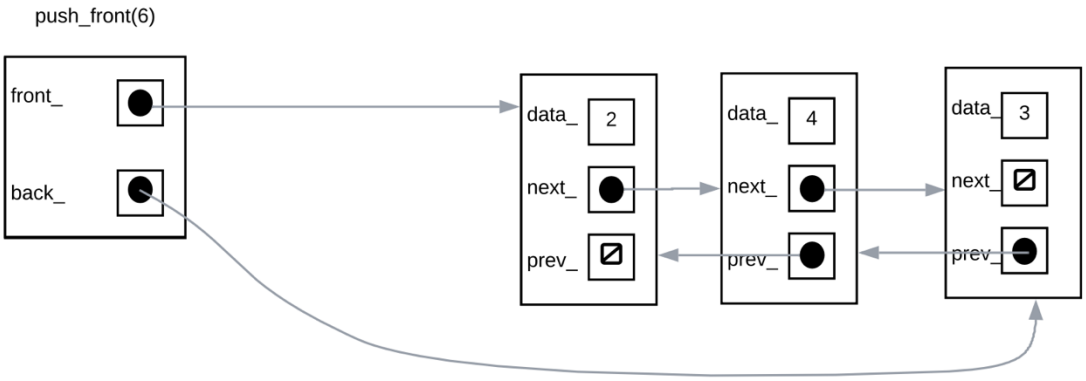
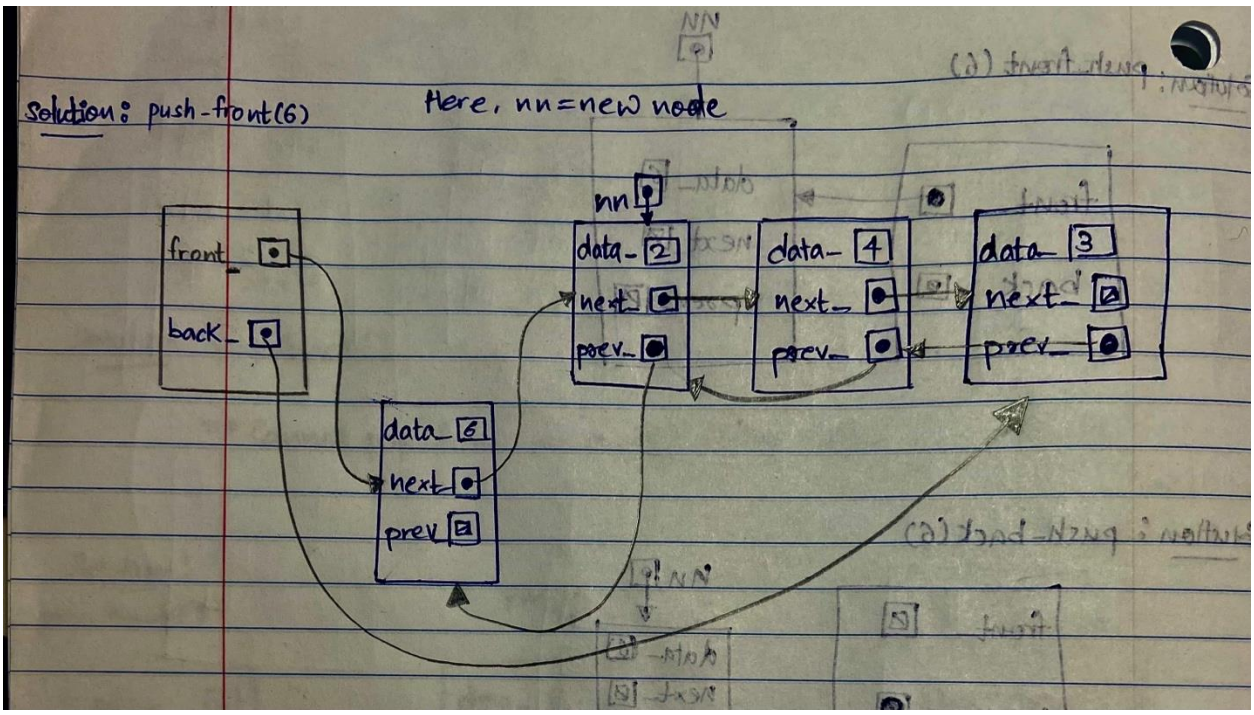
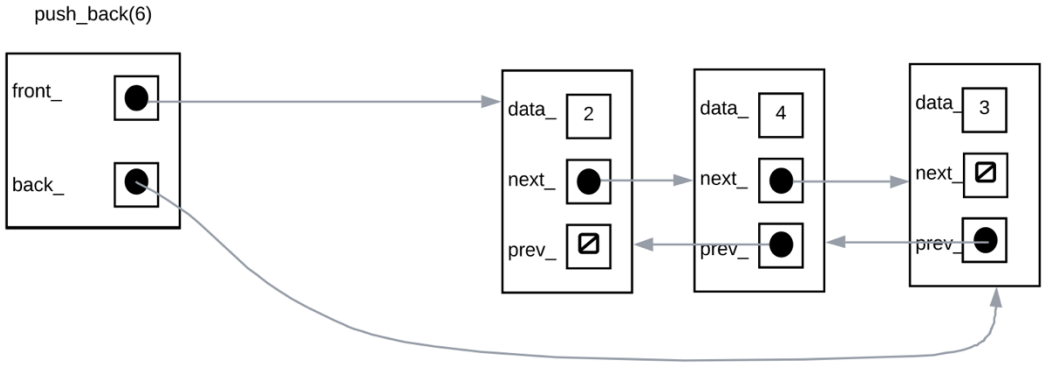
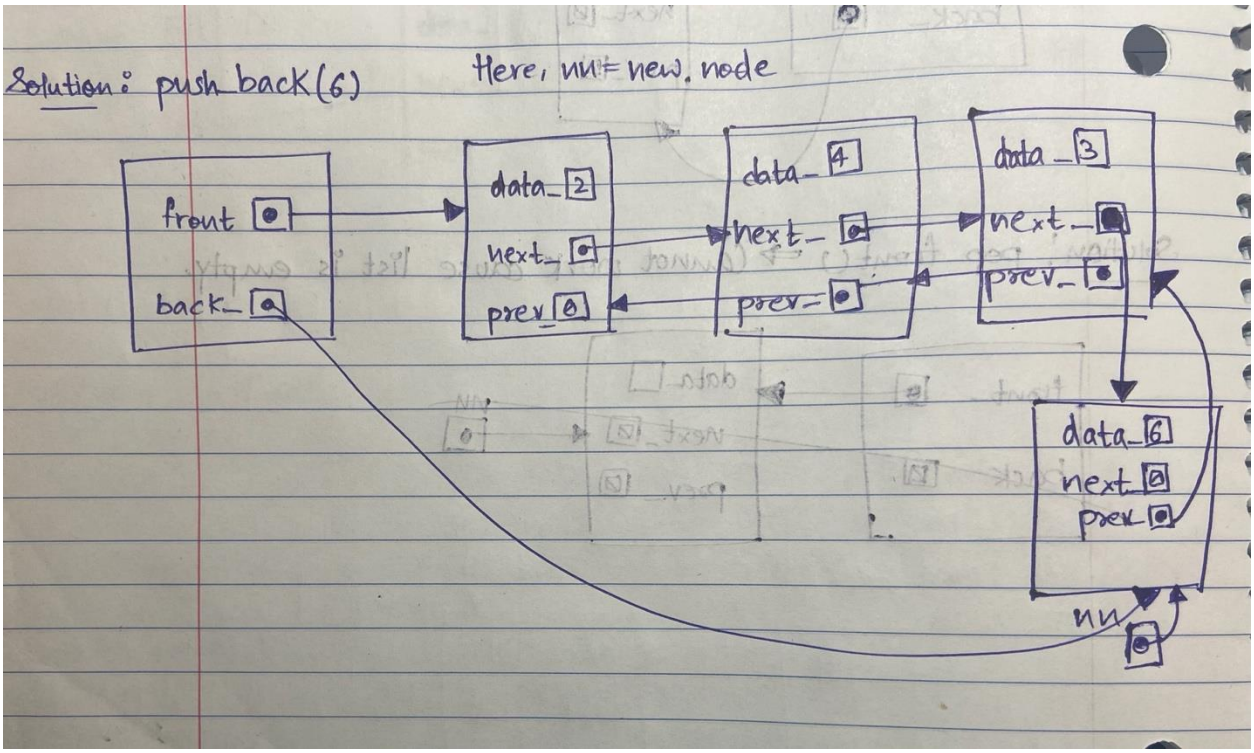
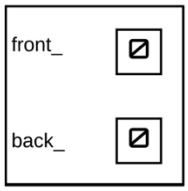

push_front(6)
front_
back_
data_ 2
next_
prev_
data_ 4
next_
prev_
data_ 3
next_
prev_

