positional lists &

Wk 3 - tut (stack & queues)

1) Implementing stack with singly linked list

push: O(n) go from head to end of the element & add the elem

pop: ~~O(n)~~

} or think of head as the last element (reversed) O(n)

O(n)

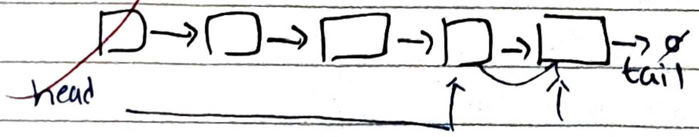
2) Implementing queue using ~~singly~~ linked list

enqueue: O(1) append at tail

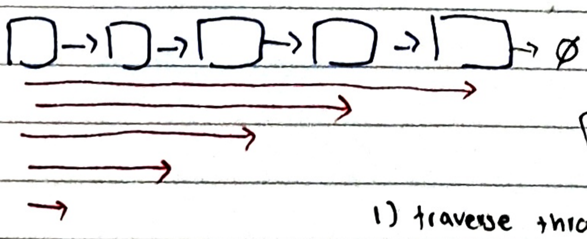
dequeue: O(1) ~~ap~~ remove ~~from~~ next ~~of~~ Node to head.

storing pointer to tail is ok I guess? explain pointer is necessary in exam if asked.

3) 3b traversing singly linked list in reverse order



* use ~~(O)~~ O(1) additional space



time
O(n²) (~~Nodes~~ len, midNode)

* 이거 더 [illegible]

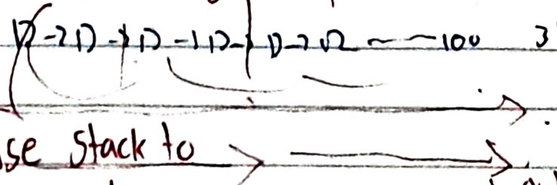
my solution

1) traverse through whole list, ~~set~~ point to last elem with "last".

2) traverse through list unttle Node with next elem pointed by "last", then "last" points to the cur node.

3) repeat step 2 until "last" points to "head"

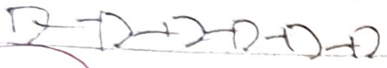
* use O(√n) additional space (add √n many new pointers)



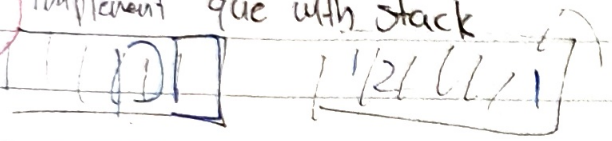
* use stack to traverse each pointer

```
traverse(list's ~~head~~ ~~midpoint~~ mid Node):
    if len(list) < 3: mid Node, next Node
        print( ~~10~~ ~~head~~ ~~list[midpoint~~
```

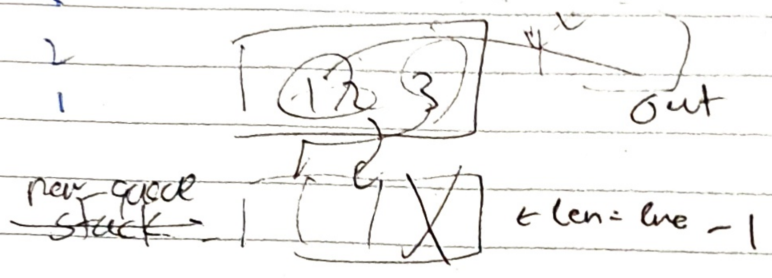
tutor's solution



4) Implement que with stack



push 1
push 2
pop 1



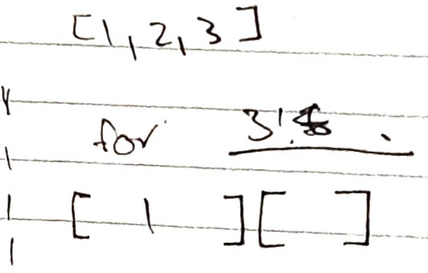
* Got hint from tutor

5) getAverage() in queue
O(1) ?

