

DAAPY 6

Wednesday, October 11, 2023 4:32 PM


JV Jagadeesh Vasudevamurthy (Host, me)

AN Anirudh Negi

JZ Jiading Zhou

JY Jingtian Yin


JC Jingyi Chen

 Mei Yin Ho

NW Ning Wang

SC Shrushti Chahande

SC Shrushti Chahande (Guest)

 Venni (cn: Wen Yu)

YJ Yang Jiang

 Yitong Wu

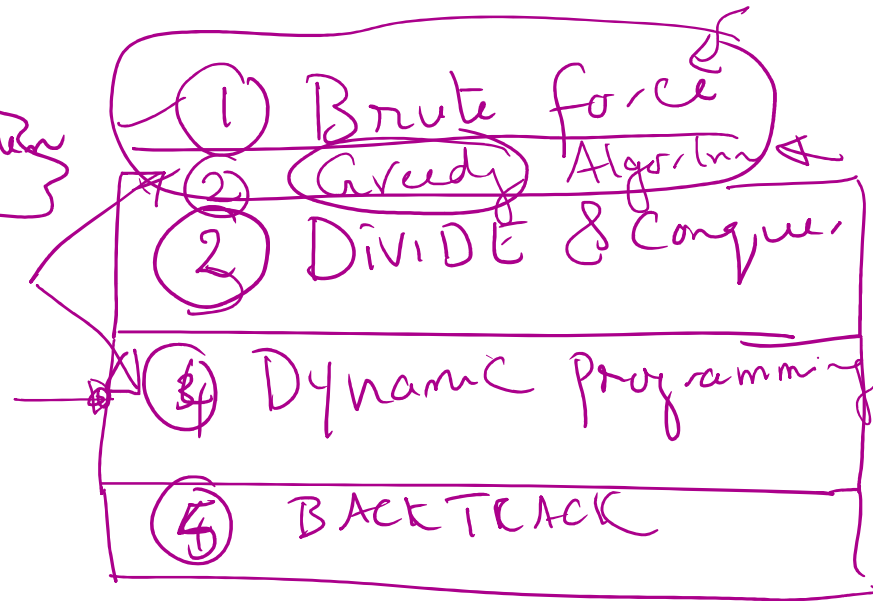
ZC Zhe Cao

HS Hongji Shi (Guest)

