Lecture 2

A $\qquad\qquad\qquad \Theta(n)$

max element

" pseudo-code "

=, <, >,

for, while, return

functions

$\overbrace{max = A[0]}$ +1

for (i=1; i<n; i++) $\cdot$ 2n+1

→ if $\overbrace{A[i] > max}^{+1}$

⊛ $\underbrace{max = A[i]}$ +1 ←

output max. +1

Invariant :

when $i = k$,

max is the maximum of

$A[0] \dots A[k-1]$.

$\underbrace{\text{Suppose max is the max. of } A[0] \dots A[k-1].}$

max. of $A[0] \dots A[k-1]$

Consider k^{th} iteration.

Sps $A[k] >$ "max"

we update max to $A[k]$

Let us count # elementary operations:

$1 + 2n + (n-1) + f + 1$ ← assumption: all elementary opns take 1 unit of time.

⊛

$0 \le f \le n-1$

$= \cancel{2n}\ 3n + 1 + f \le 3n+1 + n-1$

$= 4n = O(n)$

$O(n^2)$

$O(n^3)$!

① All elementary opns take 1 unit

② Look at the worst case.

③ Only worry about large n, and ignore constants.

10^{100} $\qquad \frac{1}{100000} n^3$

$\cancel{1000}\, n$ $\quad \frac{1}{10} n$: care about large value of n.

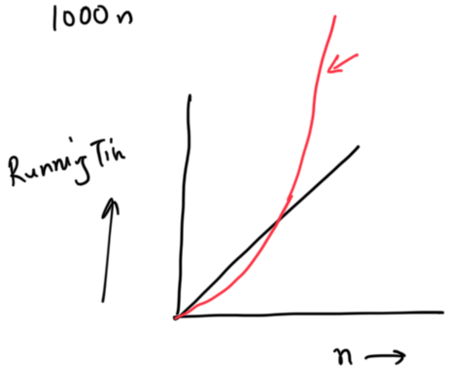
$n = 10{,}000 = 10^4$ $\quad \simeq 10^7$

$$\frac{1}{100} n^2 + 100n \approx \frac{1}{100} n^2 \approx \text{scales as } n^2$$

$$= \underbrace{O(n^2)}$$

Big "O" notation.

$1000n$ $\quad \frac{1}{10} n^2$



Definition: (Big "O" notation).

$f(n)$, $g(n)$ are two non-negative functions of $\mathbb{N}$

$100n$ $\quad n$

We say that $f(n) = O(g(n))$ if there exist constants c, N_0 such that

$$f(n) \leqslant c \cdot g(n) \quad \text{for all } n \geqslant N_0$$

" f grows slowly than g "

Is $f(n) = O(g(n))$?

1. $f(n) = 100n$, $g(n) = n$

$c = 100$, $N_0 = 1$ $\leftarrow$ $c = 500$

$$f(n) \leqslant 100\, g(n), \quad n \geqslant N_0$$

(2) $f(n) = 5n^2 + 10n + 7$, $g(n) = n^2$

$f(n) \leq 30\, g(n)$ $\quad n \geq 1.$

$c = 30$, $N_0 = 1$.

(3) $f(n) = n$ $\quad g(n) = n^2$

$f(n) \leq 1 \cdot g(n)$, $n \geq 1$

$c = 1$, $N_0 = 1$

(4) $f(n) = n$ $\quad g(n) = \begin{cases} 0 & \text{if } n \text{ is even} \\ n & \text{if } n \text{ is odd.} \end{cases}$ $\quad \times$

If $f(n) = O(g(n))$

$f(n) \leq c \cdot g(n)$ $\quad n \geq N_0$

(5) $f(n) = n$, $\quad g(n) = \begin{cases} 0 & \text{if } n \leq 1000 \\ n & \text{if } n > 1000 \end{cases}$