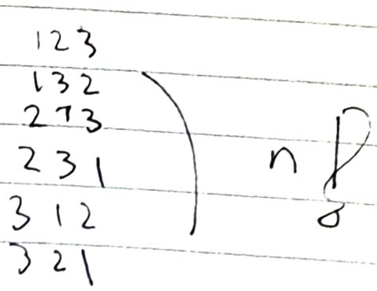
→ have a variable sum
instance

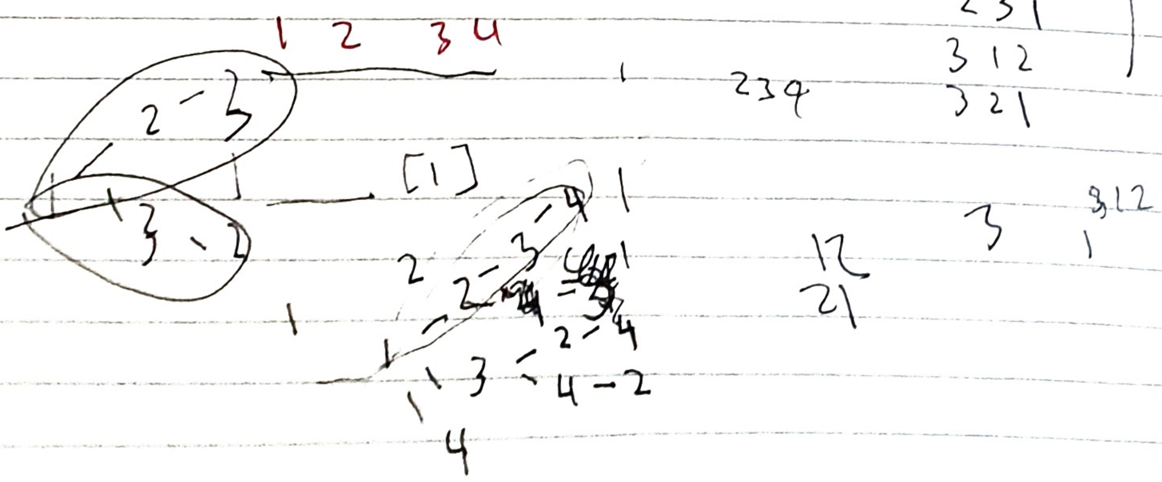
→ add value to var (sum) when enqueuing
↳ substract value from (sum) when dequeuing

getAverage() → q.sum / len(q) ← Assume length is already kept track of



6) All possible permutation {1, 2 .... n}





1
2
3
4

1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 2 3
1 4 3 2
2 1 3 4
2 1 4 3

1 , 1 (2, 1 3)

prev

prev (2, 3)

prev [[3]]

pred (2,) o 1 (3, 2]

0 : 0

1 : 4 1 : 1 1) (2 3)

[1 2 3

1 2

o o

i o

(2 , [1]

[ ]

[1]

[[

[1, 2] [2, 1]

Q6.

```
def permutations(lst):
    if len(lst) == 1 :
        return [lst]
    for i 0 ~ len(lst):
        rest = lst[:i] + lst[i+1:]
```

[[3, 1, 2], [3,

[1, 2, 3], [

[2, 1] [1, 2]

[3, 2, 1] [3, 1, 2]

[2, 3, 1] . [1, 3, 2]

[2, 1, 3] [1, 2, 3]

A B C

ABC
ACB
BAC
BCA
CAB
CBA

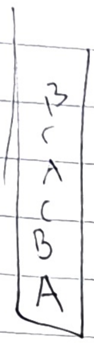
for n in A, B, C

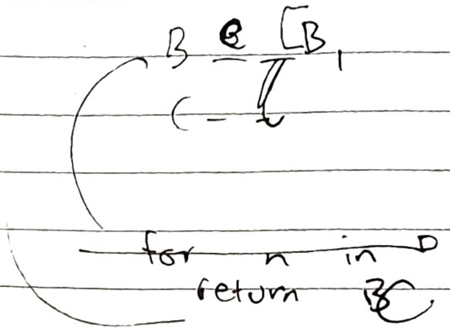
n __ __

B

for n in B, C

B C



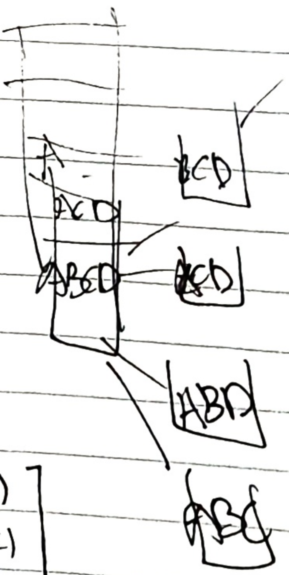


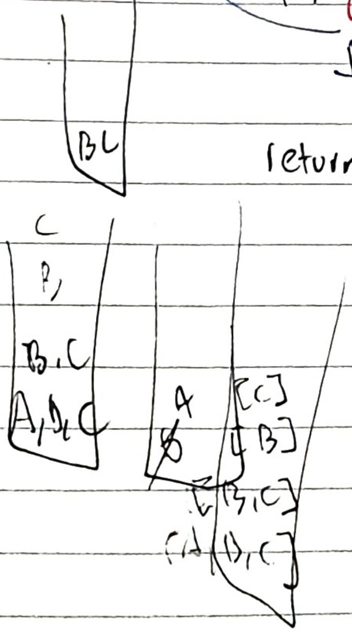
Q6 pseudo code <permutations>

perms = [ ]

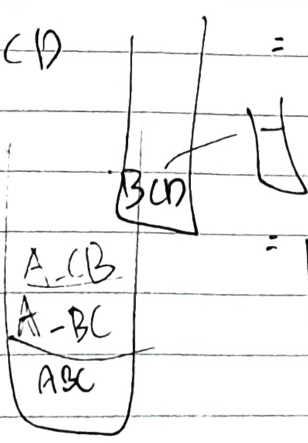
```
def permutations(input_list):
    if length of input_list is 0:
        return [input_list]
    for i from 0 to (length of input_list):
        rest = input_list[:i] + input_list[i+1:]
        rest_perms = permutations(rest)
        for rp in rest_perms:
            perms.append([input_list[i] + rp])

    return perms
```





ABCD



ABC

$= \begin{bmatrix} [A] + perm(BC) \\ [B] + perm(AC) \\ [C] + perm(AB) \end{bmatrix}$

$= \begin{bmatrix} [A] + [B] + perm(C), [A] + [C] + perm(B) \\ \\ \end{bmatrix}$