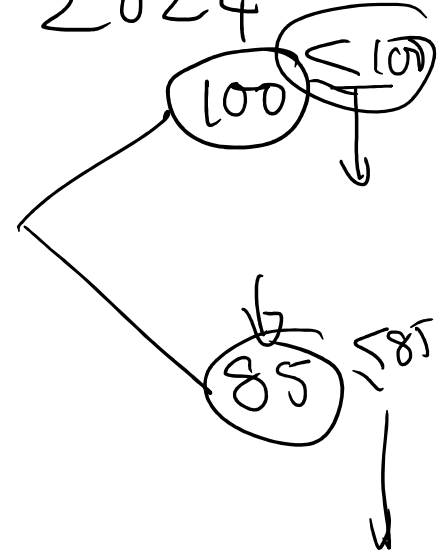
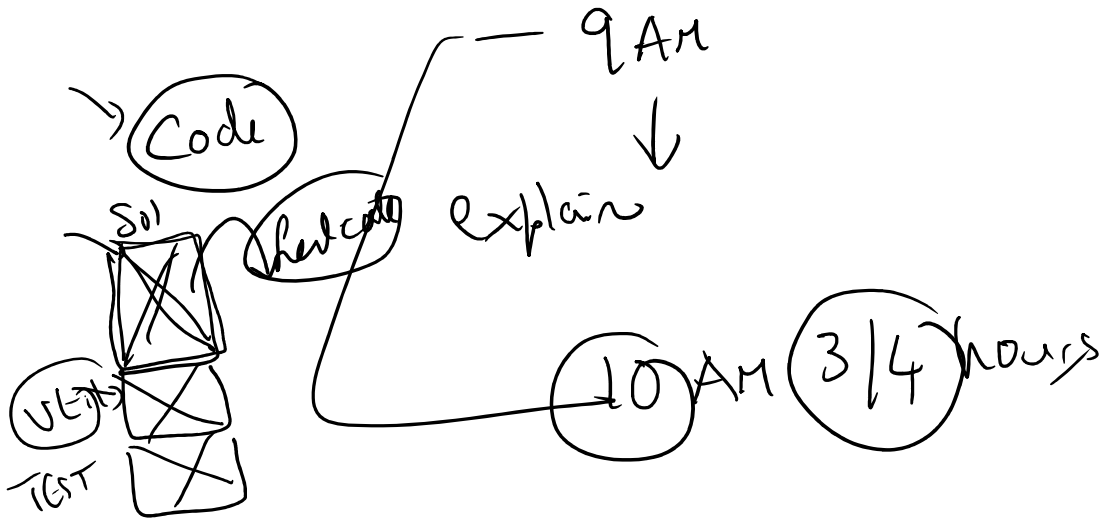


DAA PY 3

Saturday, January 27, 2024 12:04 PM

Midterm Exam is on March 3, 2024, Sunday, at 9 AM PST

March 3' 2024



Dynamic
ARRAY []

~~Append~~ $O(1)$ Amortized
~~Prepend~~ $O(n)$
 Insert- $a[i]$ $\Theta(1)$
 Find $\Theta(n)$
 Delete $\Theta(n)$
 Min $\Theta(n)$
 Max $\Theta(n)$

SList

~~append~~ $\Theta(1)$

~~Prepend~~ $\Theta(1)$

$O(n)$

$O(n)$

$O(n)$

$\Theta(n)$

$\Theta(n)$

STACK

Push $\Theta(1)$

X

Top $\Theta(1)$

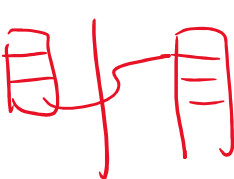


Pop $\Theta(1)$

$\Theta(1)$

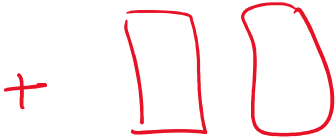
$\Theta(1)$

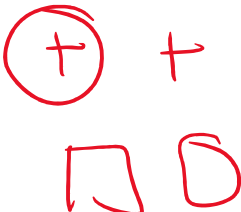
Queue

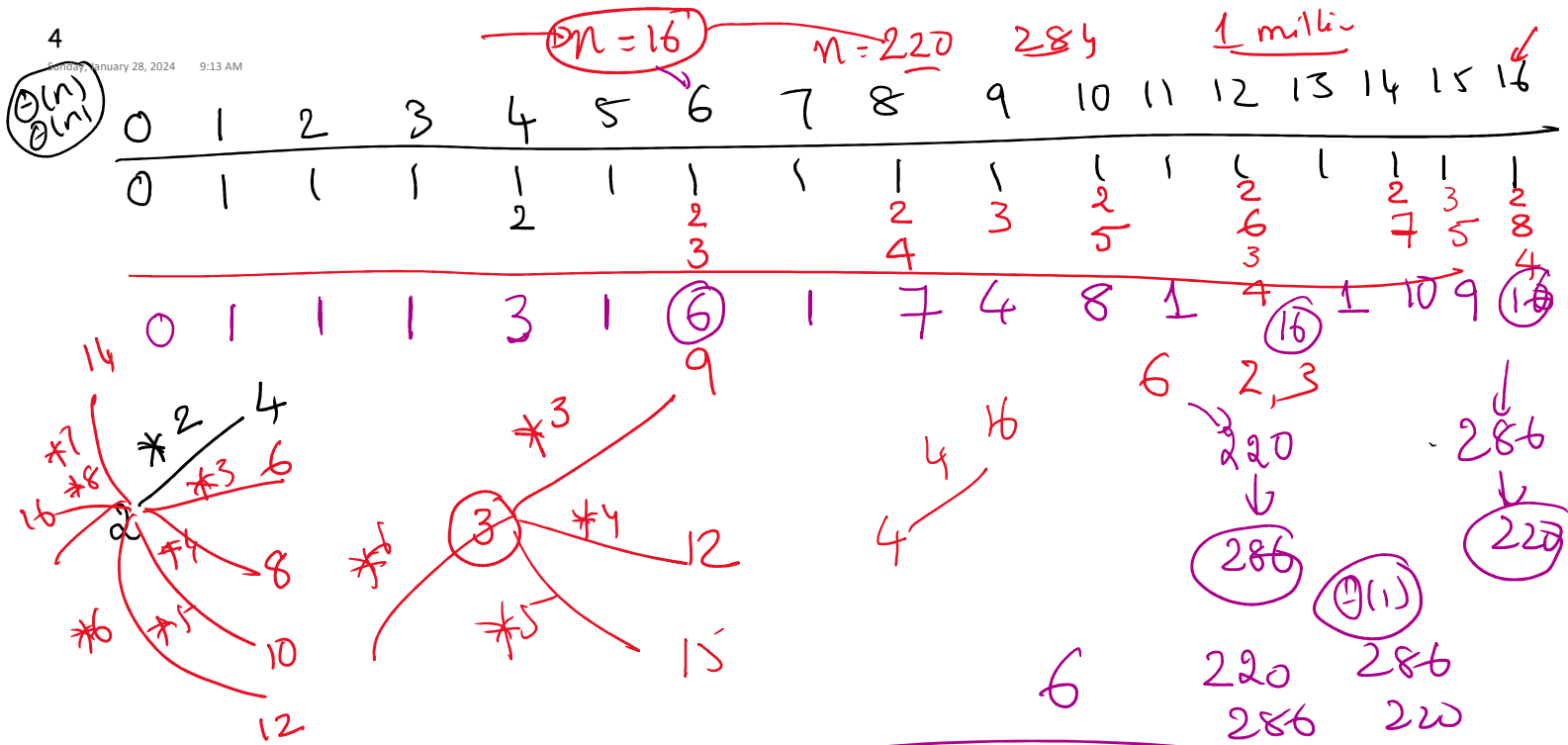
Sunday, January 28, 2024 9:06 AM

$\text{print}(d)$







$$\frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \dots + \frac{n}{\sqrt{n}}$$

$$n \left(\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{\sqrt{n}} \right)$$

$n($

HARMONIC SERIES

$$1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \approx \log_e n$$

```

    }
    return 0;
}