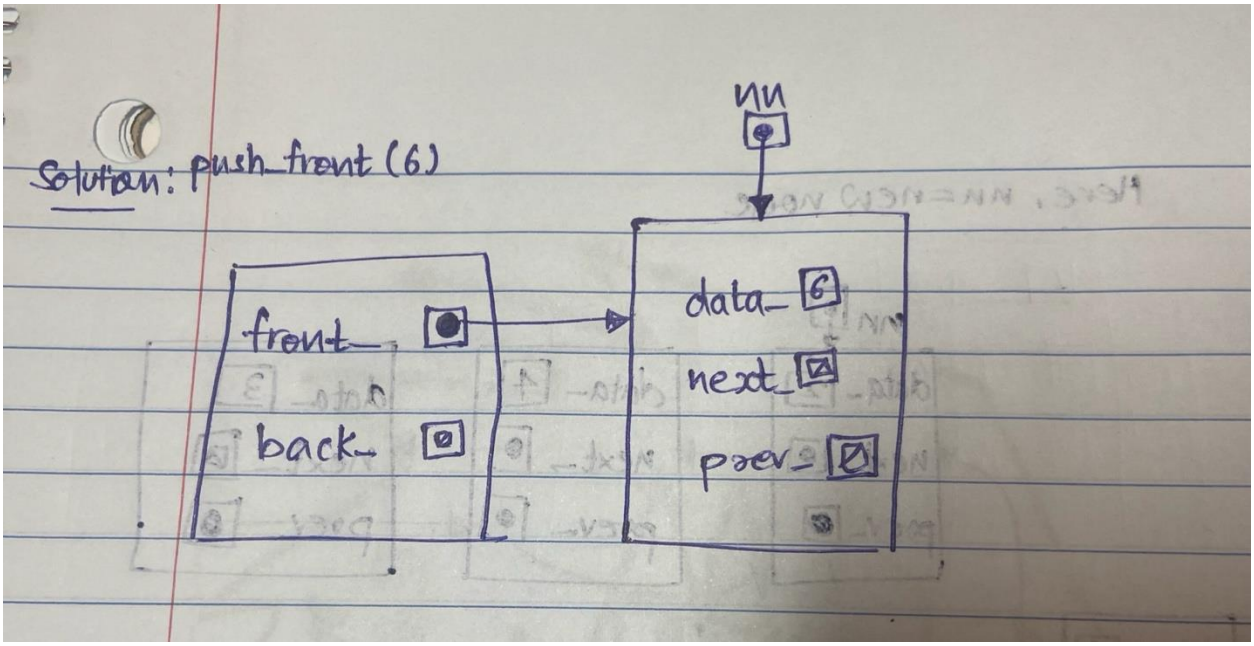

Solution: push_front(6)
Here, nn=new node
nn
front_
back_
data_ 2
next_
prev_
data_ 4
next_
prev_
data_ 3
next_
prev_
data_ 6
next_
prev_


push_back(6)
front_
back_
data_ 2
next_
prev_
data_ 4
next_
prev_
data_ 3
next_
prev_

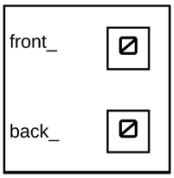

Solution: push_back(6)
Here, nn = new node
front
back_
data_ 2
next_
prev_
data_ 4
next_
prev_
data_ 3
next_
prev_
data_ 6
next_
prev_
nn

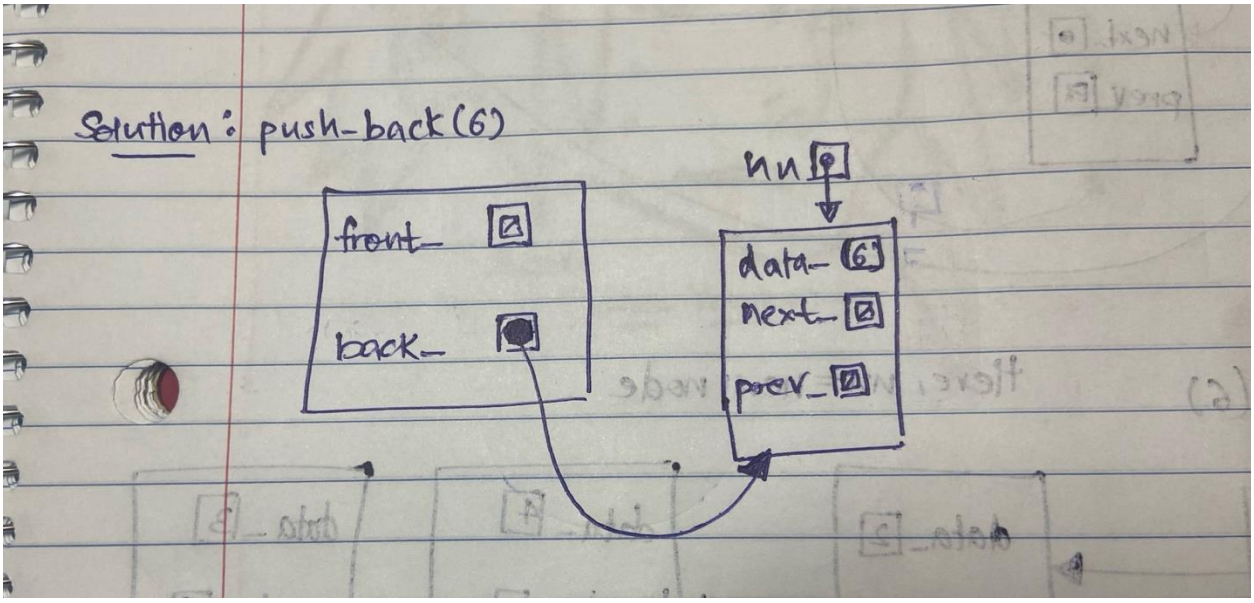
push_front(6)


front_
back_

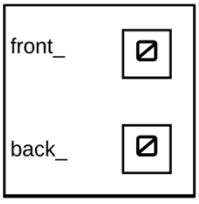

Solution: push_front (6)
NN
front_
back_
data_ 6
next_
prev_

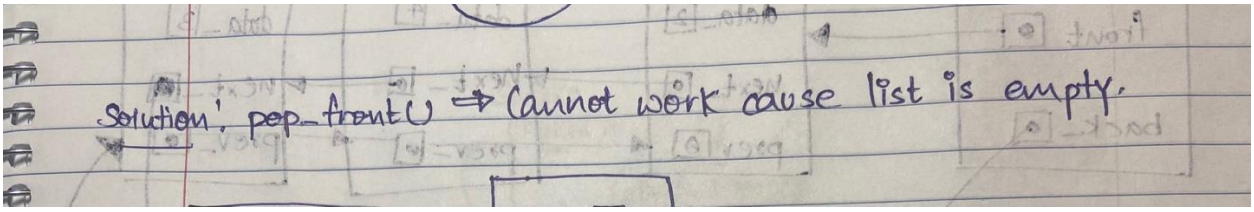
push_back(6)

