

complexity

Wednesday, September 27, 2023

5:05 PM

BUS

DYNAMIC ARRAY

LIST

$O(1)$ Amortize

A

P

$O(n)$

$O(n)$

Anyplace

$\Theta(1)$

$\Theta(1)$

$O(n)$

$O(n)$

Insert

len

$a[i]$

FIND

Delete

TRAIN SLIST

A

$O(1)$

P

$O(1)$

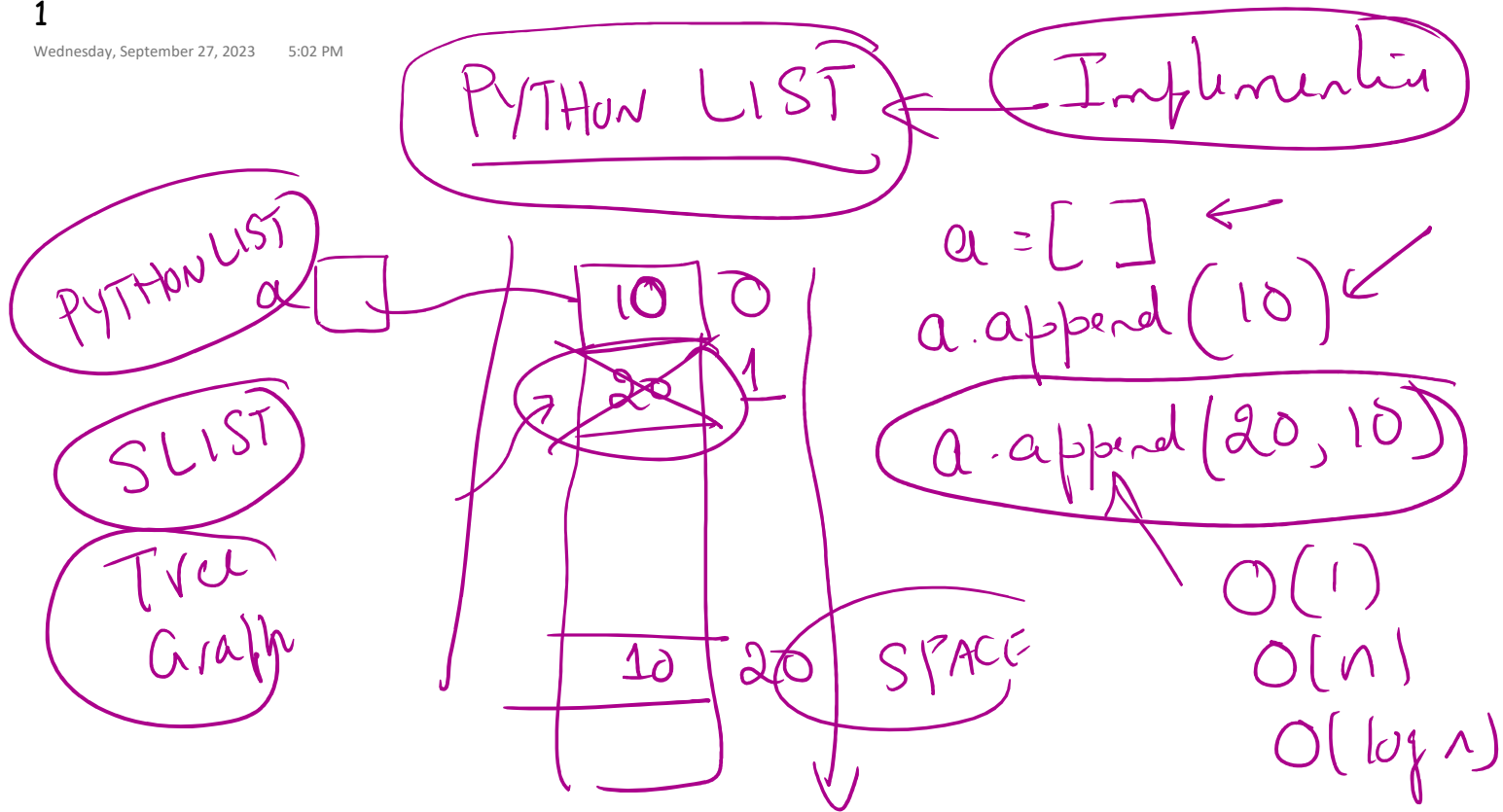
Anyplace

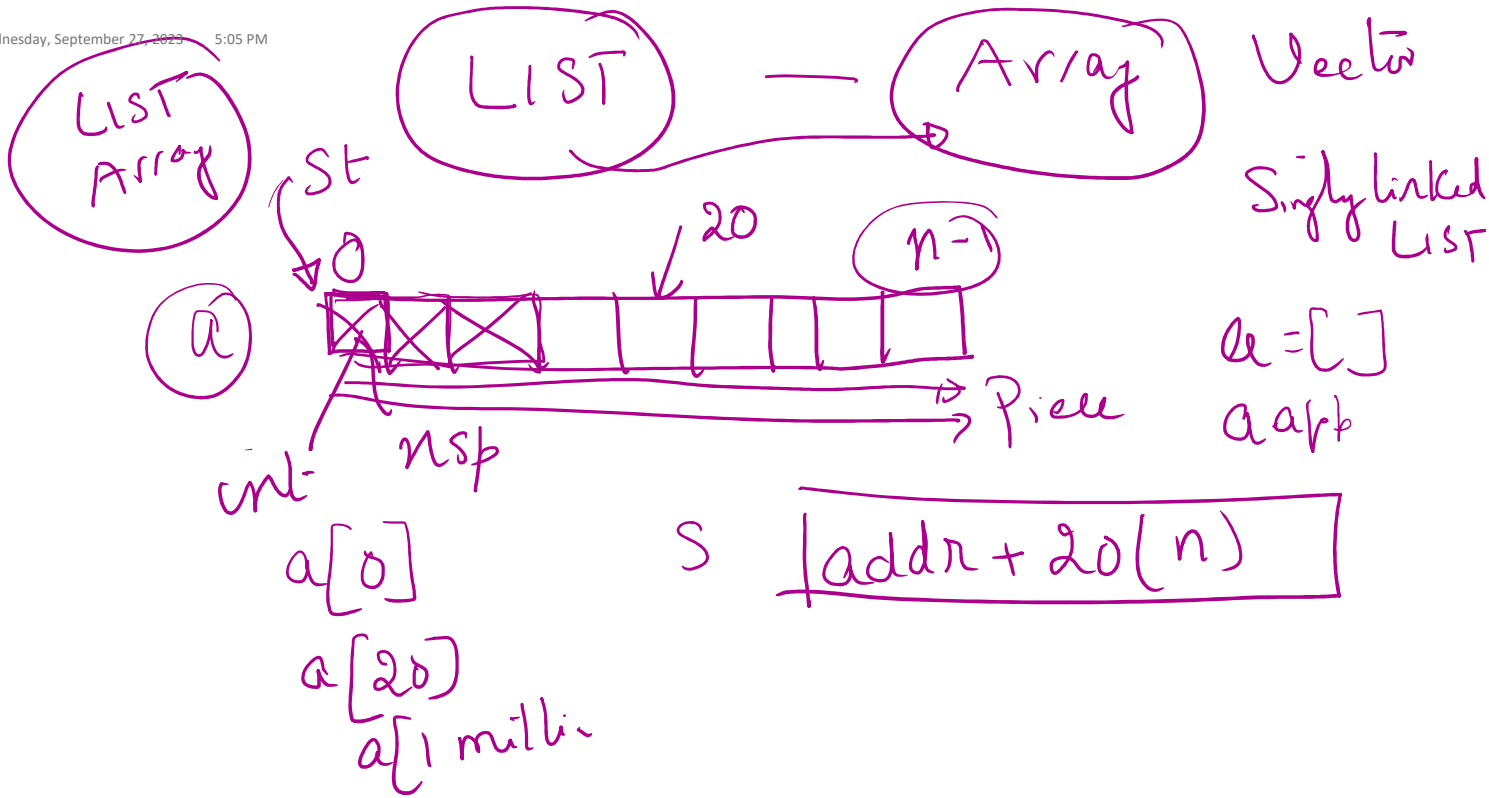