```

Total is: $O(n \cdot \sqrt{n})$

Run time for $n = 100000000$ is 6349.82 secs

_slow_work for $n = 100000000$ is 3245034969

fast_work = 828770743

Run time for $n = 100000000$ is 14.912 secs

_fast_work for $n = 100000000$ is 828770743

$$\%decrease = \frac{(828770743 - 3245034969)}{3245034969} = -74.4\%$$

$n \sqrt{n}$

n^2 $n \sqrt{n}$

2 3 ... n

$$100000000 * \log(100000000) \\ 100000000 * 8 \\ = 800000000$$

work = w

works

$n \log n$

```

def _compute_all_sum_of_factors(self):
    t1_start = process_time()
    n = self._n
    #Time: THETA(n)
    #Time: THETA(n)
    for i in range(n+1):
        self._a.append(1)
        self._increment_steps() #we are working
        self._a[0] = 0

    #Compute sum of all factors
    sqrt_of_n = self._u.sqrt_upper_bound(n)
    for i in range(2, sqrt_of_n, 1):
        k = i
        while (True):
            # This loops up to n/2
            # for n = 16, when i = 2: 2, 4, 6, 8, 10, 12, 14, 16
            # for n = 16, when i = 3: 3, 9, 12, 15
            # for n = 16, when i = 4: 4, 8, 12, 16
            self._increment_steps() #we are working
            j = i * k
            if (j >= n):
                break
            if (i == k):
                self._a[j] = self._a[j] + k # 2*2 = 4. So factor of 4 is only 2
            else:
                self._a[j] = self._a[j] + i + k; # 2*3 = 6. So factor of 6 is 2 and 3
            k = k + 1
    t1_stop = process_time()
    d = t1_stop - t1_start;
    print("Computing sum of all factors took CPU time in sec =", d)

```

n $n=10^6$



$\{1\} * n$

\sqrt{n}

$(n * n)k$

n

$n \log n$

$n=16$

6
1
2
3

4
1
2

$\frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \dots + \frac{n}{\sqrt{n}}$

$-a$



```

def _find_friends(self, i: 'int') -> 'int':
    self._increment_steps() # we are working
    ai = self._a[i]
    if (ai >= self._n):
        return 0
    aii = self._a[ai]
    if (aii == i):
        return ai
    else:
        return 0

```

for (i = 1 to 1 Bill)

270

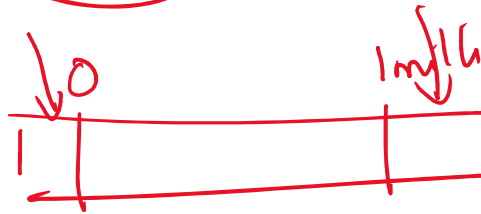
220

280

-a



Contra



a[i] Contra

Handwritten notes and code analysis:

```

def _alg(self):
    self._compute_all_sum_of_factors()
    for i in range(self._n):
        j = self._find_friends(i)
        if (j > i and j <= self._n):
            f = self._increment_number_of_friends()
            print(f, ":", i, j)

```

Annotations:

- 280** (circled) with an arrow pointing to `j = self._find_friends(i)`
- 220** with an arrow pointing to `if (j > i and j <= self._n):`
- 0** (circled) with an arrow pointing to `print(f, ":", i, j)`
- Prime** with an arrow pointing to **-a** (circled)
- 1** (circled) with an arrow pointing to `self._compute_all_sum_of_factors()`

Handwritten calculations:

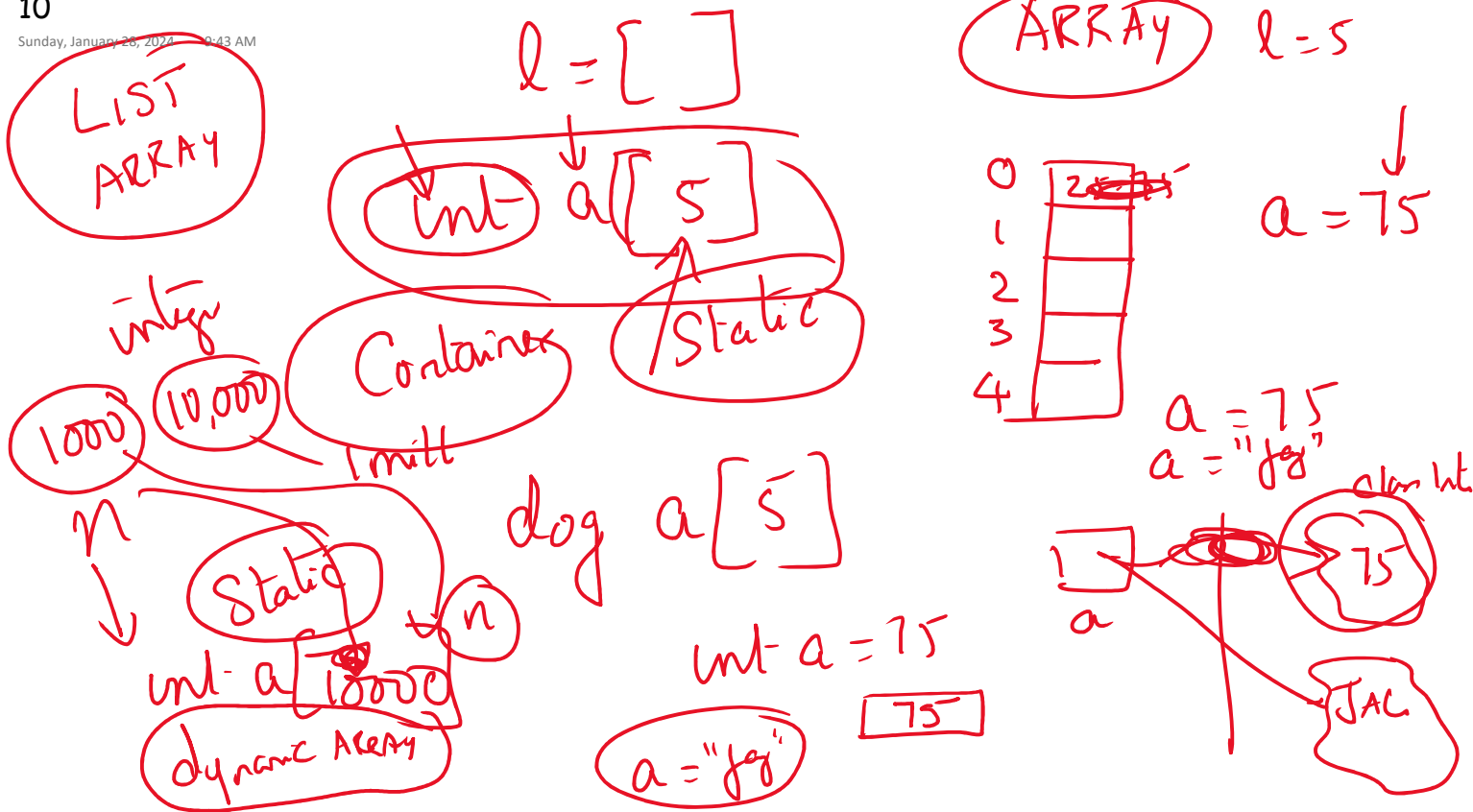
220	280
280	220

Handwritten diagram:

(6 6) → 220 280

~~280 220~~





LIST

$l = []$

K

$l.append(\text{prv.val})$

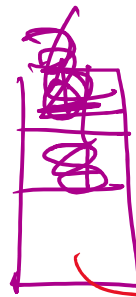
$l.append(20)$

(40)

(5)

(-50)

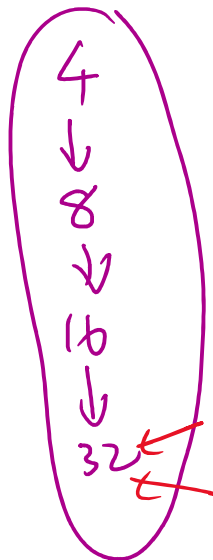
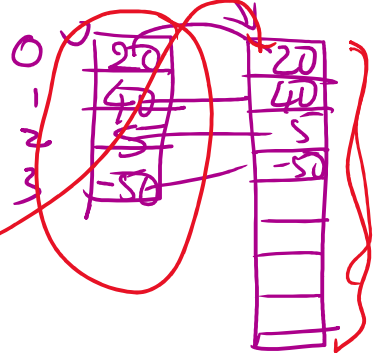
(88)



$4 \rightarrow 8$

CC

$O(n)$



(8)