6:09



Problem



Lucky 16 Cents

1 \$
100
16
84 Cents

Greedy 8 Coins

Minimal number of Coins

USING

1 5 10 25

1 1 10 1

84 Coins

1 2 3 4

25 25 25 5

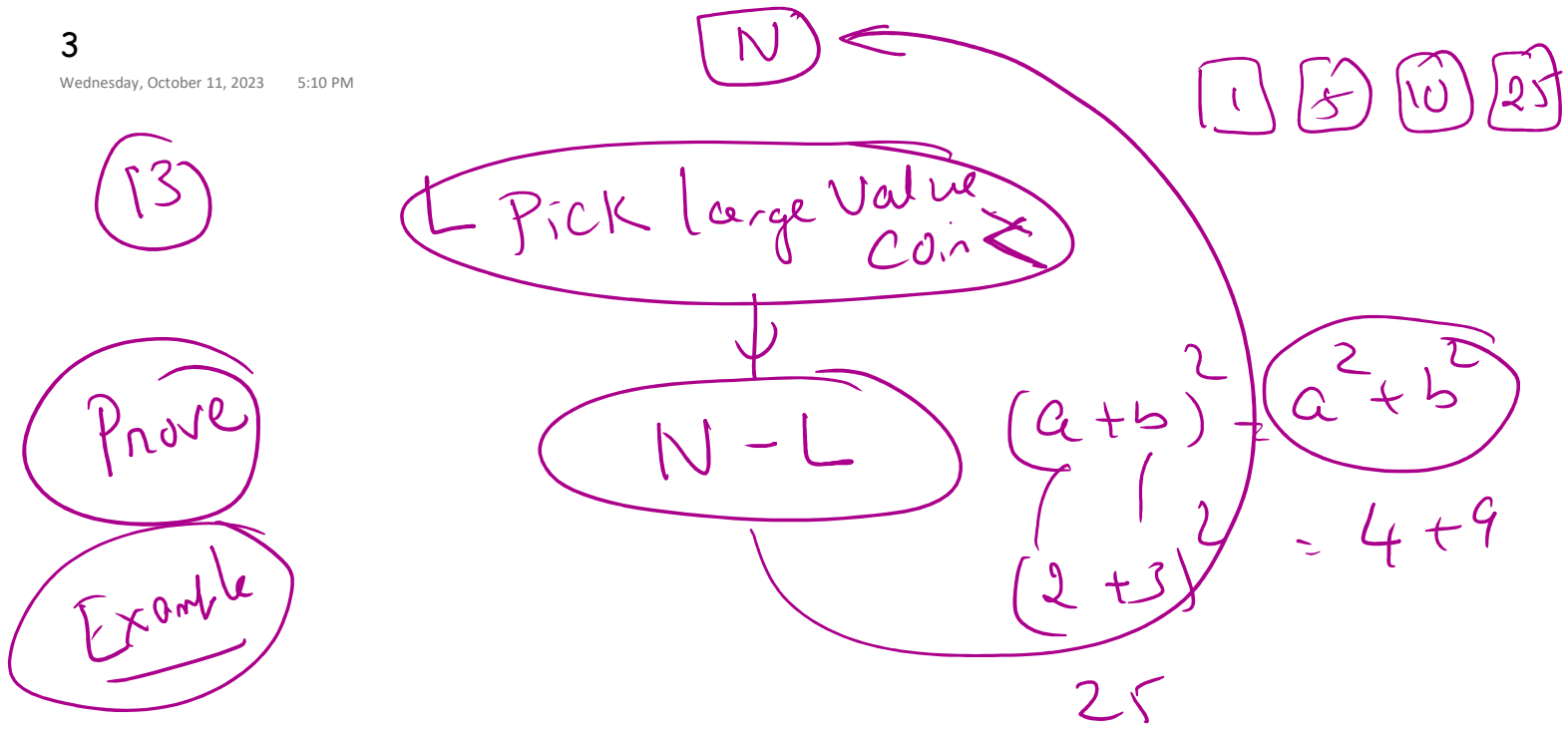
5 6 7 8

1 1 1 1

84
75
9

9 / 4

{25, 25, 25, 5, 1, 1, 1, 1}



16 Cents

16
10
5
1

16
12
4

12

10

5

1

DP

12 1 1 1 1

10 5 1

3

Greedy
Design

Coin change

Stamps

2 stamp

BIN
PACKING

As an example, suppose an instance of StampDispenser was created with stampDenominations, {90, 30, 24, 10, 6, 2, 1} and calcMinNumStampsToFillRequest (int) was called with request, 34. The call should return 2, as 34 cents can best be filled by one 24 cent stamp and one 10 cent stamp.

24 Cents

n

90 30



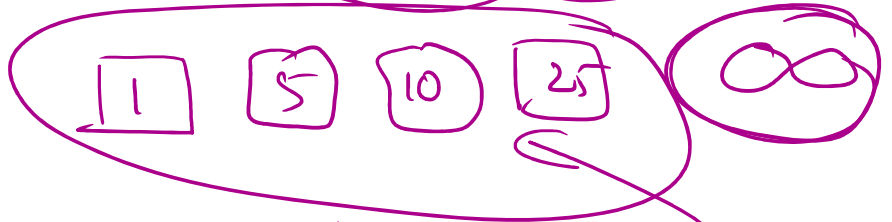
24 stamp

n = 9801

100 100 100

n = 121315612591281

12516 212

Coin
Stamp

K = 4

Stamp

100 100 100 n

n

Fibonacci number

$$F_0 = 1$$

$$F_1 = 1$$

n	0	1	2	3	4	5	6	7
Fib(n)	0	1	1	2	3	5	8	13

Fib(

Iteration

 $\Theta(n)$ 1 PrevPrev $\Theta(1)$ 1 Prev

```

int- F(int n) {
  if (n <= 2)
    return n

```

3 return (F(n-1) + F(n-2)) 4

3 { int n = 6
F(n)

5	a = F(3) b = F(2)
4	a = F(4) b = F(3)
3	a = F(5) b = F(4)