$O(n)$

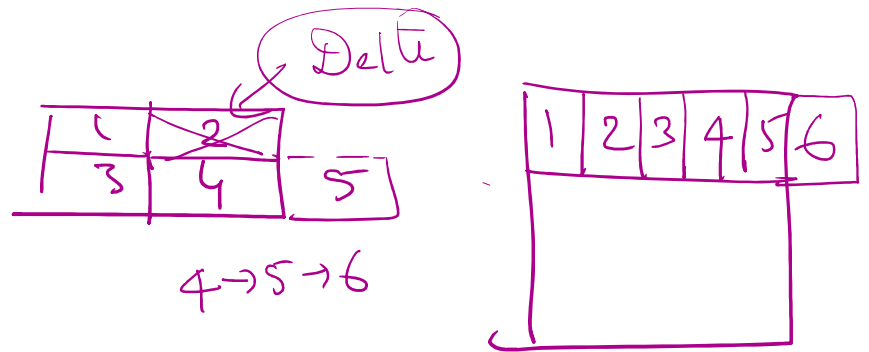
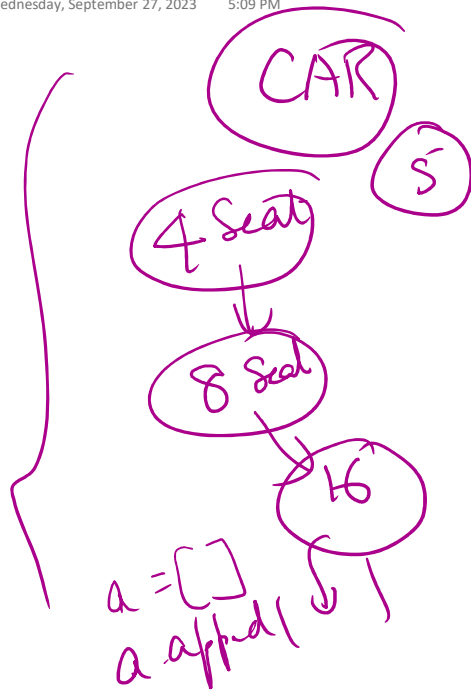
$\Theta(n)$

$O(n)$

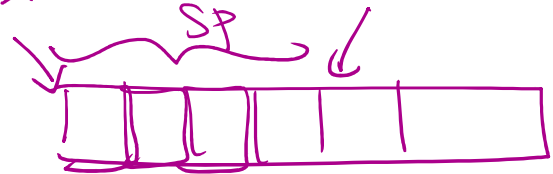
$O(n)$







PYTHON LIST



$a[0]$, $a[100]$, $a[1\text{milli}]$

Combin

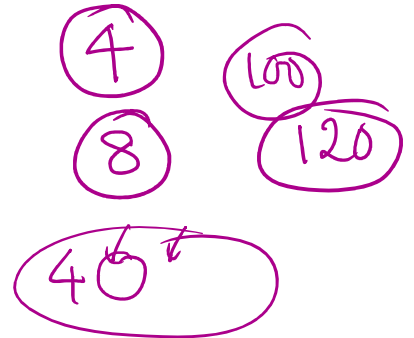
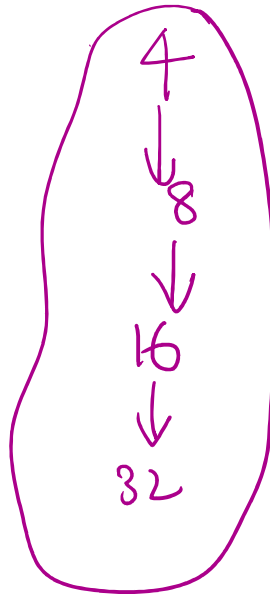
Table Doubling Algorithm