$int - a[5]$

$0 \dots 4$



2000



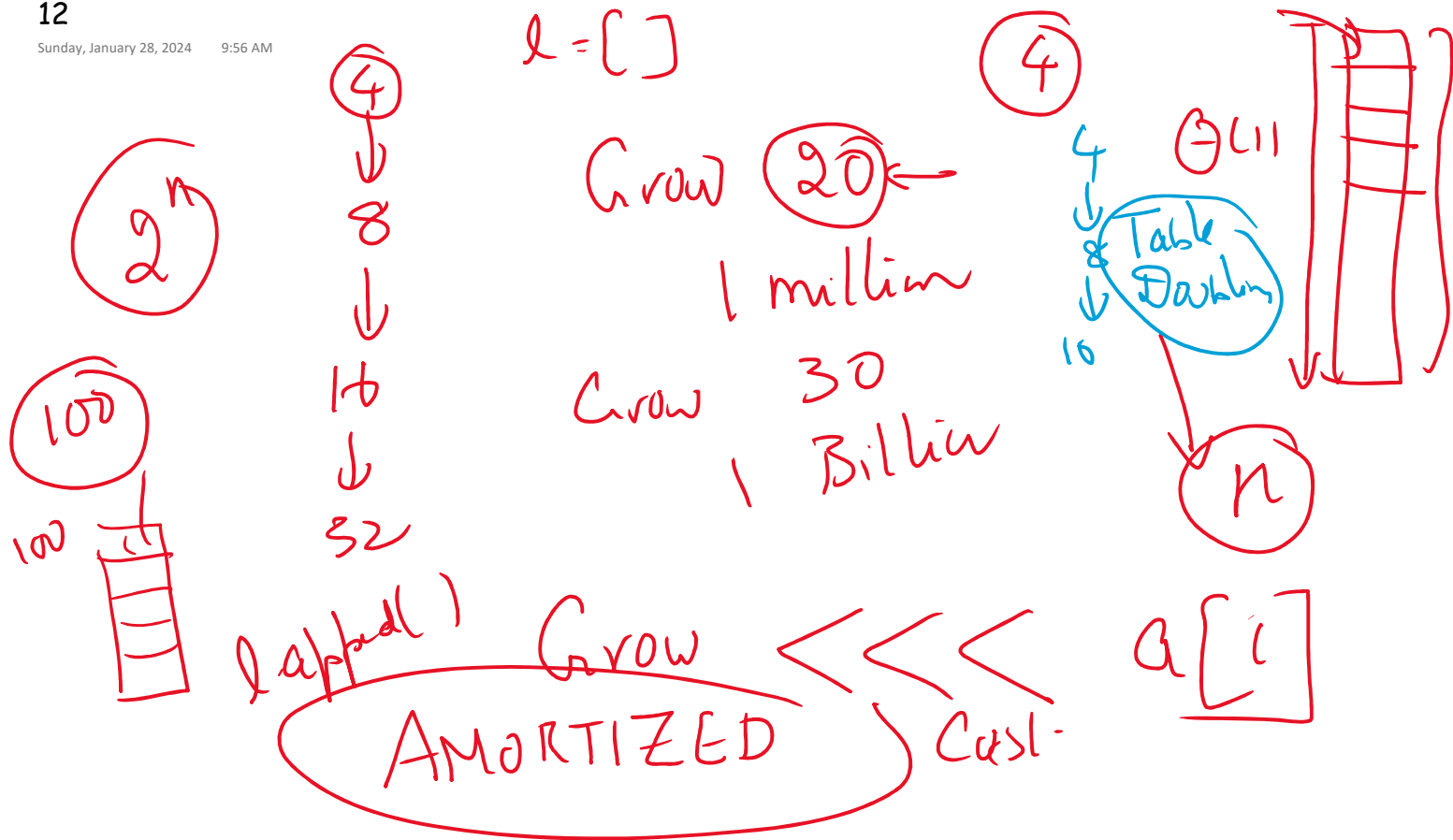
$2000 + 2$

$a[i]$
start + i

$0 \dots 4$

(1)

$\begin{cases} a[3] = 85 \\ a[2] \\ a[4] \end{cases}$



grow, append, find

```
def _test_int(self):
```

```
    # ----- Testing grow
```

```
    d = List() # My list
```

```
    l = [] # PYTHON LIST
```

```
    N = 10000
```

```
    for i in range(N):
```

```
        l.append(i * 100)
```

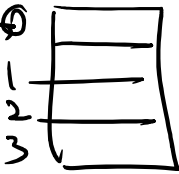
```
        d.append(i * 100)
```

```
    print(f'lines of list: {len(l)}')
```

Pos,

$l = []$
 $l.append(50)$

PYTHON LIST



✓ CLASS

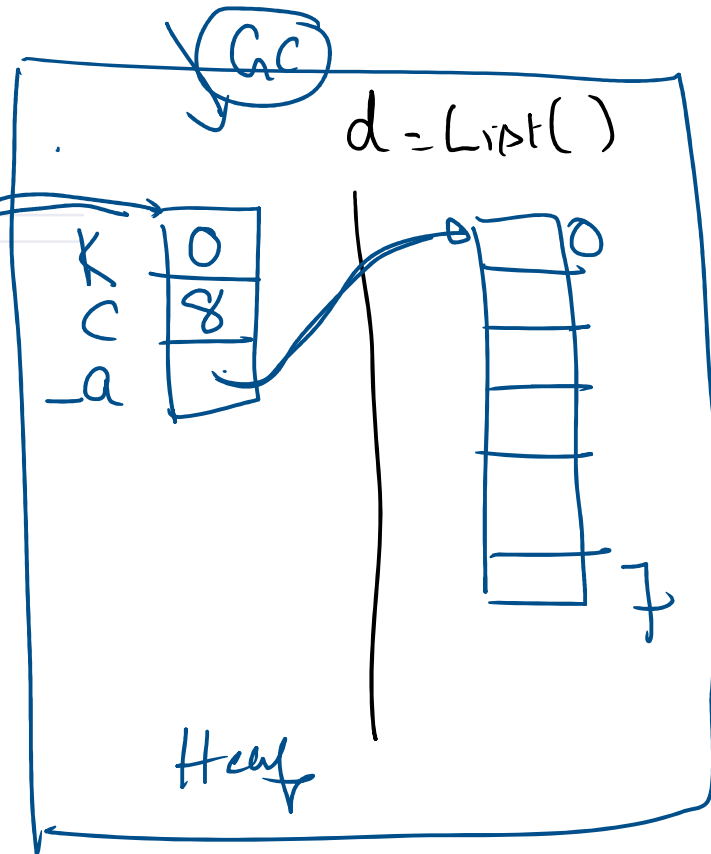
-k

```
#####
class List(object):
    def __init__(self):
        self._k = 0 ;
        self._capacity = 8
        self._a = self._allocate(self._capacity)
#####
```

d

{
d = List()

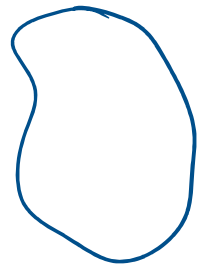
3



```
#####
# Time: O(1)
# Space: O(1)
#####
def allocate(self, new_cap):
    return (new_cap * ctypes.py_object)()
```

16 *

Low



④ 250

```
def _grow(self):
    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
```

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

With
Array list:

Shallow C

(20) $2^{20} = 1\text{milli}$
 $2^{30} = 1\text{Billion}$

k	4
e	8

0	2
1	100
2	200
3	-4

⑧

os | 4

ns | 8

b | -

2	0
100	
200	
-4	
	7


```
def __contains__(self, item) -> 'int':
```

```
    for i in range(self._k):
```

```
        if (self._a[i] == item):
```

```
            return True
```

```
    return False
```

[20 →]

LOCK 1

< nsh

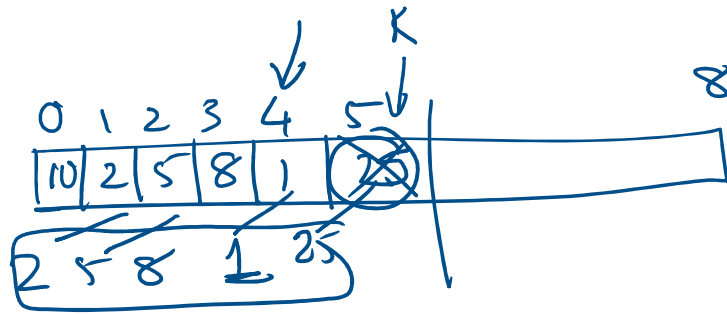
$O(n)$

n

100

120

$K = 5$



deleting LAST $O(1)$

$K = K - 1$

delete first

-- stn --

```

In [10]: IPYTHON IN LIST
----- 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]
  
```

$[1, \cancel{2}, 99, \cancel{2}]$
 $[1, 99]$

$[1, 99, 2]$

```
# [1 99 ]
```

```
# n = [1,1,1,1]
```

```
# item = 1
```

```
# []
```

```
#####
```

```
def deleteBAD(self, item):
```

```
l = List() # Note my list. Not Python list.
```

```
n = len(self)
```

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

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

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

```
    vi = self[i]
```

```
    if (vi != item):
```

```
        l.append(vi) #Space: O(n)
```

```
#WHY WE SHOULD NOT DO THIS
```

```
#self = l #This copies pointers. USELESS
```

