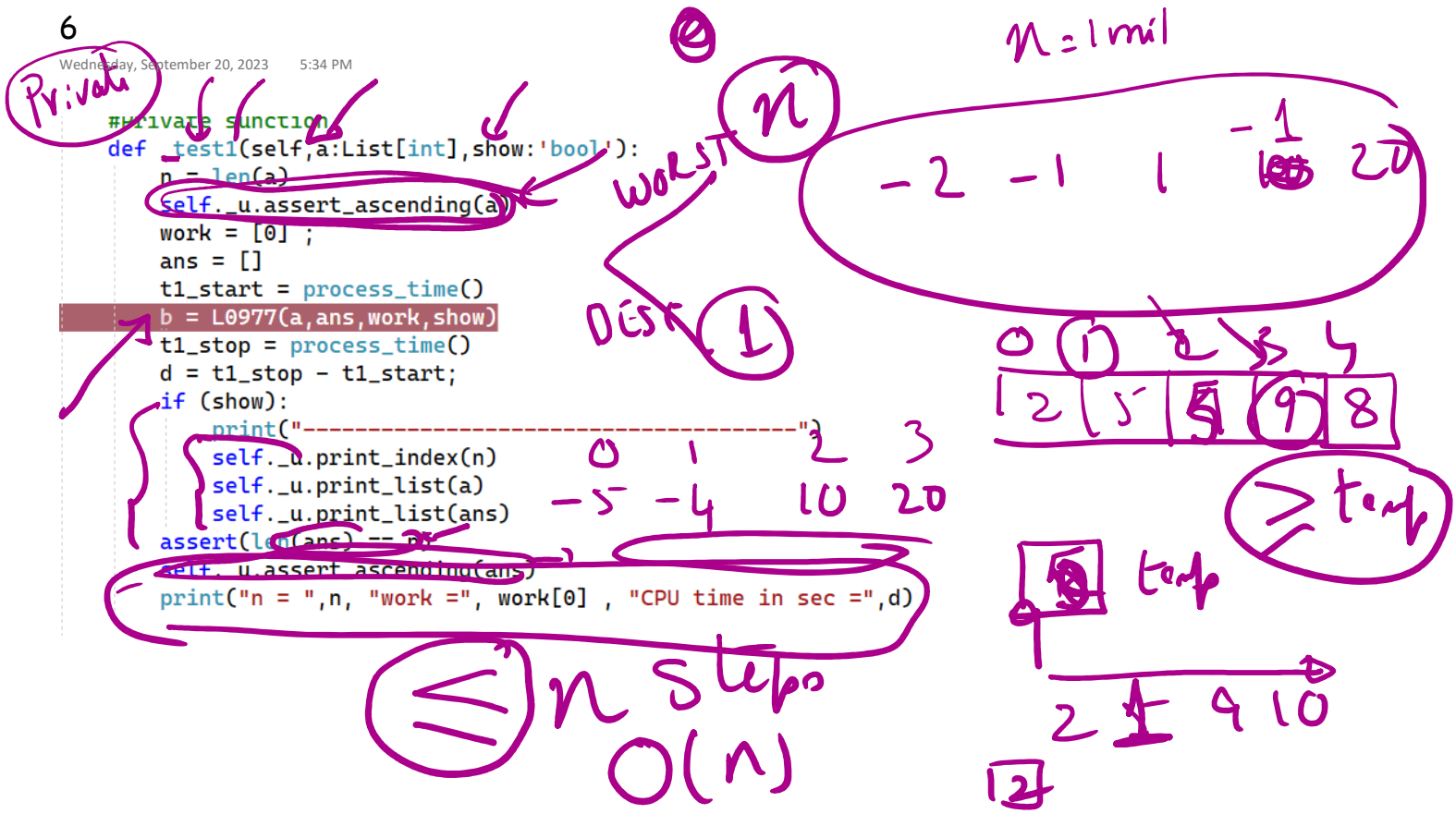
```
Input: nums = [-4,-1,0,3,10]
Output: [0,1,9,16,100]
Explanation: After squaring, the array becomes [16,1,0,9,100].
After sorting, it becomes [0,1,9,16,100].
```

Example 2:



Wednesday, September 20, 2023 5:34 PM

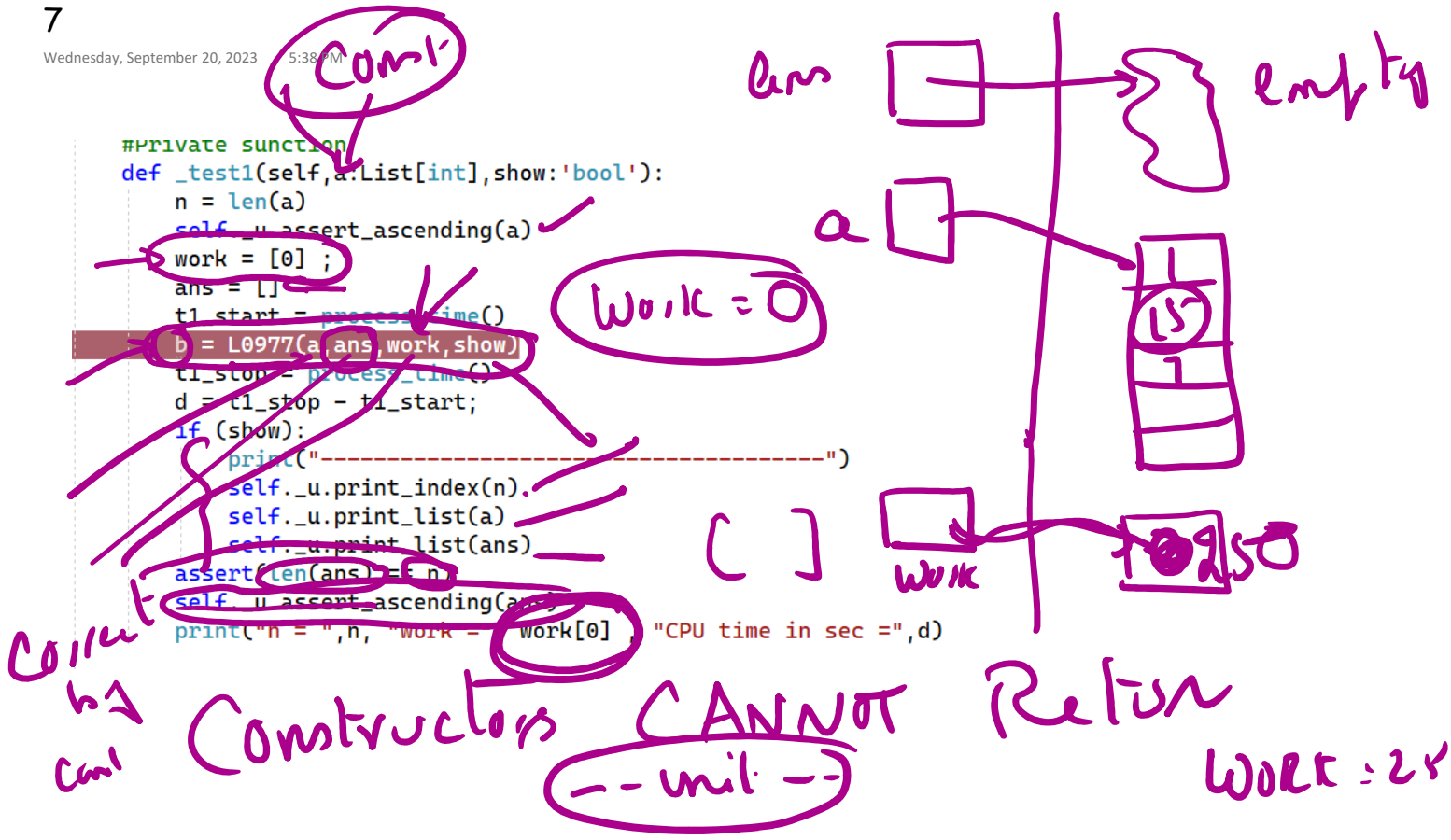
```
    #Private function
    def _test1(self,a:List[int],show:'bool'):
        n = len(a)
        self._u.assert_ascending(a)
        work = [0] ;
        ans = []
        t1_start = process_time()
        b = L0977(a,ans,work,show)
        t1_stop = process_time()
        d = t1_stop - t1_start;
        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("n = ",n, "work =", work[0] , "CPU time in sec =",d)
```



7

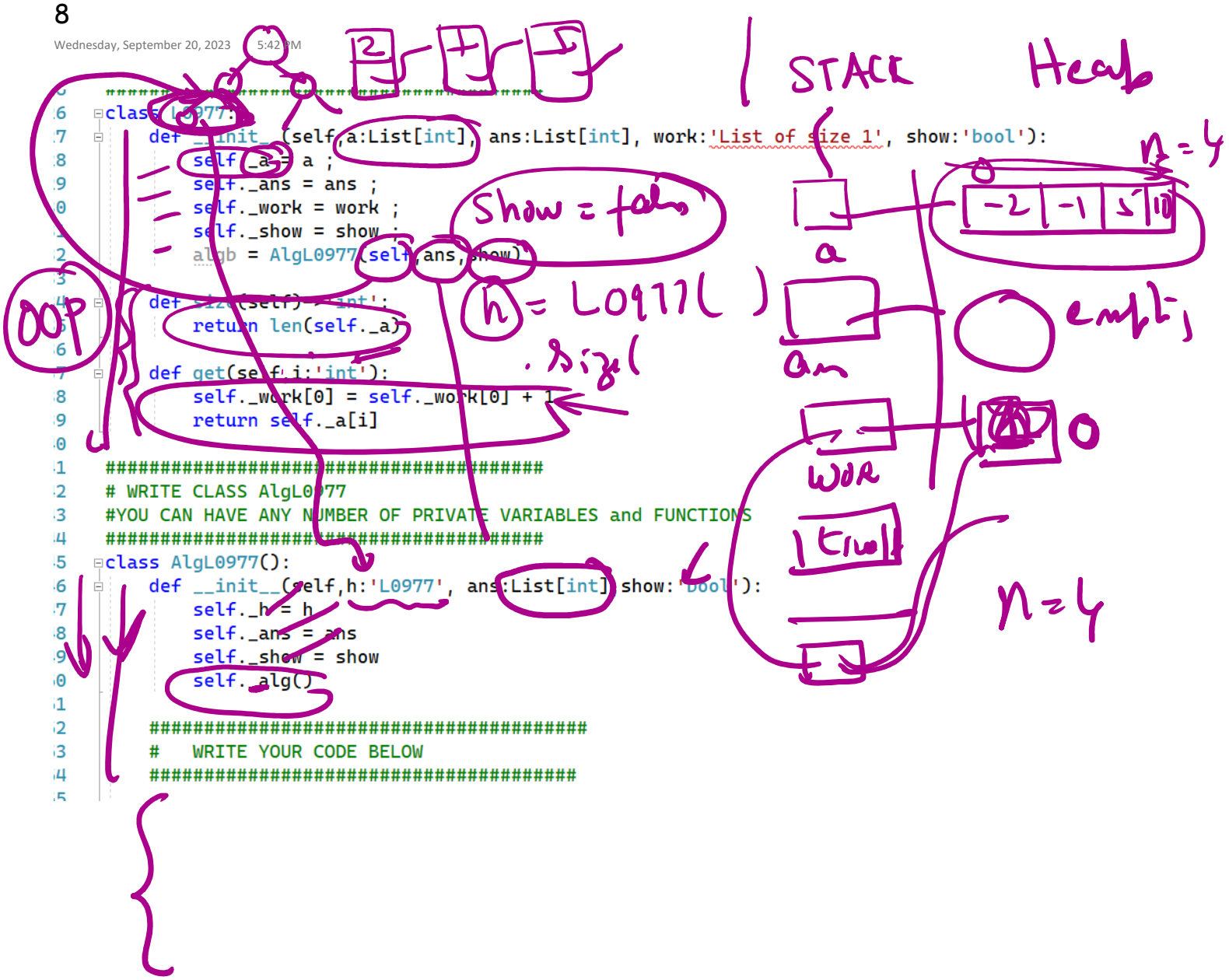
Wednesday, September 20, 2023 5:38 PM

```
    #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("n = ",n, "work = ", work[0] , "CPU time in sec =",d)
```