PYTHON
LIST

Buy

Is it
a square

TABLE
Doubling



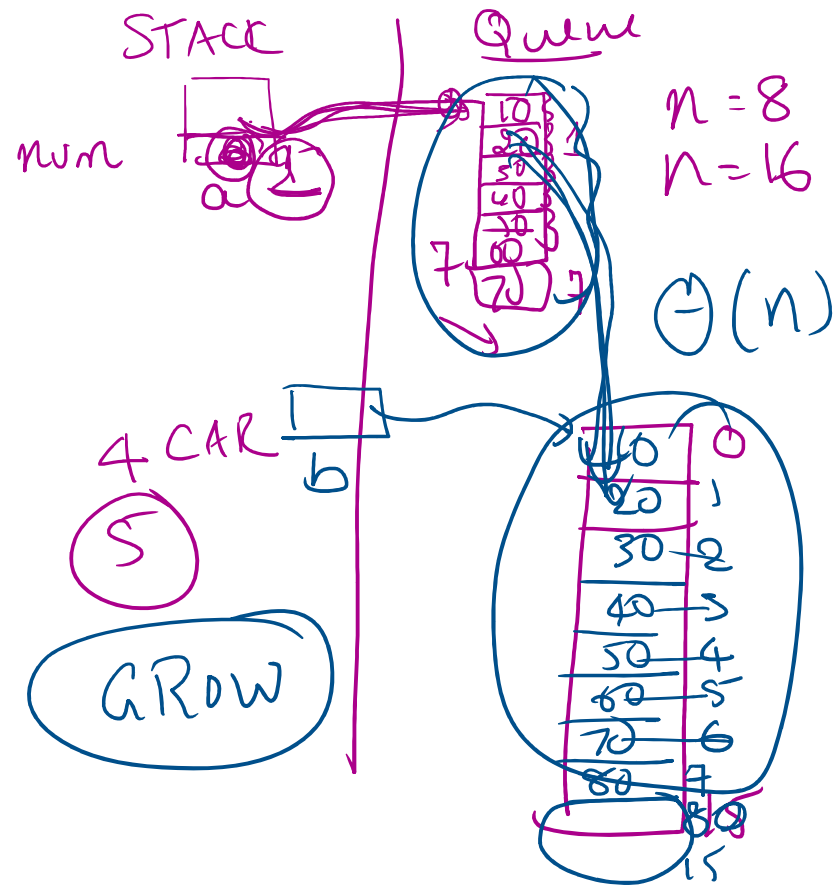
len(a)
= 1

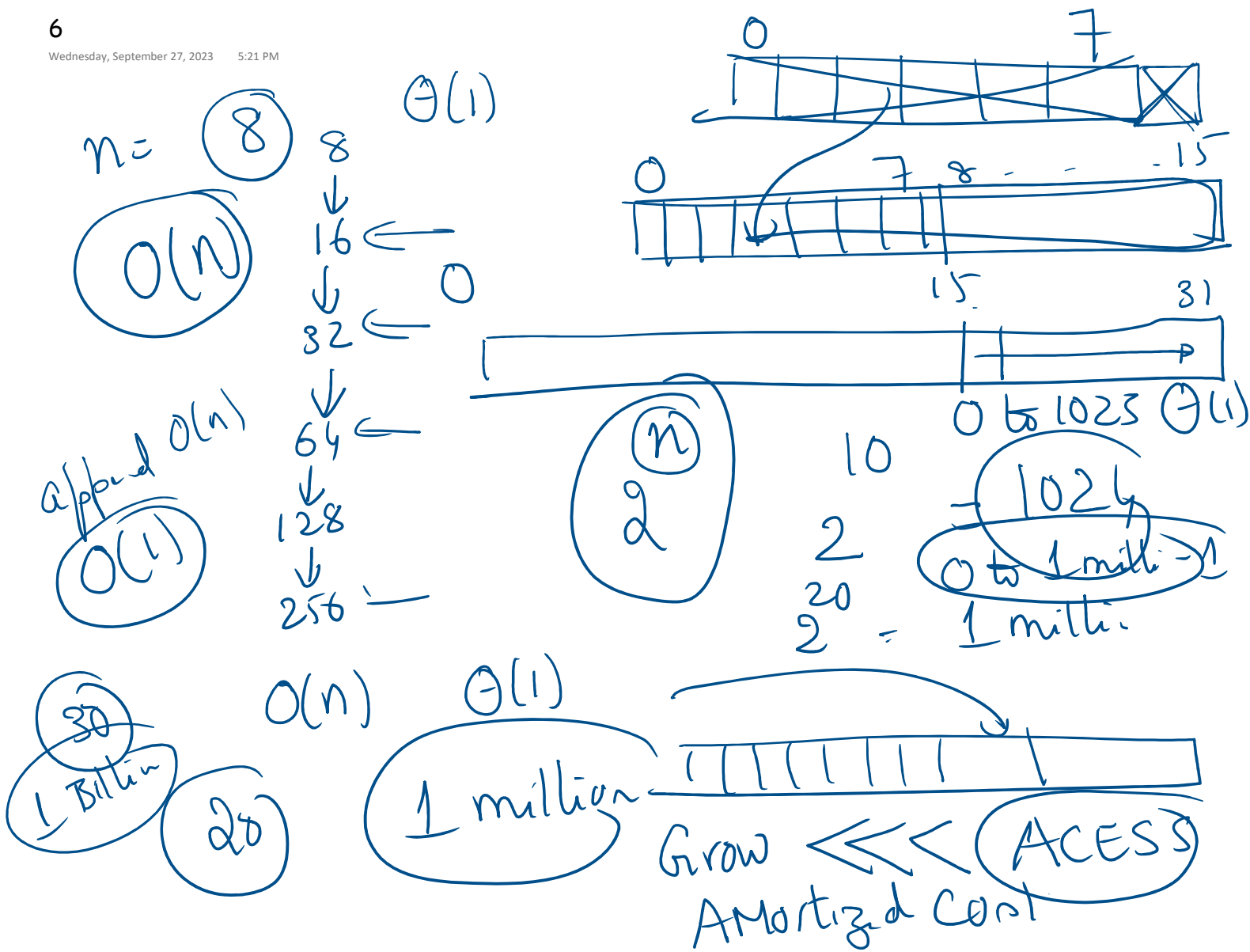
len(c)
= 2

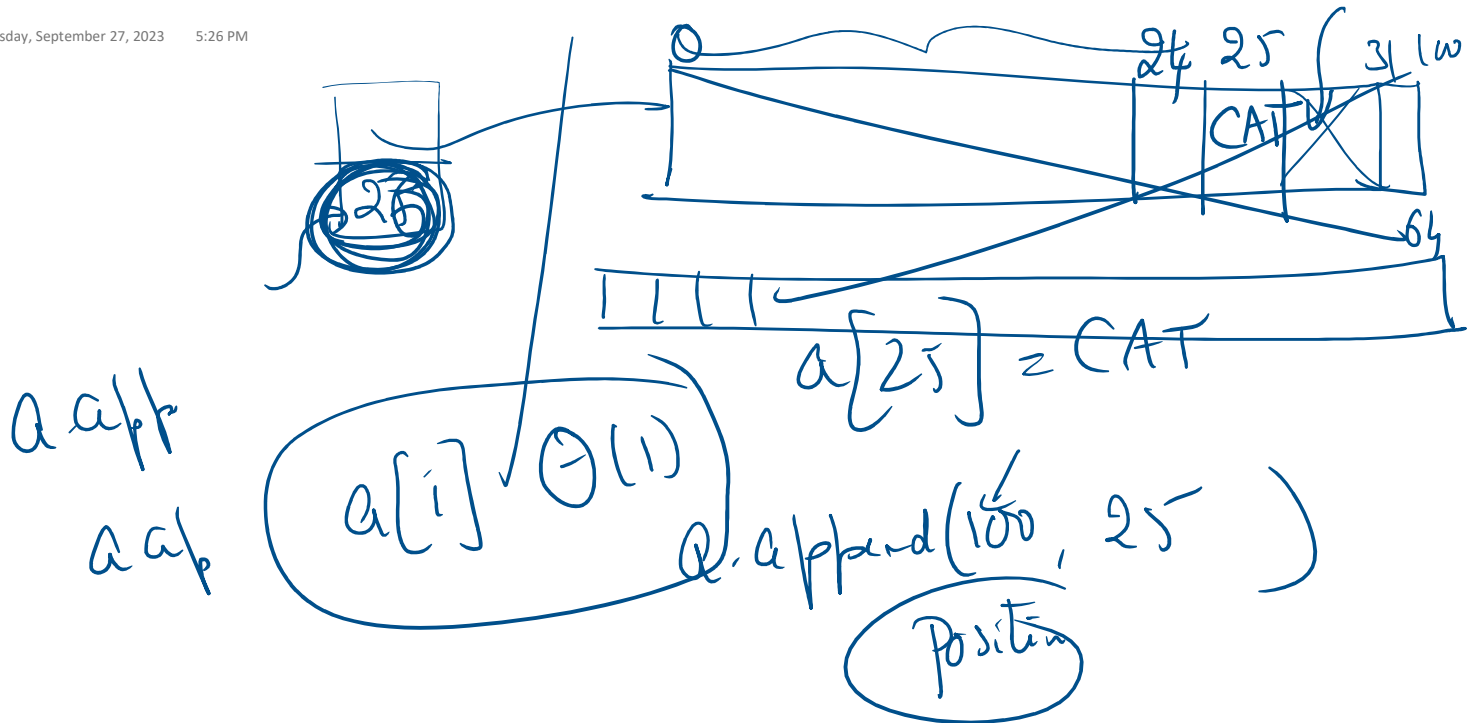
↙
a = []
a.append(10)

a.append(20)
(30)
(40)
(70)

a.append(80)
a.append(100)