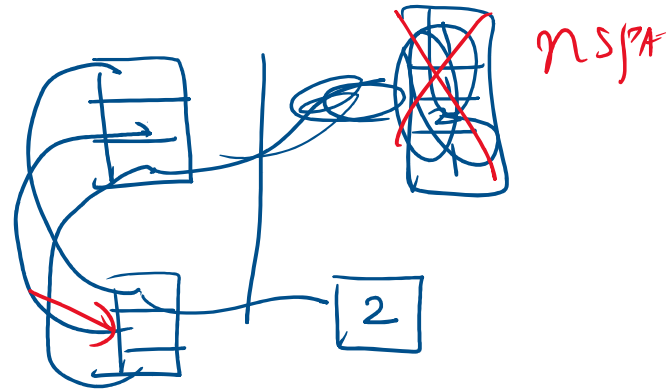
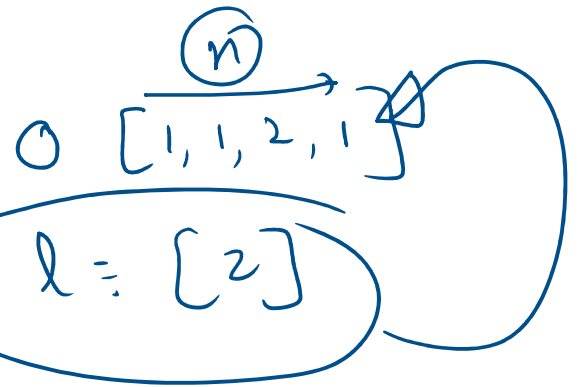
```
self._k = l._k ;
```

```
self._capacity = l._capacity
```

```
self._a = l._a ;
```

```
#old self._a will be removed by garbage collector
```

$O(n)$ SPACE

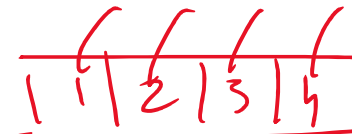
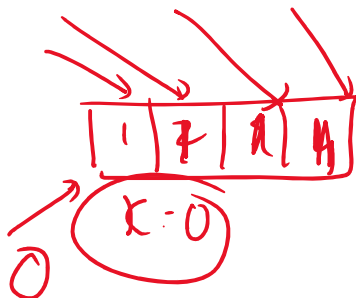
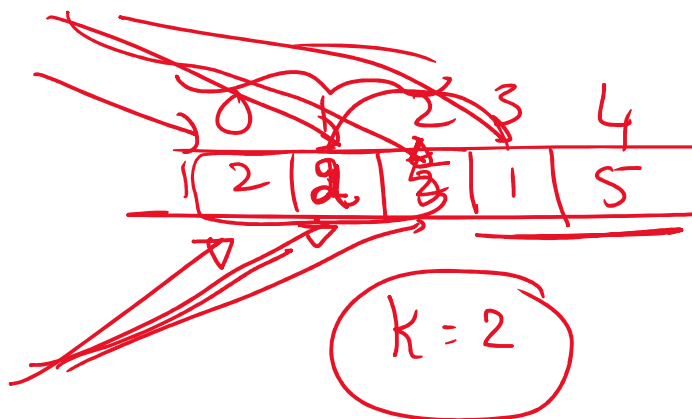


```

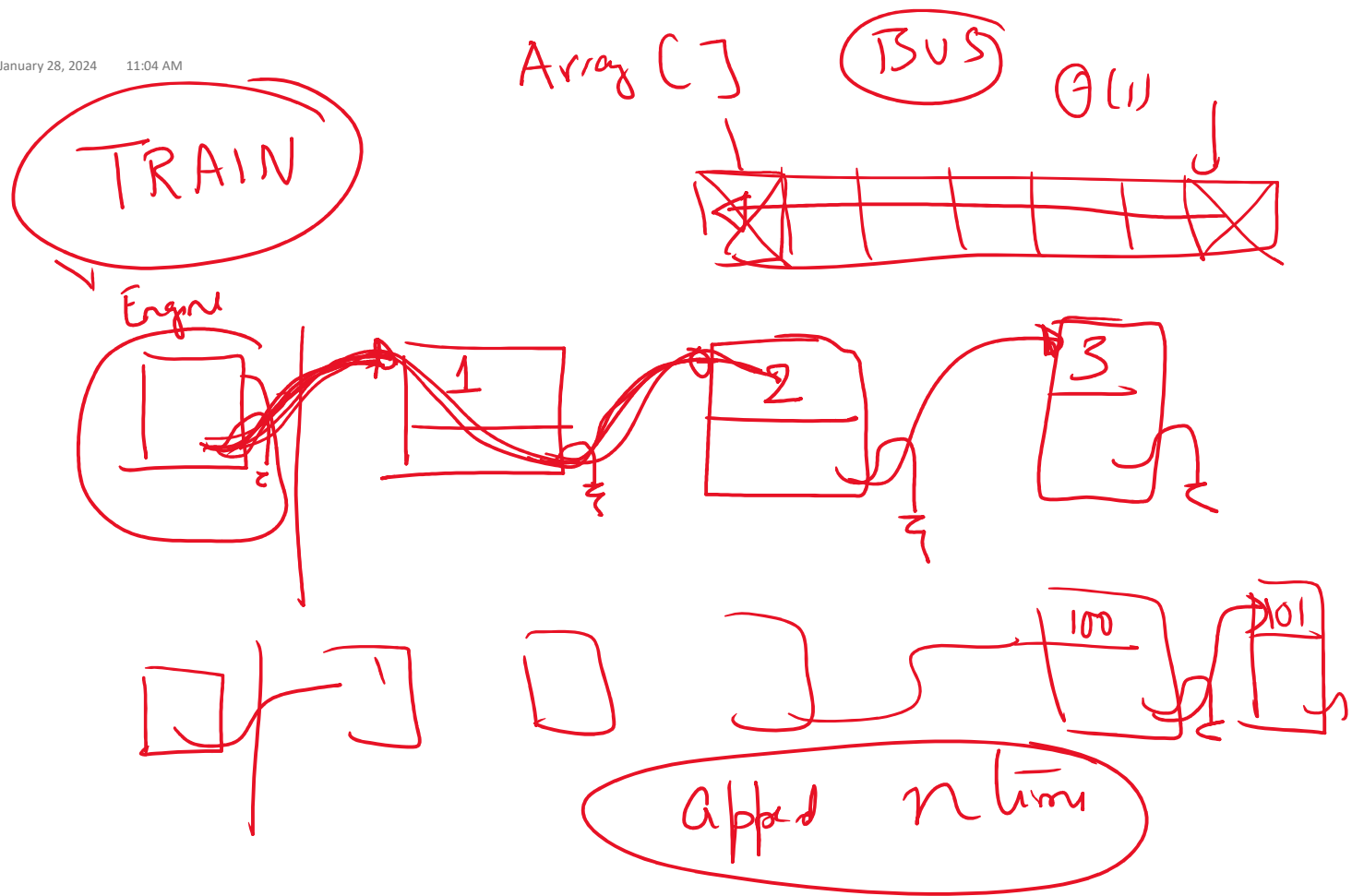
def delete(self, item):
    n = len(self)
    #Space: O(1)
    #Time: (n)
    k = 0
    for i in range(n):
        vi = self[i]
        if (vi != item):
            assert(k <= i)
            self[k] = vi
            k = k + 1
    self._k = k

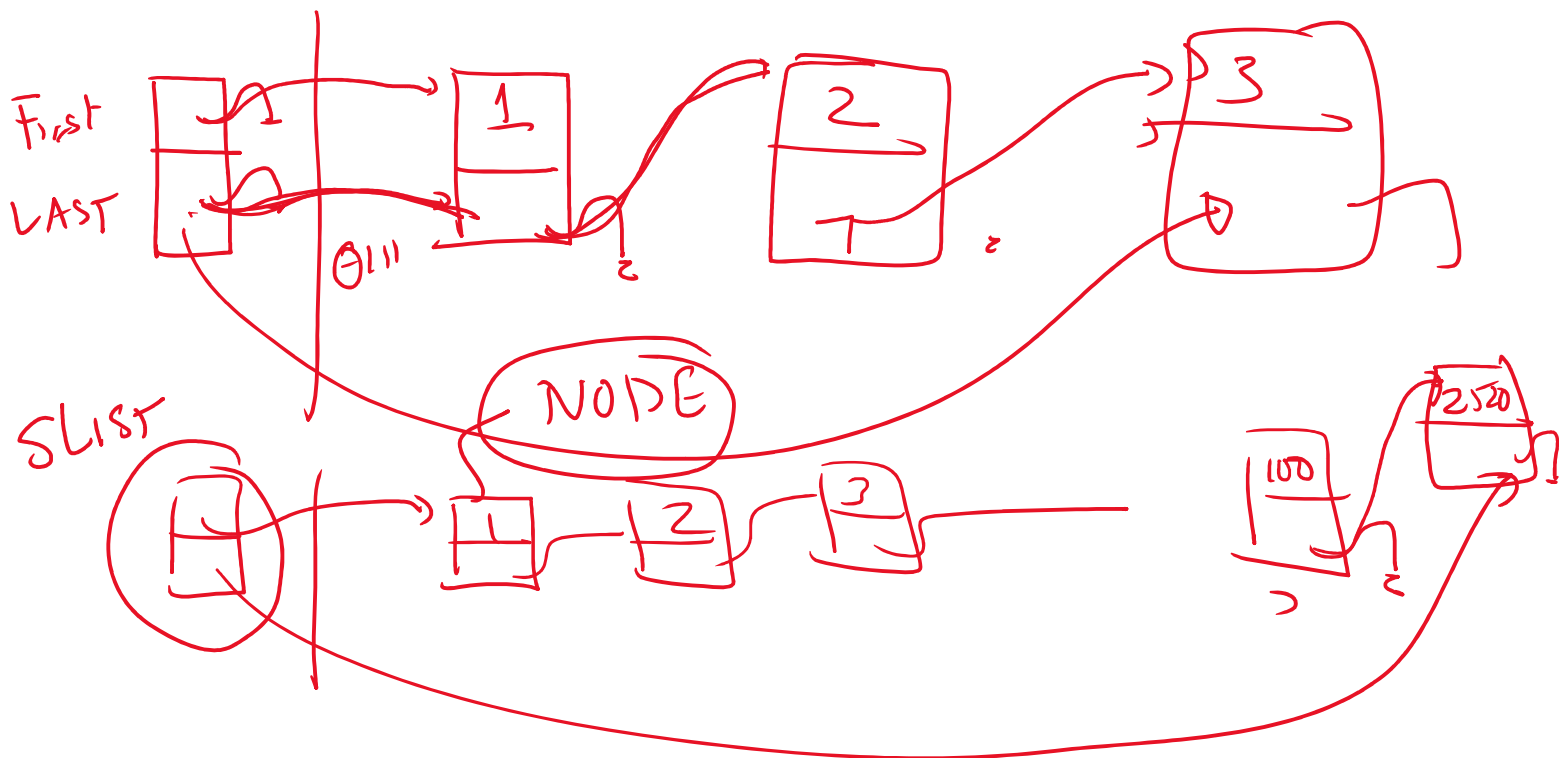
```