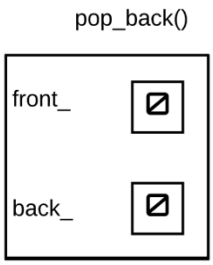


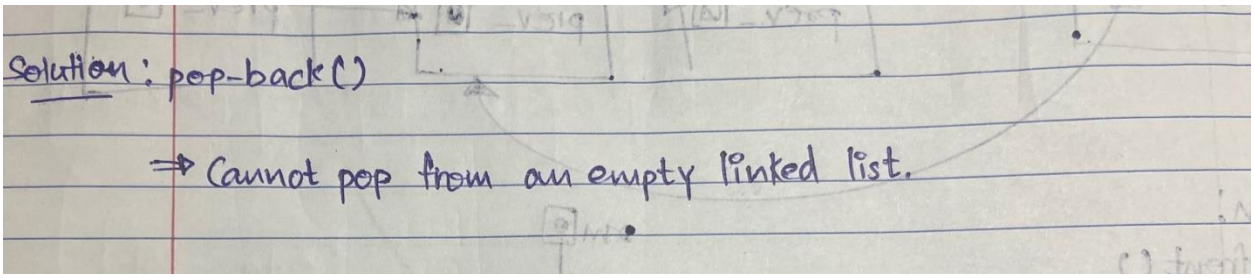


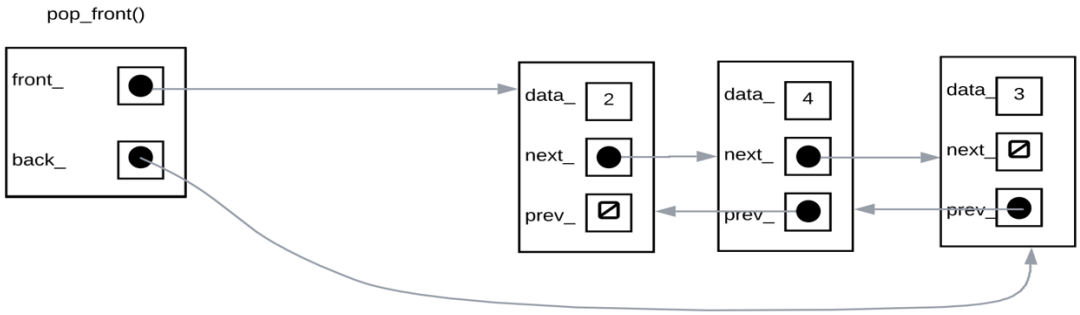
pop_front()

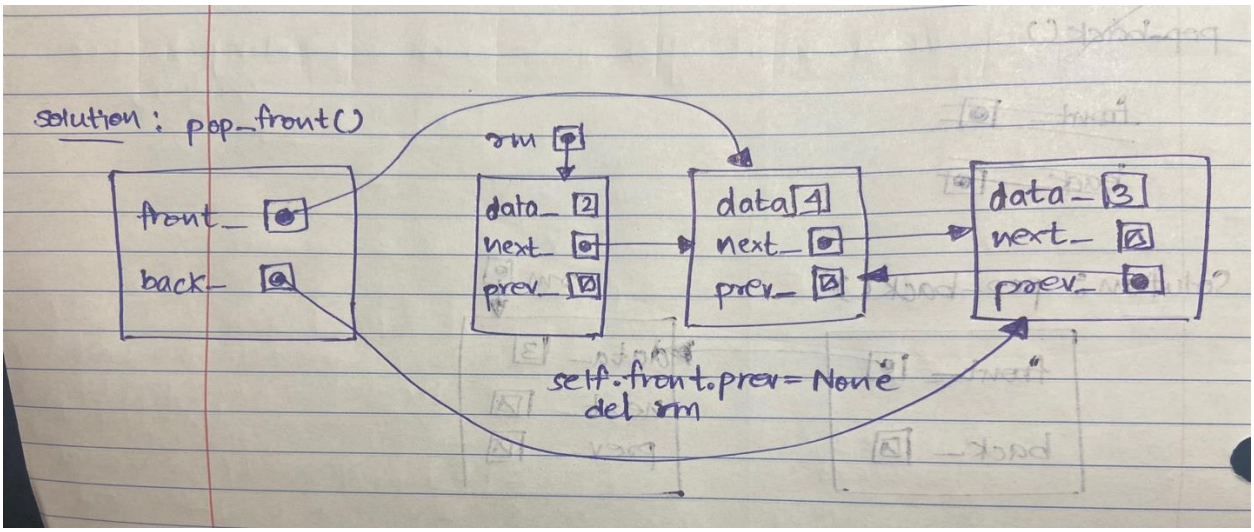


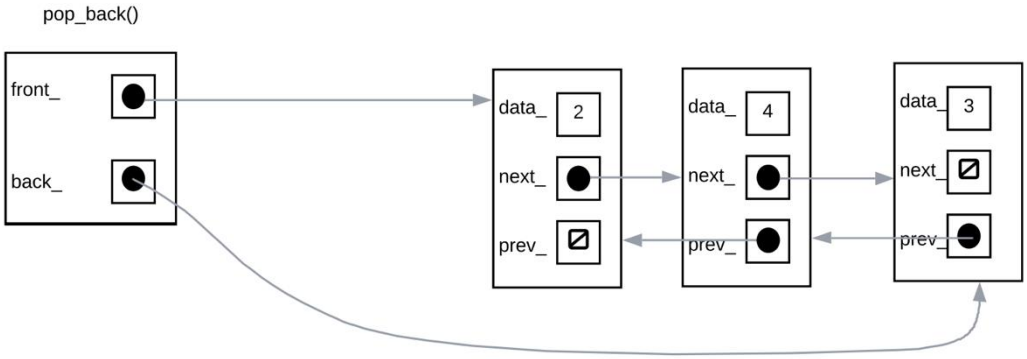


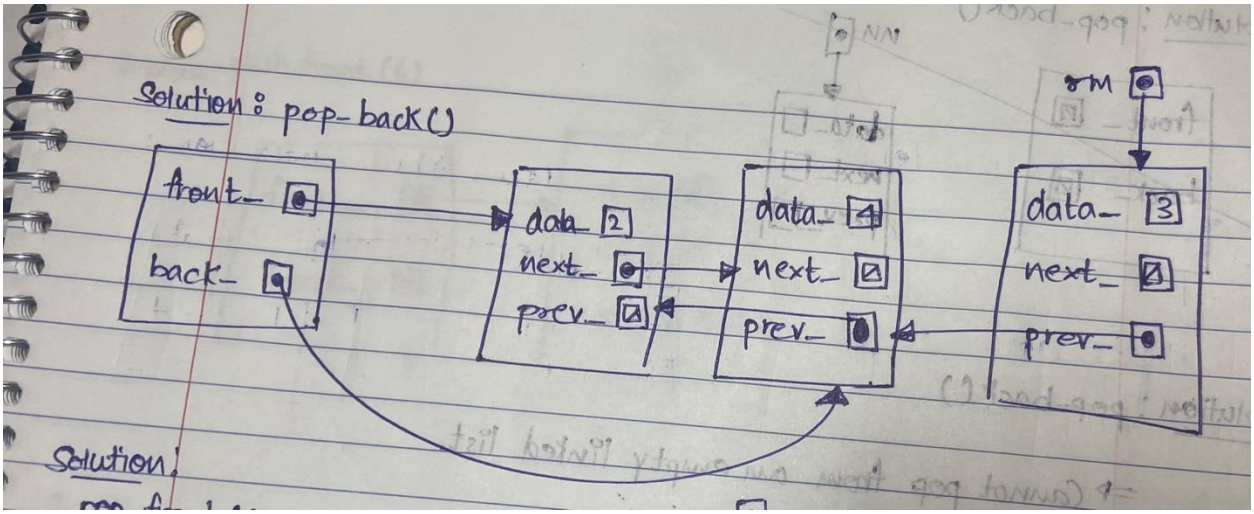

pop_back()
front_
back_

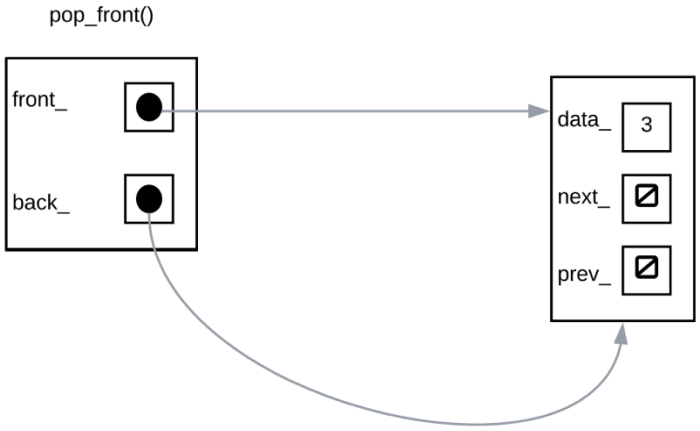

Solution : pop-back()
⇒ Cannot pop from an empty linked list.