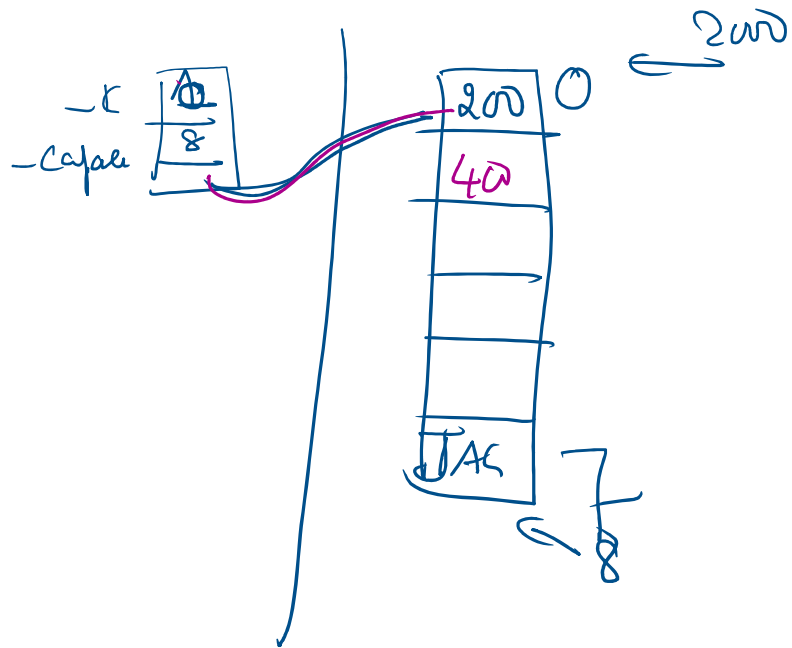
10

Wednesday, September 27, 2023 5:40 PM

```
def append(self, item):
    if (self._k == self._capacity):
        self._grow()
    self._a[self._k] = item
    self._k = self._k + 1
```

①

a. affere (2w)
(4w)



→ 184

2^n

- IC
- capacitor
- a

```
# Time:  $O(n)$ 
# Space:  $O(n)$ 
```

```
os = self._capacity
```

$$nS = OS * 2$$

```
print("Grow from", os, "to", ns, ". This is not O(1)")
```

```
b = self._allocate(ns) # New bigger array
```

```
for k in range(os): # Reference all existing values
```

```
b[k] = self._a[k]
```

```
self._a = b # Call b as the new bigger array
```

```
self._capacity = ns # Reset the capacity
```

$n = 20$
1 million
 $n = 30$
1 Billion

ii) 0.8

ns | 16

b

 ~~$a[i]$~~
 $\ominus[1]$

A hand-drawn number line on lined paper. It consists of a vertical column of eight boxes. To the right of each box is a corresponding number from 0 to 7. The boxes contain the following numbers: 0, 100, 200, 300, 400, 500, 600, and 700. Arrows point from the boxes to the numbers on the right. A large bracket on the left side of the boxes spans from the 0 box to the 700 box.

Handwritten notes and diagrams illustrating the implementation of the `__getitem__` method for a custom list-like object.

Handwritten Annotations:

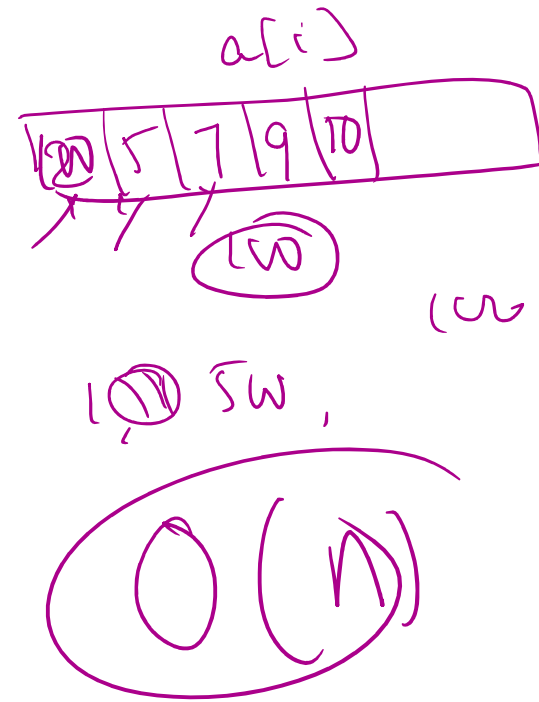
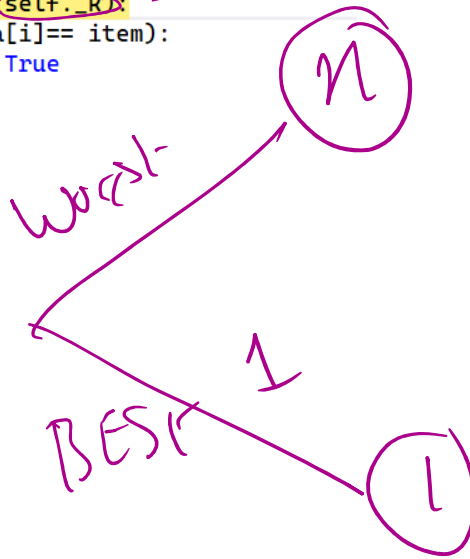
- A circle containing $O(1)$ is drawn to the left of the code block.
- Handwritten expressions $d[5]$ and $d[10000]$ are written above the code block.
- Handwritten expressions $d[-5]$ and $d[0]$ are written to the right of the code block.
- A diagram shows a vertical rectangle divided into three horizontal sections. The top section is labeled $-k$ and has a circle around it. An arrow labeled car points to the middle section. A line connects the middle section to a larger rectangle on the right, which is also divided into three horizontal sections.

