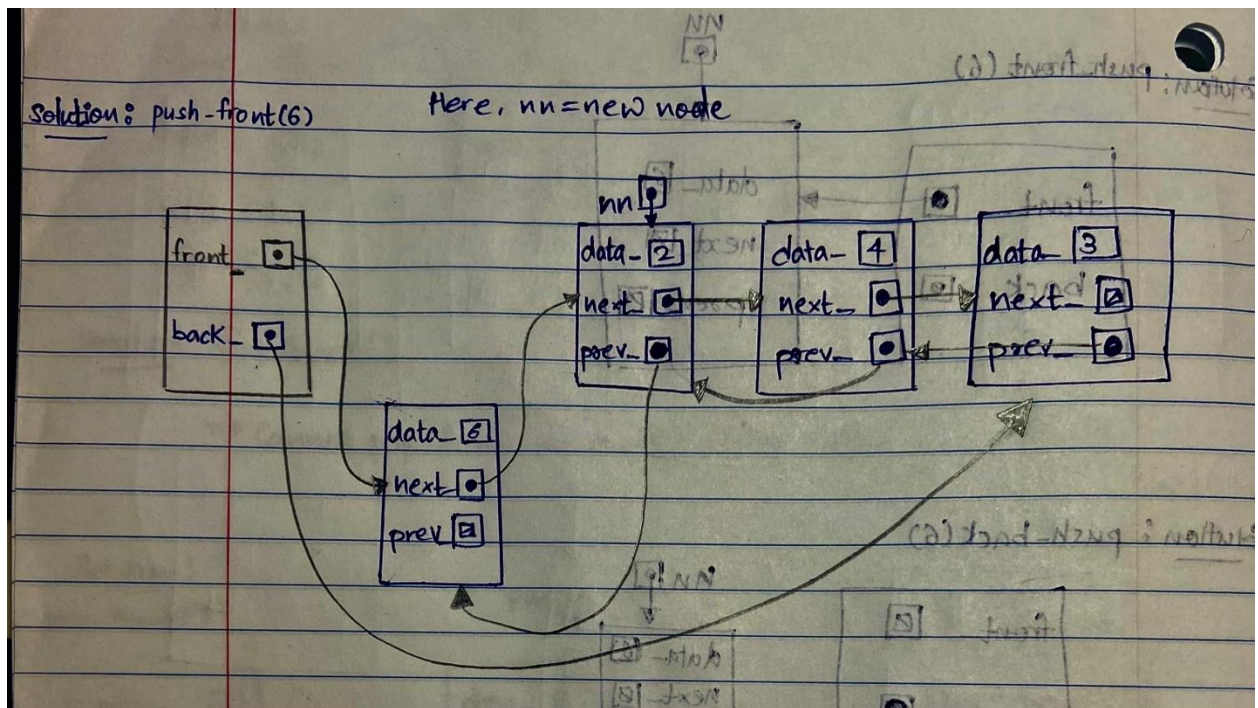
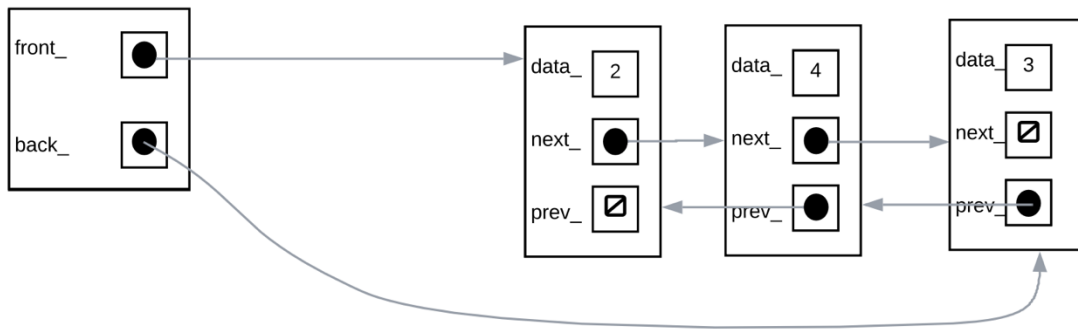
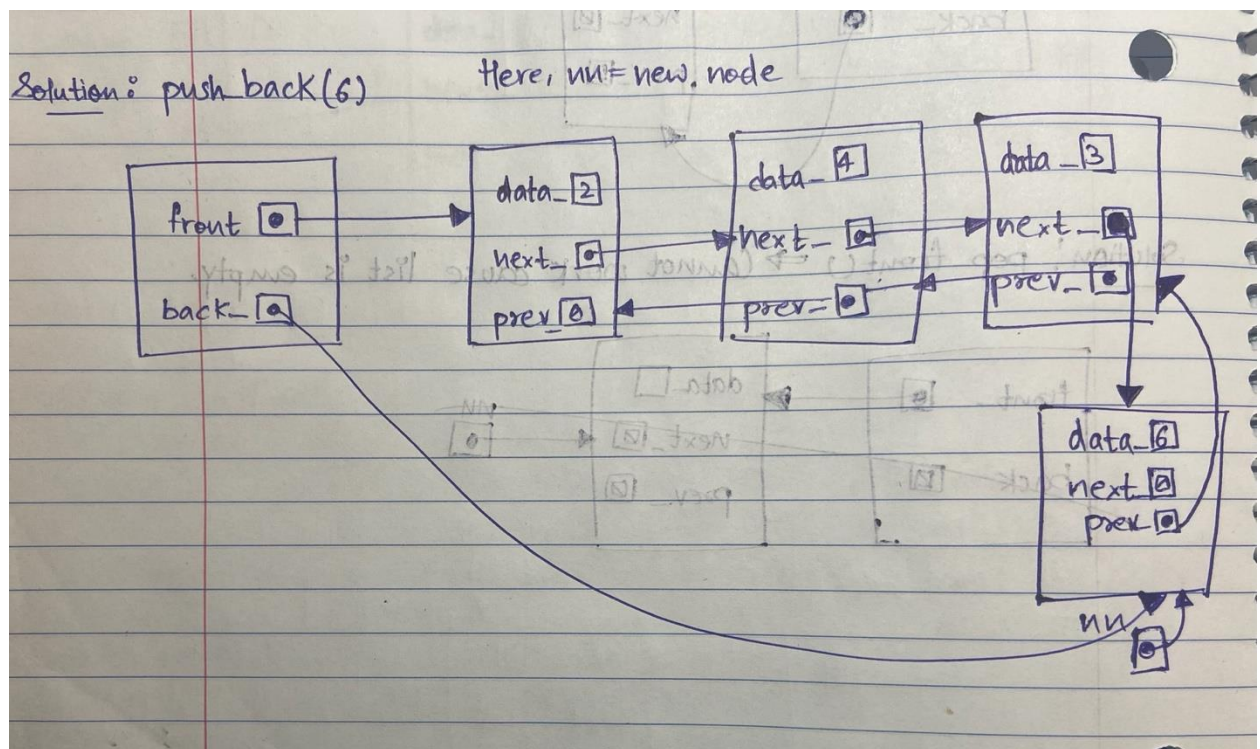
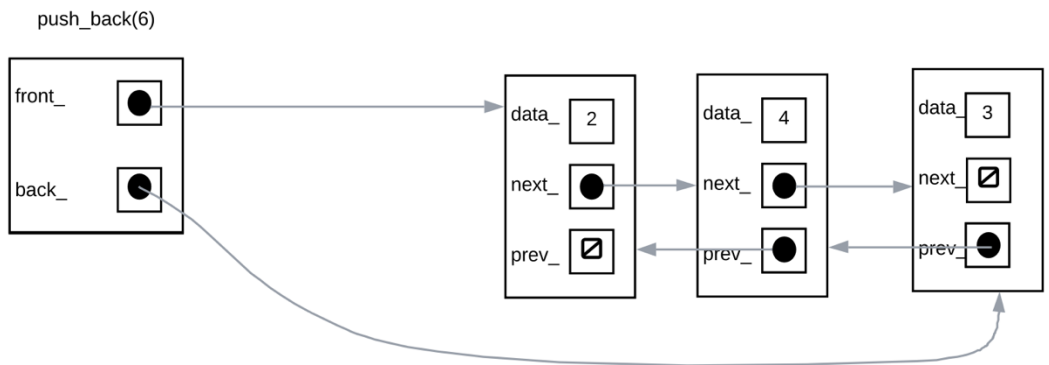
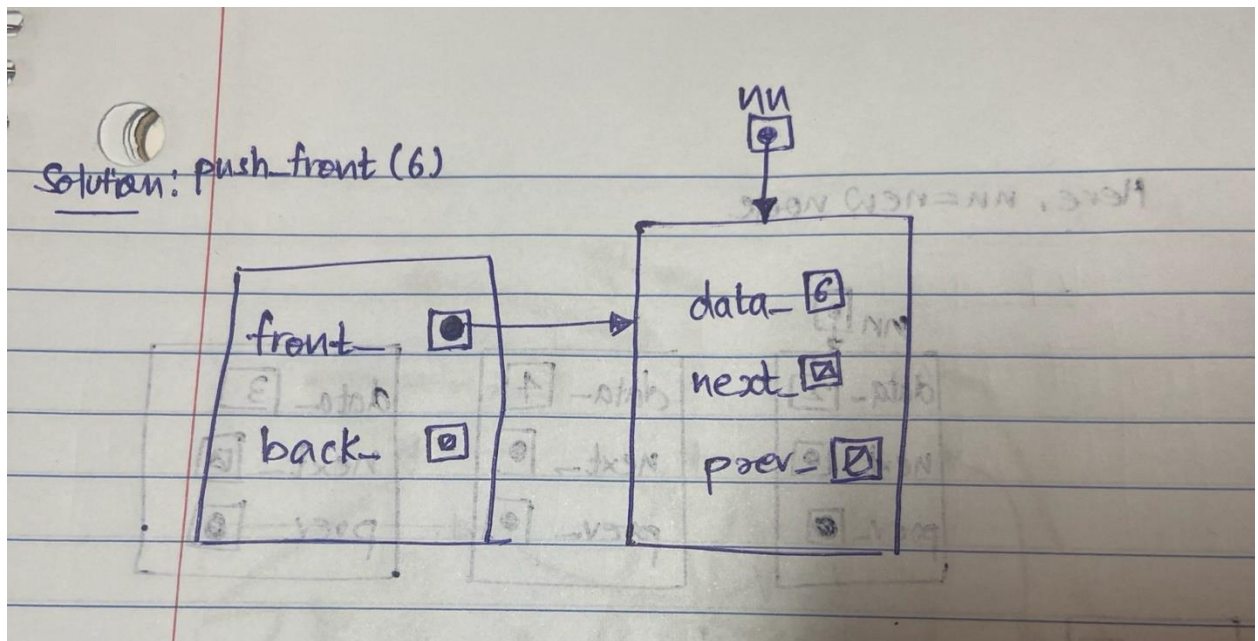
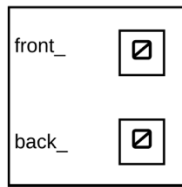