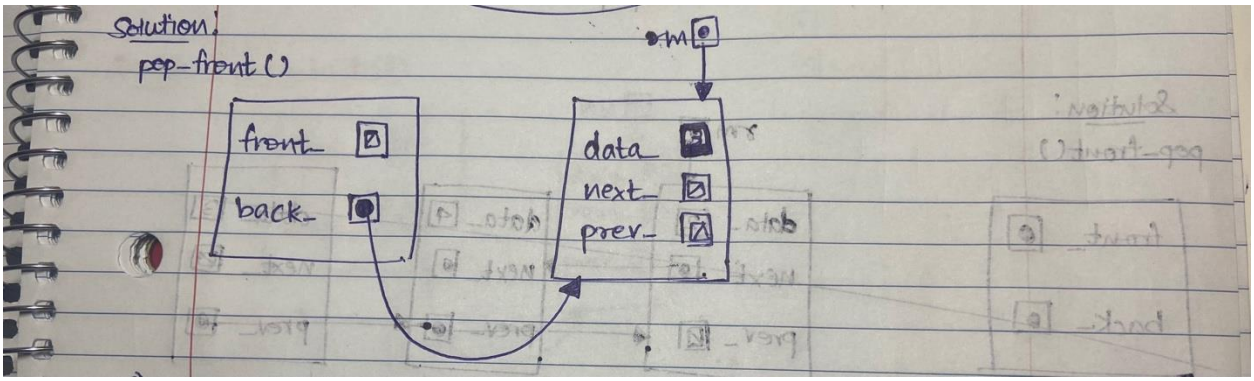

pop_front()
front_
back_
data_
2
next_
prev_
data_
4
next_
prev_
data_
3
next_
prev_

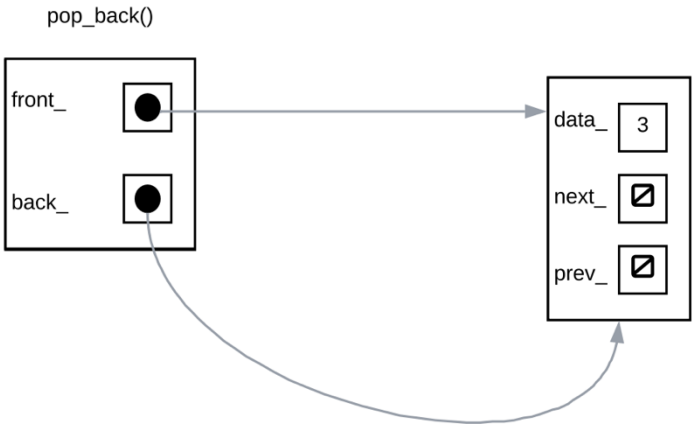

solution : pop_front()
front_
back_
rm
data_ 2
next_
prev_
data 4
next_
prev_
data_ 3
next_
prev_
self.front.prev = None
del rm

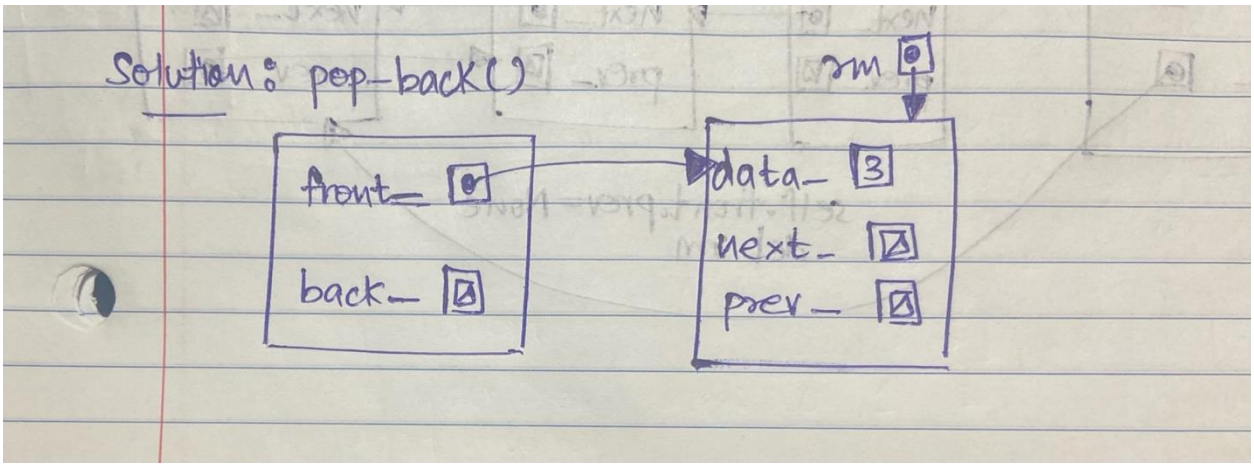

pop_back()
front_
back_
data_
2
next_
prev_
data_
4
next_
prev_
data_
3
next_
prev_

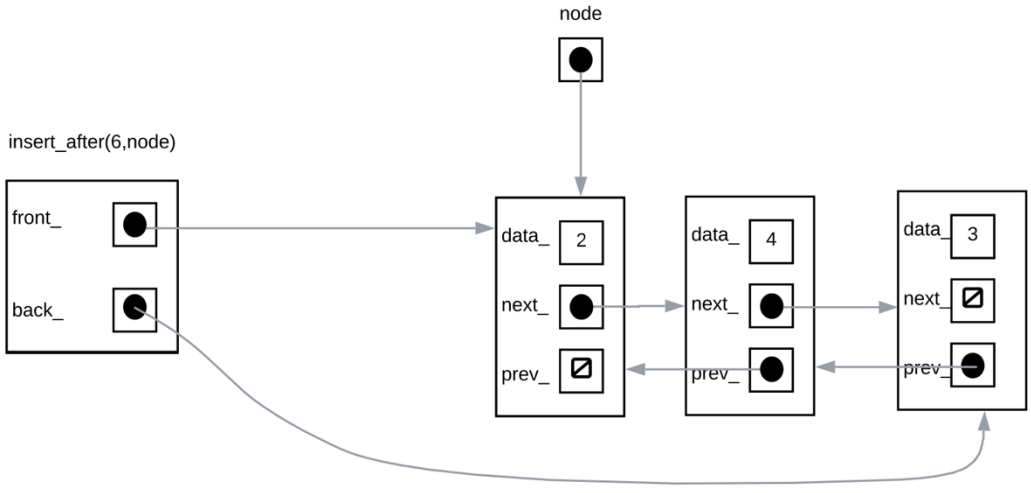

Solution : pop_back()
front_
back_
data_
2
next_
prev_
data_
4
next_
prev_
data_
3
next_
prev_
Solution:

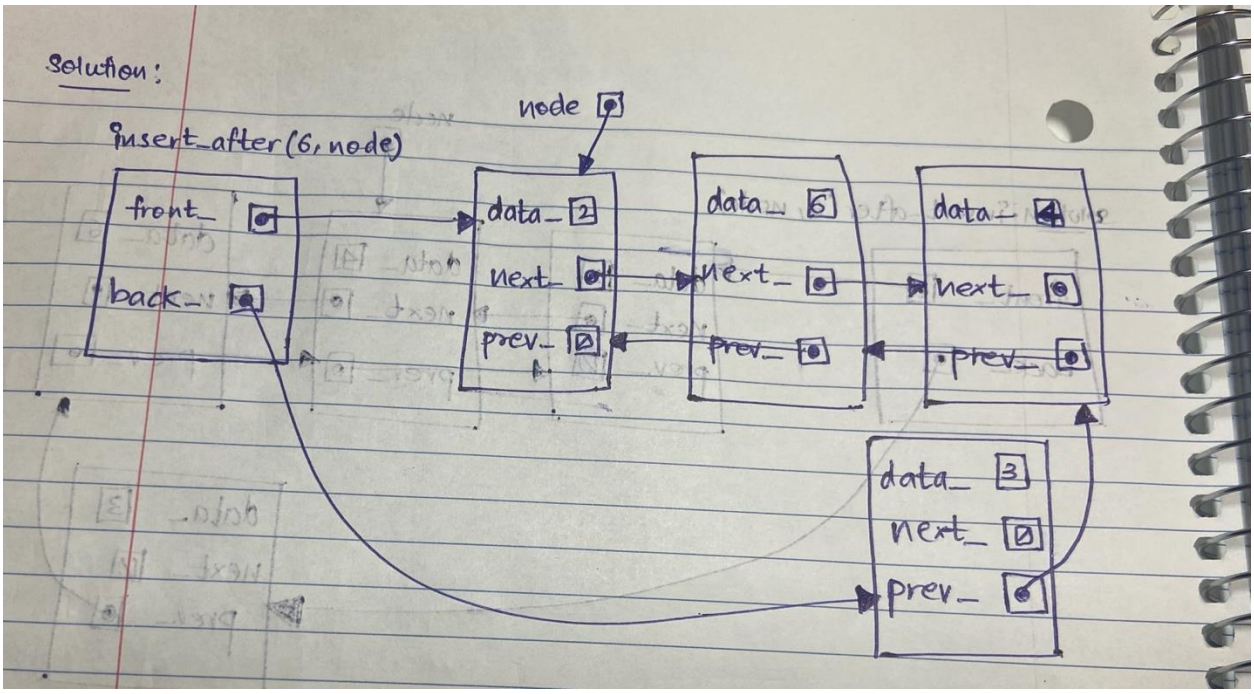

pop_front()
front_
back_
data_
3
next_
prev_

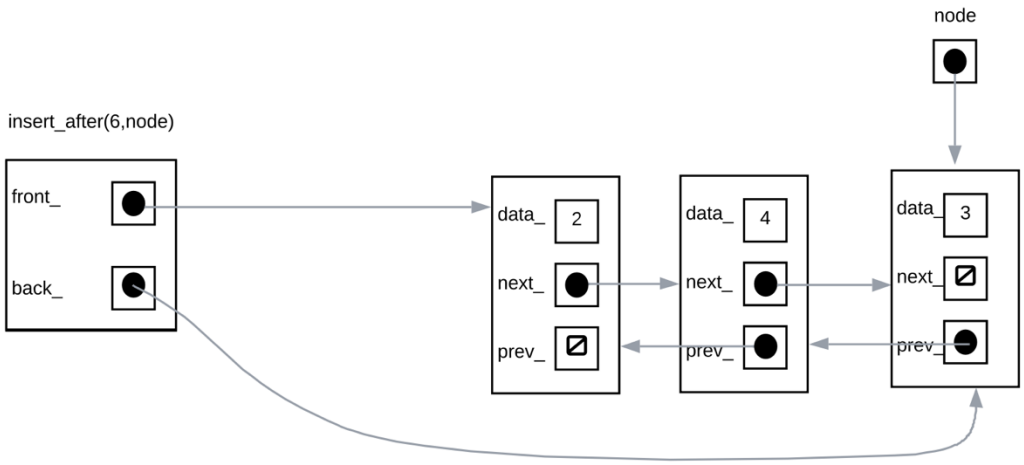

Solution:
pop-front()
front_
back_
data_
next_
prev_

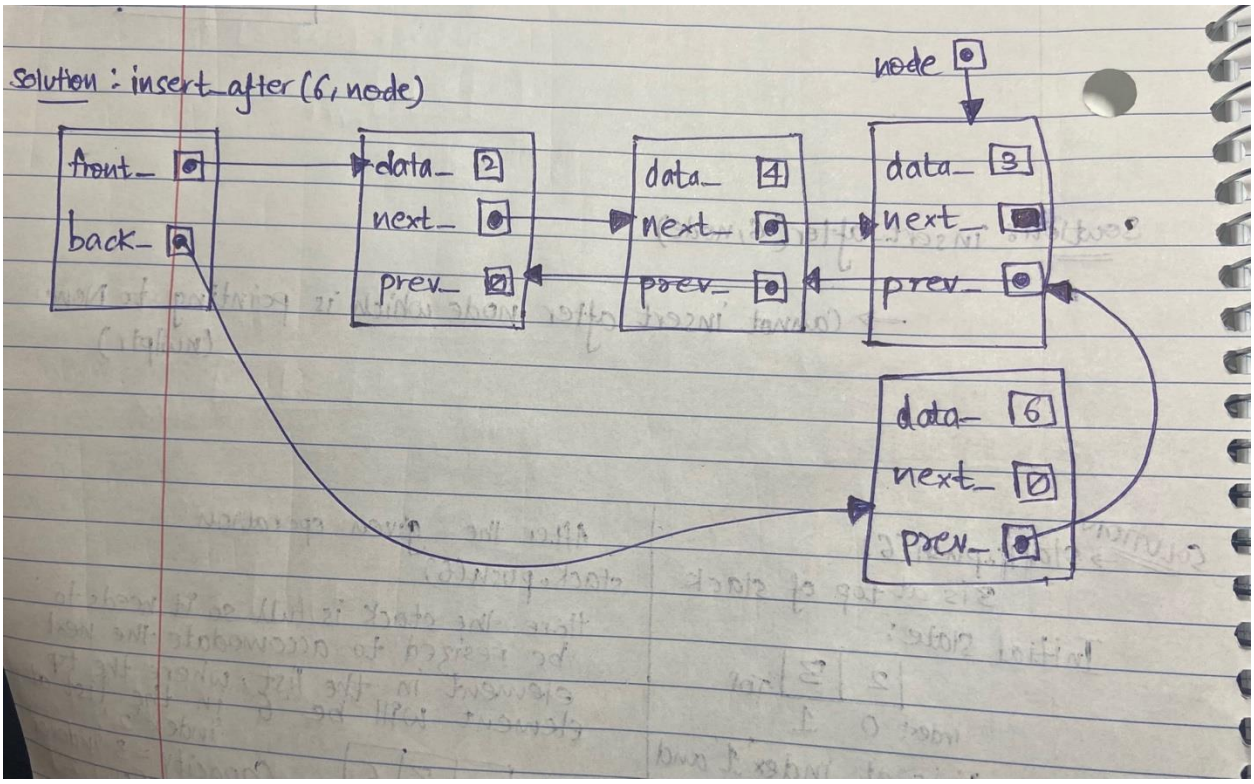

pop_back()
front_
back_
data_
3
next_
prev_

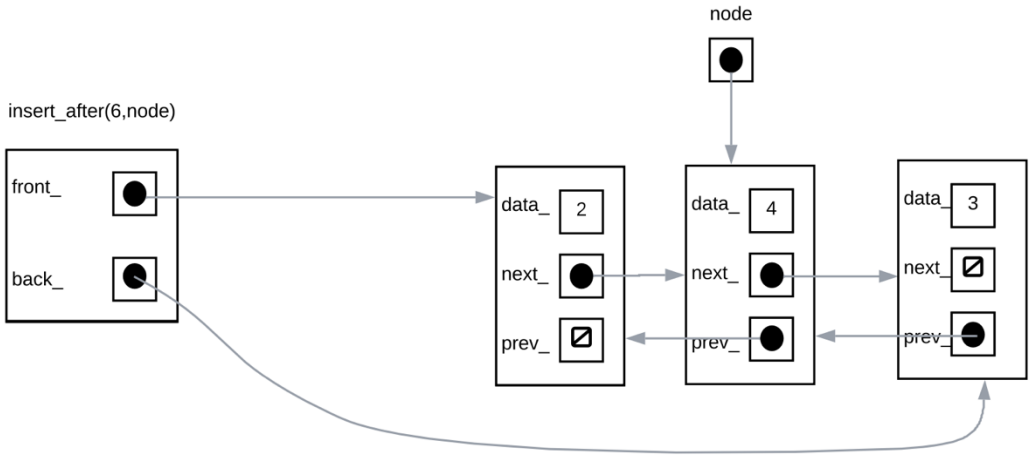

Solution: pop_back()
rm
front_
back_
data_
3
next_
prev_

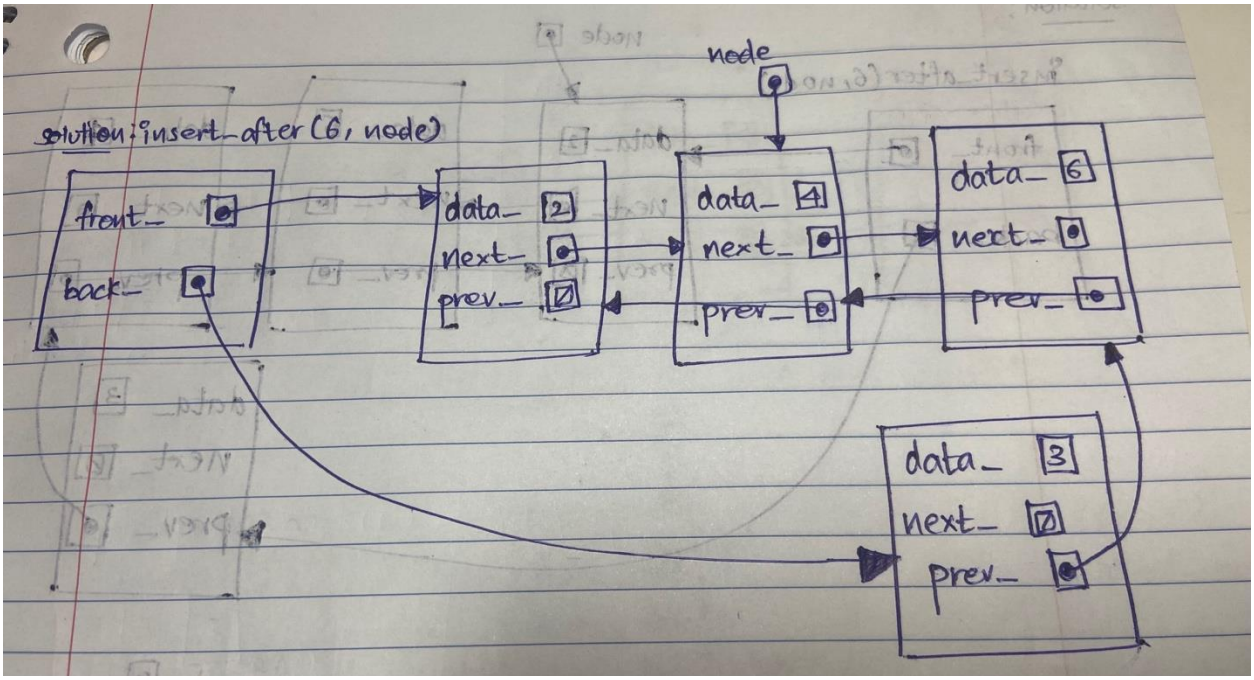

node
insert_after(6,node)
front_
back_
data_
2
next_
prev_
data_
4
next_
prev_
data_
3
next_
prev_

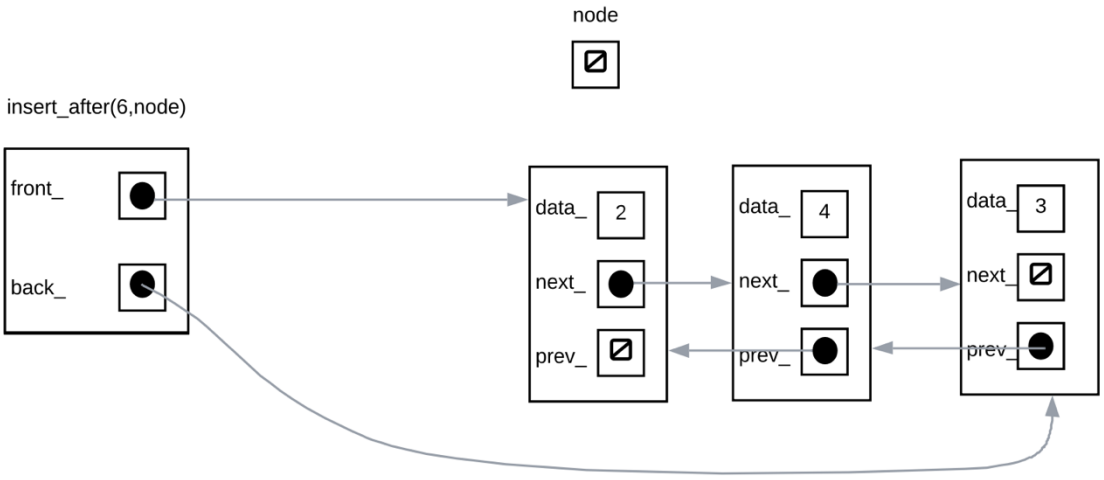

Solution:
insert_after(6, node)
node
front_
back_
data_ 2
next_
prev_
data_ 6
next_
prev_
data_ 4
next_
prev_
data_ 3
next_
prev_

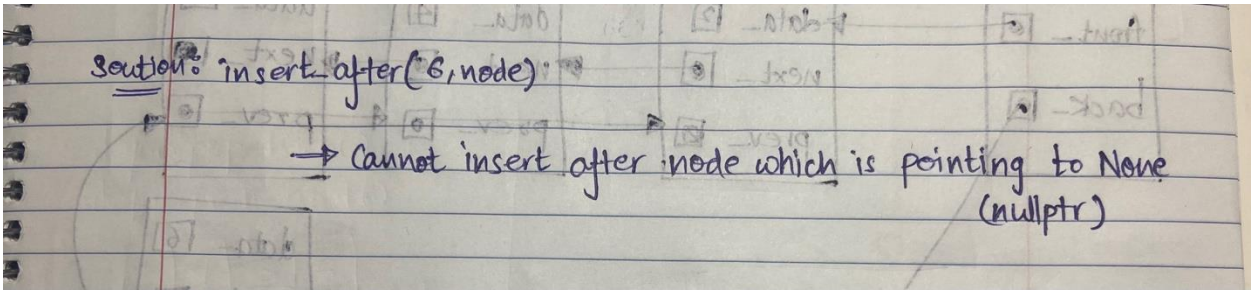