$$\Theta F(n) = (F(n-1) + F(n-2))$$

n

↓

$n-1$

↓

$n-2$

$\Theta(n)$

*1,2

*1,5

*1,4

5

$$F(0) = 0$$

$$F(1) = 1$$

Const'n

$\log n$

n

↓

$n/2$

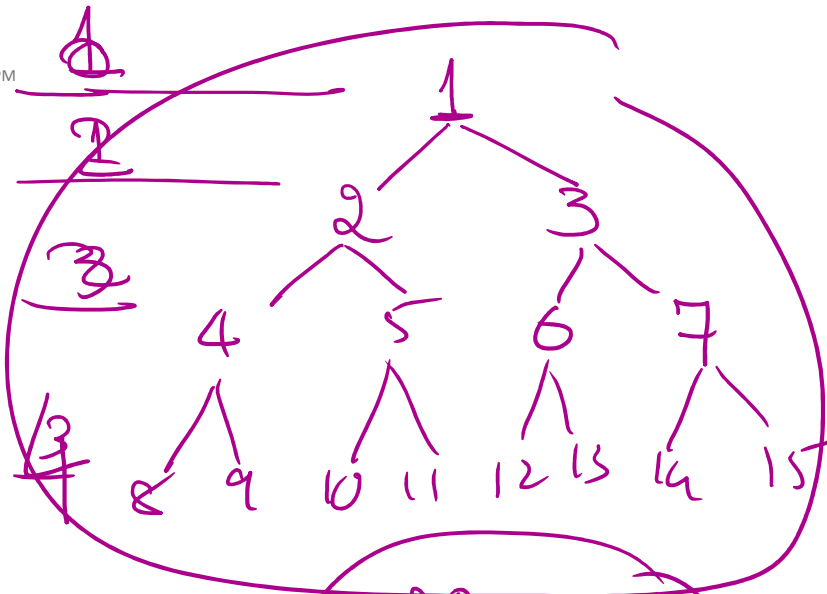
↓

$n/4$

↓

Binary tree

$$\Theta(2^n)$$



2020

$$4 - 1$$

$$20$$

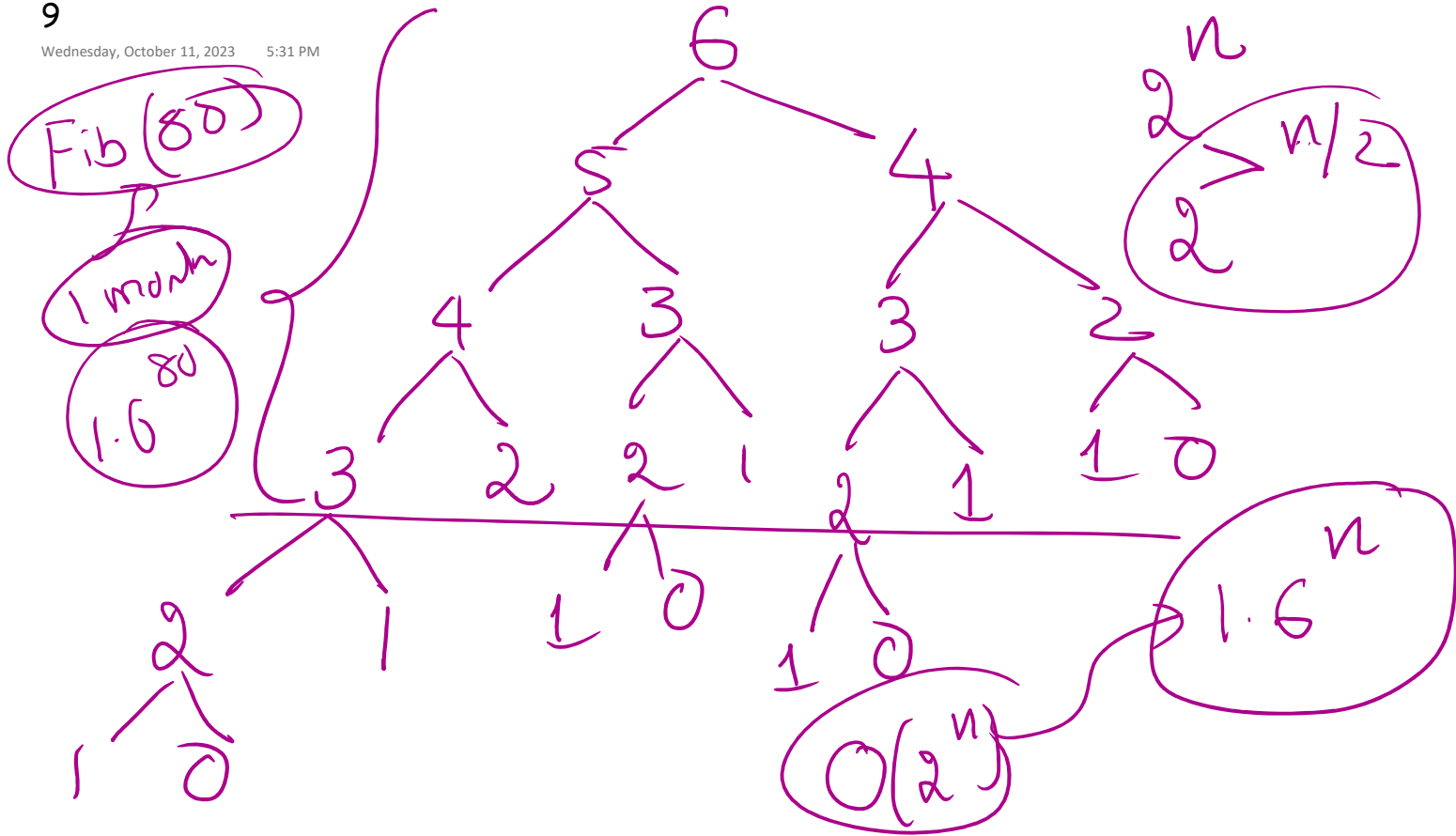
$$20 - 1$$

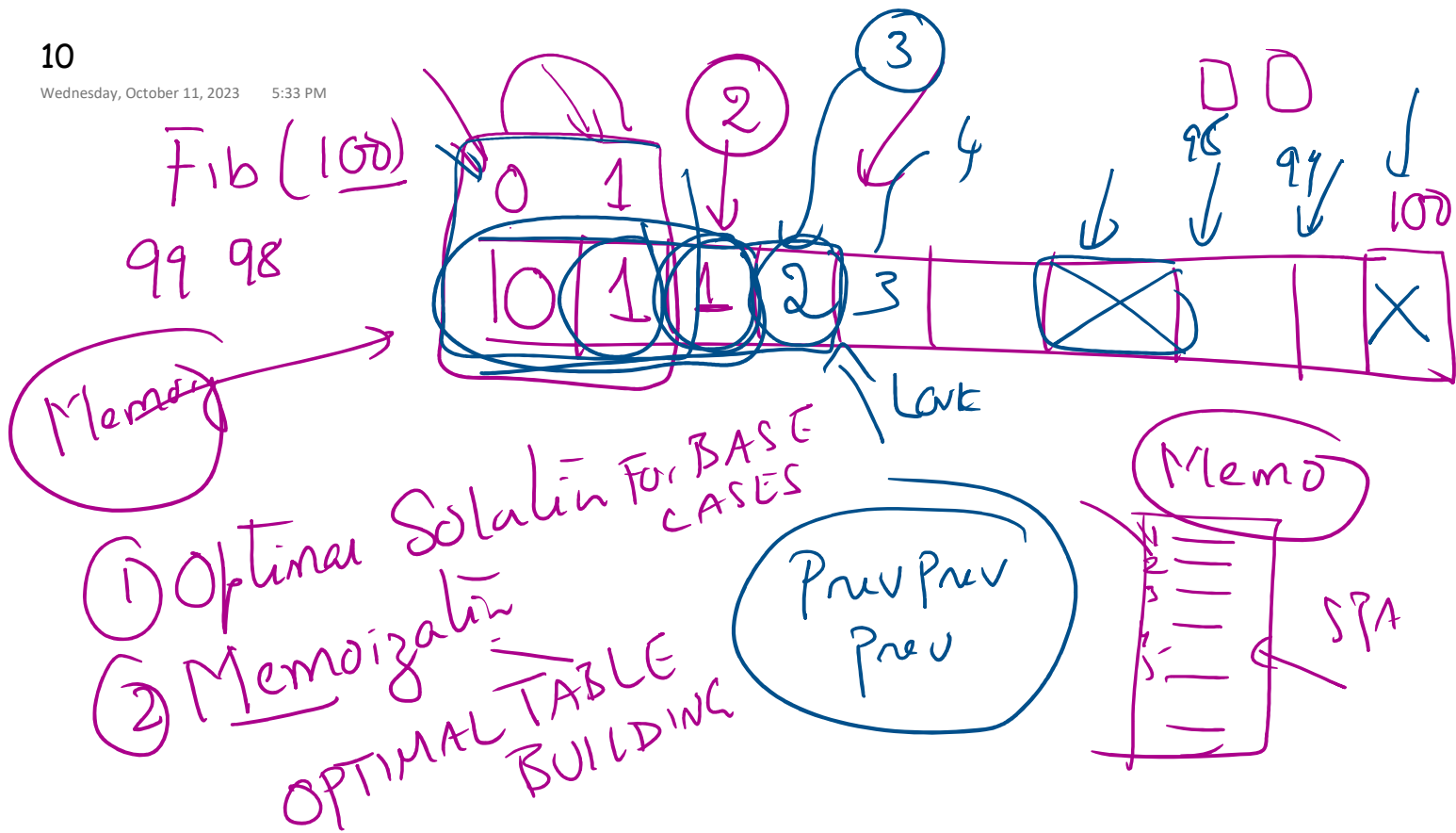
1 million

$$30$$

$$30 - 1$$

1 Billion





11

K

Wednesday, October 11, 2023 5:38 PM

6

1 3 4

1 1 1 0 0 0

1 4 1

2

3 3

$n = 1891$

90

$n=6$ $n=121$

(1) (3) (4)