push\_front(6)

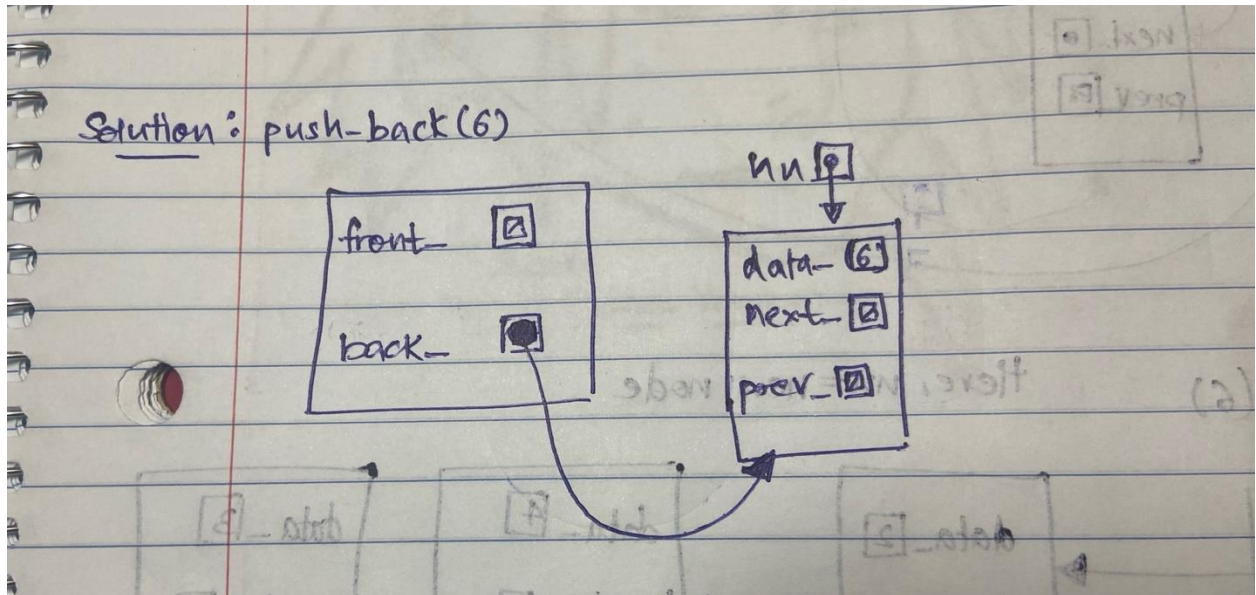
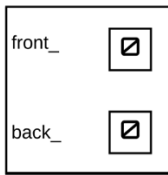




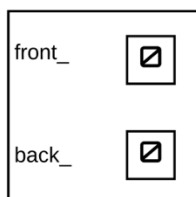
push\_front(6)



push\_back(6)



pop\_front()



Solution: pop\_front()  $\Rightarrow$  Cannot work cause list is empty.