$c = 1$, $N_0 = 1000$

Lecture 3

$$4n + 5 = O(n) = O(n^2) = O(n^3) = O(2^n).$$

<u>Def</u> ("Ω") $f(n)$, $g(n)$ are two non-negative function of $\mathbb{N}$.
We say that $f(n) = \Omega(g(n))$ if
there exist c, N_0 such that

$$f(n) \geq c \cdot g(n) \quad \text{for all } n \geq N_0.$$

<u>eg</u>: (1) $f(n) = n^2$, $\quad g(n) = 10n$

$n^2 \geq 10n$ $\quad \forall\, n \geq 10$

$c = 1, \quad N_0 = 10$

If $f(n) = O(g(n))$ then $g(n) = \Omega(f(n))$

A, x Does x appear in A.

```
for   i = 0 ... n-1
        if  (A[i] == x )
                 return  TRUE

return FALSE
```

Best case $O(1)$

Worst $O(n)$.

$I_1, \; I_2, \ldots, \; I_K$

$\max(T_1, \ldots, T_K) \geq cn$

Example: $f(n) = n^2$

$f(n) = \Omega(n^2)$

$\Omega(n)$

$\Omega(n \log n)$

$\Omega(\sqrt{n})$

Is it possible that

$f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$?

$f(n) = n^2 \qquad g(n) = 3n^2 + 10n$

$f(n) = \Theta(g(n))$.

$2^{\sqrt{\log_2 n}} \qquad n$

$f(n) \qquad g(n)$

1 Is $f(n) = O(g(n))$? ✓

$$2^{\sqrt{\log n}} \leq 2^{\log n} = n$$

② $\log_2 n = O(\log_{10} n)$?

$$\log_a n = (\log_a b) \log_b n$$

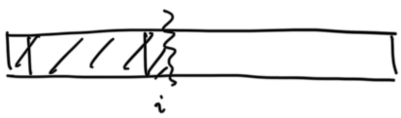
$\log_a n = \Theta(\log_b n)$

$n \log n$

③ $2^{\sqrt{\log_{1.5} n}} = O(n)$?

$\log_2 n,\ n,\ n\log_2 n,\ n^2$

Insertion Sort.



A[0] ... A[i] is sorted

↓

A[0] ... A[i+1] is sorted.

A[0]			A[i]	A[i+1]	
1	5	7	9	15	8

p

8 15

8 9

for $i = 0$ to $n-2$ ←

p = i+1 ← 1

$6i+1$ while (p > 0 && A[p] < A[p-1])

swap (A[p], A[p-1]) ← 3 } ≤ 6i
p-- ← 1

$$\sum_{i=0}^{n-2} (6i+1) = c \cdot n^2 + c'n + c''$$

$$= O(n^2)$$

$\Omega(n^2)$?

Worst case running time is $\Theta(n^2)$.

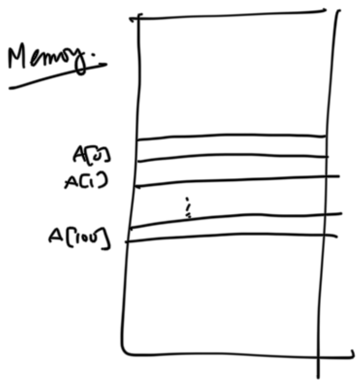
Python Programming (Chapters 1, 2).

ABSTRACT DATA TYPE

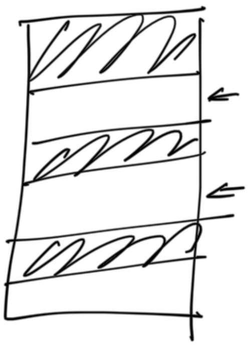
ARRAY : store an ordered sequence of elements

Access i^{th} element and modify it $O(1)$

A[i]

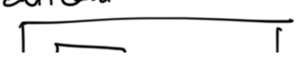


int A [100]



Lecture 4

CreditCard



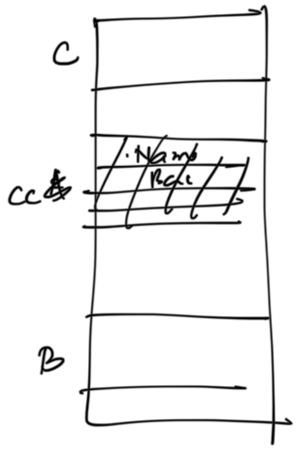
Abstraction.