 $2n$
 $\Theta(n)$ SPACE K-ARRAY

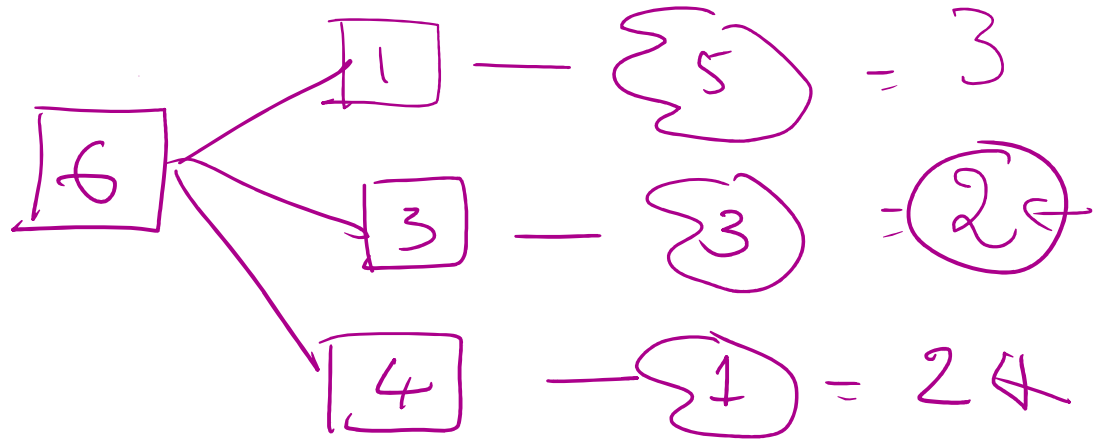
V-ARRAY

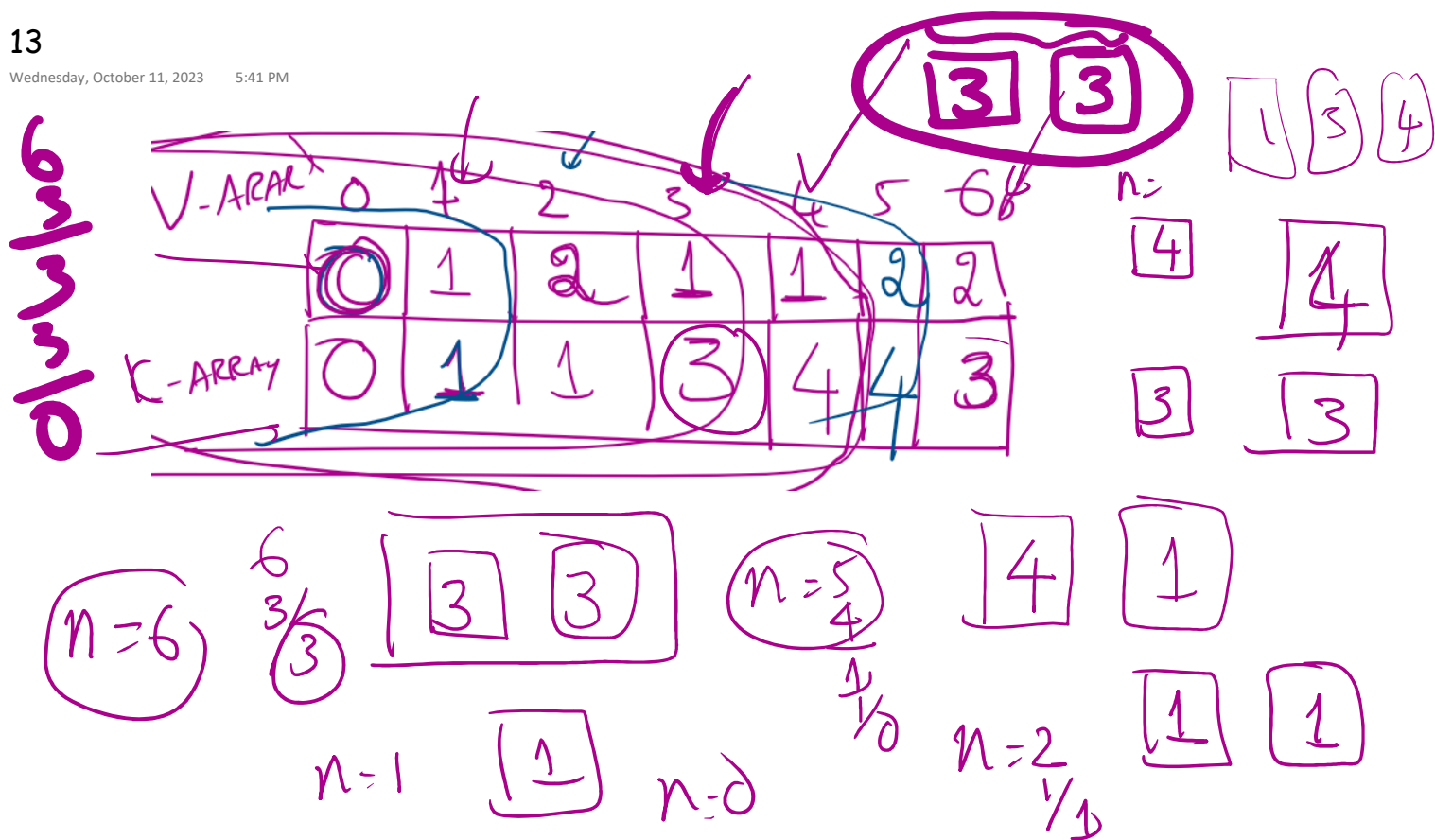
	0	1	2	3	4	5	6
0	0	1	2	1	1	2	2
1	0	1	1	3	4	4	3