8

Wednesday, September 20, 2023 5:42 PM

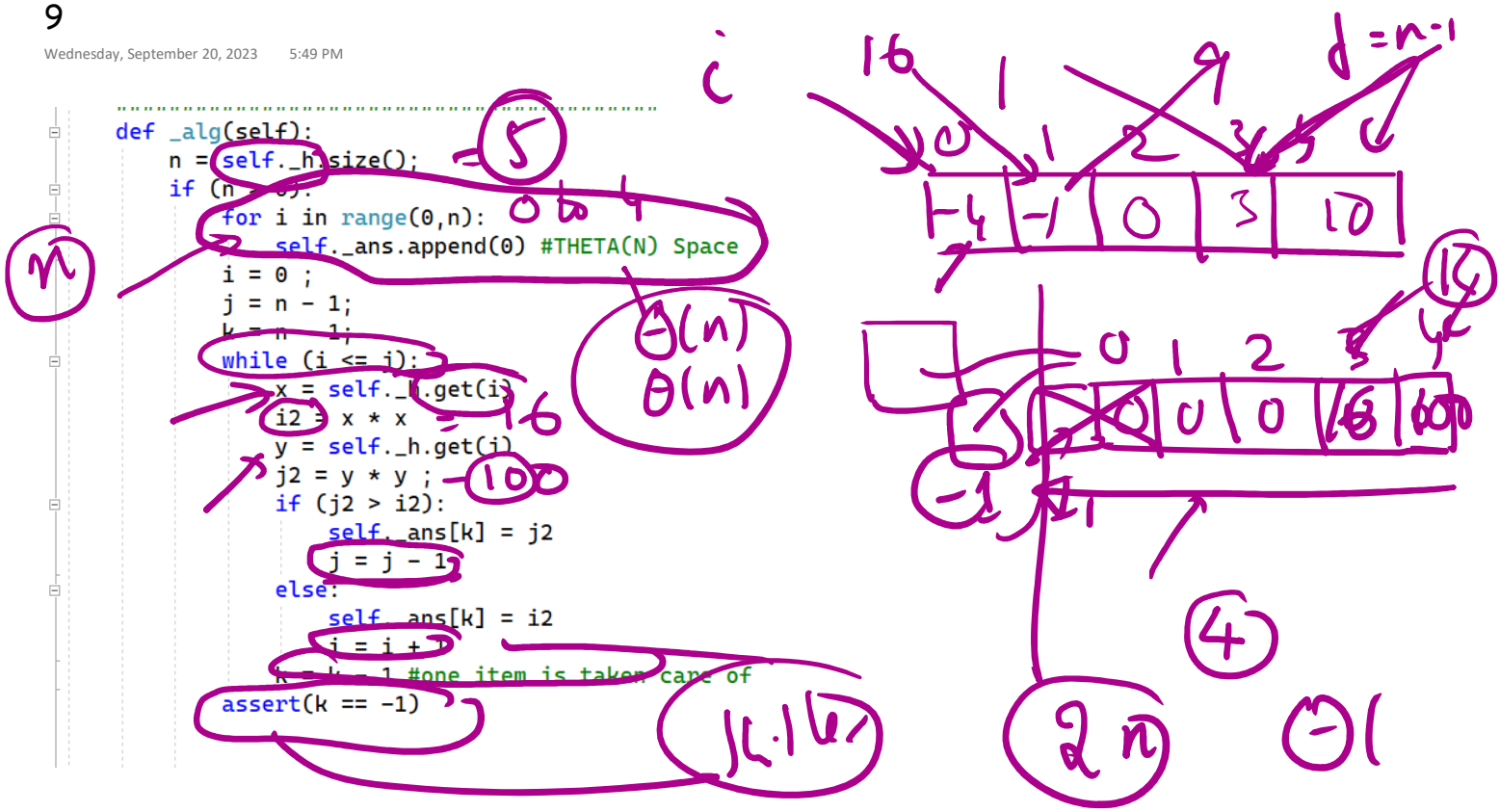


```
class L0977:
    def __init__(self,a:List[int], ans:List[int], work:'List of size 1', show:'bool'):
        self._a = a ;
        self._ans = ans ;
        self._work = work ;
        self._show = show ;
        algb = AlgL0977(self,ans,show)

    def size(self) -> 'int':
        return len(self._a)

    def get(self,i:'int'):
        self._work[0] = self._work[0] + 1
        return self._a[i]

########################################
# WRITE CLASS AlgL0977
#YOU CAN HAVE ANY NUMBER OF PRIVATE VARIABLES and FUNCTIONS
########################################
class AlgL0977():
    def __init__(self,h:'L0977', ans:List[int], show:'bool'):
        self._h = h
        self._ans = ans
        self._show = show
        self._alg()

    ########################################
    #   WRITE YOUR CODE BELOW
    ########################################
```

9

Wednesday, September 20, 2023 5:49 PM

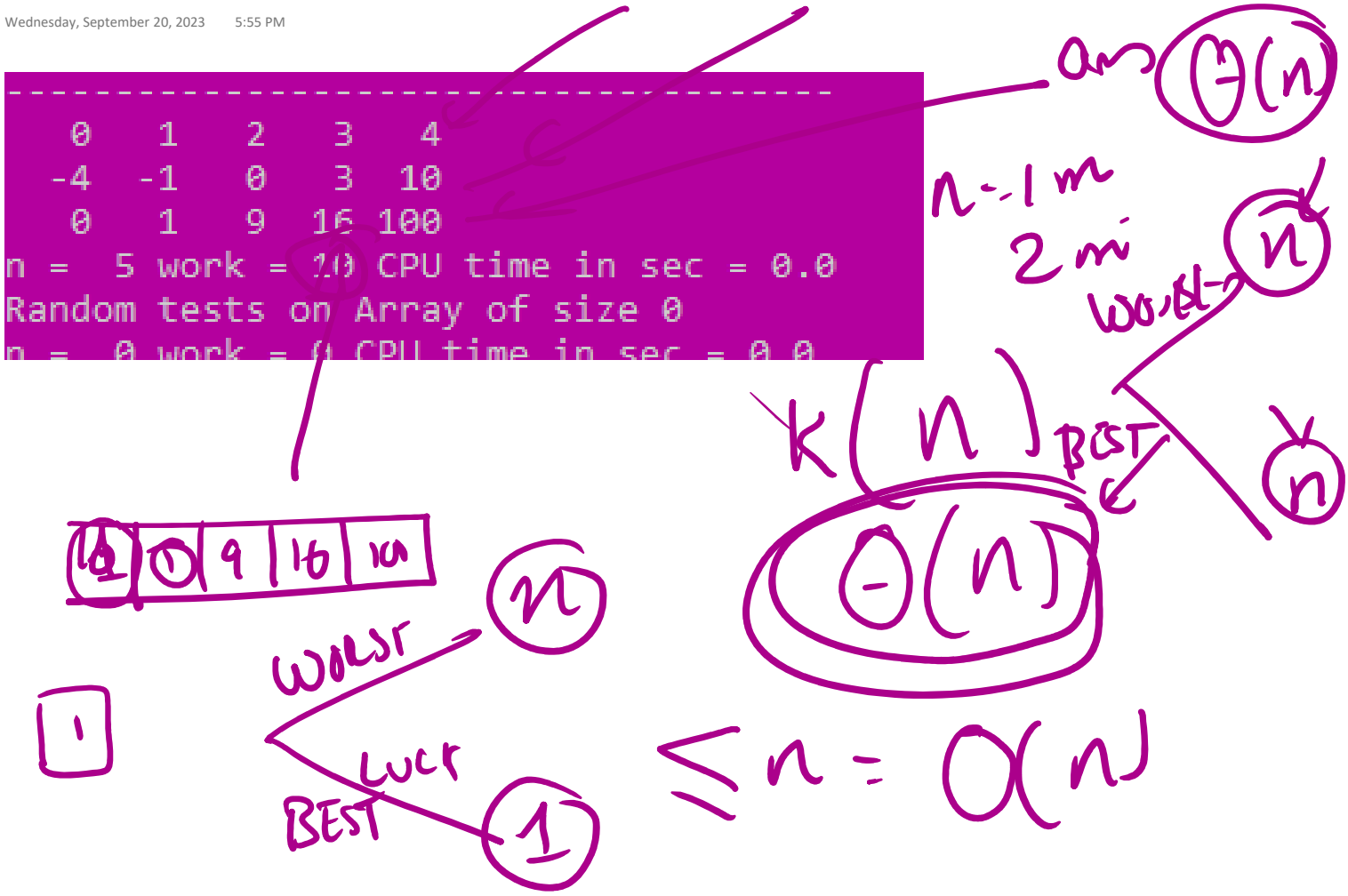
```
"""
def _alg(self):
    n = self._h.size();
    if (n > 0):
        for i in range(0,n):
            self._ans.append(0) #THETA(N) Space
        i = 0 ;
        j = n - 1;
        k = n - 1;
        while (i <= j):
            x = self._h.get(i)
            i2 = x * x
            y = self._h.get(j)
            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)
```



10

Wednesday, September 20, 2023 5:55 PM

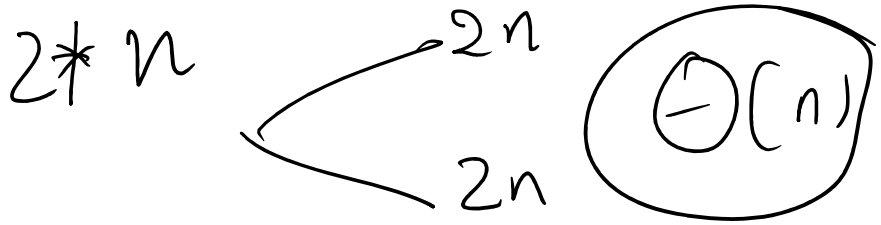
```
-----------------------------------------
   0   1   2   3   4
  -4  -1   0   3  10
   0   1   9  16 100
n =  5 work = 10 CPU time in sec = 0.0
Random tests on Array of size 0
n =  0 work = 0 CPU time in sec = 0.0
```



11

Wednesday, September 20, 2023 6:00 PM

```
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
```



Wednesday, September 20, 2023 6:00 PM

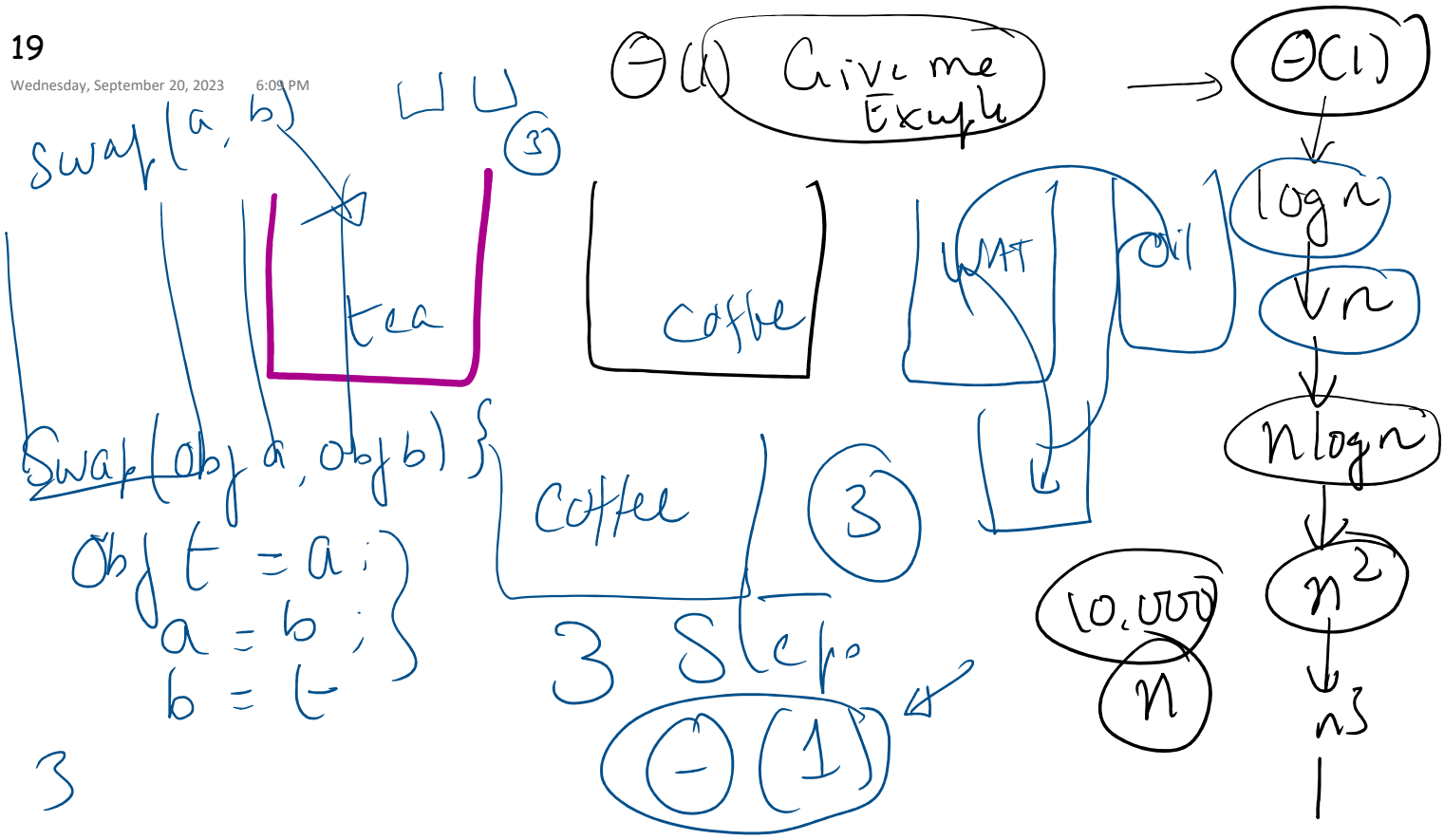
```
def _alg(self):
    n = self._h.size();
    if (n > 0):
        for i in range(0,n):
            self._ans.append(0) #THETA(N) Space
        i = 0 ;
        j = n - 1;
        k = n - 1;
        while (i <= j):
            x = self._h.get(i)
            i2 = x * x
            y = self._h.get(j)
            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)
```