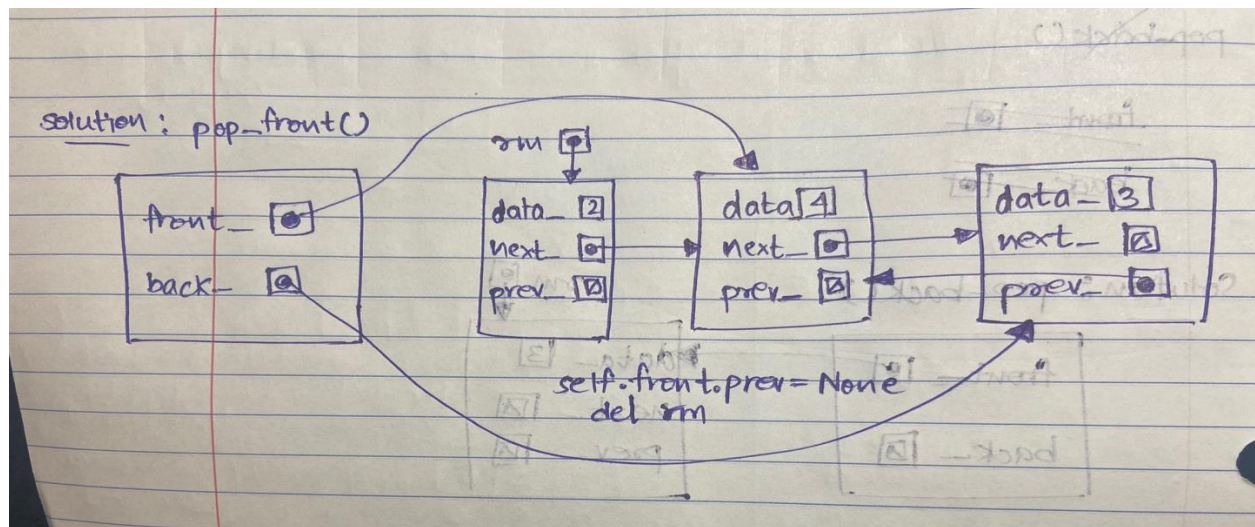
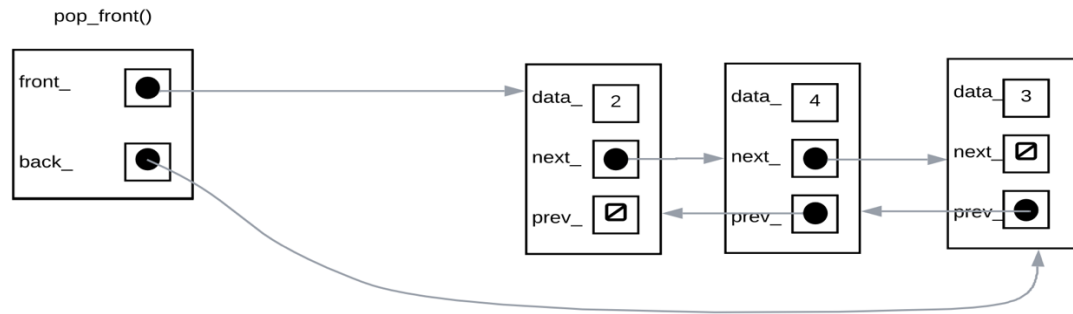


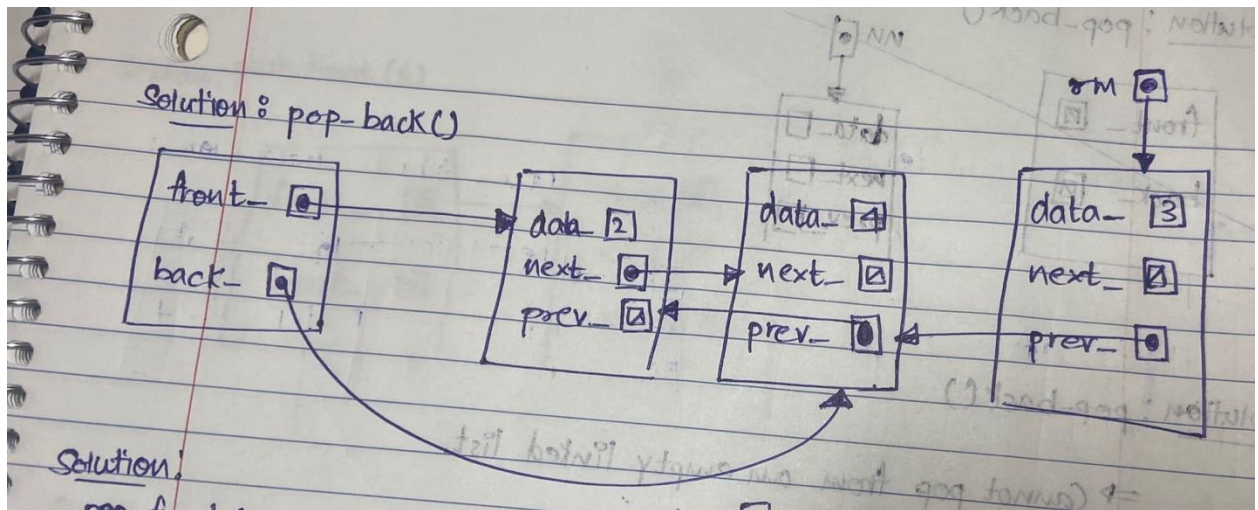
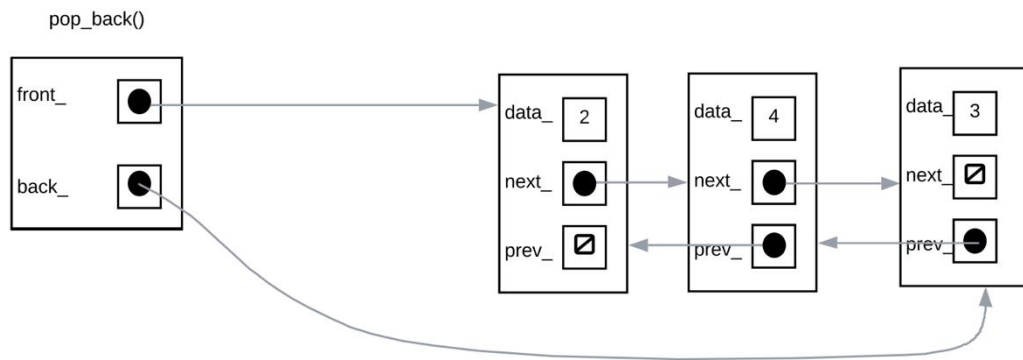
pop\_back()