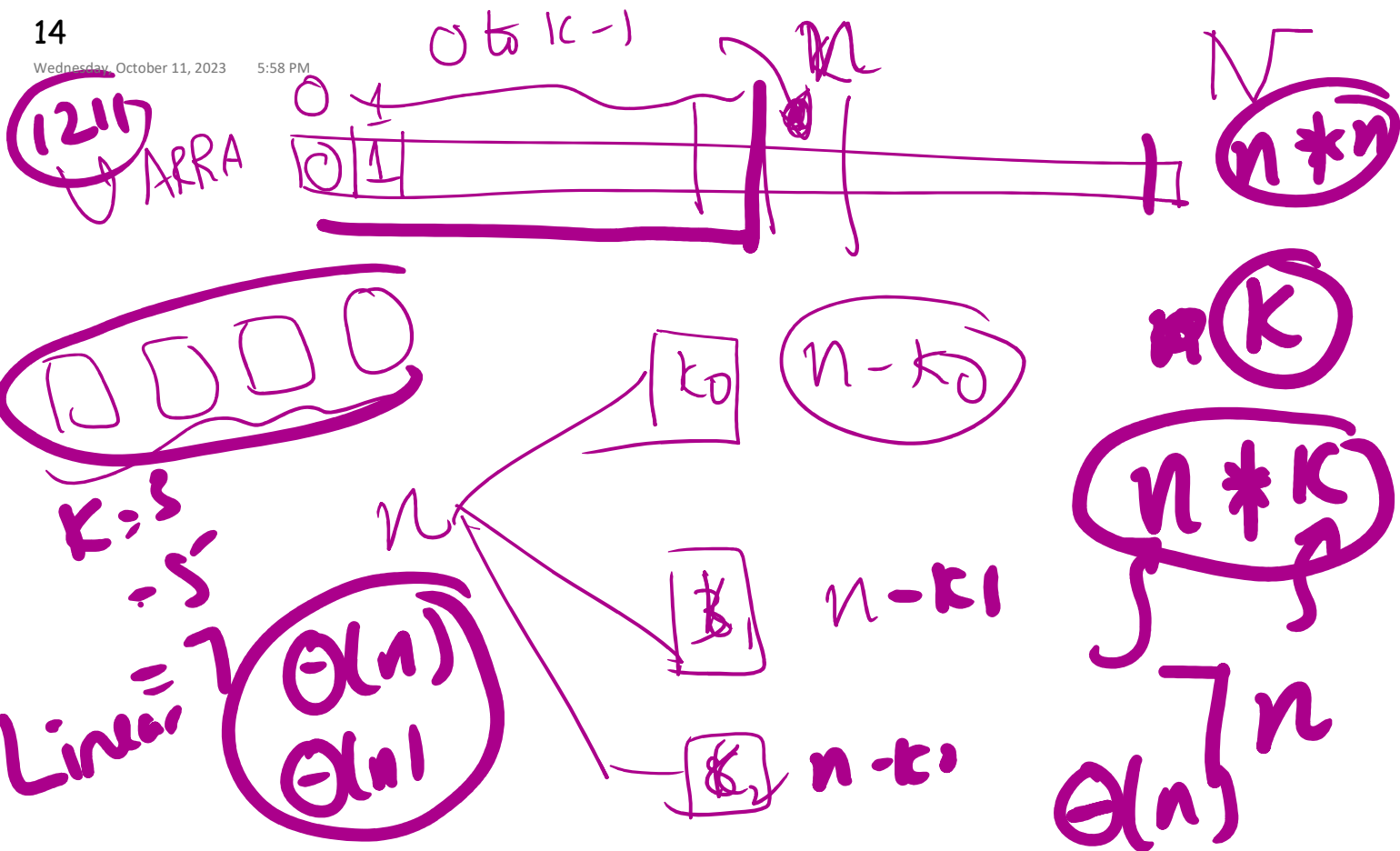
(2)

OPTIMAL
SOLUTION FOR BASE
CASE
MEMOIZATION

OPTIMAL
TABLE
BUILDING







$$n = 2589$$

Give minimum change for 34 cents using coins {1, 2, 6, 10, 24, 30, 90}

i =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
m array	0	1	1	2	2	3	1	2	2	3	1	2	2	3
k array	0	1	2	1	2	1	6	1	2	1	10	1	2	1

$$\begin{array}{r} 19 \\ \hline 18 \\ 2 \\ \hline 16 \\ 6 \\ \hline 10 \\ 10 \end{array}$$

14	15	16	17	18	19	20								
3	4	2	3	3	4	2								
2	1	6	1	2	1	10								
20	21	22	23	24	25	26	27	28	29	30	31	32		
2	3	3	4	1	2	2	3	3	4	1	2	2		
10	1	2	1	24	1	2	1	2	1	30	1	2		

$$26 \frac{2}{24}$$


```

class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:

```

{23} 3 cent
 8 6 5 1

```

13
14 class Solution:
15     ## YOU CANNOT CHANGE THIS INTERFACE
16     ## LEETCODE INTERFACE
17     def coinChange(self, coins: List[int], amount: int) -> int:
18         ## YOU CANNOT CHANGE ANYTHING IN THIS PROCEDURE
19         work = [0]
20         changes = [] # If change cannot be given, you must put -1 in changes[0]
21         show = False
22         p = L0322(coins, amount, changes, work, show)
23         num_change = len(changes)
24         if (num_change == 1):
25             if (changes[0] == -1):
26                 num_change = -1
27         return num_change