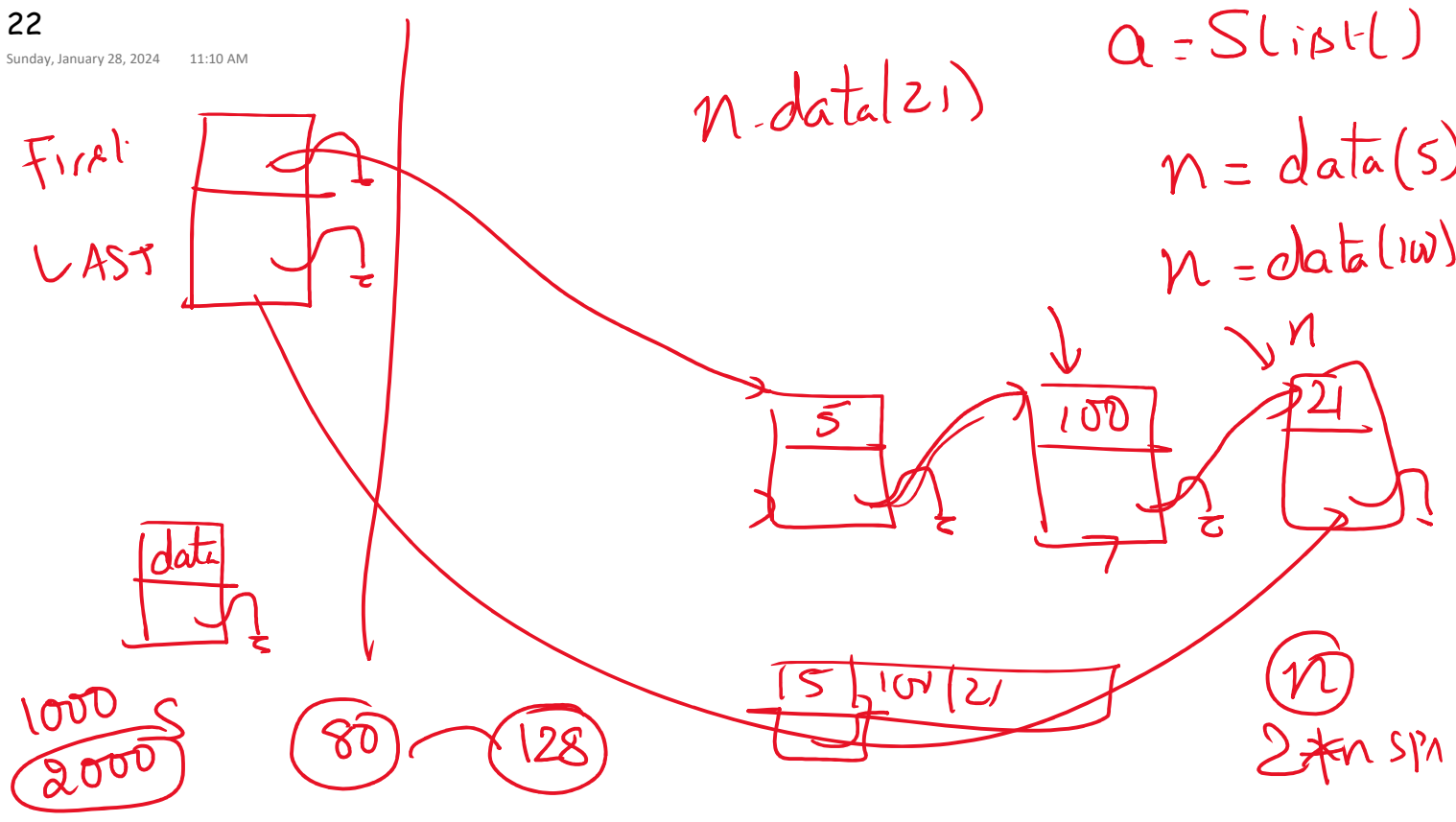
In place

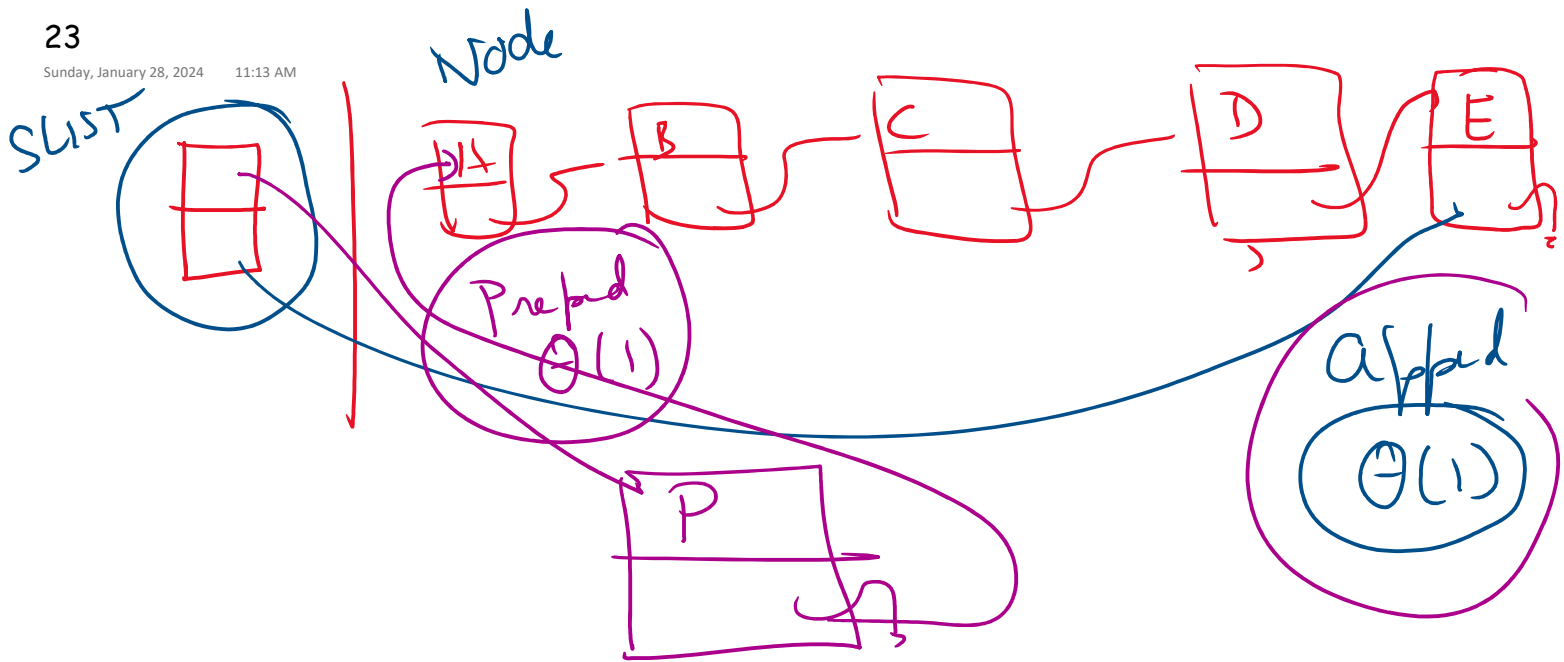


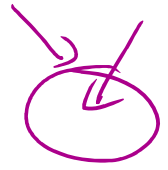
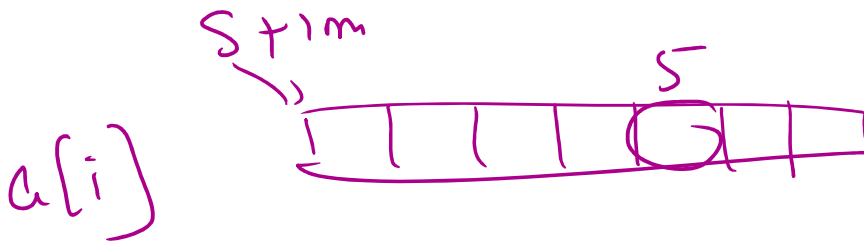
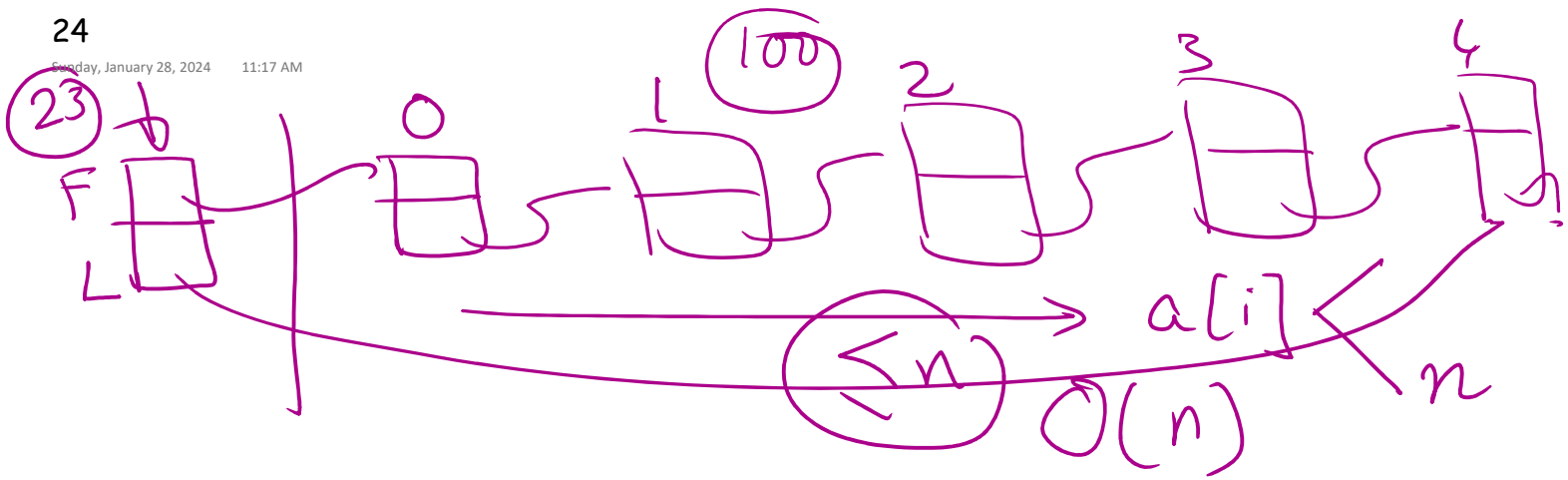
5

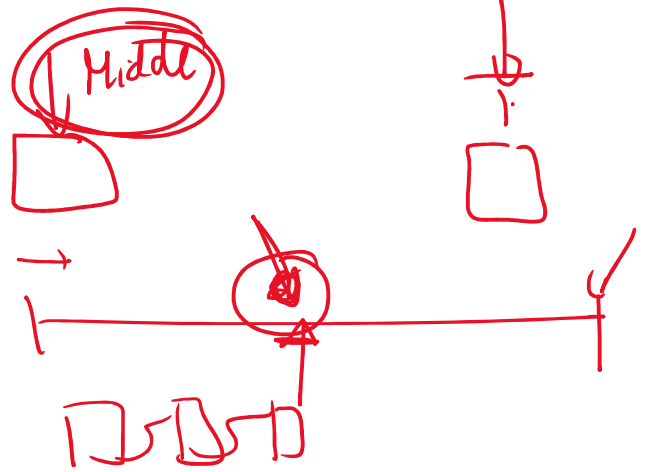