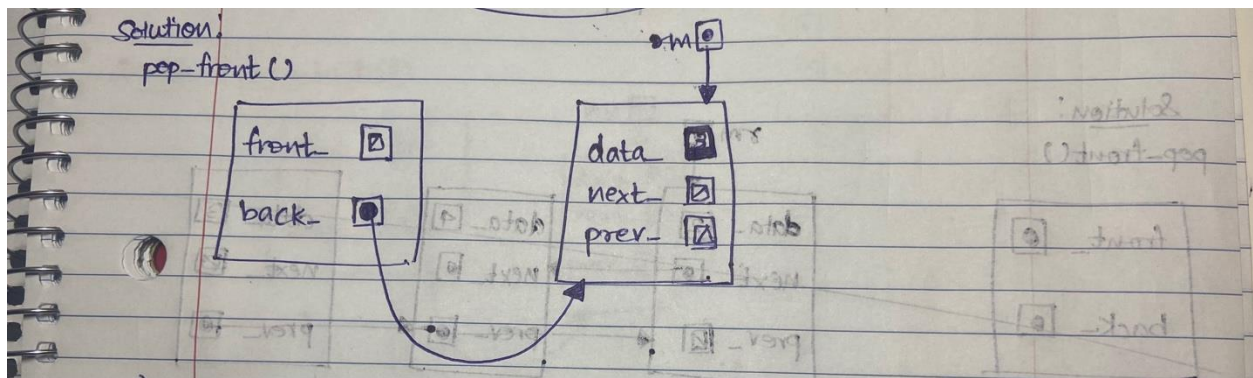
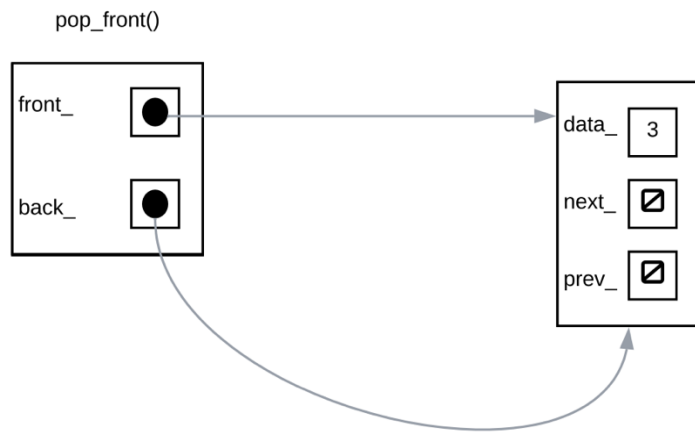
front_	<input checked="" type="checkbox"/>
back_	<input checked="" type="checkbox"/>

Solution: pop-back()

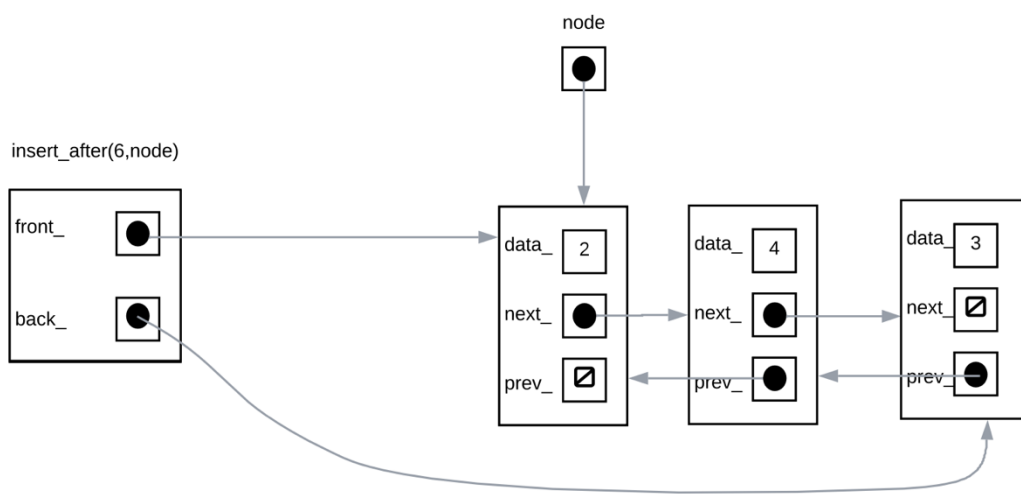
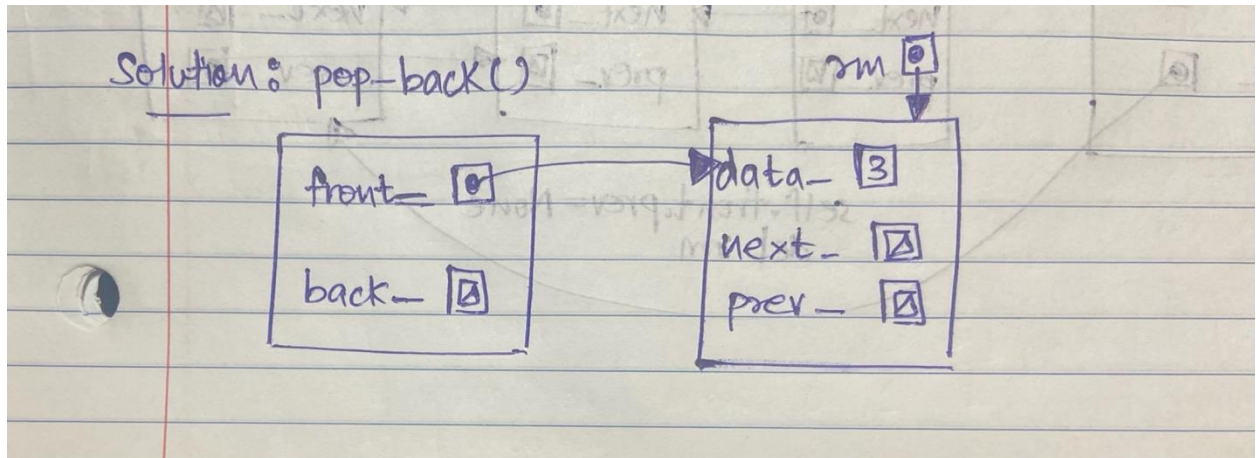
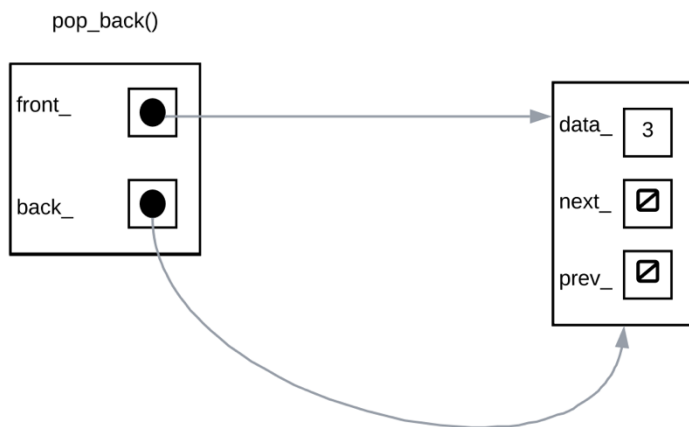
⇒ Cannot pop from an empty linked list.