Code Block:

```
#####  
# Time:O(1)  
# Space:O(1)  
#####  
def __getitem__(self, index):  
    assert(index >= 0 and index < self._k)  
    return self._a[index]  
#####
```

$O(n)$

```
def __contains__(self, item) -> 'int':  
    for i in range(self.k):  
        if (self._a[i] == item):  
            return True  
    return False
```



```

n = [1, 2, 99, 2]
k = 2
self._test1(n, k, 2)

```

```

n = [1, 1, 1, 1]
k = 1
self._test1(n, k, 0)

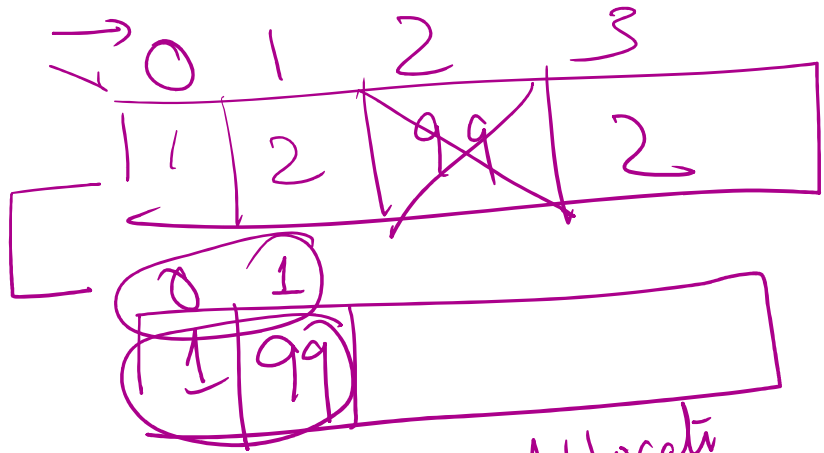
```

n

n

k

9



Allocati

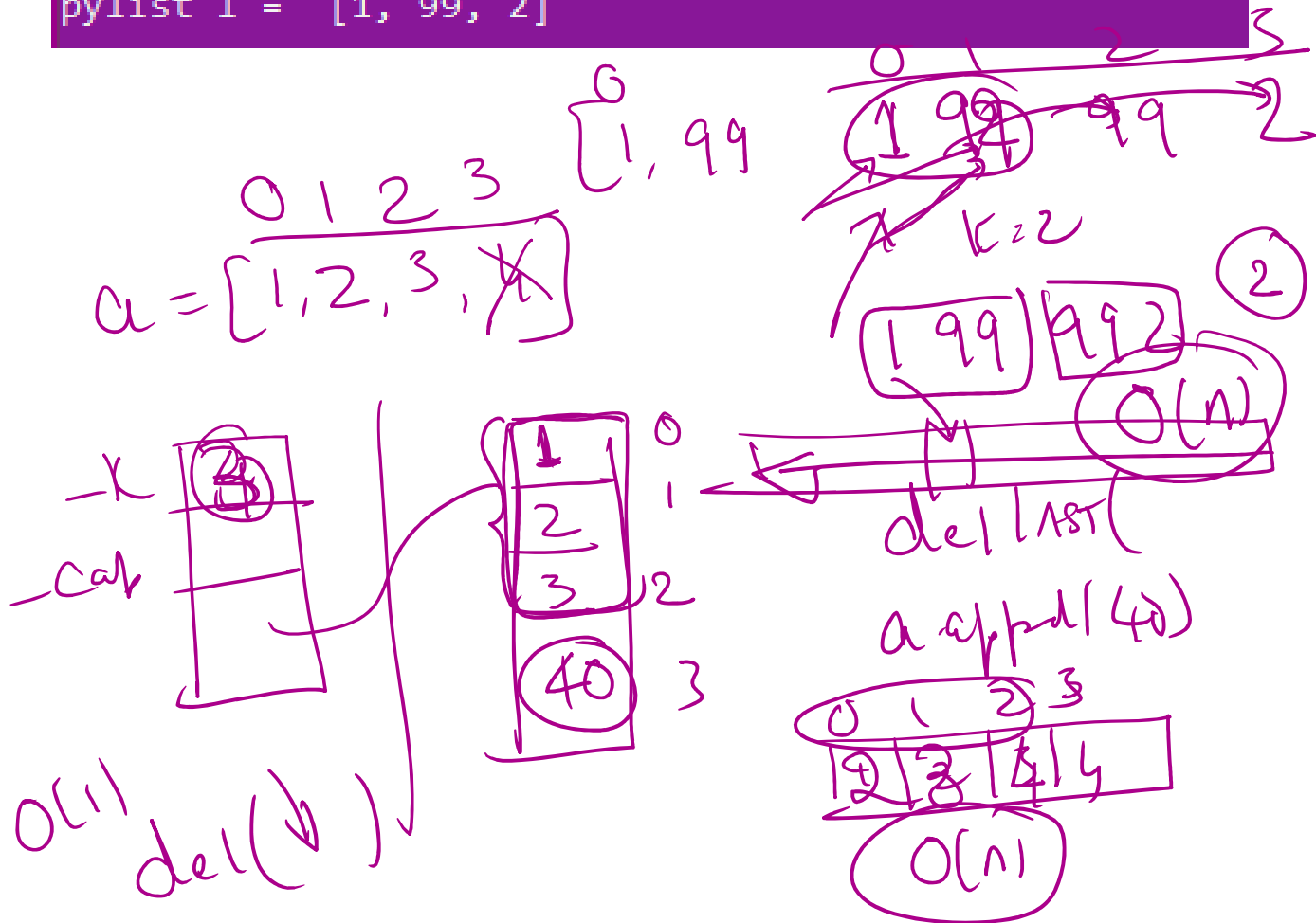
2

~~[1, 2, 99, 2]~~ ~~[1, 2, 99, 2]~~
~~[1, 99, 2]~~

```

----- Testing delete -----
-----Before delete -----
mylist d = [1 2 99 2 ]
pylist l = [1, 2, 99, 2]
-----After deleting all 2 from the array-----
mylist d = [1 99 ]
pylist l = [1, 99, 2]

```



$\left\{ \begin{array}{l} \text{for } (10) \\ \text{Count} = \text{Count} + 1 \\ \log n \end{array} \right\} O(n)$

$n = 1000$
 n^2

```

def _is_prime(self, n: "int") -> "bool":
    # for (int i = 2; (i * i) <= n; ++i)
    i = 2
    while (i * i) <= n: # square root without calling library
        self._increment_work_done()
        if n % i == 0:
            return False
        i = i + 1
    return True

```

```

def _nsquare(self):
    self._append_prime_number(2) # 2 is a prime
    k = self._n + 1
    for i in range(3, k, 2):
        p = self._is_prime(i)
        if p == True:
            self._append_prime_number(i)

```

$i = 2; i \leq \sqrt{n}$
 \sqrt{n}
 $\sqrt{n} \cdot \sqrt{n} = n$

$(2) \sqrt{n} * \sqrt{n} = n$

$(2) \ 50 \ 100$
 (1000)

$\sqrt{1000} \ (33)$

$O(n) \ O(n)$

3

$n / \rightarrow \sqrt{n}$

$\sqrt{1000}$
 $33 * 33 = 995$
 $34 * 34$

$n\sqrt{n}$

10000

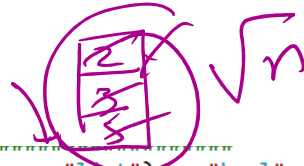
$10 * 10 = 100$
 $11 * 11 =$

```

def _is_divisible_by_prime(self, n: "int", p: "list") -> "bool":
    # for (int i = 0; (i * i) <= n; ++i)
    i = 0
    while (p[i] * p[i]) <= n: # square root without calling library
        self.increment_work_done()
        if n % p[i] == 0:
            return False
        i = i + 1
    return True

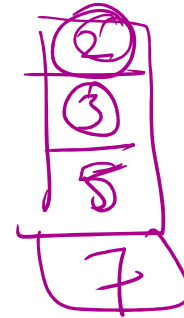
def _up_to_primes(self):
    self._append_prime_number(2) # 2 is a prime
    k = self._n + 1
    for i in range(3, k, 2):
        p = self._is_divisible_by_prime(i, self._prime_number_list)
        if p == True:
            self._append_prime_number(i)