```

1, 3, 4
 n = 6

0 5

2

n 3 -1
 n 2 1

1 1 5

3
 9 **class** L0322:
 0 **def** __init__(self, coins: List[int], amount: 'int', changes: 'list of int', work: 'List of size 1', show: 'boolean'):
 1 #NOTHING CAN BE CHANGED
 2 self._d = coins
 3 self._n = amount
 4 self._ans = changes
 5 self._work = work
 6 self._show = show
 7 # YOU MUST GENERATE V table and k table
 8 self._v = []
 9 self._k = []
 0 # You can have any number of data structures here
 1 # MUST WRITE TWO ROUTINES
 2 self._alg()
 3 self._get_solution()
 4
 5 **def** _increment_work(self):
 6 self._work[0] = self._work[0] + 1
 7

Handwritten notes and diagrams:
 - {1, 4, 13}
 - 100
 - 2, 2, 2, 2, 2
 - 15, 14
 - 8, 2
 - V table:

V	0	1
K	0	1

 - A long horizontal rectangle with a wavy line below it.
 - - alg()
 - get_s

alg

```

59 print( WRITE CODE )
60 #####
61 # NOTHING CAN BE CHANGED IN THIS ROUTINE BELOW
62 #####
63 def _get_solution(self):
64     if (self._show):
65         a = []
66         for i in range(self._n + 1):
67             a.append(i)
68         print(a)
69         print(self._v)
70         print(self._k)
71         if (self._n < 1000):
72             for i in range(self._n + 1):
73                 if (self._show):
74                     print("minimum change for", i, "cents can be achieved using", self._v[i], "coins.")
75                 self._get_solution1(i)
76         else:
77             self._get_solution1(self._n)
78
79 #####
80 # TIME O(n)
81 # SPACE THETA(1)
82 # How will you give change for p cents
83 # WRITE CODE BELOW
84 #####
85 def _get_solution1(self, p: 'int'):
86     print("WRITE CODE")
87

```

$v = []$
 $k = []$

$n = 7$

0, 1, 2, 7

0, 1,

$n = 25$



```

w = [1]
c = 0
e = 0
self._test1(w,c,e)

w = [1]
c = 1
e = 1
self._test1(w,c,e)

w = [1]
c = 2
e = 2
self._test1(w,c,e)

w = [2]
c = 3
e = -1
self._test1(w,c,e)

self._show = False
w = [186,419,83,408]
c = 6249
e = 20
self._test1(w,c,e)

self._show = False
w = [474,83,404,3]
c = 264
e = 8
self._test1(w,c,e)

```

{1}

0

{13}

1

1

2

3

-1

6249

186

419

83

408

419
419

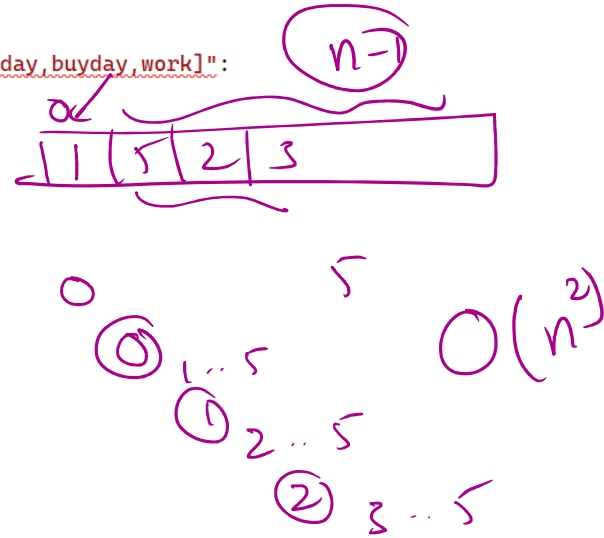
6:50

Record

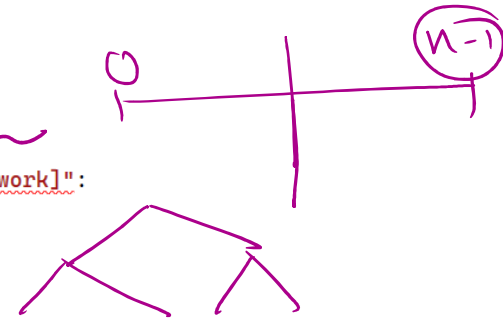
```

#####
# TIME:THETA(N^2)
# Space:THETA(1)
# I have implemented the code. NOTHING CAN BE CHANGED HERE
#####
def nsquare_time_constant_space(self, a: List[int]) -> "[sellday,buyday,work]":
    n = len(a)
    gp = 0
    bday = 0
    sday = 0
    work = 0
    for i in range(0, n - 1):
        for j in range(i + 1, n):
            work = work + 1
            p = a[j] - a[i]
            if p > gp:
                gp = p
                bday = i
                sday = j
    return [sday, bday, work]

```



```
def _nlogn_time_logn_space(self, a: List[int]) -> "sell day, buy day, work":  
    n = len(a)  
    if n == 0:  
        return 0  
    work = [0]  
    return self._dq(a, 0, n - 1, work)
```



```
def _dq(self, a: List[int], l: "int", h: "int", work: "list of size 1") -> "[sellday, buyday, work]":
    if l == h:
```

```
        # exactly one element ;
        # Profit is zero
        ans = [h, l, work[0]]
        return ans
```

```
    # Divide
```

```
    work[0] = work[0] + 1
```

```
    m = ((h - l) // 2) + l
```

```
    # Left side profit index
```

```
    li = self._dq(a, l, m, work) # li = sellday, buyday, work
```

```
    # Right side profit index
```

```
    ri = self._dq(a, m + 1, h, work) # ri = sellday, buyday, work
```

```
    # CONQUER
```

```
    work[0] = work[0] + 1
```

```
    # Find minimum number index on left side
```

```
    minlefti = self._get_min_or_max_index(a, l, m, work, self._min)
```