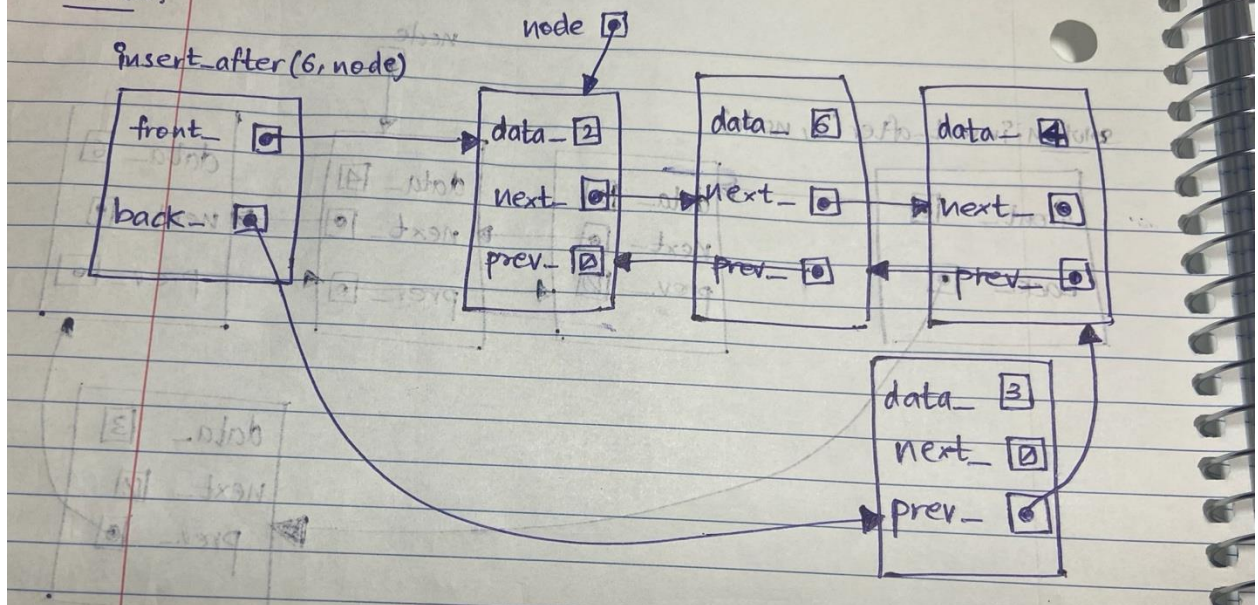


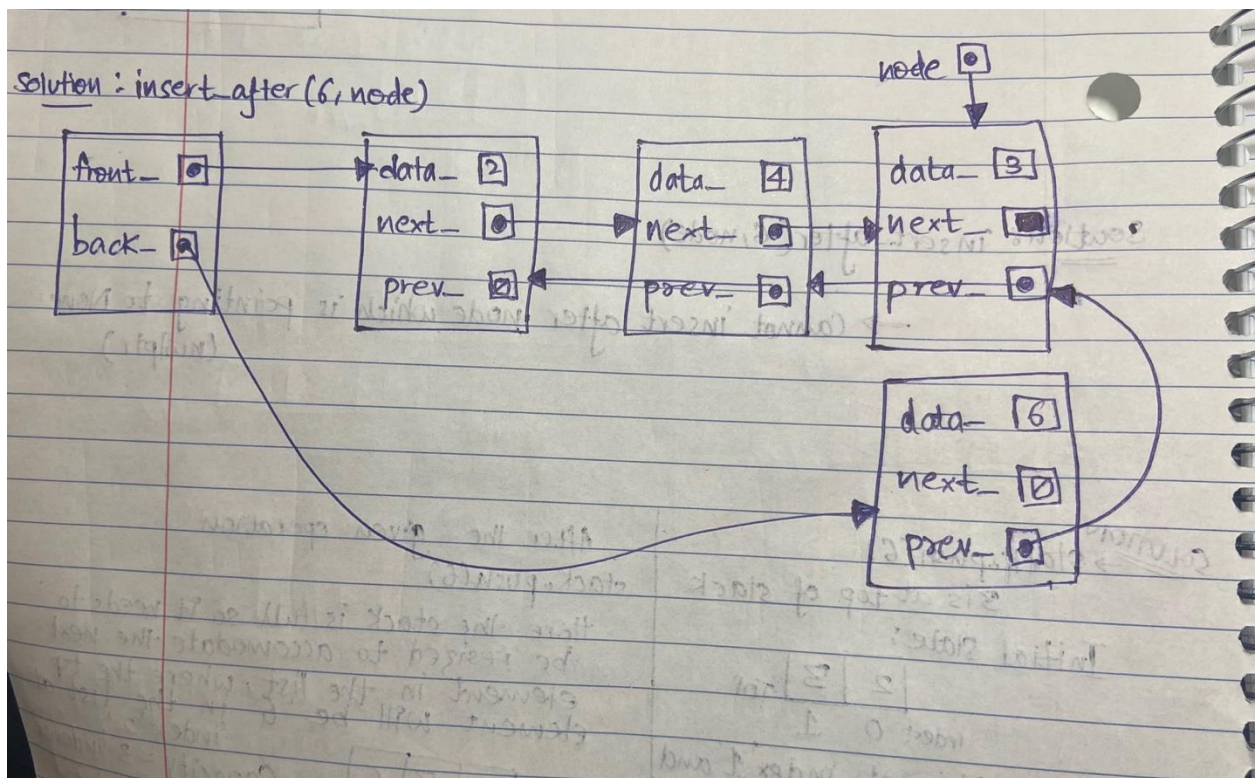
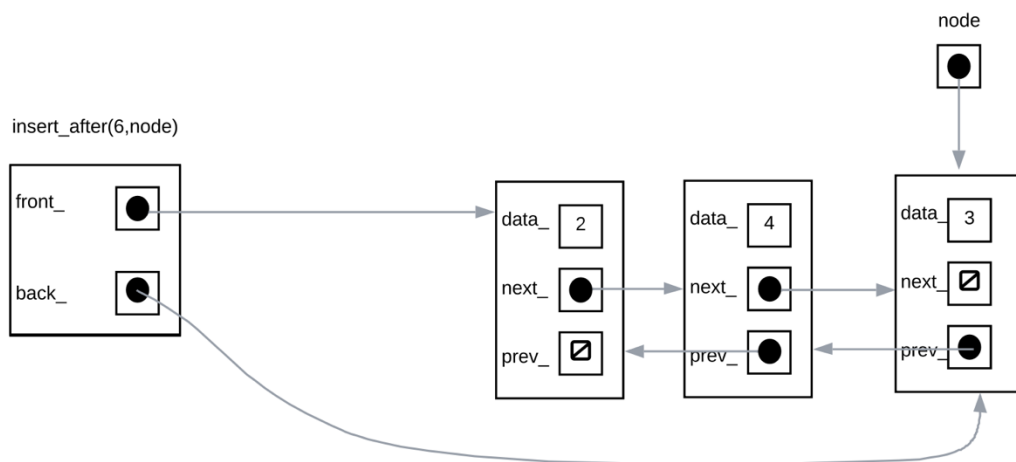