13

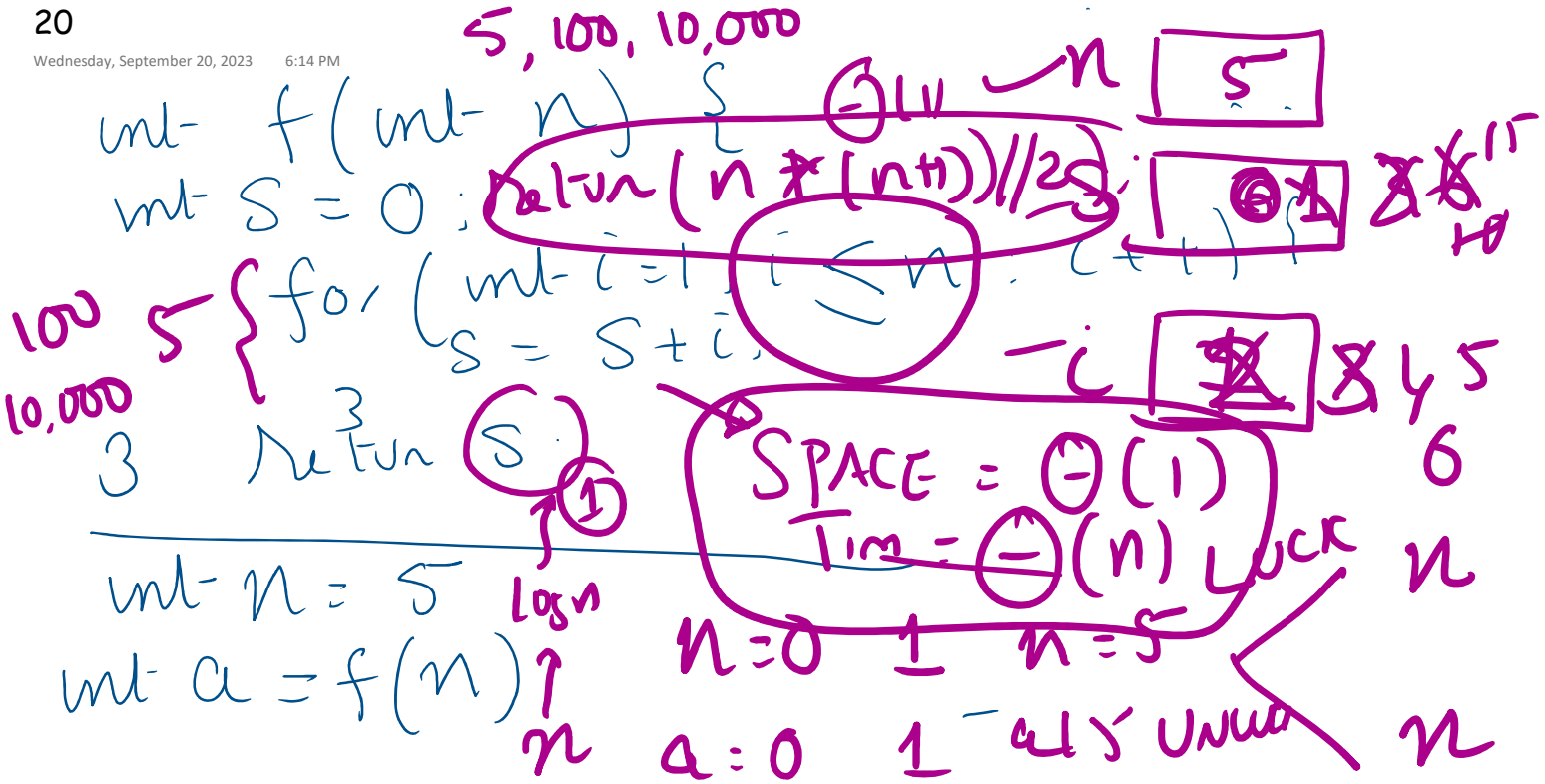
Wednesday, September 20, 2023 6:06 PM

```
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
```

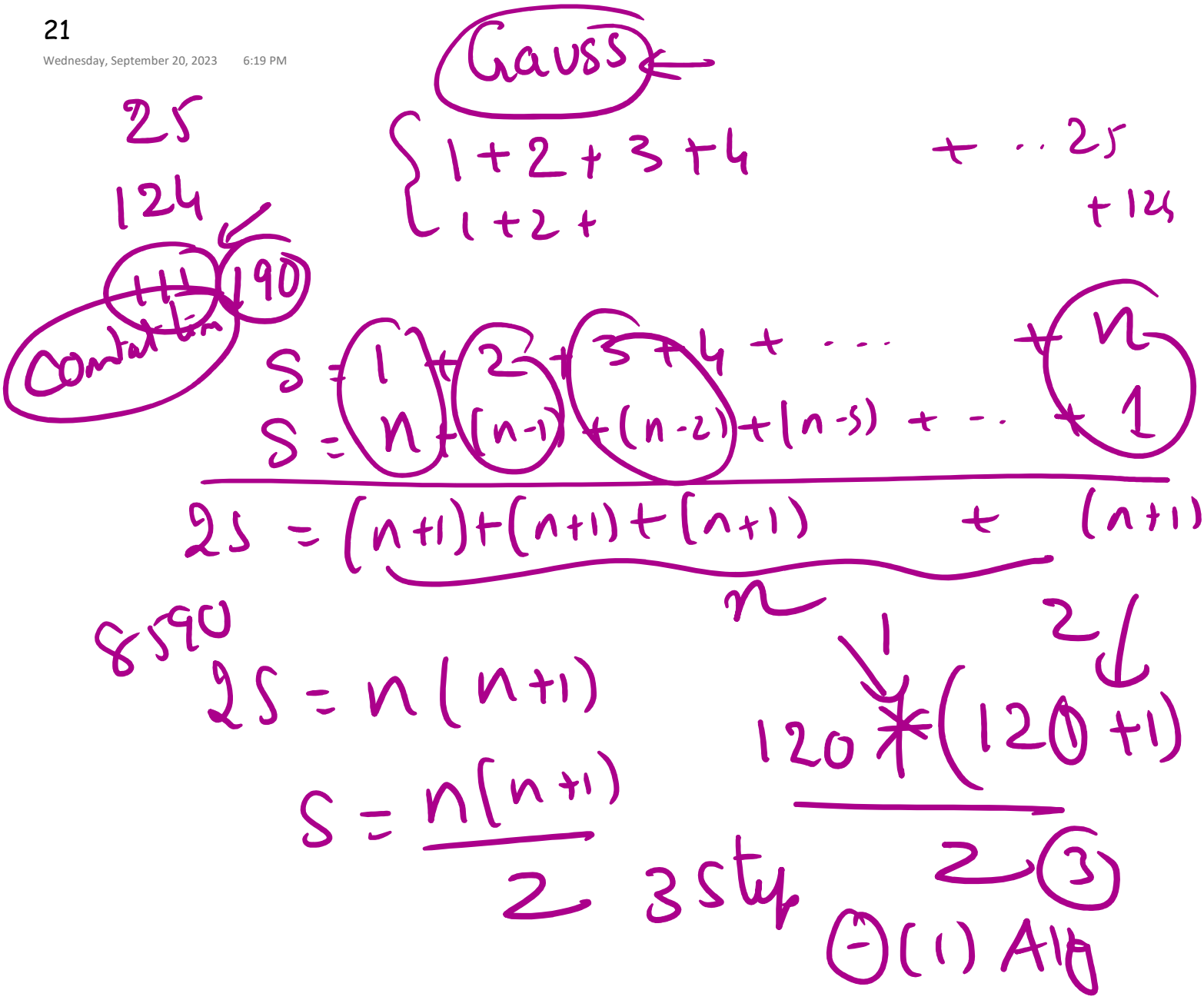

swap(a, b)
tea
3
Swap(Obj a, Obj b) {
Obj t = a;
a = b;
b = t
3
coffee
3
3 Steps
O(1)
Give me Example
O(1)
log n
n
n log n
n²
n3
1
MAT
Oil
10,000
n