Credit Card

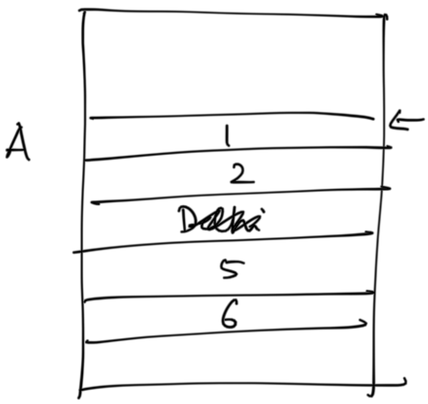
Class CreditCard

Objects

$\boxed{\text{int } x;}$



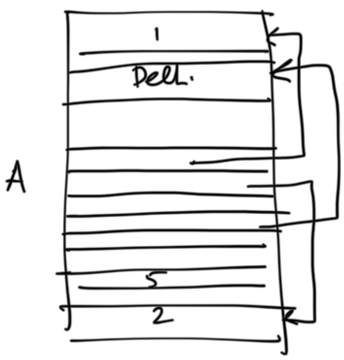
CreditCard A, B, C



integer : 32 bits

A[i] ~~int 32~~

X + 32i



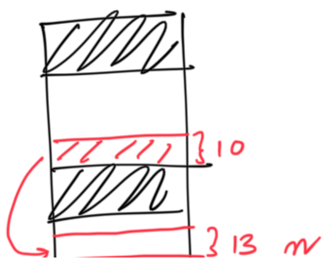
Python List

A = [1, 2]

A[i]

A = [1, 2, 3, 4, 10, 7, 8]

15



A = [1]

k

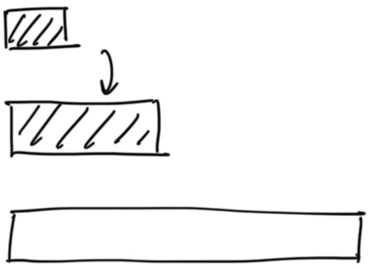
4

$$2 \downarrow i$$
$$i+1$$

$$\sum_{i=1}^{k} i = \Theta(k^2).$$

Lecture 5

for $i = 1, 2, \ldots, N$
    $A[i]$ = values

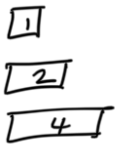
A


$\rightarrow 10 + 20 + \ldots + 10k$
$= 10k^2 = n^2/10$

$N = 2^k$

$1, 2, 4, 8, \ldots$

$1 + 2 + 4 + 8 + \ldots + 2^i + \ldots + 2^{k-1}$



$1 + 2 + 4 + \ldots + 2^{k-1} \leq 2^k \leq 2N$

$O(N)$ $\qquad 2^{k-1} \leq N < 2^k$

Total time $\leq 3N$
on an average, each operation takes 3 units of time
$O(1)$.

f( ) {

g( ) ←

n

h( ) {

k( )

2

}

g( ) {

h( )

}

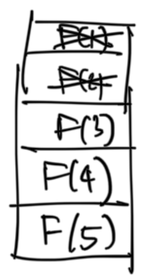
Last In First Out

Recursive Functions

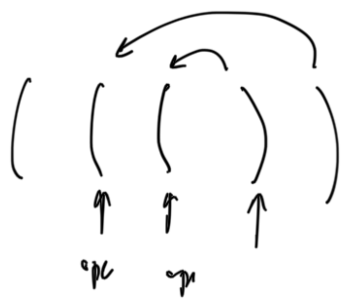
Factorial(n):

if n==1 return 1
else
return n * Factorial(n-1).