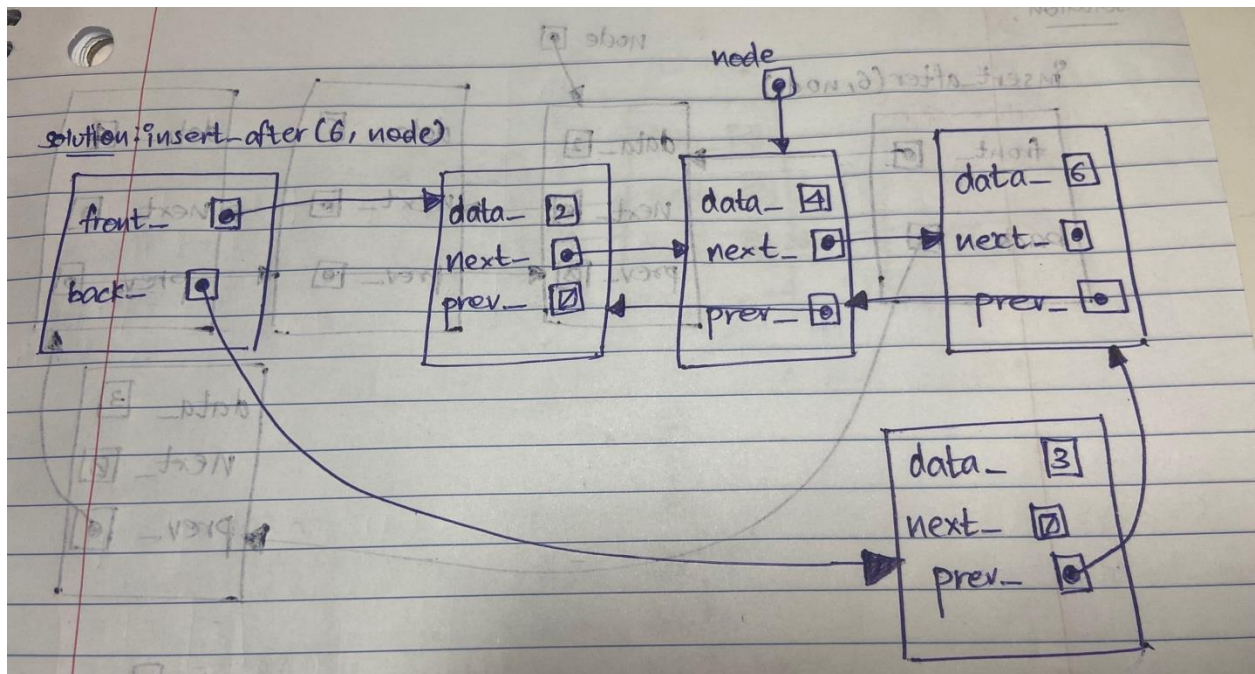
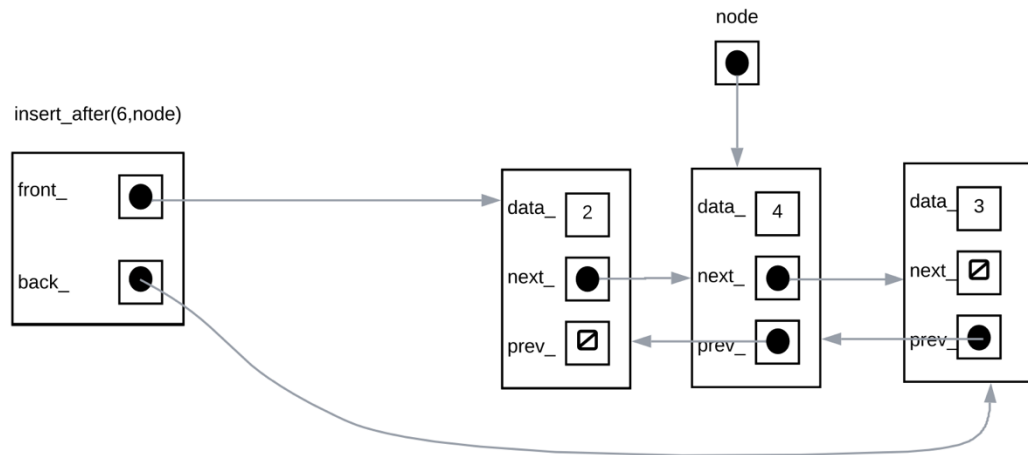