Wednesday, September 20, 2023 6:14 PM



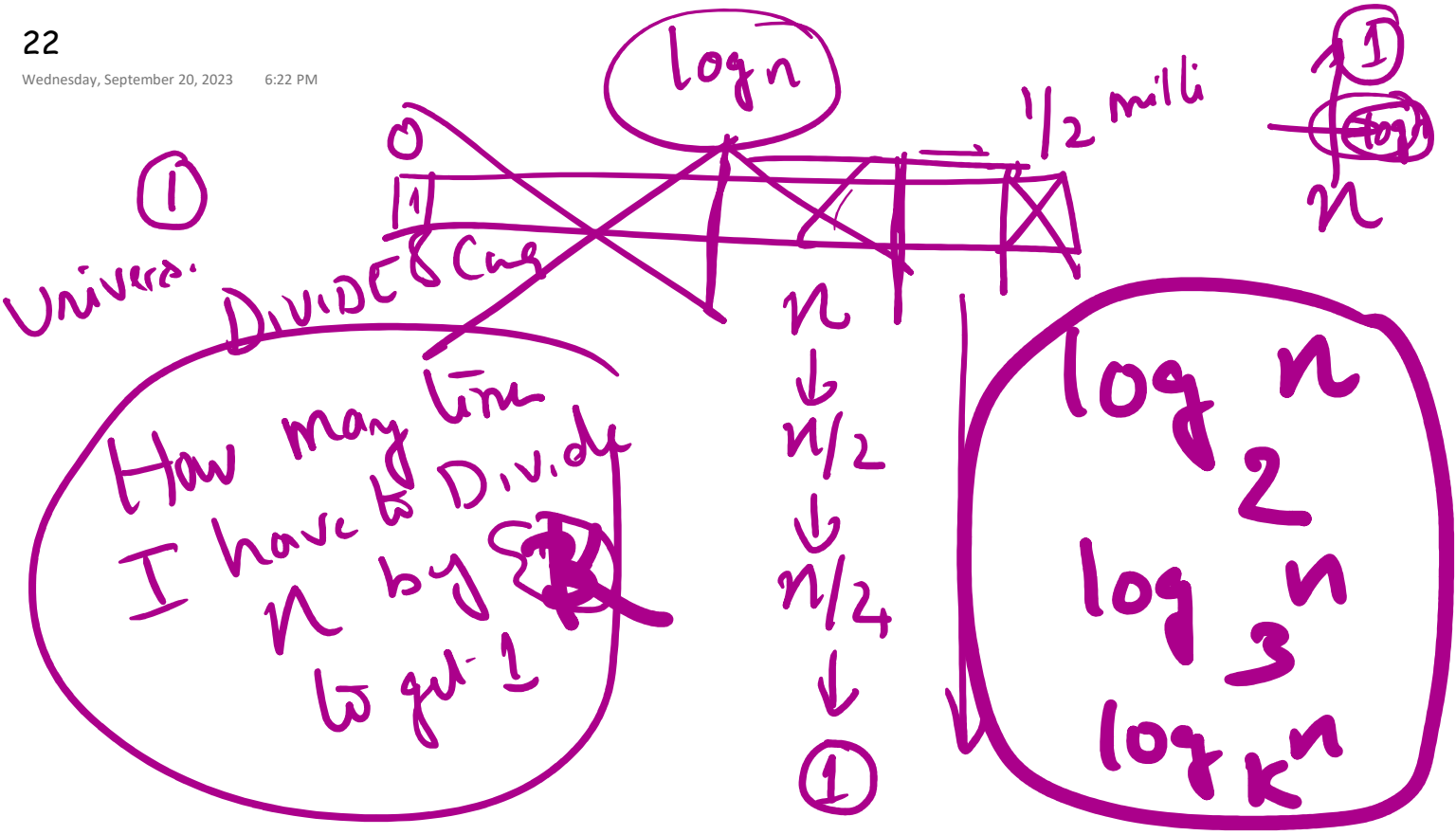
21

Wednesday, September 20, 2023 6:19 PM

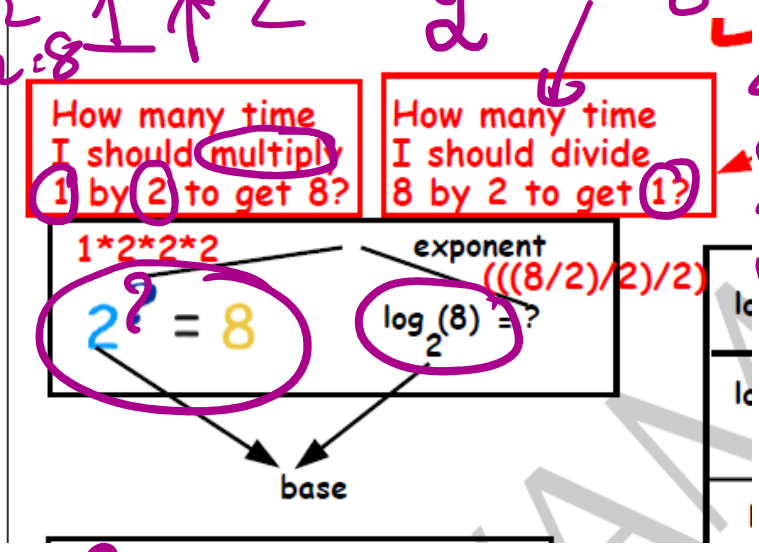


22

Wednesday, September 20, 2023 6:22 PM



Wednesday, September 20, 2023 6:26 PM



Wednesday, September 20, 2023 6:30 PM

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?

$1 * 2 * 2 * 2$ — exponent (((8/2)/2)/2)

$2^? = 8$ $\quad \log_2(8) = ?$

base

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

$\log_e 7.389 == \ln(7.389) \sim= 2$ Math

e = Euler Number = 2.71828

$\log_2 16 == \lg(16) = 4$ lg CS

$5^? = 625$ $\quad \log_5(625) = ?$

$5*5*5*5 = 625$

$\log_5(625) = 4$

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

$2^0 = 1$ $\quad \log_2 1 = 0$

$2^1 = 2$ $\quad \log_2 2 = 1$

what is $\log_2 64$ $\quad 2^8 = 64$

$\log_2 64 = 8$

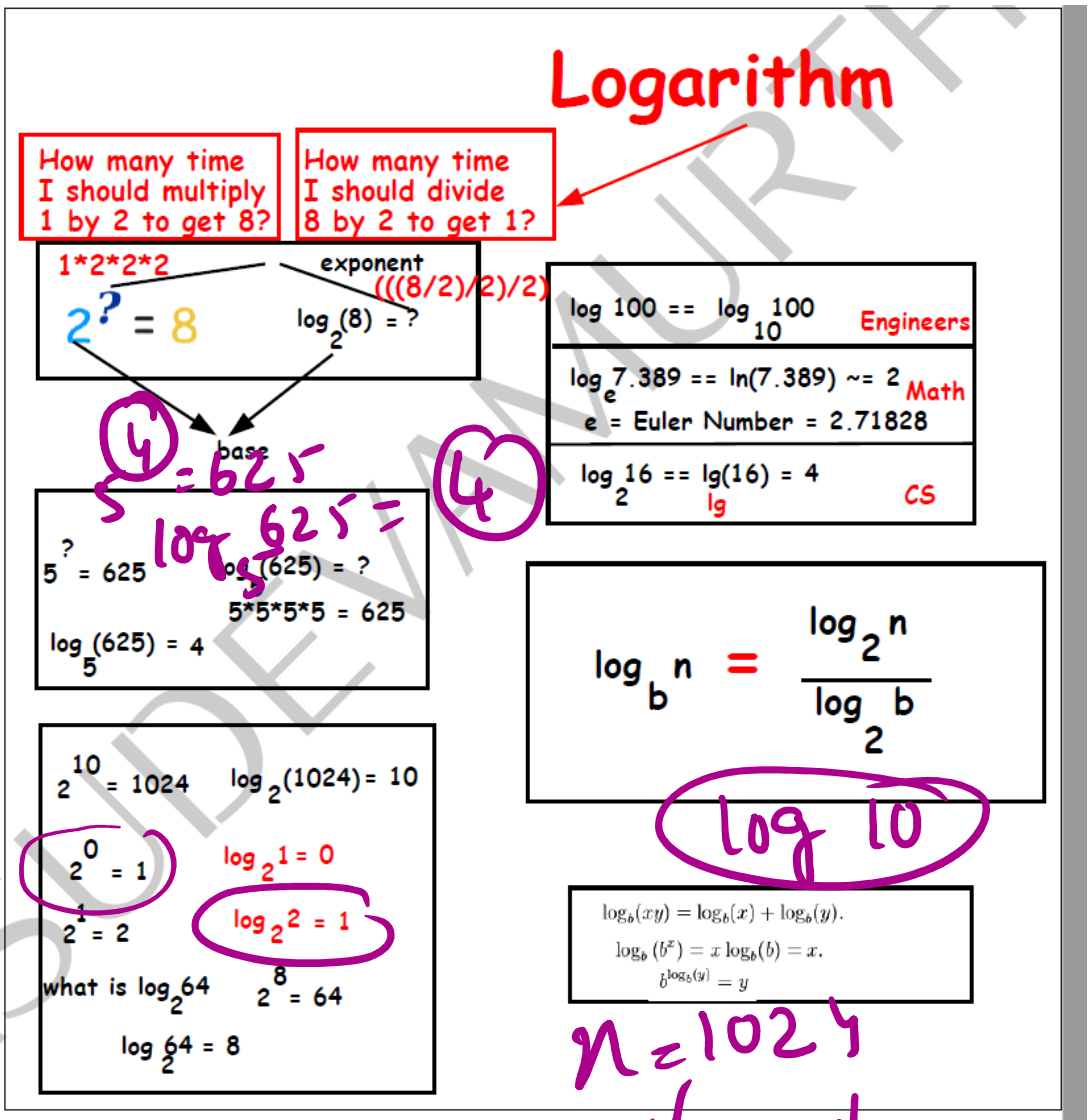
$$\log_b n = \frac{\log_2 n}{\log_2 b}$$

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

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

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

Wednesday, September 20, 2023 6:34 PM



Wednesday, September 20, 2023 6:36 PM

Bina Search Tr

n

$\log_2 n = 1$

$\log_k n = \frac{\log_2 n}{\log_2 k}$

$\log_k n = 1$ $\log_3 n = 1$

$\log_3 n = k \log_2 n$

$\log_3 n = 1$

$\frac{\log_2 3}{\log_2}$

Wednesday, September 20, 2023 6:39 PM

Table of base 10, base 2 and base e (ln) logarithms:

x	$\log_{10} x$	$\log_2 x$	$\log_e x$
1	0.000000	0.000000	0.000000
2	0.301030	1.000000	0.693147
3	0.477121	1.584963	1.098612
4	0.602060	2.000000	1.386294
5	0.698970	2.321928	1.609438
6	0.778151	2.584963	1.791759
7	0.845098	2.807355	1.945910
8	0.903090	3.000000	2.079442
9	0.954243	3.169925	2.197225
10	1.000000	3.321928	2.302585
20	1.301030	4.321928	2.995732
30	1.477121	4.906891	3.401197
40	1.602060	5.321928	3.688879
50	1.698970	5.643856	3.912023
60	1.778151	5.906991	4.094345
70	1.845098	6.129283	4.248495
80	1.903090	6.321928	4.382027
90	1.954243	6.491853	4.499810
100	2.000000	6.643856	4.605170
200	2.301030	7.643856	5.298317
300	2.477121	8.228819	5.703782
400	2.602060	8.643856	5.991465
500	2.698970	8.965784	6.214608
600	2.778151	9.228819	6.396930
700	2.845098	9.451211	6.551080
800	2.903090	9.643856	6.684612
900	2.954243	9.813781	6.802395
1000	3.000000	9.965784	6.907755
10000	4.000000	13.287712	9.210340