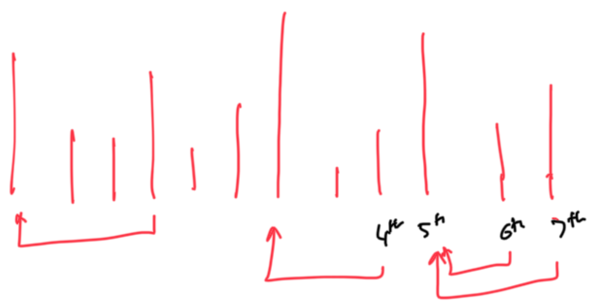
$O(n)$



x = 1
for i = 1 ... n
x = x * i



Lecture 7: Stacks.



$X[\ ]$: $X[i]$ price on day i

$S[i] = 4$

$S[i]$: # days we have to go back to find the first day when the prices are above $X[i]$

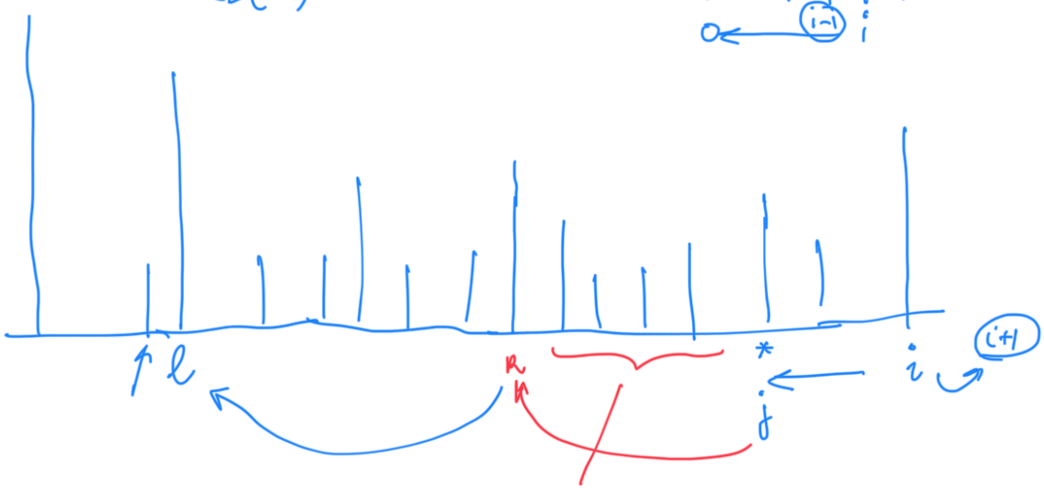
Obvious Alg: $X[0] = \infty$ $\quad X[1], \ldots, X[n]$

```
for i=1 ... n                  <- n itns
    /* to compute S[i] */
        j = i-1 ;  S[i] = 1
        while ( X[j] <= X[i] )     } <= n
            j--
            S[i]++
```

$O(n^2) \to O(n)$?

$\Omega(n^2)$?

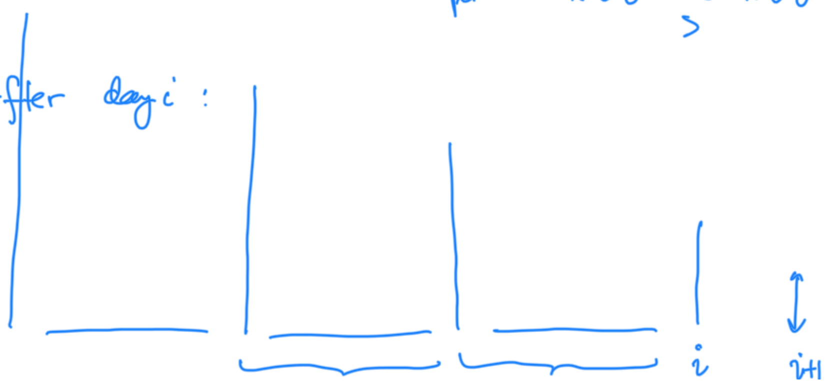
$\sum_{i=1}^{n} i \geq \frac{n^2}{2}$.

