Q1:

1.

```
def reverse1(lst):
    rev_lst = []
    i = 0
    while(i < len(lst)):
        rev_lst.insert(0, lst[i])
        i += 1
    return rev_lst
```

O(1)
O(1)
O(n²)
time complexity: Θ(n²)

iterate for n times
arithmetic, related to len(rev_lst)
O(1)

2.

```
def reverse2(lst):
    rev_lst = []
    i = len(lst) - 1
    while (i >= 0):
        rev_lst.append(lst[i])
        i -= 1
    return rev_lst
```

O(1)
O(1)
O(n)
time complexity: Θ(n)

iterate for n times
O(1)
O(1)

Q2: C.

1. append pop

$= n + (1+2+4+\cdots+n) + n + (1+4+16+\cdots+n)$

$\leq 2n + 2^{\lfloor \log_2(n) \rfloor + 1} - 1 + 2^{\lfloor \log_4(n) \rfloor + 1} - 1$

$\leq 2n + n + n = 4n \quad \therefore O(n)$

2. the worst case for sequence n:

every append/pop is after pop/append
that needs to resize the array for every operand

$n \times n = n^2 \quad \therefore \Omega(n^2)$

Q3: b.

```
def find_duplicates(lst):
    repeat = [i for i in range(len(lst))]                #O(n)
    for elem in lst:                                     #O(n)
        if repeat[elem] != None and repeat[elem] != 0:   ##O(1)
            repeat[elem] = None                          ##O(1)
        elif repeat[elem] == None:                       ##O(1)
            repeat[elem] = 0                             ##O(1)
            repeat.append(elem)                          ##O(1)
    return repeat[len(lst):]                             #O(n)

#Time complexity: O(n)
```

Q4: a.

worst case running:
Θ(n²)

```
def remove_all(lst, value):
    end = False
    while(end == False):        iterate n times
        try:                    O(n²)
            lst.remove(value)   O(n)
        except ValueError:
            end = True
```

c.

```
def remove_all(lst, value):                                                    ⚠ 17
    i = 0                                                  #O(1)
    count = 0                                              #O(1)
    try:
        while i < len(lst):                                #O(n)
            if lst[i] == value:                            ##O(1)
                j = 1                                      ##O(1)
                while lst[i + j] != value:                 ##O(1) iteration times=seperation of two consecutive values in lst
                    j += 1                                 ##O(1)
                lst[i-count:i+j-1-count] = lst[i+1:i+j]    ##O(1)
                count += 1                                 ##O(1)
                i += j - 1          ##O(1) reduce iteration time for outer while loop=iteration time in inner while loop
            i += 1                                         ##O(1)
    except IndexError:
        if lst[-1] == value:                               ##O(1)
            count += 1                                     ##O(1)
    finally:
        for n in range(count):                             ##O(n)
            lst.pop()                                      ##O(1)
        return lst

#Time complexity=O(n)
```