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

$O(\log n)$

Independent

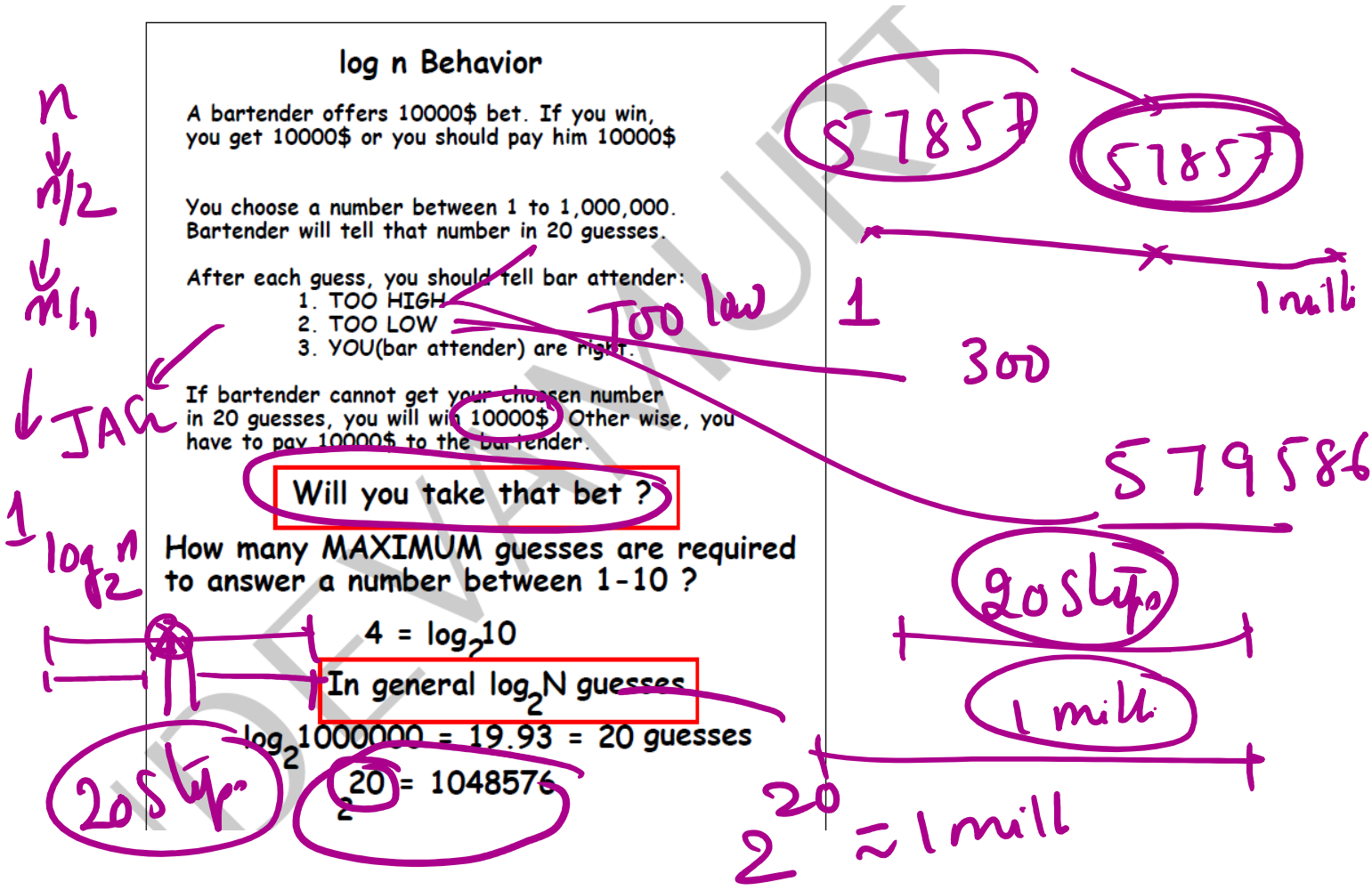
$O(\log n)$

Base does not matter

increases by constant amount independent of n

$\log_k n = \frac{\log_2 n}{\log_2 k}$

Wednesday, September 20, 2023 6:41 PM

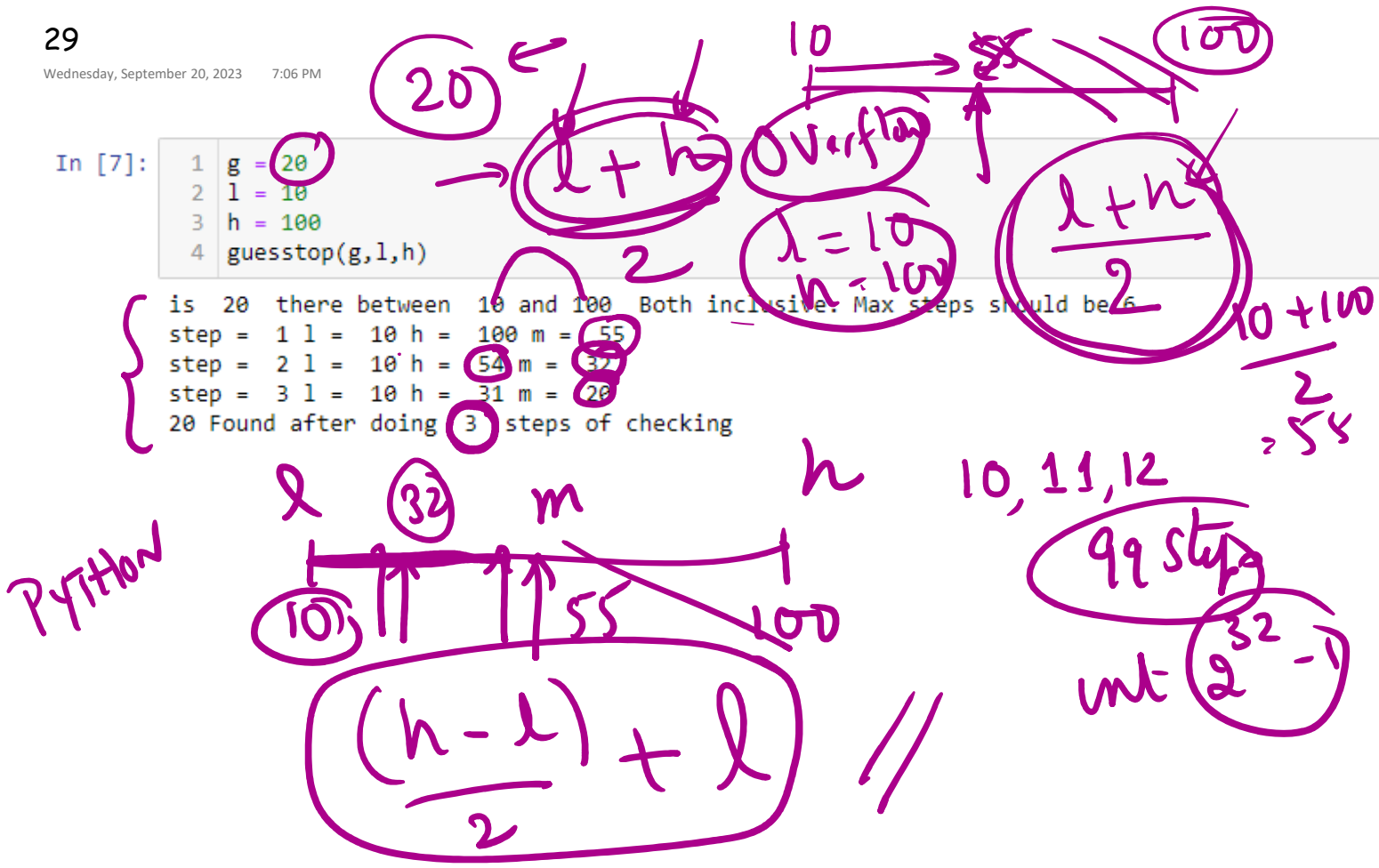


29

Wednesday, September 20, 2023 7:06 PM

In [7]:
```
g = 20
l = 10
h = 100
guesstop(g,l,h)
```

```
is  20  there between  10 and 100  Both inclusive. Max steps should be 6
step =  1 l =  10 h =  100 m =  55
step =  2 l =  10 h =  54 m =  32
step =  3 l =  10 h =  31 m =  20
20 Found after doing  3  steps of checking
```



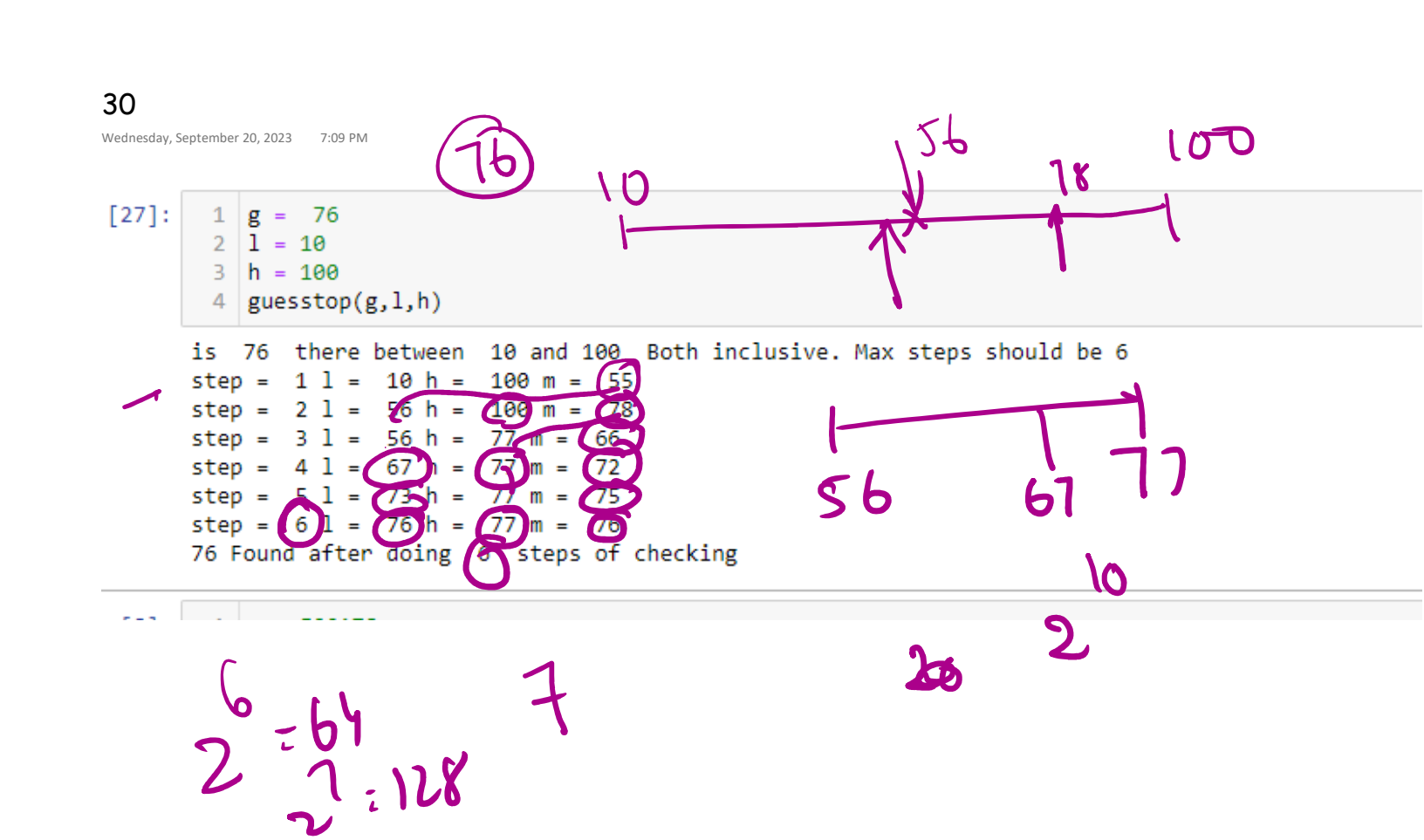
30

Wednesday, September 20, 2023 7:09 PM

[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
```



31

Wednesday, September 20, 2023 7:11 PM

In [28]:
```
g = 500178
l = 1
h = 1000000
guesstop(g,l,h)
```

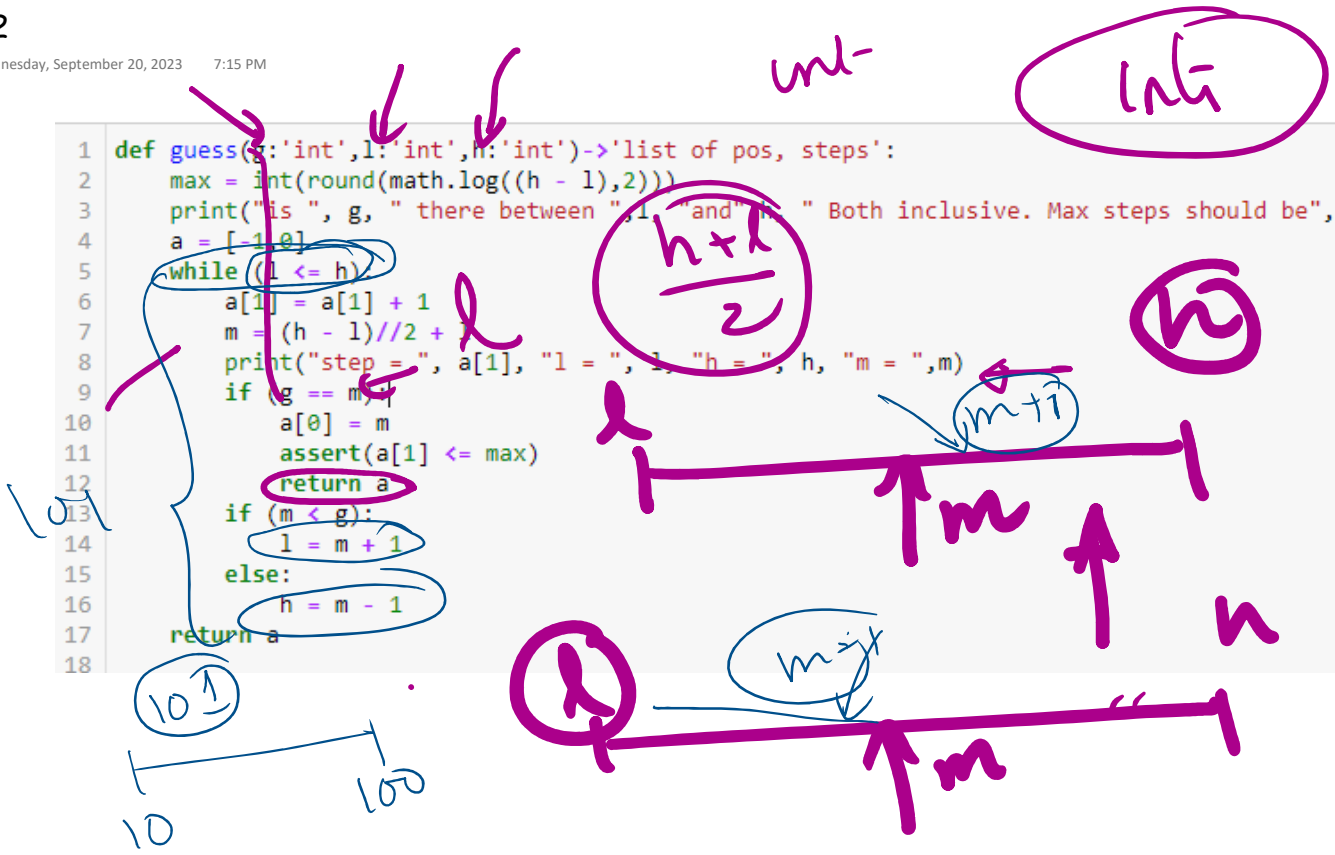
```
is  500178  there between  1 and 1000000  Both inclusive. Max steps should be 20
step =  1 l =  1 h =  1000000 m =  500000
step =  2 l =  500001 h =  1000000 m =  750000
step =  3 l =  500001 h =  749999 m =  625000
step =  4 l =  500001 h =  624999 m =  562500
step =  5 l =  500001 h =  562499 m =  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 [0]:
```
g = 101
```

32

Wednesday, September 20, 2023 7:15 PM

```python
def guess(g:'int',l:'int',h:'int')->'list of pos, steps':
    max = int(round(math.log((h - l),2)))
    print("is ", g, " there between ",l, "and",h, " Both inclusive. Max steps should be",max)
    a = [-1,0]
    while (l <= h):
        a[1] = a[1] + 1
        m = (h - l)//2 + l
        print("step = ", a[1], "l = ", l, "h = ", h, "m = ",m)
        if (g == m):
            a[0] = m
            assert(a[1] <= max)
            return a
        if (m < g):
            l = m + 1
        else:
            h = m - 1
    return a
```