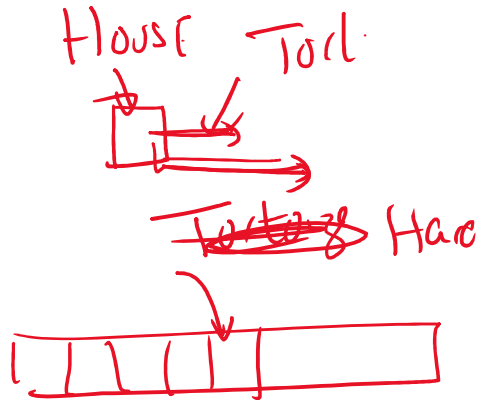
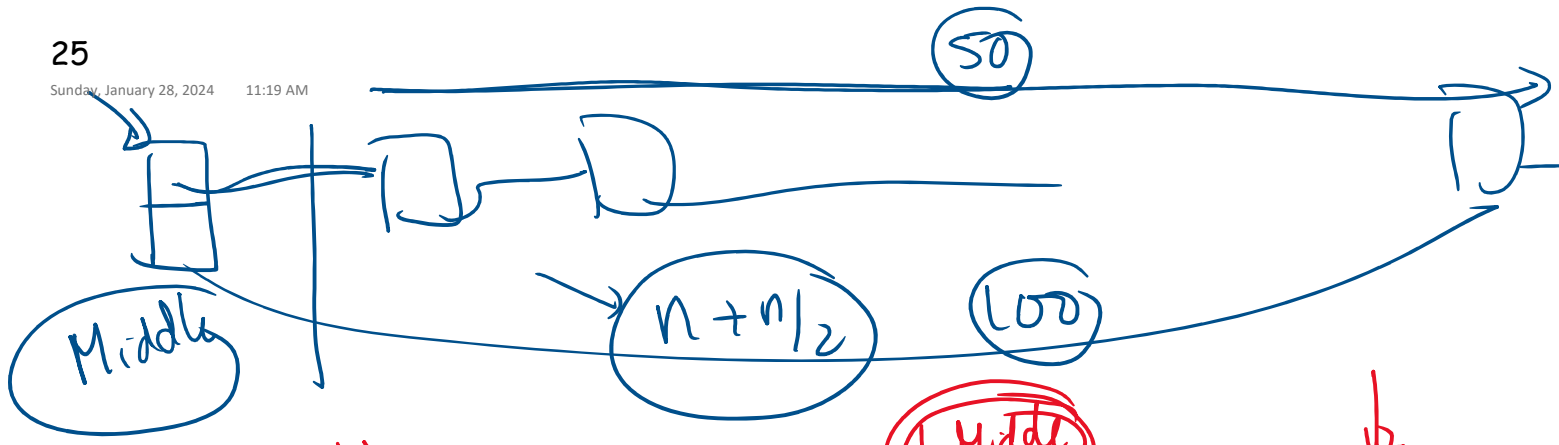






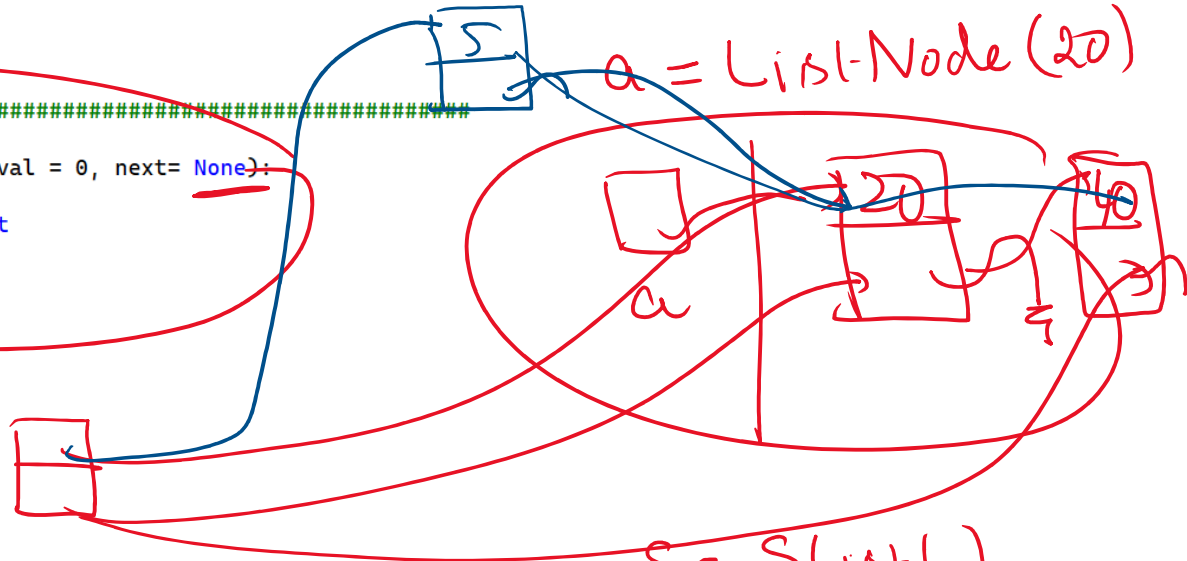






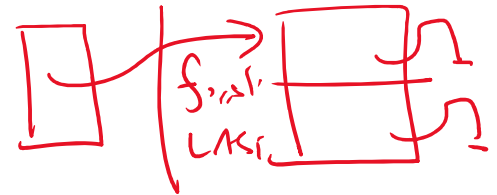
```
#####
class ListNode:
    def __init__(self, val = 0, next= None):
        self.val = val
        self.next = next
```