node
insert_after(6,node)
front_
back_
data_ 2
next_
prev_
data_ 4
next_
prev_
data_ 3
next_
prev_


Solution : insert_after (6, node)
node
front_
back_
data_ 2
next_
prev_
data_ 4
next_
prev_
data_ 3
next_
prev_
data_ 6
next_
prev_


node
insert_after(6,node)
front_
back_
data_ 2
next_
prev_
data_ 4
next_
prev_
data_ 3
next_
prev_


node
solution: insert_after(6, node)
front_
back_
data_ 2
next_
prev_
data_ 4
next_
prev_
data_ 6
next_
prev_
data_ 3
next_
prev_


node
insert_after(6,node)
front_
back_
data_
2
next_
prev_
data_
4
next_
prev_
data_
3
next_
prev_

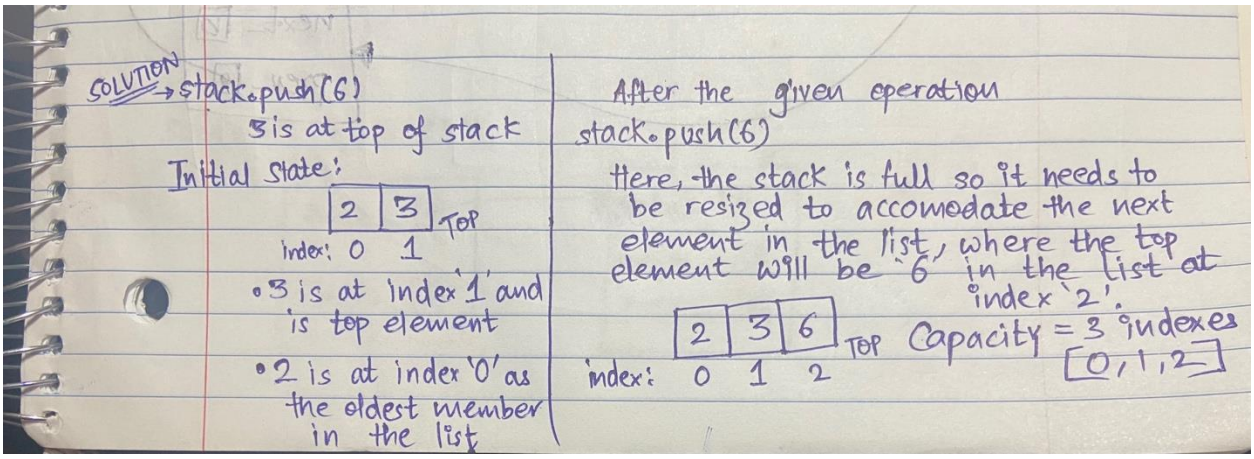

Soution: insert_after(6,node)
⇒ Cannot insert after node which is pointing to None
(nullptr)