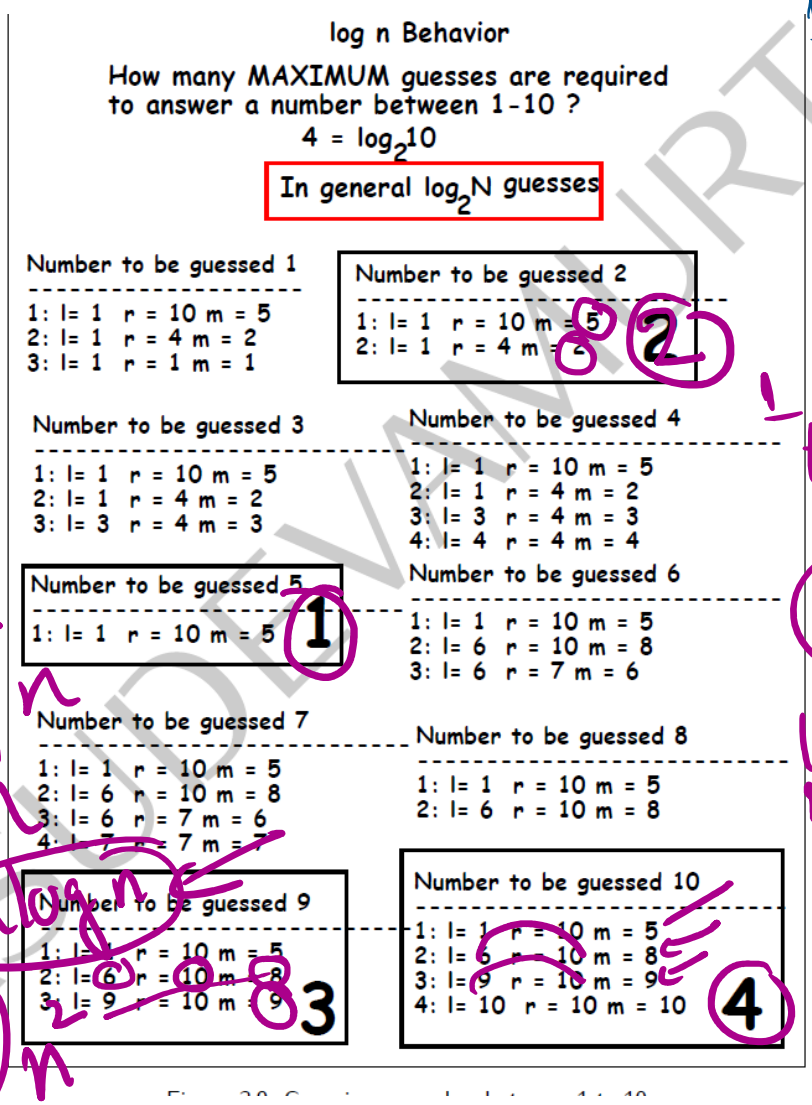


33

Wednesday, September 20, 2023 7:19 PM

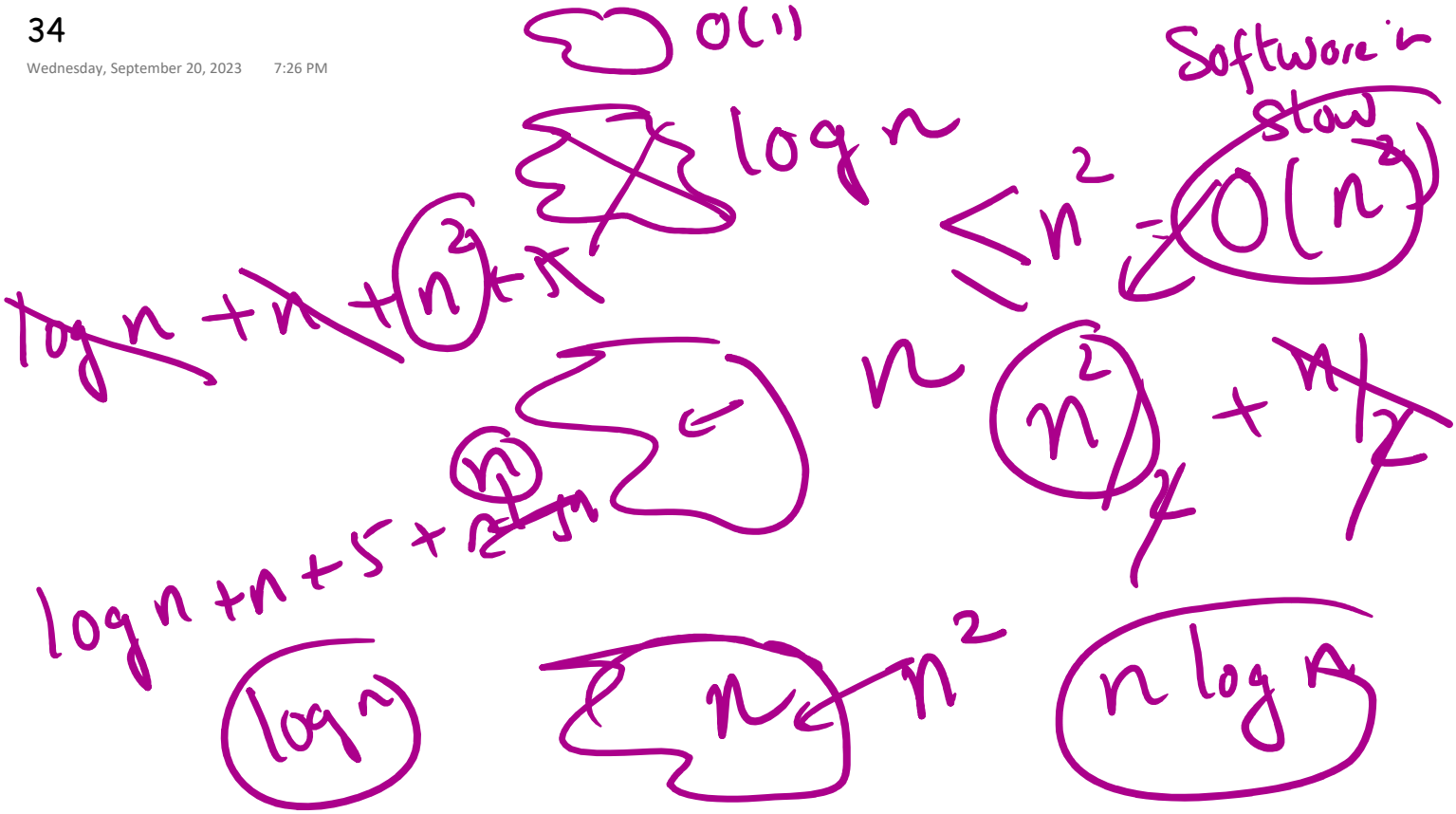
[31]:
```
g = 101
l = 1
h = 100
guesstop(g,l,h)
```

```
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
```

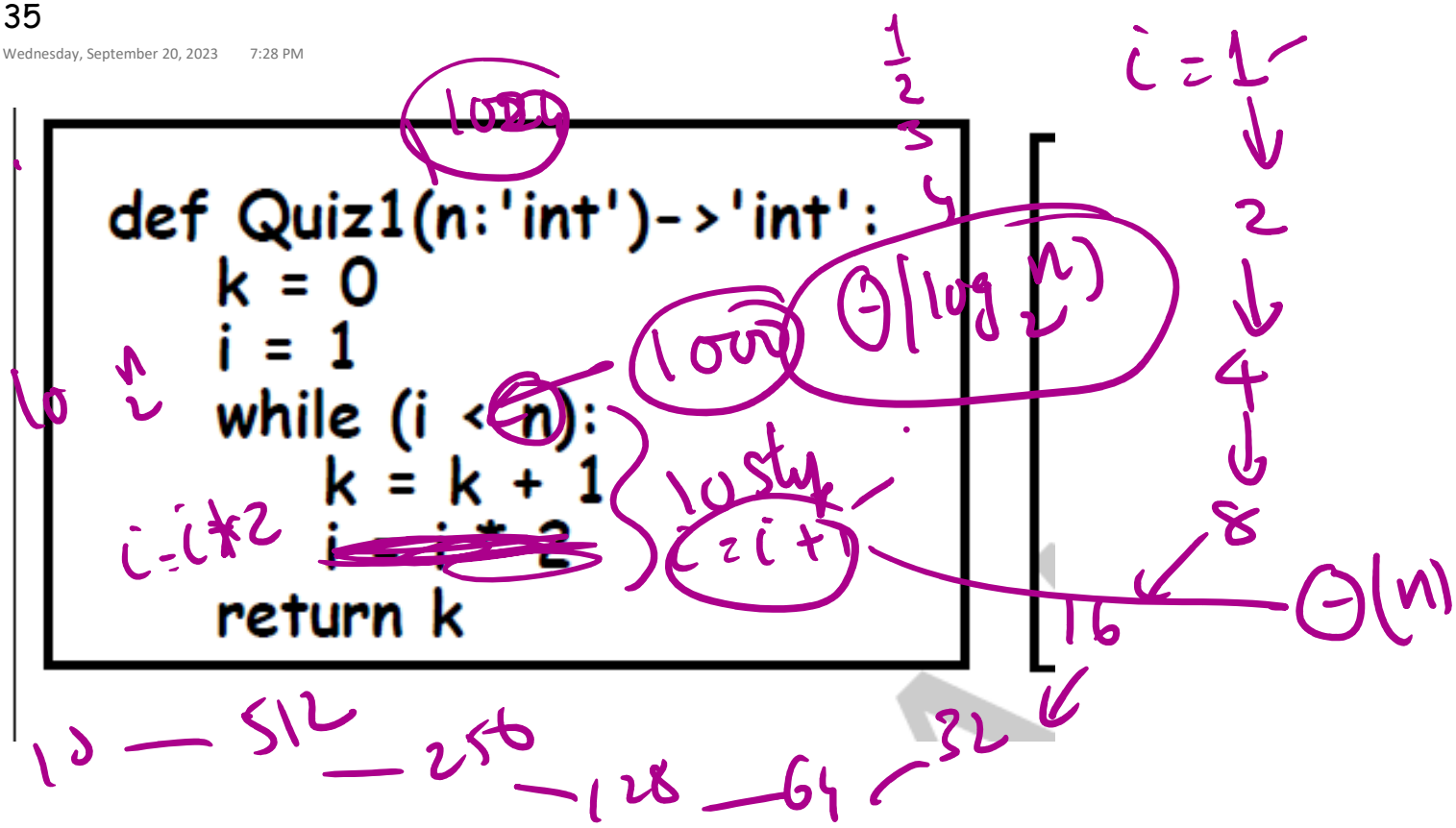


34

Wednesday, September 20, 2023 7:26 PM



Wednesday, September 20, 2023 7:28 PM



Wednesday, September 20, 2023 7:30 PM

n = 1024

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

1024

log n

n = n - 1

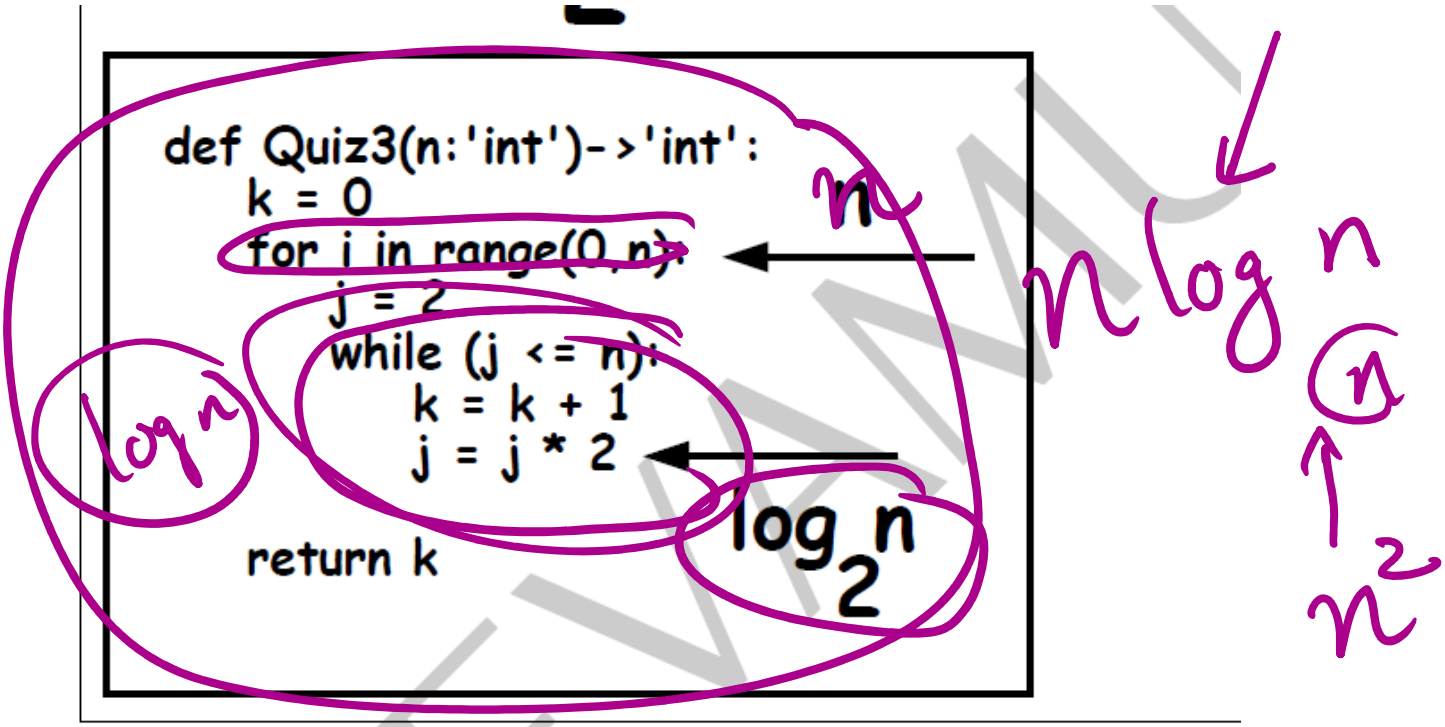
1024 → 512 → 256 → 128 → 64 ← 32 ← 16 ← 8 ← 4 ← 2 ← 1