```



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

$$n = 7$$



3, 5, 100

```
def _sieve_of_eratosthenes(self):
```

```
    # O(n) space
```

```
    # O(n) time
```

```
    l = [1]
```

```
    for i in range(self._n + 1):
```

```
        self._increment_work_done()
```

```
        l.append(True)
```

```
    l[0] = False
```

```
    l[1] = False
```

```
    i = 2
```

```
    while (i * i) <= self._n: # square root without calling library
```

```
        if l[i] == True:
```

```
            j = i
```

```
            while (i * j) < (self._n + 1):
```

```
                # if i = 10
```

```
                # i = 10 10 10 10
```

```
                # j = 10 11 12 13
```

```
                # (i*j) = 100 110 120 130 #100,110,120,130 is NOT prime
```

```
                self._increment_work_done()
```

```
                l[i * j] = False
```

```
                j = j + 1
```

```
    i = i + 1
```

```
    # NOW get number of primes
```

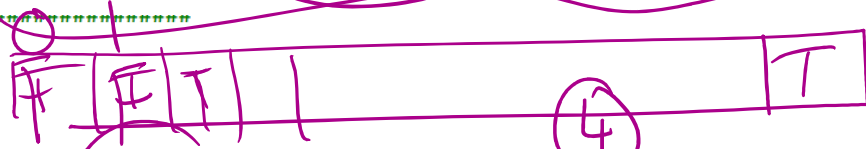
```
    for i in range(self._n + 1):
```

```
        self._increment_work_done()
```

```
        if l[i] == True:
```

```
            self._append_prime_number(i)
```

$n = 16$



\sqrt{n}

$\frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{\sqrt{n}}$

$O(\log \log n)$

131

5

$\frac{n}{\sqrt{n}}$

$\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \frac{n}{7} + \dots$

14

16

6:55 PM

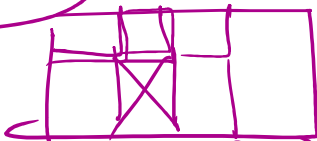
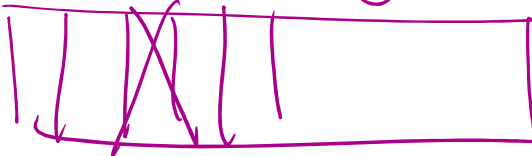
Record