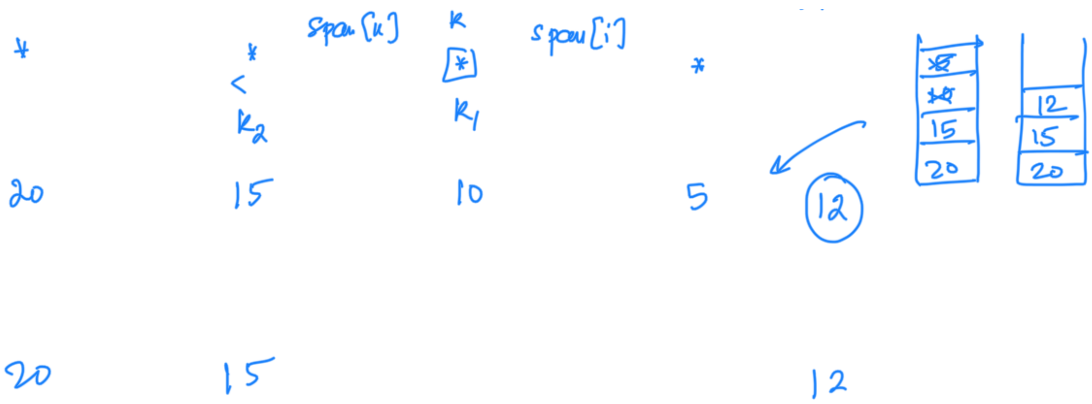


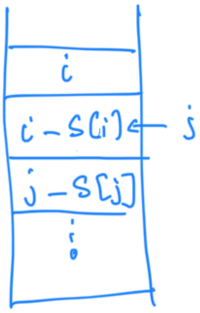
all days here have price $\leq X[j] < X[i]$

compare $X[i]$ and $X[k]$.
$>$

After day i :







After day [i]

	1	2	3	4	5	6	7
∞	25	11	8	7	16	5	10

A.push(X[0])

for i = 1 ... n

while (X[A.top] ≤ X[i+1])

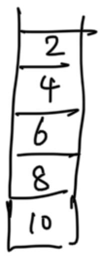
A.pop(); ←

A.push(i+1)

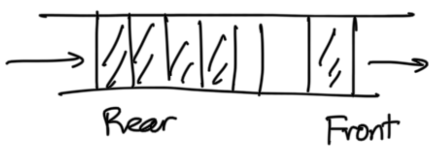
Only n push operations

⇒ ≤ n pop "

⇒ O(n) time



STACK

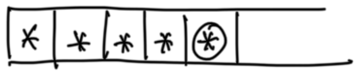


QUEUE

LIFO FIFO

ARRAY

Stack



Queue

Enqueue(o)

R++

A[R] = o

Dequeue()

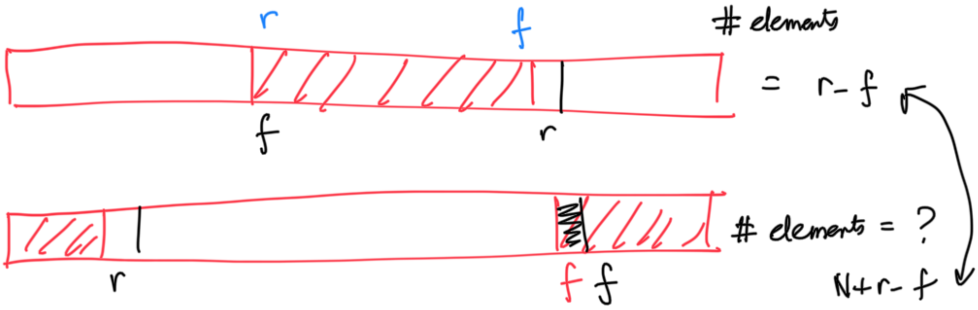
o = A[F]

F++

return o

Lecture 8

A



elements $= r - f$

elements = ?