$a = \text{List-Node}(20)$



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

$S = \text{Slist}()$



--Slist-- Print(s)

```

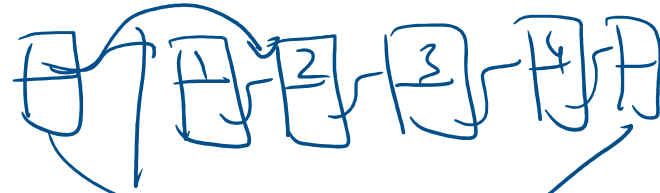
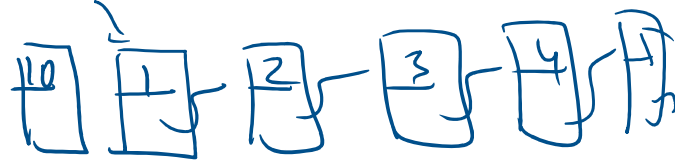
p3.9.7 (default, Sep 10 2021, 10:59:28) [MSC v.1916 64-bit (AMD64)]
>----- test append/prepend/find -----
[1, 2, 3, 4, 5] is stored as 1->2->3->4->5->NULL
size of slist = 5
After prepending [10, 11, 12] s looks like 12->11->10->1->2->3->4->5->NULL
size of slist = 8
Find if there passed
Find if NOT there passed

```

```

----- test delete -----
[1, 2, 3, 4, 5] is stored as 1->2->3->4->5->NULL
Size of s = 5
After removing 1 : 2->3->4->5->NULL
After removing 5 : 2->3->4->NULL
After removing 3 : 2->4->NULL