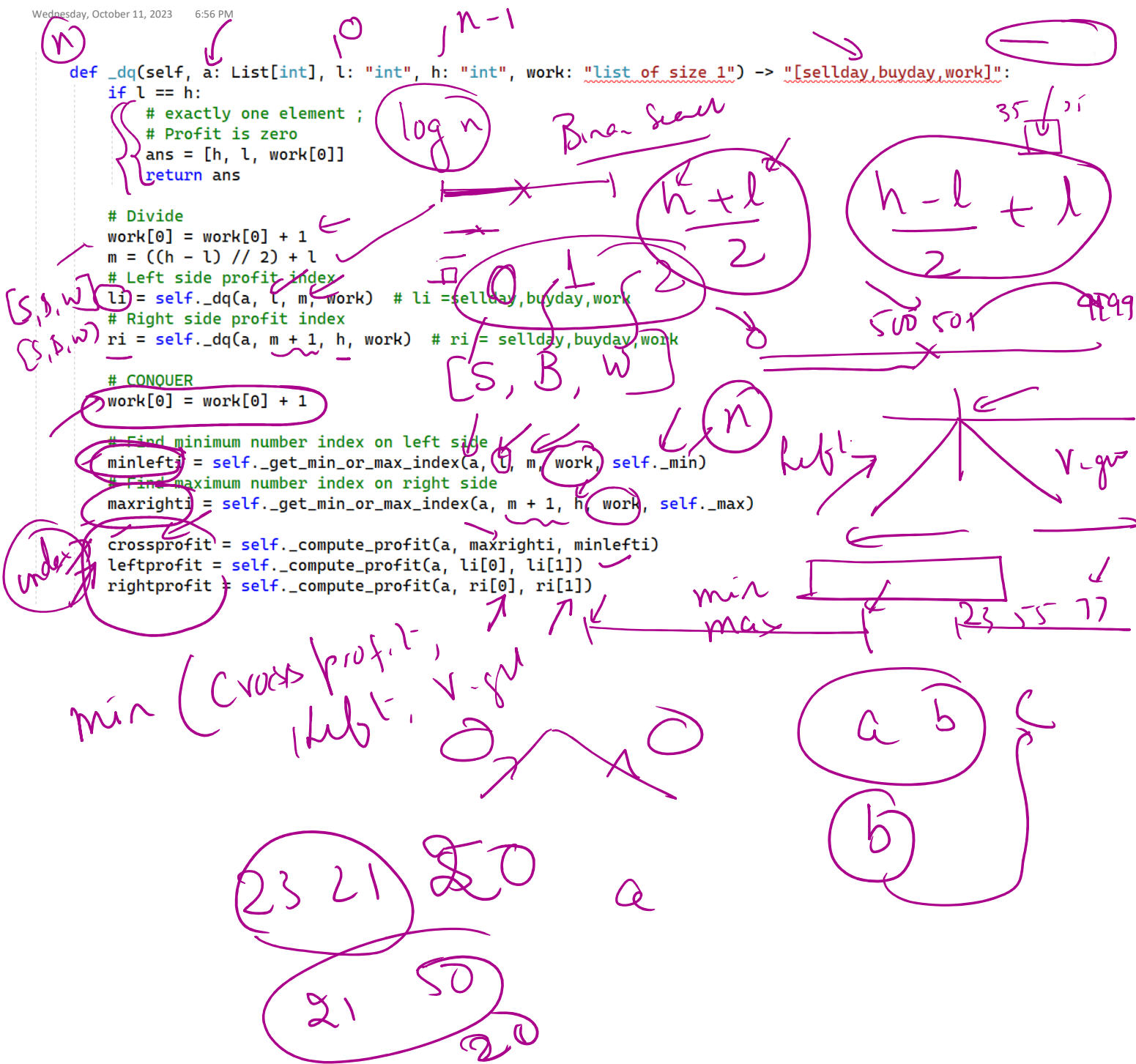
```
    # Find maximum number index on right side
```

```
    maxrighti = self._get_min_or_max_index(a, m + 1, h, work, self._max)
```

```
    crossprofit = self._compute_profit(a, maxrighti, minlefti)
```

```
    leftprofit = self._compute_profit(a, li[0], li[1])
```

```
    rightprofit = self._compute_profit(a, ri[0], ri[1])
```

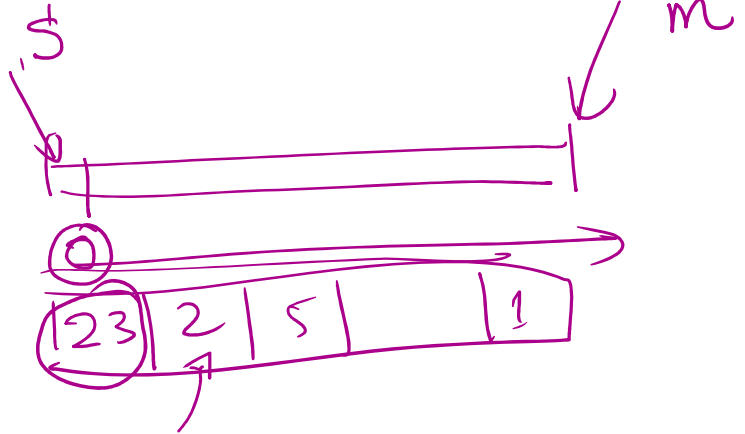


```

#####
# TIME:THETA(n)
# Space:THETA(1)
# Find min or max from s to m included
#####
def _get_min_or_max_index(self, a: List[int], s: "int", m: "int", work: "list of size 1", minormax: "int") -> "int":
    v = a[s]
    vi = s
    for i in range(s + 1, m + 1):
        work[0] = work[0] + 1
        if minormax == self._min:
            if a[i] < v:
                v = a[i]
                vi = i
        else:
            if a[i] > v:
                v = a[i]
                vi = i
    return vi # return index