log 1024

20 steps

37

Wednesday, September 20, 2023 7:32 PM



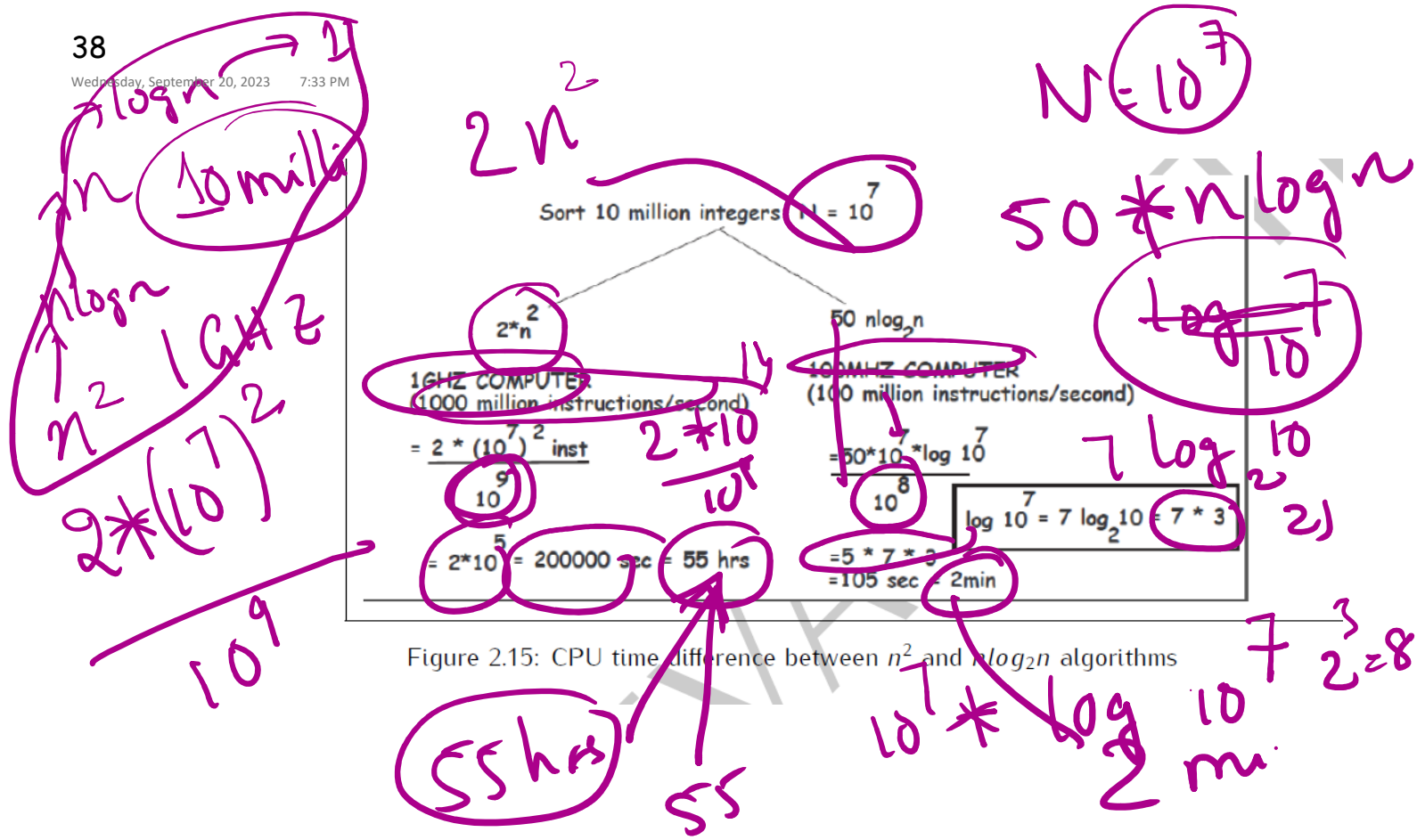


Figure 2.15: CPU time difference between n^2 and $n log_2 n$ algorithms

39

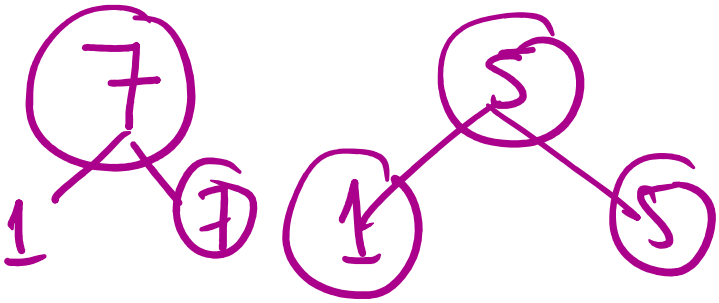
Wednesday, September 20, 2023 7:36 PM

1 is NOT A

The prime numbers until 25 are
1 2 3 4 5 6 7 8 9
2, 3, 5, 7, 11, 13, 17, 19, 23

#(P25) = 9

25



n = 25

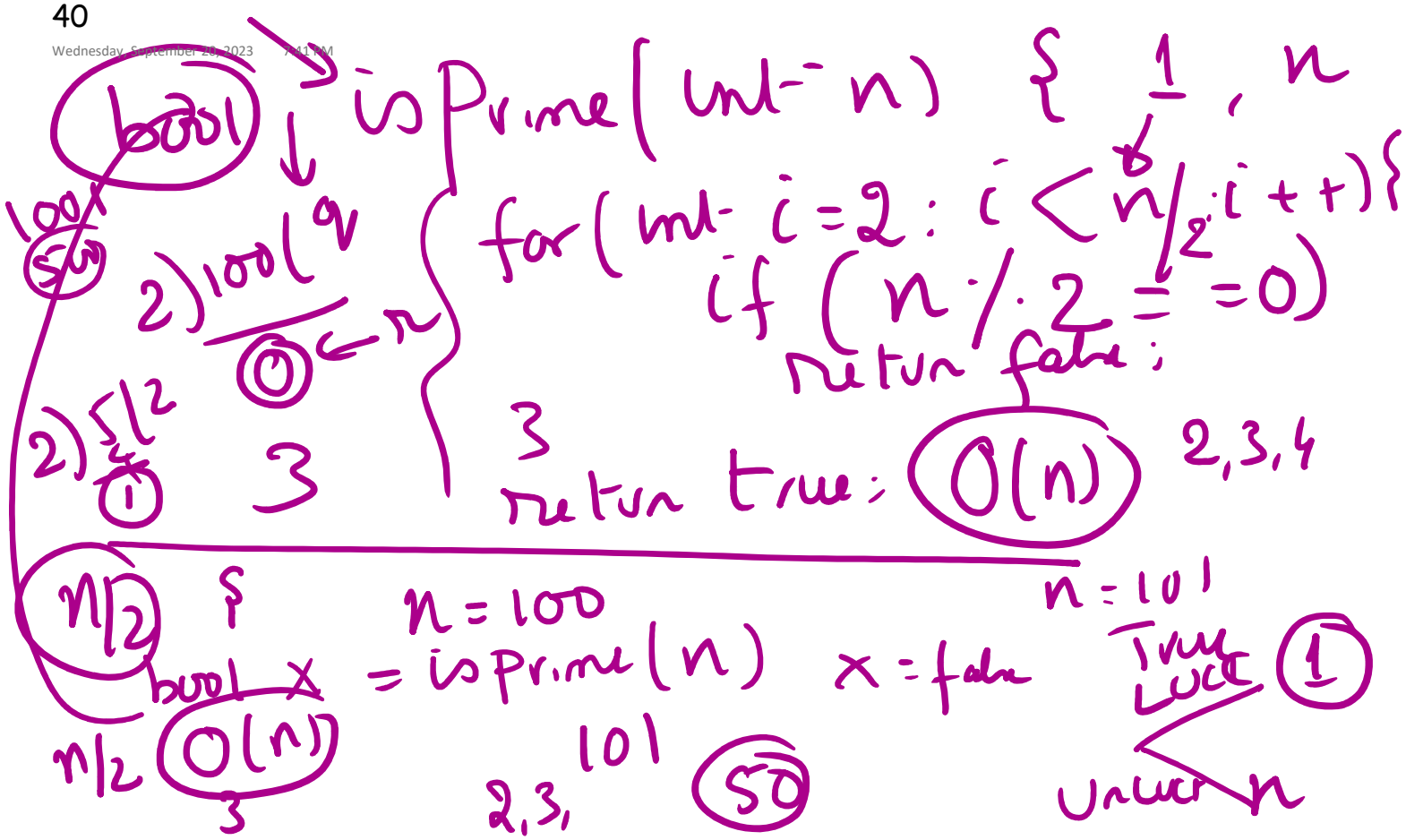
n = 1000
n = 1 m

1, 5

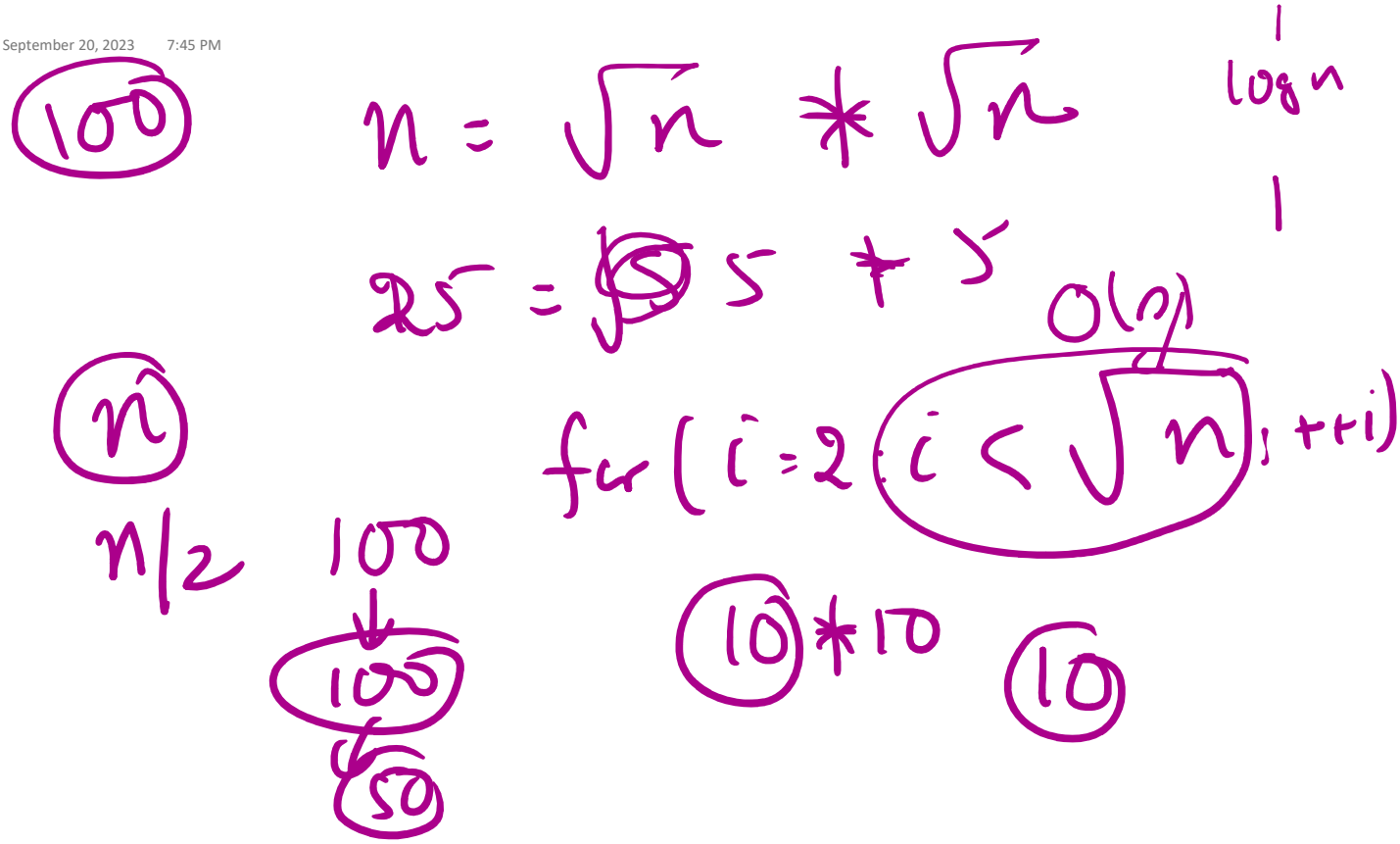
2

2, 3, 5 7 9 11 13 17
19 23

Wednesday, September 20, 2023 7:11 PM



Wednesday, September 20, 2023 7:45 PM

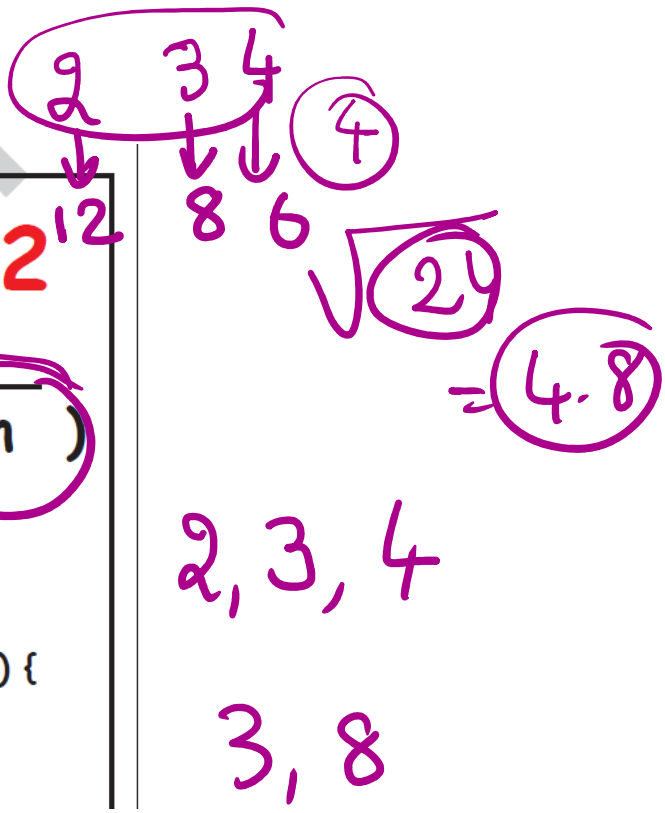


42

Wednesday, September 20, 2023 7:47 PM

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 <= SQRT(n)

   n  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 ;
     }
   }
```



Wednesday, September 20, 2023 7:48 PM

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

   n  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 $\sqrt{n}$)

100

10 90 50

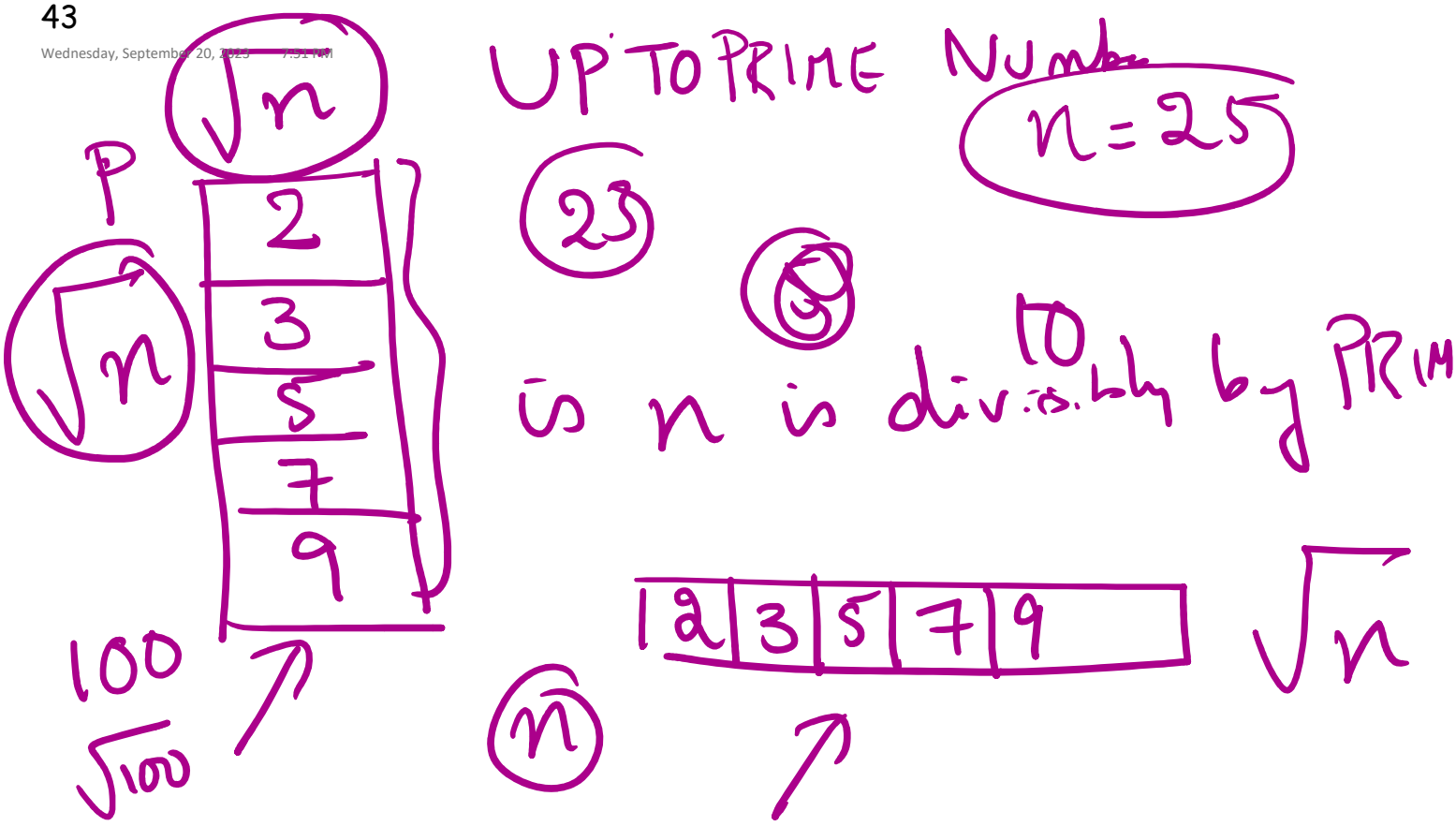
25

0 1 2 3 4 5

2, 3, 5, 7, 11, 13, 17, 19, 23

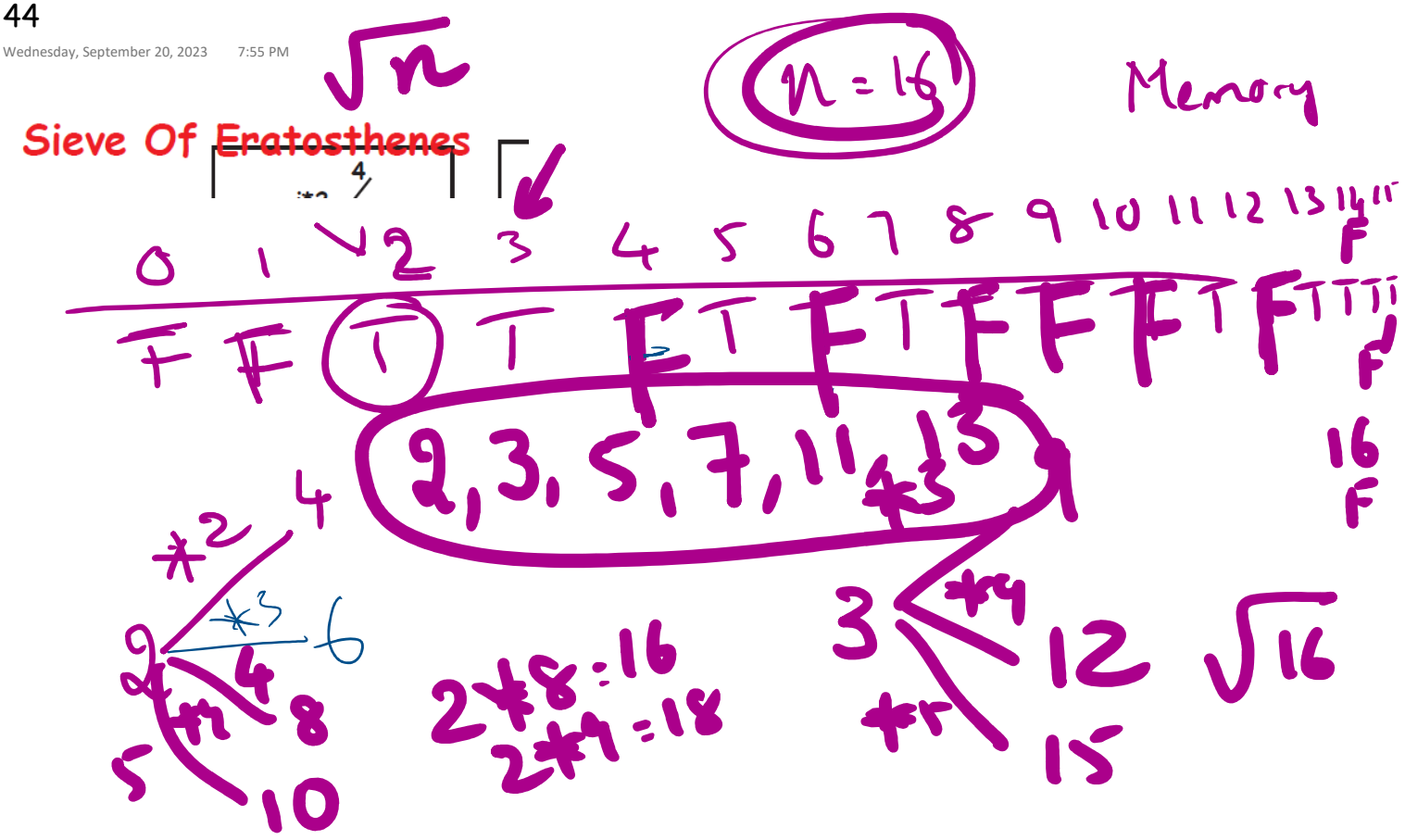
6 7 8

n-8


UP TO PRIME Number
n = 25
√n
P
2
3
5
7
9
√n
100
√100
23
n
1 2 3 5 7 9
√n
is n is divisible by PRIME

Wednesday, September 20, 2023 7:55 PM

Sieve Of Eratosthenes



45

Wednesday, September 20, 2023 8:01 PM

```
##############################################################

class Solution:
    def __init__(self,n:'int',prime_number_list:'list')->None:
        self._n = n
        self._prime_number_list = prime_number_list #You need to append this using _append_prime_number
        self._work_done = 0 #You need to increment using _increment_work_done()
        ##YOU CANNOT have anything here

    ############################################################
    # All work done goes through this routine.
    # Nothing can be changed
    ############################################################
    def _increment_work_done(self):
        self._work_done = self._work_done + 1

    ############################################################
    # All prime numbers are appended to the list.
    # Nothing can be changed
    ############################################################
    def _append_prime_number(self,p):
        self._prime_number_list.append(p)
```

46

Wednesday, September 20, 2023 8:04 PM

```
27
28      ##Required function to implement
29      def nsquare(self)->'int':
30          ## NOTHING CAN BE CHANGED HERE
31          ## YOU CANNOT CALL math.log from Python library
32          self._nsquare()
33          return self._work_done
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
```