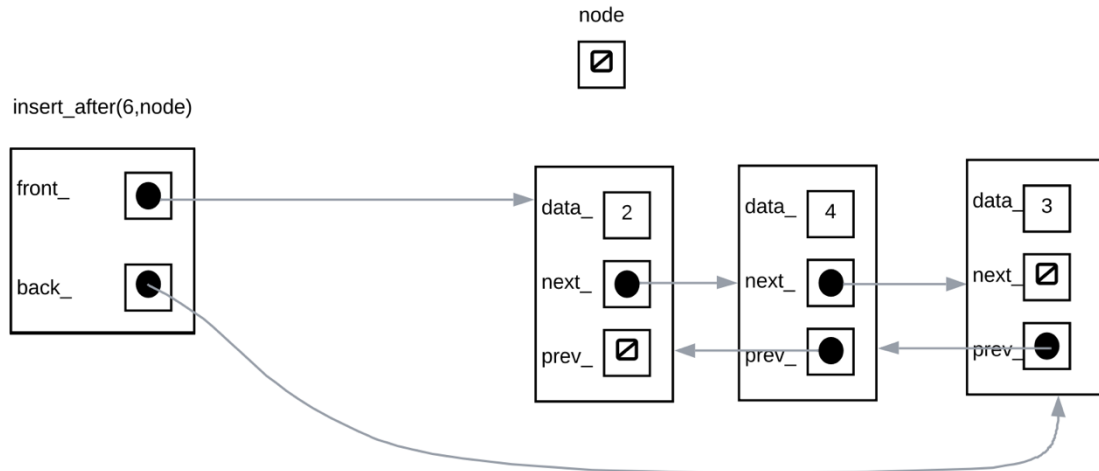
Solution:











Solution: insert\_after(6,node)

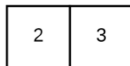
→ Cannot insert after node which is pointing to None (nullptr)



Stack: In the diagrams below list what data members you need to track and what their values are in its initial state and their state after each of the operations are applied to the diagram. If the array needs to be resized, draw the new array with the correct capacity

stack.push(6)

3 is at top of stack



SOLUTION → stack.push(6)

3 is at top of stack

Initial state:

2	3
---	---

index: 0 1 Top

- 3 is at index '1' and is top element
- 2 is at index '0' as the oldest member in the list

After the given operation stack.push(6)

Here, the stack is full so it needs to be resized to accommodate the next element in the list, where the top element will be '6' in the list at index '2'.

2	3	6
---	---	---

index: 0 1 2 Top Capacity = 3 indexes [0, 1, 2]

```
stack.pop()
stack.pop()
stack.push(6)
```

initially 5 is at top of stack

2	4	3	5
---	---	---	---

Solution:

stack.pop()

stack.pop()

stack.push(6)

initially 5 is at top of stack

2	4	3	5
index: 0	1	2	3

STATES

Operation (I): stack.pop()

For operation (I), the top element at index 3 is going to be removed. So,

2	4	3	
index: 0	1	2	3

Operation (II): stack.pop() = another top element preceded by top index '3' will be removed

2	4		
index: 0	1	2	3

Operation (III): stack.push(6)

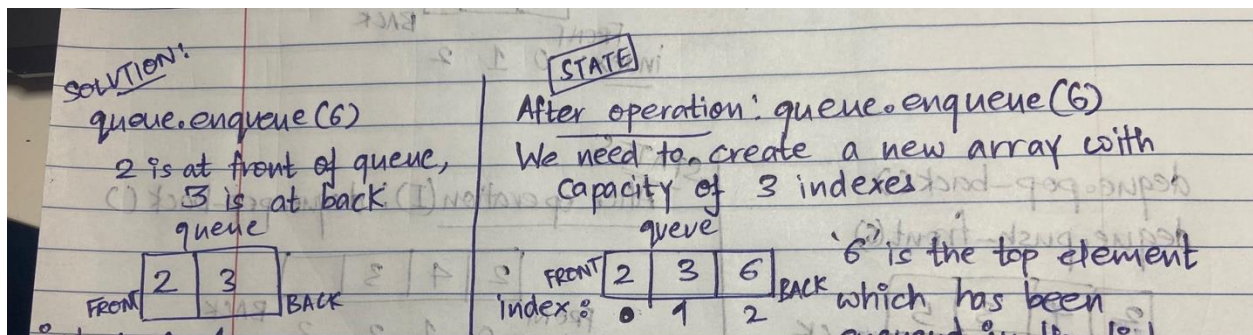
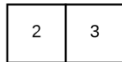
2	4	6	
index: 0	1	2	3

element 6 will be pushed at '2' index of list and will be the top element.

Queues: In the diagrams below list what data members you need to track and what their values are in its initial state and their state after each of the operations are applied to the diagram. If the array needs to be resized, draw the new array with the correct capacity

queue.enqueue(6)

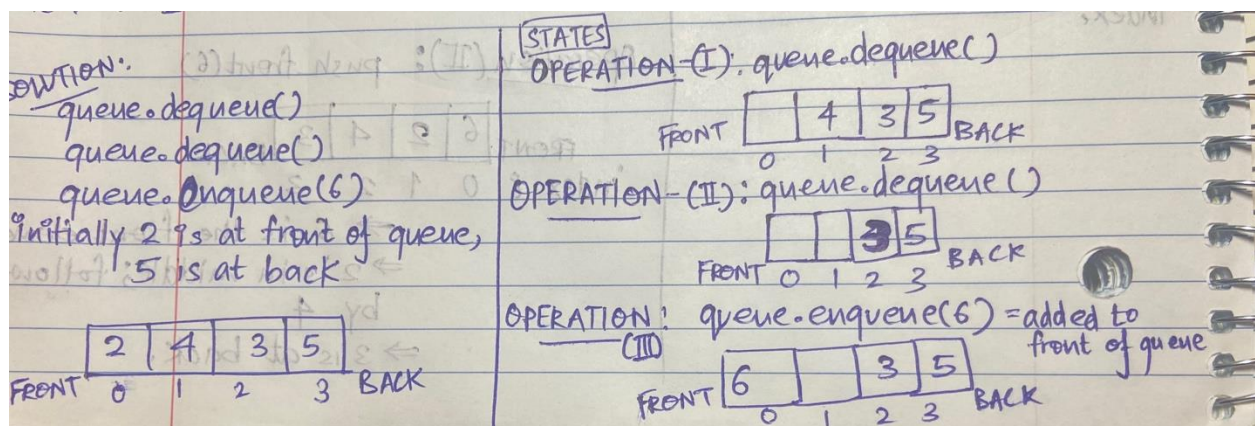
2 is at front of queue, 3 is at back



```
queue.dequeue()
queue.dequeue()
queue.enqueue(6)
```