```

$\Theta(n)$




```
def _ntime_constant_space(self, a: List[int]) -> "[sellday, buyday, work]":
```

```
    n = len(a)
```

```
    if n == 0:
```

```
        return [0, 0, 0]
```

```
    work = 1
```

```
    gp = 0
```

```
    sellday = 0
```

```
    buyday = 0
```

```
    lowest_stock_day = 0
```

```
    lowest_stock_day_price = a[0]
```

```
    for i in range(1, n):
```

```
        work = work + 1
```

```
        price_today = a[i]
```

```
        if price_today < lowest_stock_day_price:
```

```
            lowest_stock_day = i
```

```
            lowest_stock_day_price = price_today
```

```
        else:
```

```
            p = self._compute_profit(a, i, lowest_stock_day)
```

```
            if p > gp:
```

```
                gp = p
```

```
                buyday = lowest_stock_day
```

```
                sellday = i
```

```
    ans = [sellday, buyday, work]
```

```
    return ans
```

day → PRICE → 100

n^2
 $n \log n$
 n

n

100 99 98
1 2 3 4 5

25 20 1 25

1
100

17.6 0/1 Knapsack problem

Value ARRAY
BAG OF SIZE 0

$v_1 = 5$	$v_2 = 3$	$v_3 = 4$
$w_1 = 3$	$w_2 = 2$	$w_3 = 1$

	TV	LAPTOP	GOLD	TV GOLD
Value	5	3	4	9
Weight	3	2	1	4

8 \$

(5)

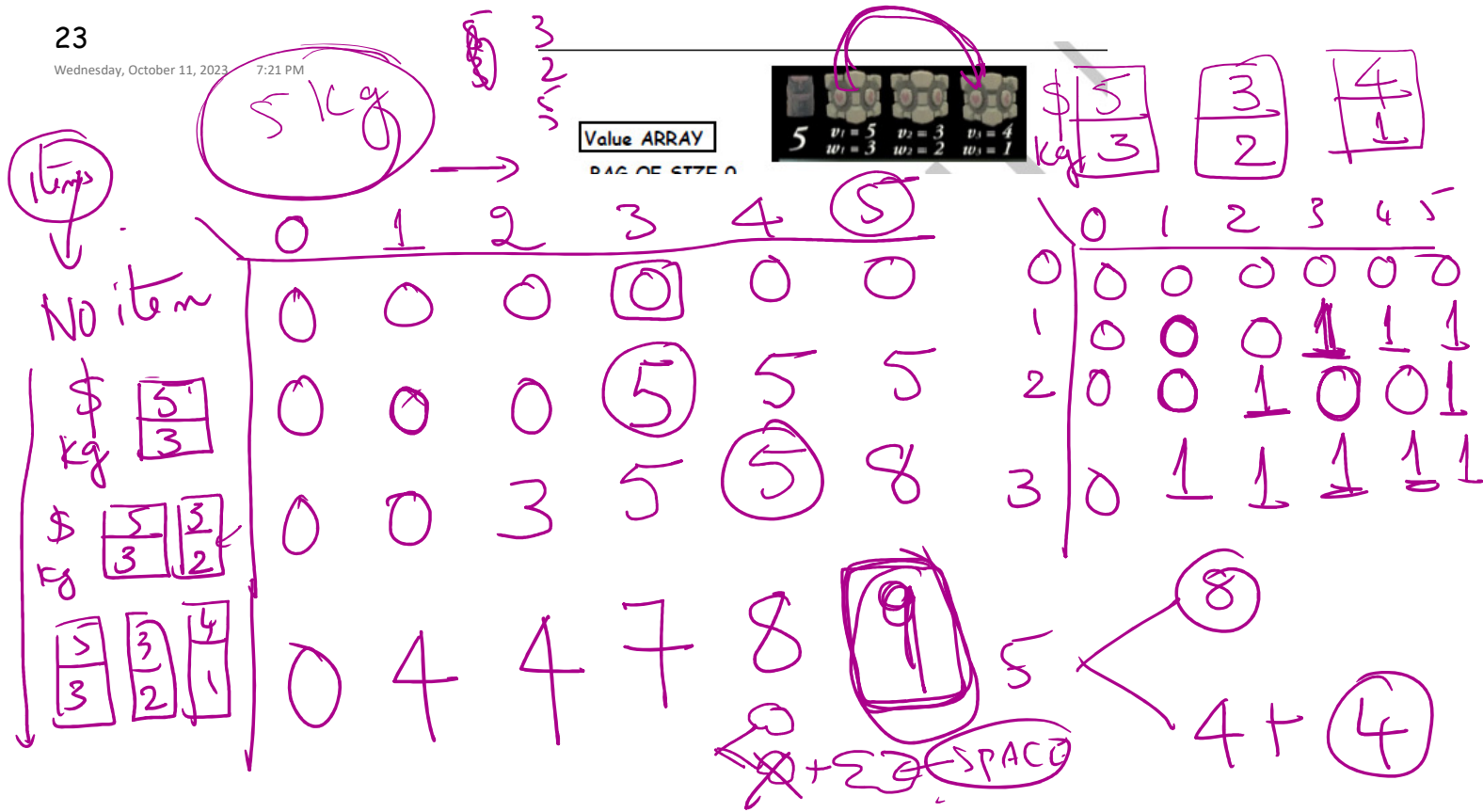
(15 10)

□□□□□

Coin change

1, 5, 10, 25

MANY



Keep Array		BAG SIZE					
		0	1	2	3	4	5
items	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1
2	0	0	1	0	0	1	1
3	0	1	1	1	1	1	1

$n = \text{item size}$ $c = \text{max bag size}$ Max profit is 9
 1. {3,5} is 1. Item 3 of weight 1, value 4 is selected.
 Remaining bag space = $5 - 1 = 4$.
 Remaining profit = $9 - 4 = 5$
 2. {2,4} is 0. Nothing is selected.
 3. {1,4} is 1. Item 1 of weight 3, value 5 is selected.
 Remaining bag space = $4 - 3 = 1$.
 Remaining profit = $5 - 5 = 0$
 Remaining profit is Zero. Algorithm terminates

0/1

BAK

	0	1	2	3	4	5
No item	0	0	0	0	0	0
\$ 5 kg 3	0	0	0	5	5	5
\$ 5 3 kg 3 2	0	0	3	5	8	8
\$ 5 3 4 kg 3 2 1	0	4	4	7	8	9