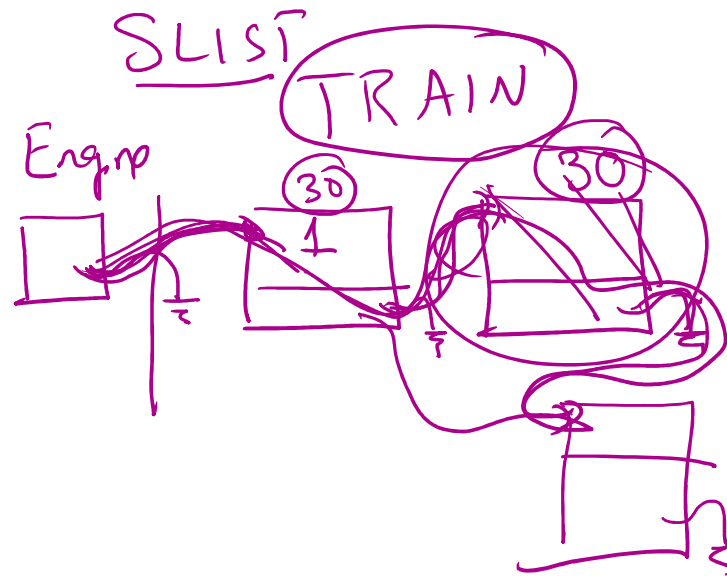
Singly linked List

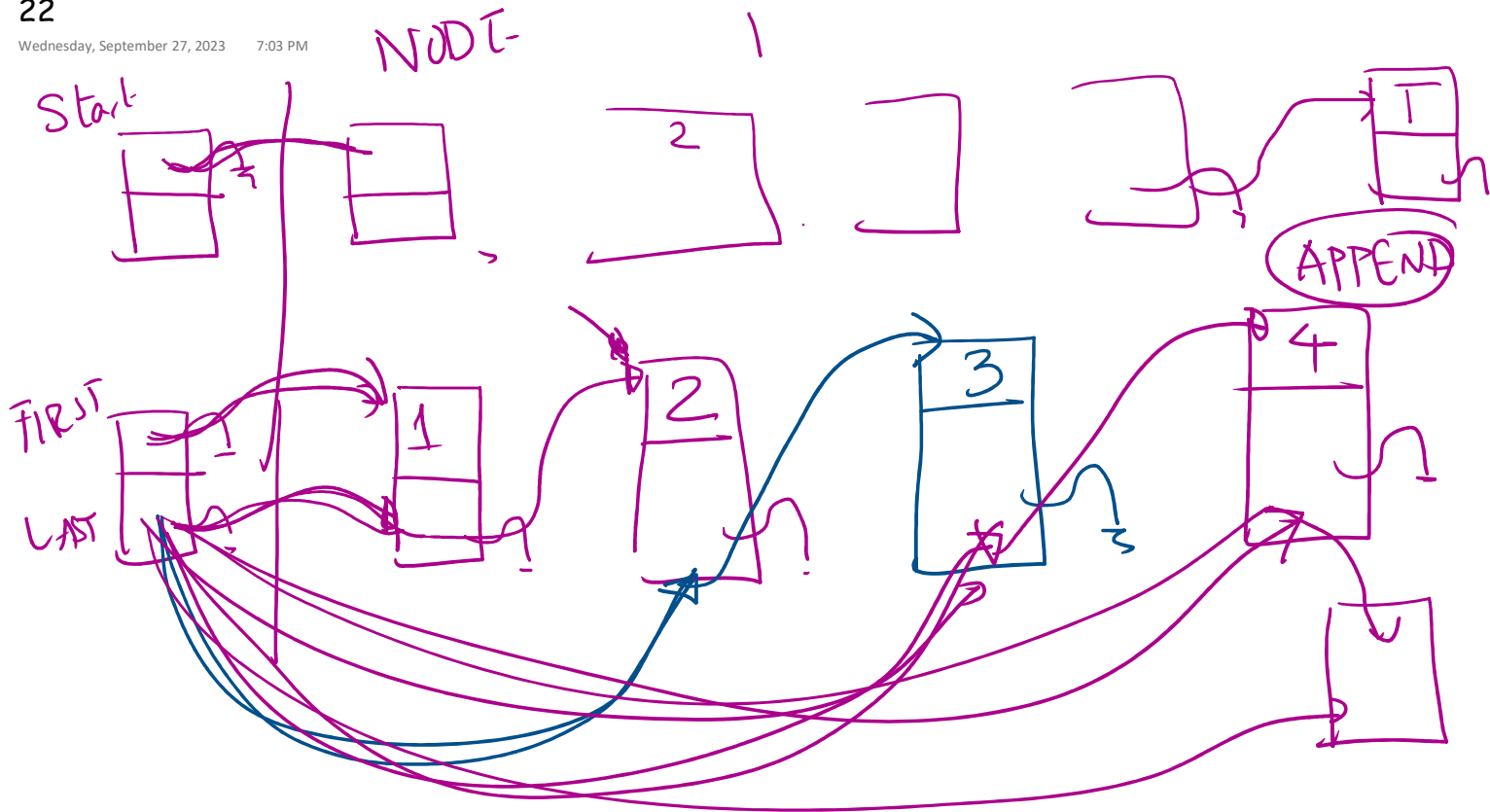
LIST
SLIST

✓	2	Add Two Numbers	41.3%	Medium	
✓	19	Remove Nth Node From End of List	42.6%	Medium	
	21	Merge Two Sorted Lists	63.2%	Easy	
	23	Merge k Sorted Lists	51.0%	Hard	
	24	Swap Nodes in Pairs	63.1%	Medium	
	25	Reverse Nodes in k-Group	56.6%	Hard	
	61	Rotate List	36.8%	Medium	
	82	Remove Duplicates from Sorted List II	46.5%	Medium	
	83	Remove Duplicates from Sorted List	51.5%	Easy	
	86	Partition List	55.1%	Medium	
	92	Reverse Linked List II	47.2%	Medium	
	109	Convert Sorted List to Binary Search Tree	60.9%	Medium	
	114	Flatten Binary Tree to Linked List	63.3%	Medium	
	116	Populating Next Right Pointers in Each Node	61.5%	Medium	
	117	Populating Next Right Pointers in Each Node II	51.2%	Medium	
	138	Copy List with Random Pointer	54.1%	Medium	
	141	Linked List Cycle	48.8%	Easy	
	142	Linked List Cycle II	50.0%	Medium	
	143	Reorder List	54.5%	Medium	
	146	LRU Cache	41.7%	Medium	
	147	Insertion Sort List	52.1%	Medium	
	148	Sort List	56.4%	Medium	