initially 2 is at front of queue,  
5 is at back

2	4	3	5
---	---	---	---

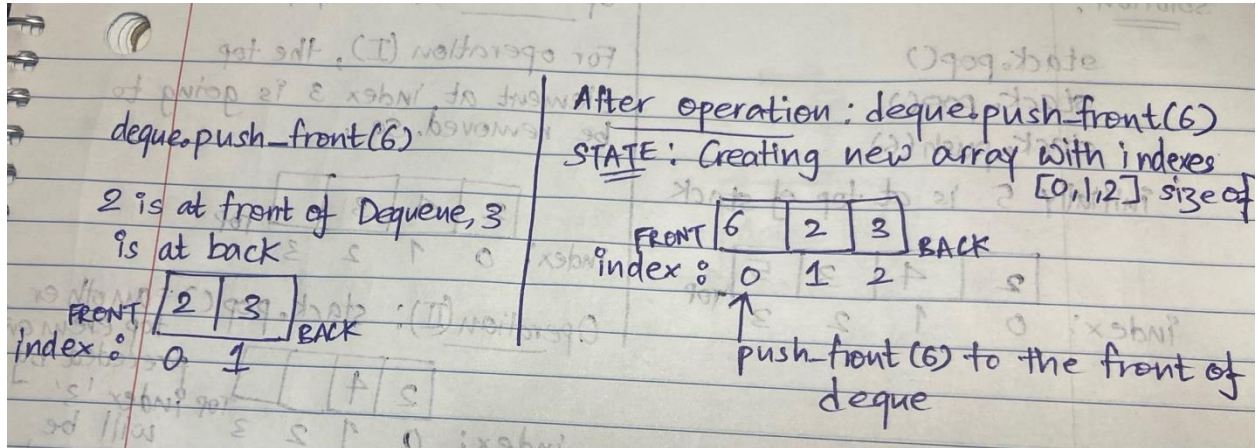
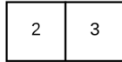




Deque: In the diagrams below list what data members you need to track and what their values are in its initial state and their state after each of the operations are applied to the diagram. If the array needs to be resized, draw the new array with the correct capacity

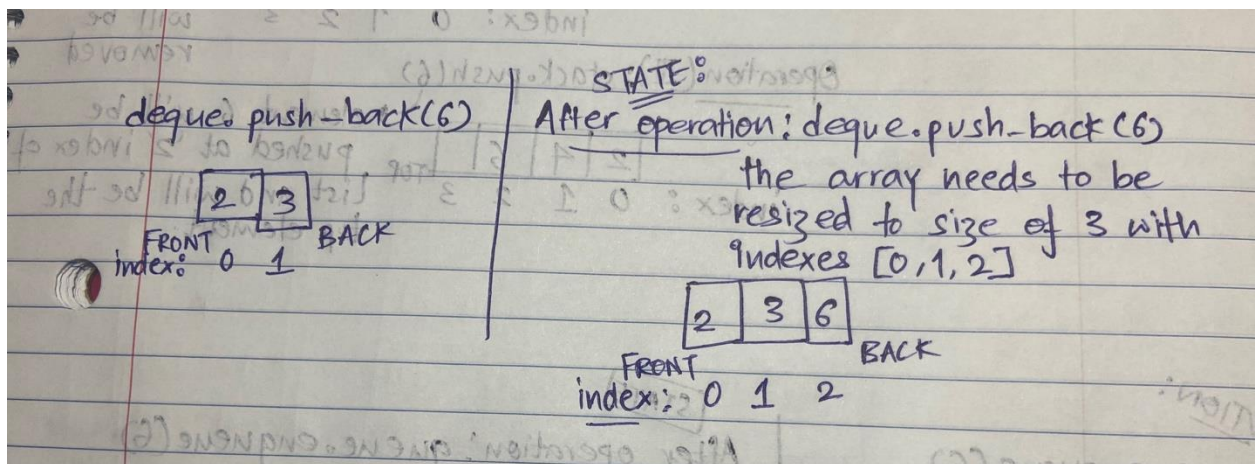
deque.push\_front(6)

2 is at front of Deque, 3 is at back



deque.push\_back(6)

2 is at front of Deque, 3 is at back





```
deque.pop_back()
deque.push_front(6)
```

initially 2 is at front of deque, 5 is at back

2	4	3	5
---	---	---	---

STATES

deque.pop\_back()  
deque.push\_front(6)

2	4	3	5
---	---	---	---

FRONT index: 0 1 2 3 BACK

After operation (I): deque.pop\_back()

2	4	3	
---	---	---	--

FRONT index: 0 1 2 3 BACK

operation (II): push\_front(6)

6	2	4	3
---	---	---	---

FRONT index: 0 1 2 3 BACK

⇒ 6 is the front element  
⇒ 2 is in middle; followed by 4  
⇒ 3 is at back.

```

deque.pop_front()
deque.push_back(6)
deque.pop_front()
deque.push_back(7)

```

initially 2 is at front of deque,  
5 is at back

2	4	3	5
---	---	---	---

```

deque.pop_front()
deque.push_back(6)
deque.pop_front()
deque.push_back(7)

```

FRONT 

2	4	3	5
---	---	---	---

 BACK  
index: 0 1 2 3

OPERATION-(I): deque.pop\_front()

	4	3	5
--	---	---	---

  
FRONT index: 0 1 2 3 BACK

OPERATION-(II): deque.push\_back(6)

⇒ The array needs to be resized to size of 5 with indexes [0, 1, 2, 3, 4]

	4	3	5	6
--	---	---	---	---

  
FRONT index: 0 1 2 3 4 BACK

OPERATION-(III): deque.pop\_front()

		3	5	6
--	--	---	---	---

  
FRONT index: 0 1 2 3 4 BACK

OPERATION-(IV): deque.push\_back(7)

array resized  
to size of 6  
with indexes  
[0, 1, 2, 3, 4, 5]

		3	5	6	7
--	--	---	---	---	---

  
FRONT index: 0 1 2 3 4 5 BACK

overflow(grid,the\_queue) - apply the overflow function to the grid below and show all the grids the function would add to the queue. Number the grid in the order they are added to the queue. Also state the return value. Note that some grids may remain empty

-2	1	-3	-3	0
2	0	3	2	0
0	0	-3	0	0
0	0	1	0	0








The Grade#7 will remain empty as per the solution.



Initial grid

-2	1	-3	-3	0
2	0	3	2	0
0	0	-3	0	0
0	0	1	0	0

Graph #1

0	-5	0	-1	0
-2	0	-2	0	0
0	-1	3	0	1
-1	0	0	0	1

Graph #2

-1	0	-1	-1	0
-2	-1	-2	0	0
0	-1	3	0	1
-1	0	0	0	1

Graph #3

2	1	1	1	0
0	4	1	0	0
0	0	-3	0	3
-1	0	-2	3	0

Graph #4

0	3	1	1	0
2	0	2	0	1
0	1	-3	2	0
-1	0	3	0	2

Graph #5

1	0	2	1	0
2	1	2	0	1
0	1	4	2	1
-1	1	0	2	0

Graph #6

1	0	2	1	0
2	1	3	0	1
0	2	0	3	1
-1	1	1	2	0