$N + r - f$

$(N + r - f) \bmod N$

✓ Enqueue(x)

$A[r] = x$

$r = (r+1) \bmod N$ → if $r \neq N-1$
$r = r+1$
else $r = 0$

$O(1)$.

✓ Dequeue()

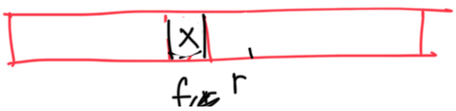
if $(f == r)$ X

$o = A[f]$

$f = (f+1) \bmod N$

Condition for Q to be full := $f = (r+1) \bmod N$

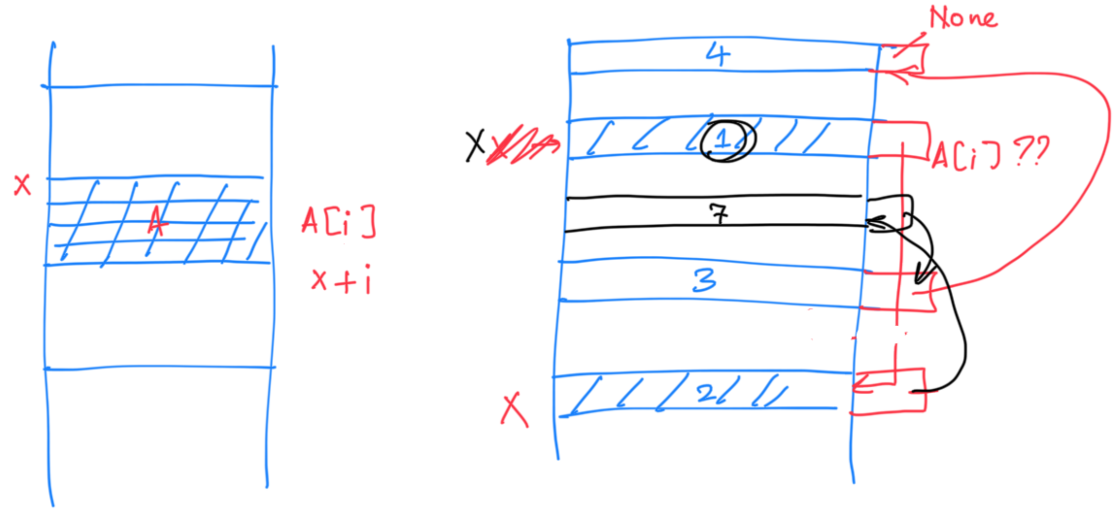
Queue is empty:
$f = r$



Stacks
Queues

Arrays

Linked Lists

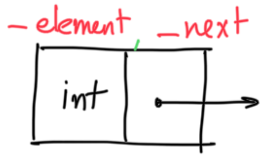


Array   Linked List

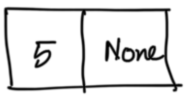
```
class Node :

    def __init__(self, element, next) :

        self._element = element ;
        self._next    = next    ;
```



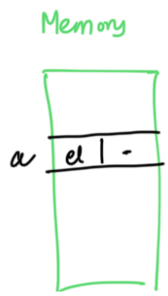

```
Node(5, None)
```

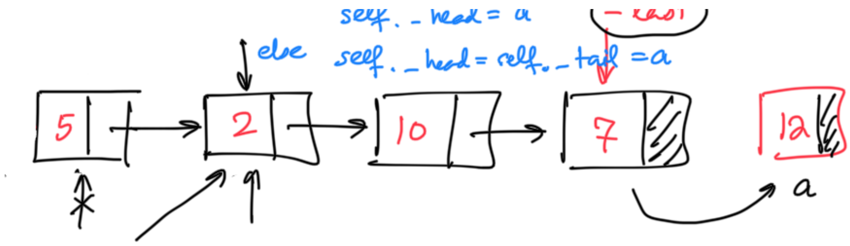



```
class LinkedList :

    def __init__(self) :          ] constructor
        self._head = None
        self._tail = None

    def _insertFirst(self, element) :
        a = Node(element, self._head)
        if self._head ≠ None
```