Wednesday, September 27, 2023 6:59 PM

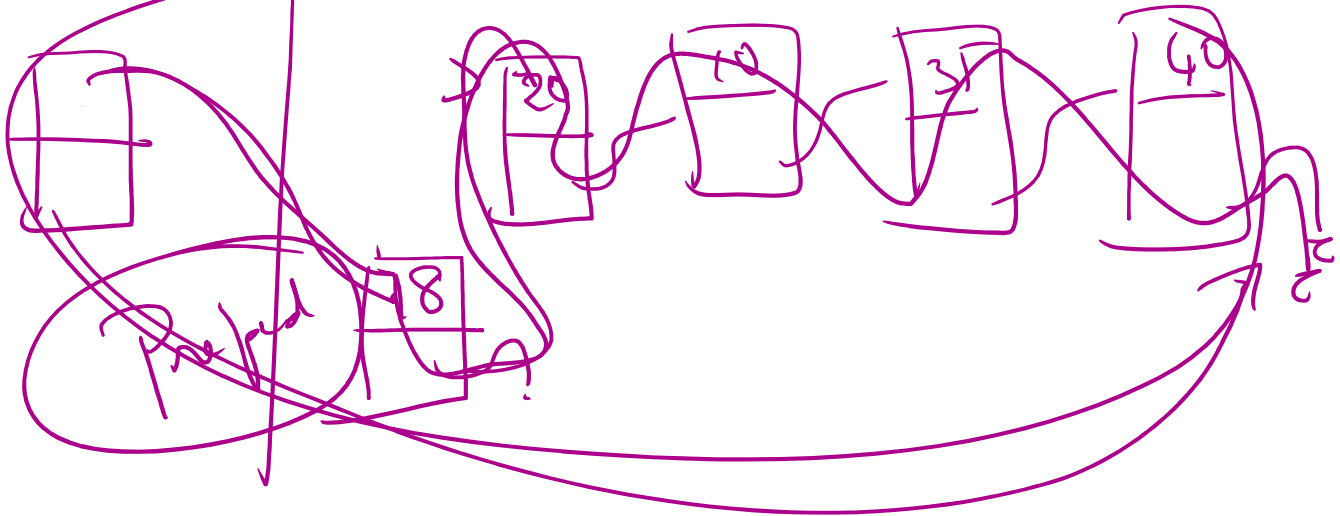
LIST ARRAY 8
 16
 32
 BUS



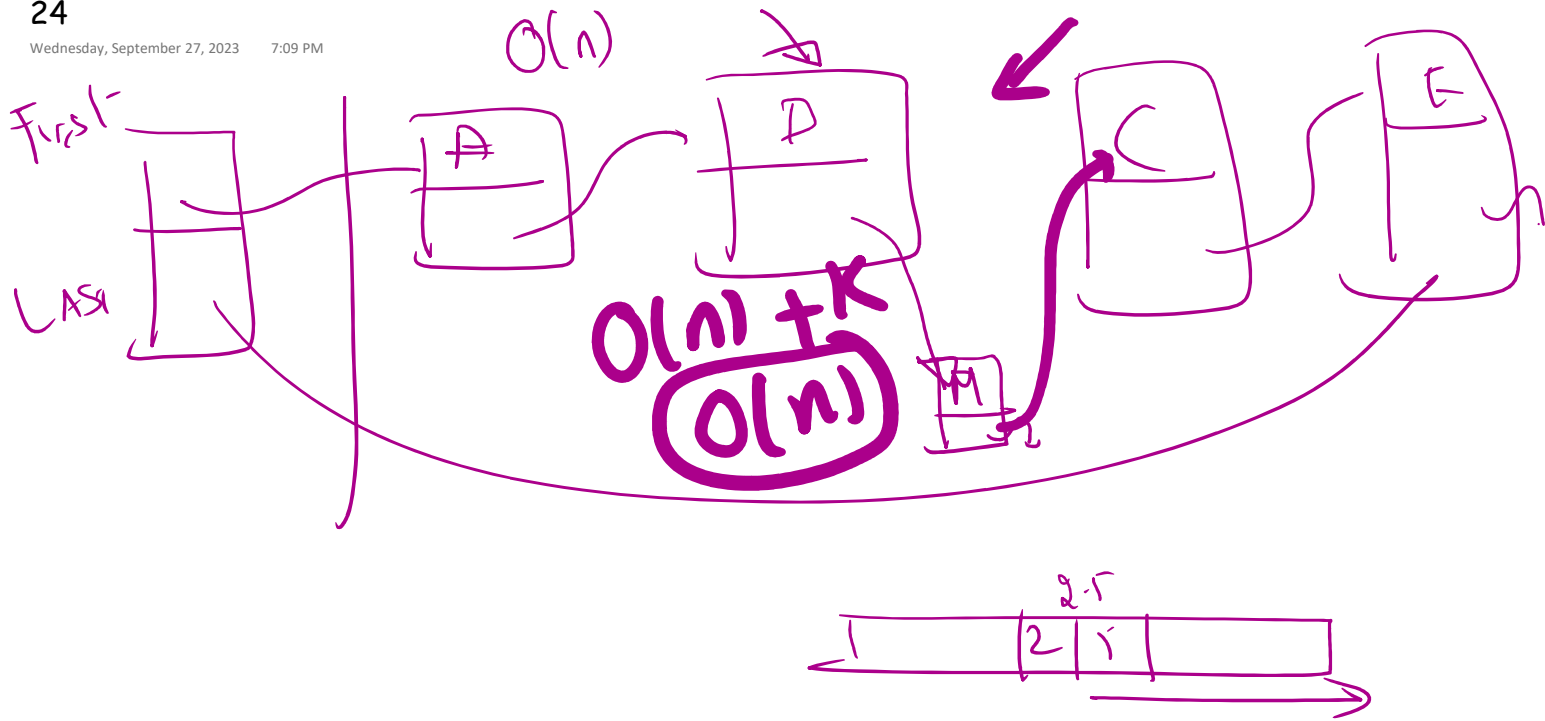




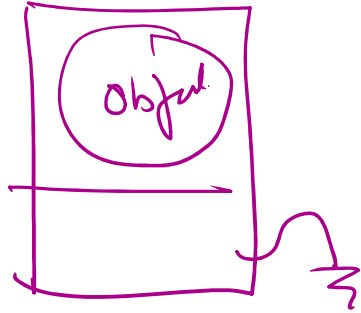
$O(n)$

0	1	2	3	
12	10	35	40	
18	20	10	55	40

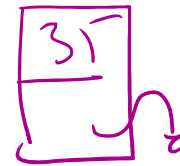
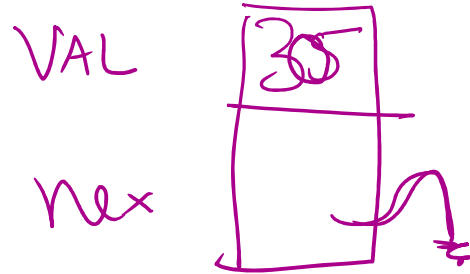




```
#####
class ListNode:
    def __init__(self, val = 0, next= None):
        self.val = val
        self.next = next
#####
```



ListNode n(35);



```
#####
class Slist():
    def __init__(self):
        #NOTHING CAN BE CHANGED HERE
        self._first = None
        self._last = None
#####
```

Slist s;

-first

-last



App



```

s = Slist()
a = [1, 2, 3, 4, 5]
s.build_slist_from_list(a)
print(a, "is stored as", s)

```

for (e in a)

①, 2, 3, 4, 5

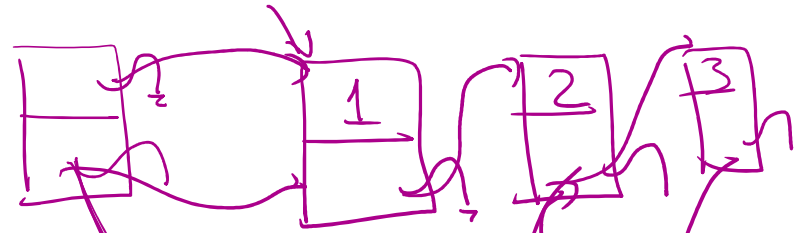
[1, 2, 3, 4, 5]

--str--

-first-

-last-

--str--

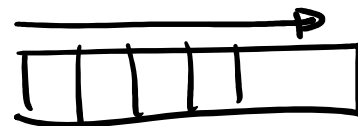


1 → 2 → 3 → 4 → 5 → None

len = 5

1 → 2 → 3 → 4 → 5 → None

1



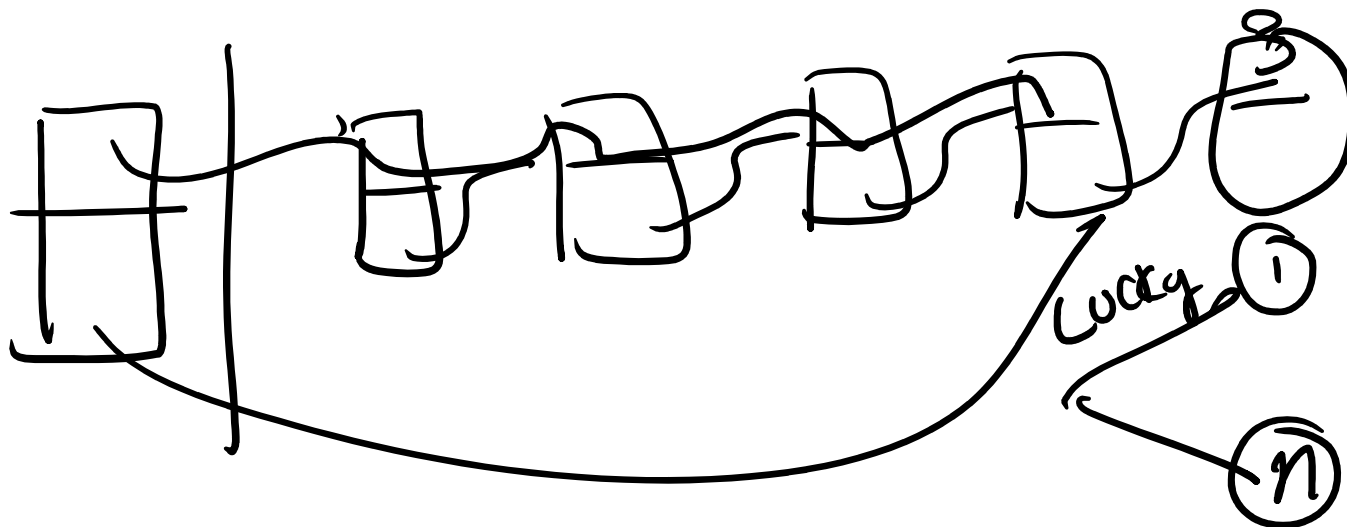
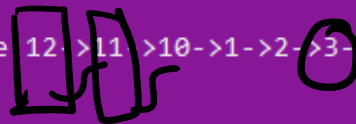
----- test append/prepend/find =====

[1, 2, 3, 4, 5] is stored as 1->2->3->4->5->NULL

size of slist = 5

After prepending [10, 11, 12] s looks like 12->11->10->1->2->3->4->5->NULL

size of slist = 8



```

s = slist()
a = [1,2,3,4,5]
s.build_slist_from_list(a)
print(a,"is stored as",s);
num = len(s);
print("Size of s =",num)
assert(num == len(a))
x = 1
s.delete(x)
print("after removing",x, ":",s)
num = len(s)
assert(num == len(a)-1)
x = 5
s.delete(x)
print("after removing",x, ":",s)
num = len(s)
assert(num == len(a)-2)
x = 3
s.delete(x)
print("after removing",x, ":",s)
num = len(s)
assert(num == len(a)-3)

```

