3C 1)

append operation 's amortise time is O(1) because it doesn't need to go through the list but the end, even sometime need to resize (enlarge), the average performance is still O(1) constant time

append(1)
append(2)
....
append(n)

O(1) × n
: O(n)

pop()(default) 's amortise time is O(1) because it add item in the end of list, even sometime need to shrink the list, the average performance is still O(1), constant time

pop()
:
× n
:
pop()

O(1) × n
O(n)

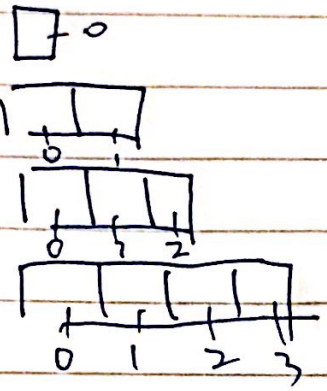
O(n) + O(n) => 2n operation

2) ① if the lst goes shrinking to half when the actual size of list is less than half of the capacity, the runtime of pop will increase to 2n from $\frac{3}{4}n$, but will save storage

shrink $\frac{1}{2}$: 16 8 4 2 1

shrink $\frac{1}{4}$: 16 4 1

② if we append object by just increase one capacity to resize, it will be a $O(n^2)$ operation:



$1+2+3+4\cdots+n = O(n^2)$

also, if we shrink in every pop() operation, it will be $O(n^2)$

4b Seperate loop operations for three times
$O(3n) \Rightarrow O(n)$

5a: while loop (n) worst
inside remove operation go through list (n) worst

worst $O(n^2)$

5C: worst case go through two seperate while loops, $O(2n)$