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

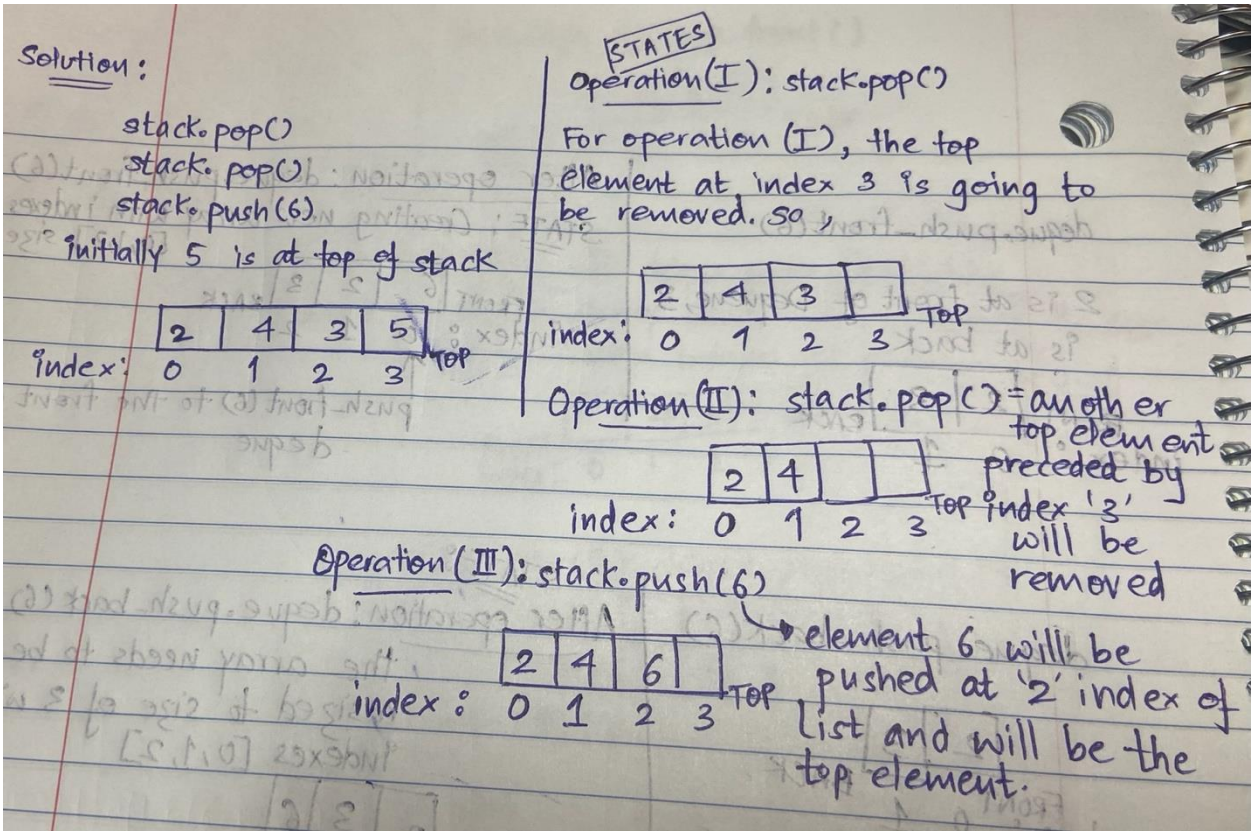
2	3



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

initially 5 is at top of stack

2	4	3	5

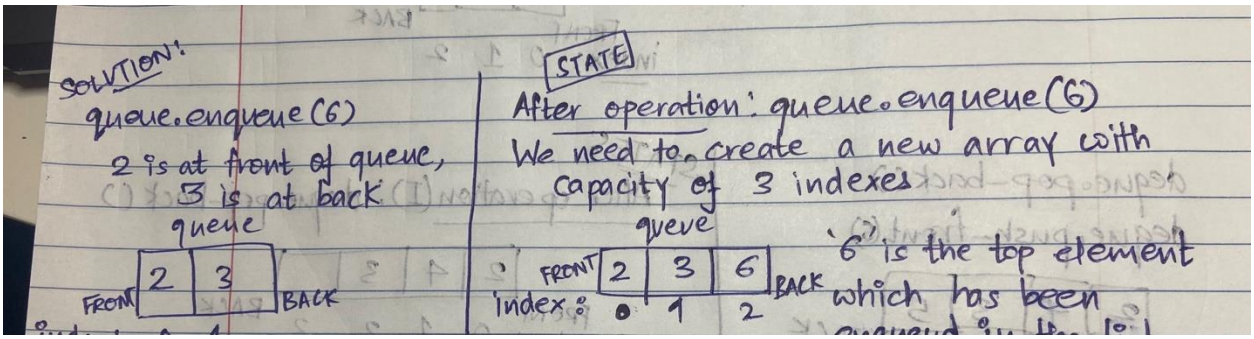


Queues: In the diagrams below list what data members you need to track and what their values are in its initial state and their state after each of the operations are applied to the diagram. If the array needs to be resized, draw the new array with the correct capacity

queue.enqueue(6)

2 is at front of queue, 3 is at back

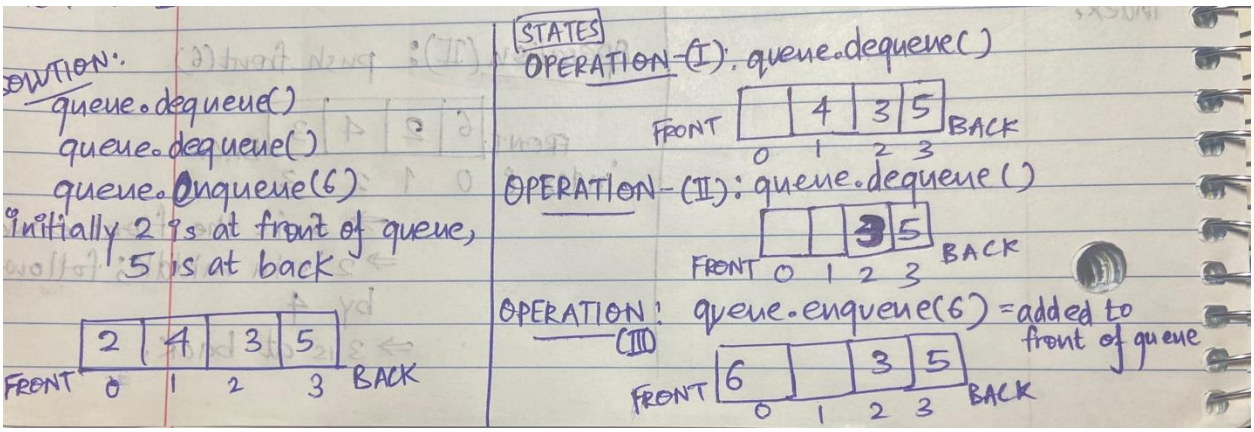
2	3



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

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

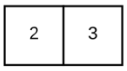
2	4	3	5

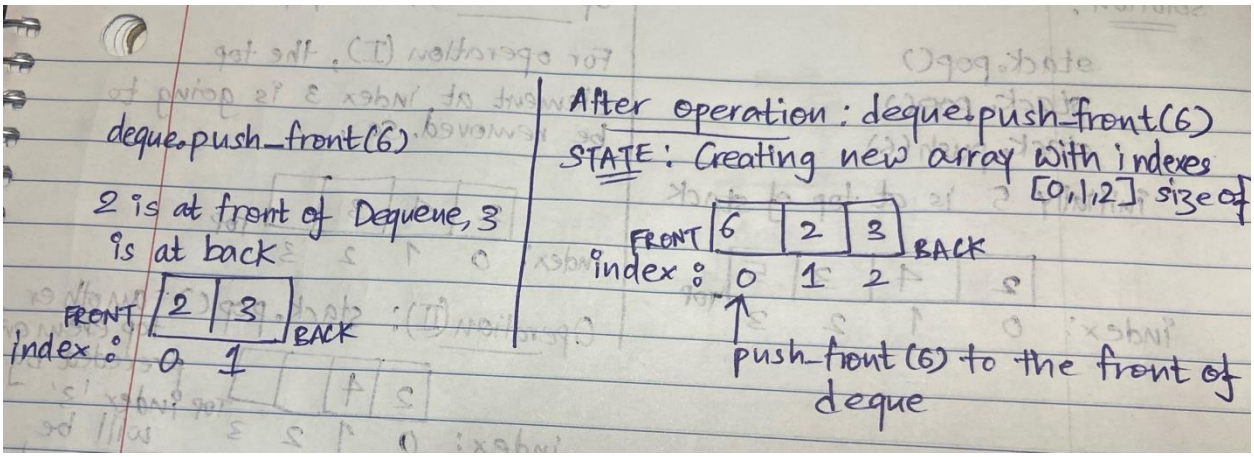


Deques: In the diagrams below list what data members you need to track and what their values are in its initial state and their state after each of the operations are applied to the diagram. If the array needs to be resized, draw the new array with the correct capacity

deque.push_front(6)