```



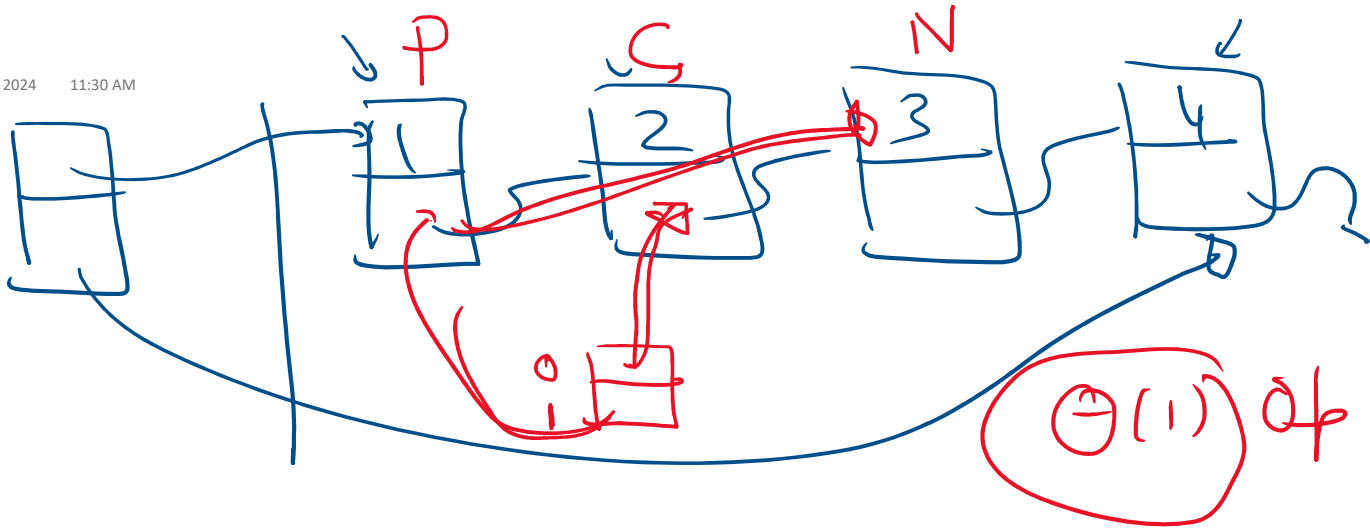
2 → 3 → 4 → 5

2 → 3 → 4 → null

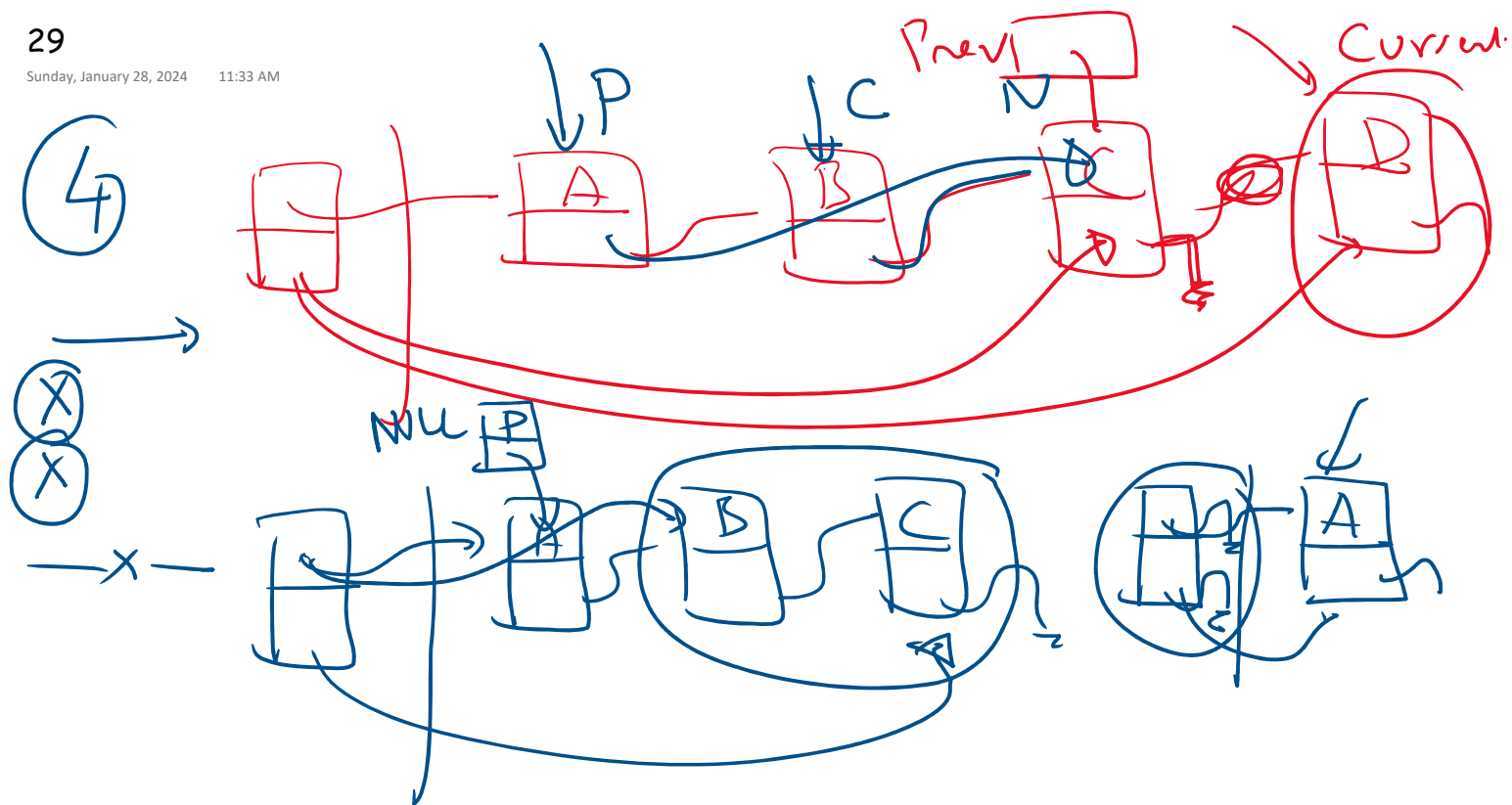
2 → 4 → null

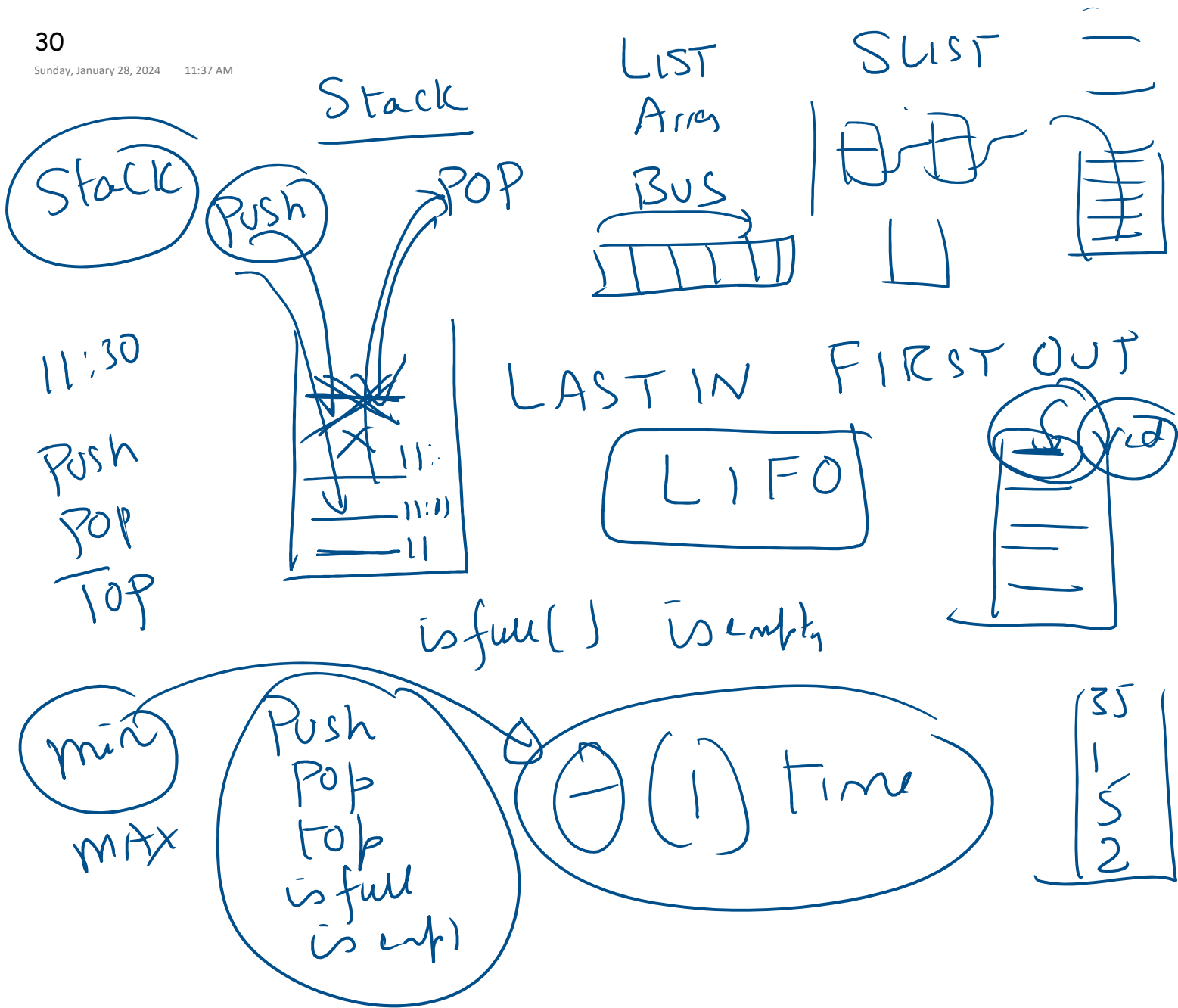
28

Sunday, January 28, 2024 11:30 AM



Find
 $O(n)$





Stack using Python List
Note Stack has infinite size

```
class Stack(self):  
    def __init__(self):  
        self.__a = []  
ALL operation must be CONSTANT  
def push(self, T):  
def pop(self)->'T':  
    #return None is empty  
    #else return T  
def top(self)->'T':  
    #return None if empty  
    #else return T  
def empty(self)->'Bool':  
def is_full(self)->'Bool':  
def space(self)->'int':  
    #Max space used in entire life  
    return self.__maxspace  
def __len__(self)->'int':
```

The image shows a handwritten implementation of a Stack using a Python List. The code is written in a class named `Stack`. The `__init__` method initializes an empty list `self.__a`. The `push` method is defined but its body is not visible. The `pop` method returns the top element or `None` if the stack is empty. The `top` method returns the top element or `None` if the stack is empty. The `empty` method returns a boolean value. The `is_full` method is defined but its body is not visible. The `space` method returns the maximum space used in the entire life of the stack. The `__len__` method is defined but its body is not visible. The code is annotated with blue circles and arrows highlighting specific parts: the list initialization, the `pop` and `top` methods, the `empty` method, and the `space` method. A note at the top states "Stack using Python List" and "Note Stack has infinite size". A note in the middle states "ALL operation must be CONSTANT".

Figure 7.1: class Stack

Self. $a = []$

11

None

0 1 2 3

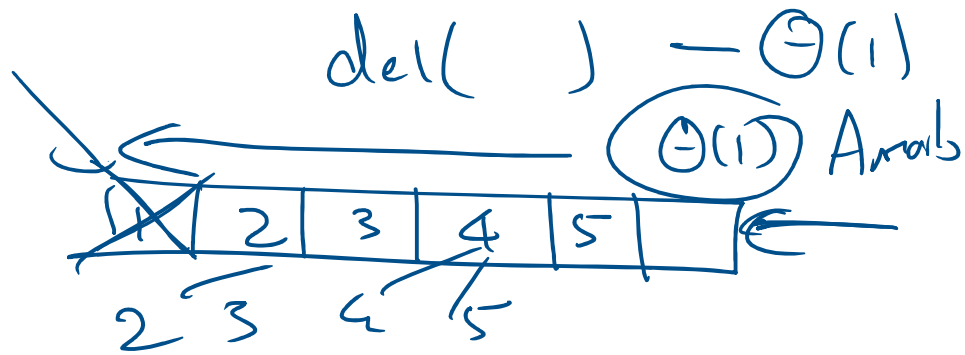
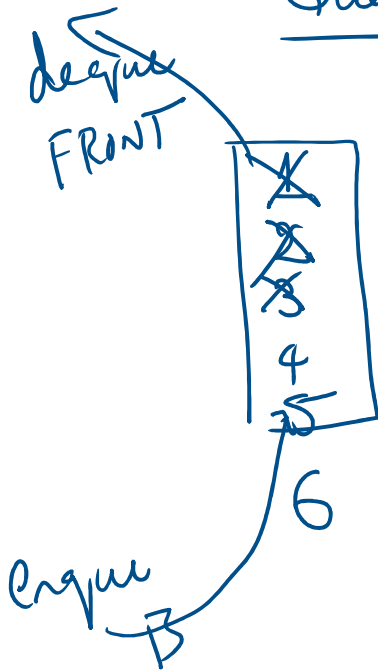
5	10	11	2		
---	----	----	---	--	--

$$\begin{bmatrix} 2 \\ 11 \\ 10 \\ 5 \end{bmatrix}$$

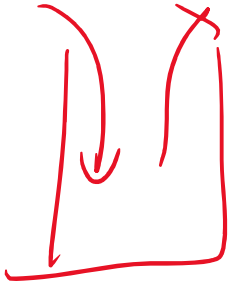
Push
POP
TOP
is up
is full

→ min
 $\Theta(1)$



Queue

enqueue
dequeue



PUSH

POP

a(i) TOP

isfull

is empty

min

max



enqueue
dequeue

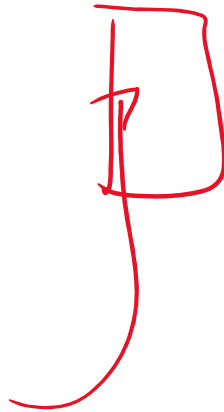
FRONT
BACK

isfull

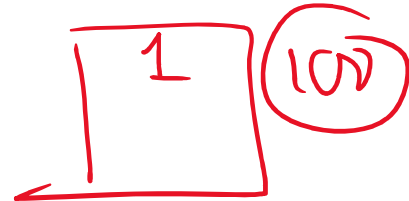
is empty

min

max



Queue using Python List
 Note Queue has a known size n
 class Queue(n:'int'):
 def __init__(self):
 self._a = []
 ALL operation must be CONSTANT
 def enqueue(self, T)->'bool':
 # False means Queue is full
 # True means enQueued
 def dequeue(self)->'bool':
 # does not return T
 # True: dequeued
 # False: Queue was empty
 def Front(self)->'T':
 return -1 if Queue is empty
 else return T
 def Rear(self)->'T':
 return -1 if Queue is empty
 else return T
 def isEmpty(self)->'Bool':
 def isFull(self)->'Bool':
 def __len__(self)->'int':



-- len --