_head · _head.next

```
def _removeFirst(self)
    self._head = self._head.next
    if (self._head == None)
        self._tail = None
```

if (self._head ≠ None)

$O(1)$ ||

```
def _insertLast(self, element):
```

① Find the last node:

```
    p = self._head
    while (p.next != None)
        p = p.next
    last = p
```

$O(n)$ time

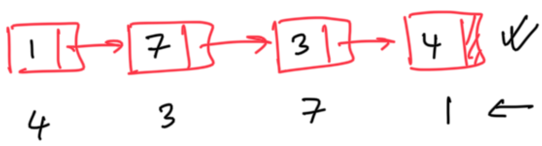
```
    a = Node(element, None)
    if self._head ≠ None
        self._tail.next = a
        self._tail = a
    else: self._tail = self._head = a.
```

$O(1)$

def _removeLast(): $\Omega(n) \rightarrow O(1)$

arrays → implement Stack, Queue

Linked Lists →



top of the stack ⟷ head of the list

pop ⟷ Remove First

push ⟷ Insert First

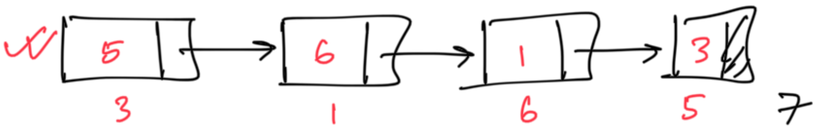
class Stack :

```
__init__(self):
self._data = new LinkedList()

def push(self, element):
        self._data.insertFirst(element)

def pop(self):
        return self._data.removeFirst();
```

Queue :

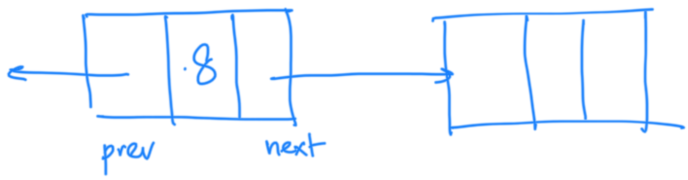


front   rear

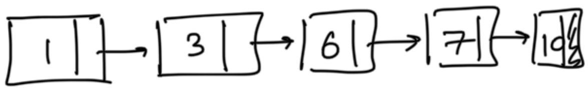
5 , 6, 1, 3 7

Enqueue ⟷ Insert Last

Dequeue ⟷ Remove First



Doubly Linked List

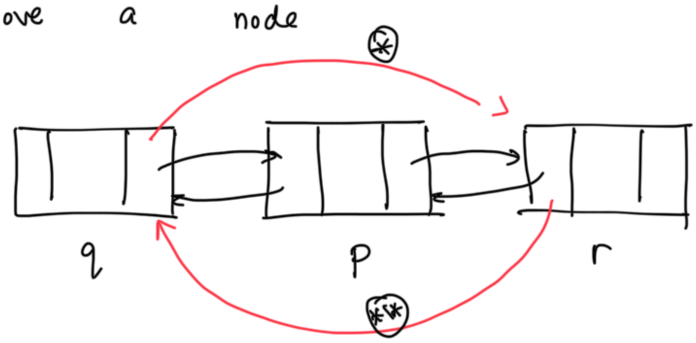


class Node:

```
def __init__(self, element, next, prev):
        self._element = element
        self._next = next
        self._prev = prev
```

header trailer

remove a node



p._prev._next = p._next (*)

p._next._prev = p._prev (**)