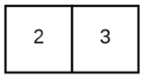
2 is at front of Deque, 3 is at back

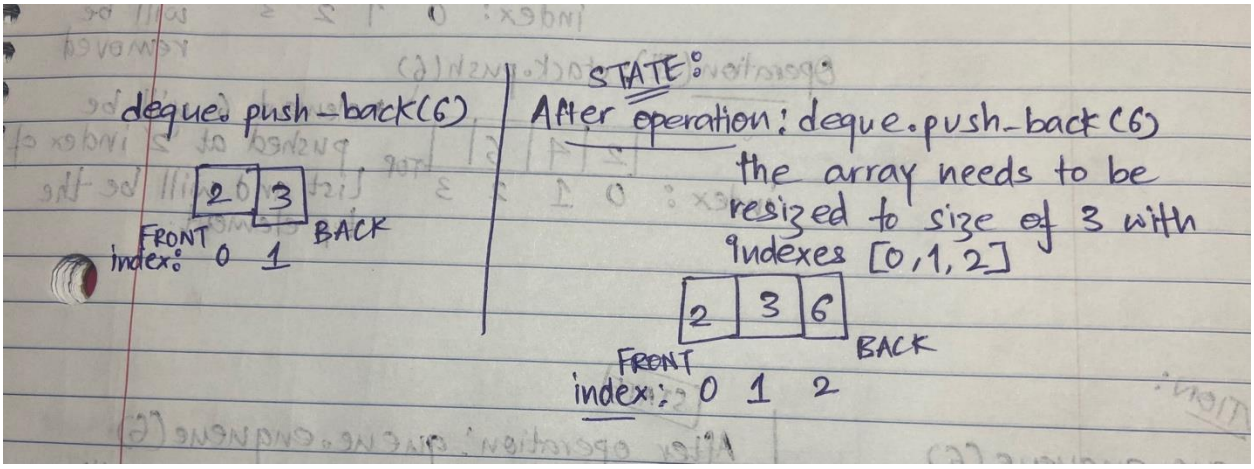




deque.push_back(6)

2 is at front of Deque, 3 is at back

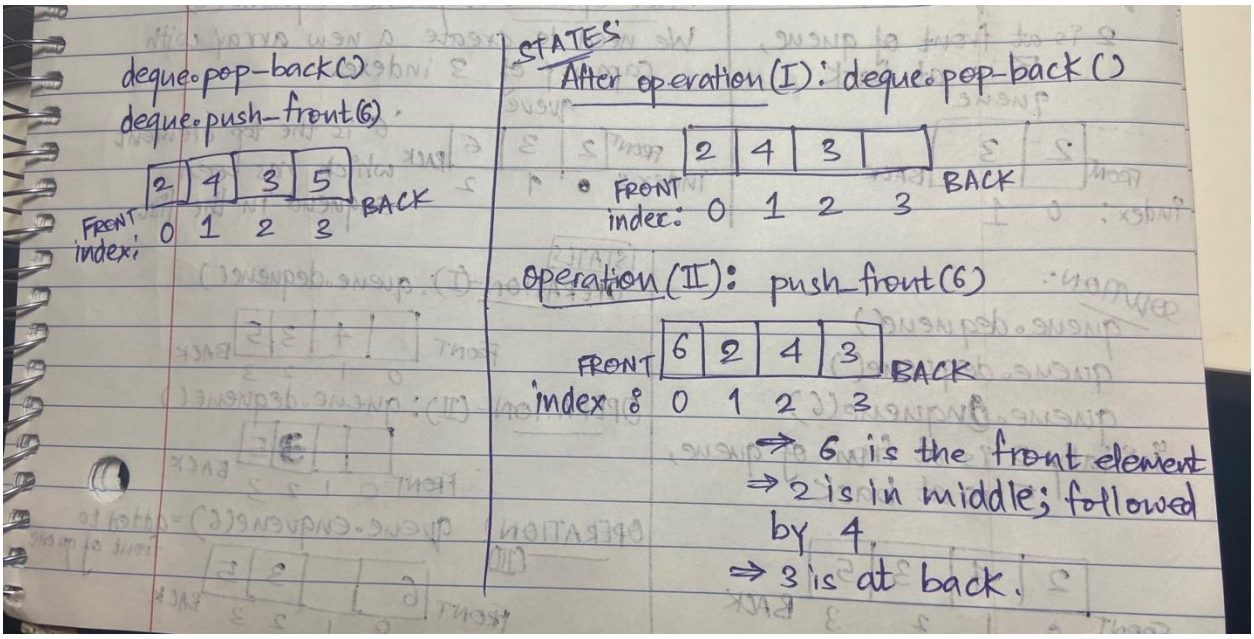




deque.pop_back()
deque.push_front(6)

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

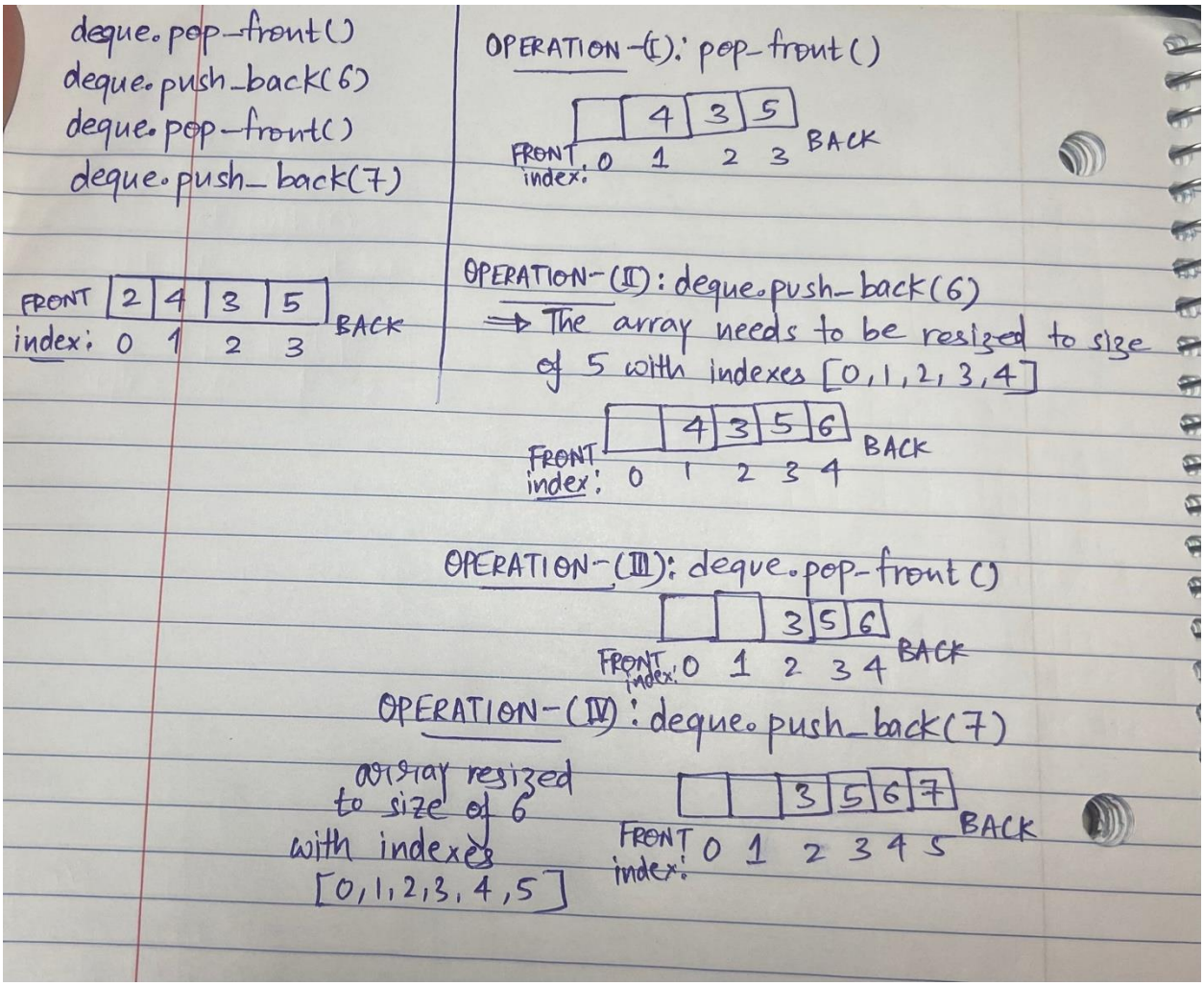
2	4	3	5



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



overflow(grid,the_queue) - apply the overflow function to the gride 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