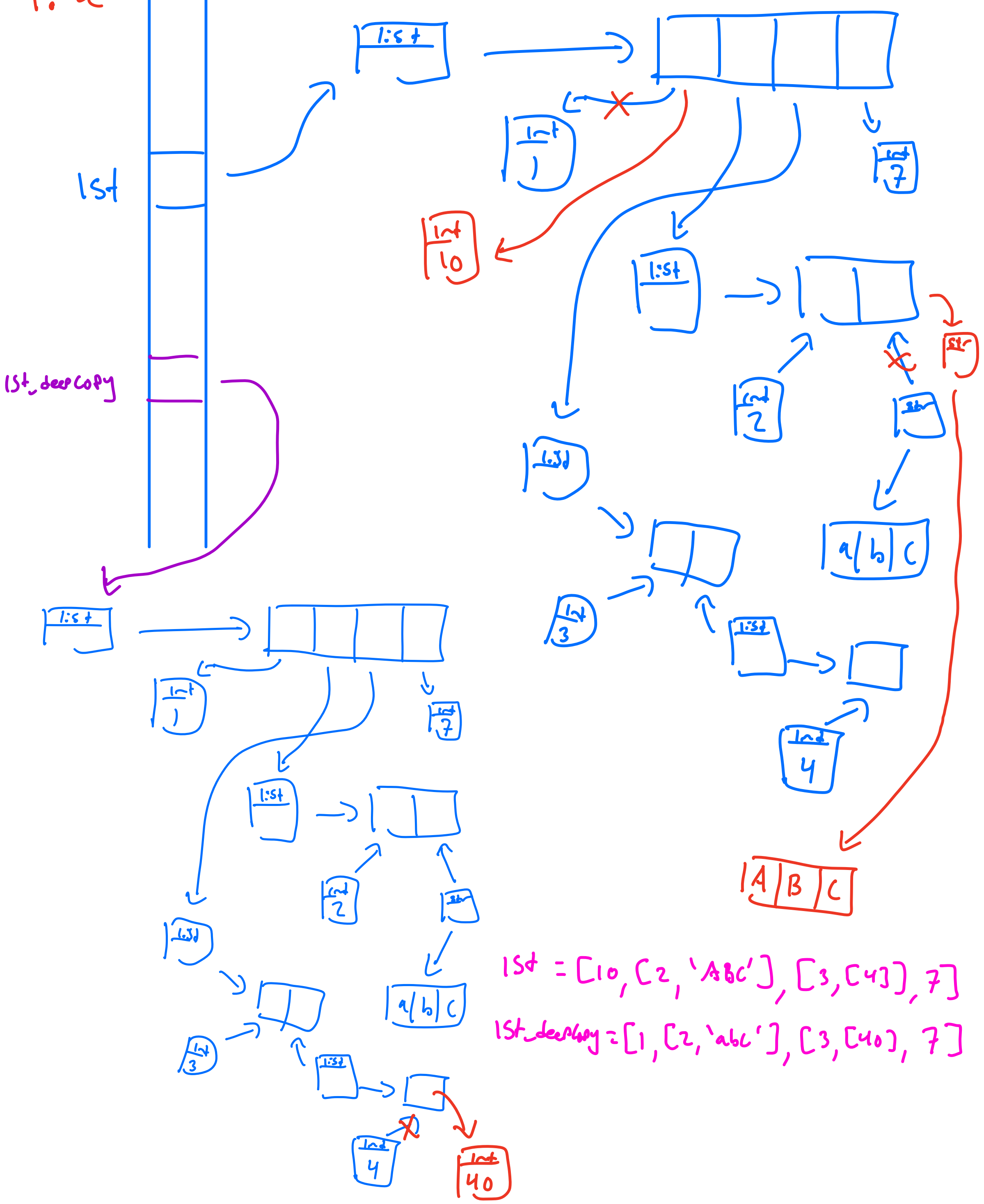
1. a

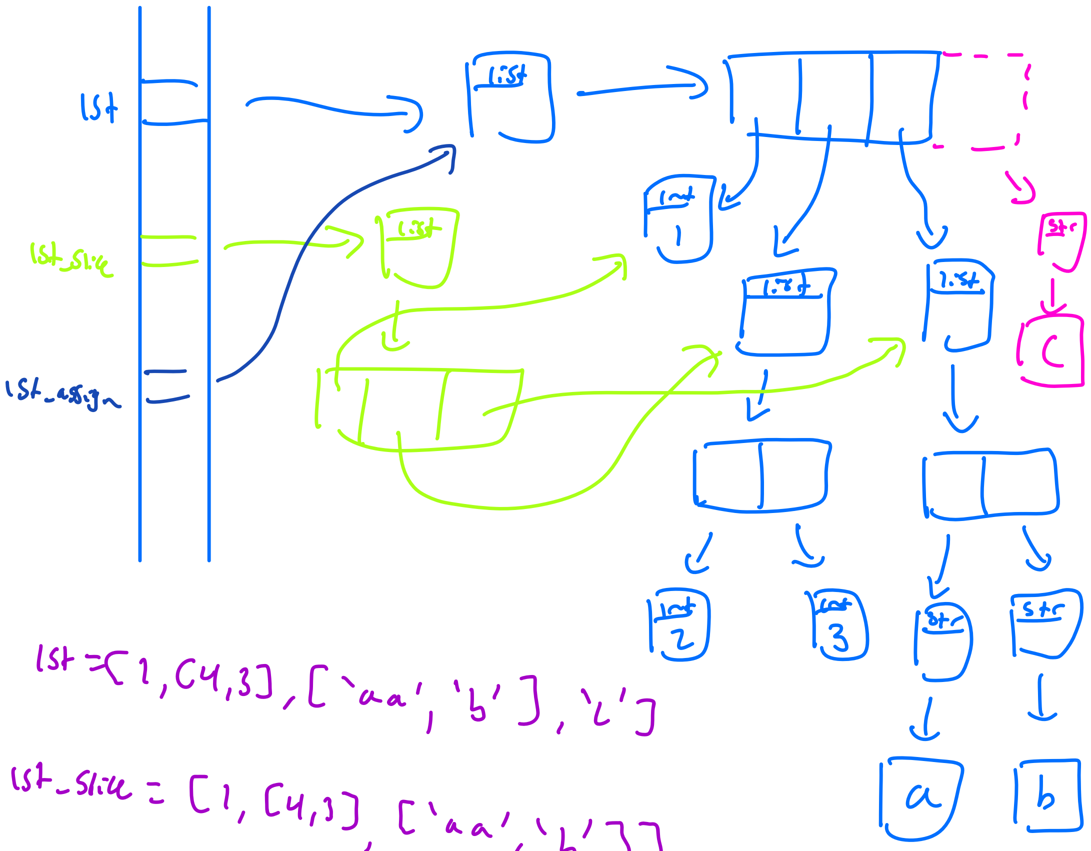
Ty Wenrick



lst = [10, [2, 'ABC'], [3, [4]], 7]

lst_deepcopy = [1, [2, 'abc'], [3, [40]], 7]

1. b



lst =[1,[4,3],['aa','b'],'c']

lst_slice = [1, [4,3], ['aa','b']]

2. n^2+5n-2 is $O(n^2)$

$n^2+5n-2 < n^2+5n^2+2n^2 < c*n^2$ for all $n > n_0$

$n^2+5n-2 < n^2+5n^2+2n^2$

$c = 1+5+2 = 8$

$n^2+5n-2 < 8n^2$

$c=8 \quad n_0=1$

b. $\frac{n^2-1}{n+1}$ is $O(n)$

$\frac{n^2-1}{n+1} < c*n$ for all $n > n_0$

$n-1 < c*n$

$n < 2*n+1$

$c=2$ $n_0=1$

c. $\frac{n^2+1}{n+1} < C*n$

$n^2+1 < c*n(n+1)$

$n^2+1 < c(n^2+n)$

$n^2+1 < n^2+n$

$c=1$ $n_0=1$

d. $\sqrt{5n^2-3n+2}$ $\Theta(n)$

Big O $d=\sqrt{10}$ $n_0=1$ $(\sqrt{10}\approx 3.16)$

S

ran out of time for this question.

3. a) $8n^2(\sqrt{n})$ is $O(n^3)$

TRUE

$8n^2(\sqrt{n}) < c * n^3$ for all $n > n_0$

$8n^2(\sqrt{n}) < 8n^3$

$8n^{2.5} < 8n^3$

$c = 8$ $n_0 = 1$

b) $8n^2(\sqrt{n})$ is $\Theta(n^3)$

False

n gets infinitely larger

no tight bound

c. TRUE

$16\log(n^2) + 2 < c * \log(n)$

$16 * 2\log(n) + 2 < c * \log(n)$

4. Given n numbers

$1+1+1+1+1 \ldots +1 = n = \Theta(n)$

$n+n+n+n \ldots +n = n^2 = \Theta(n^2)$

$1+2+3+4+5 \ldots +n = \frac{n(n-1)}{2} = \Theta(n^2)$

Given log(n) numbers where n < 2

$1 + 2+4+8+\ldots+n$

time limit up

adding powers of 2 $\underline{2n-1} = \Theta(n)$

← notes

$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} \ldots + 1 = \Theta(n)$

$n\left(1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16} \ldots\ldots + \frac{1}{n}\right)$

$n(1+1 \cdot \frac{1}{n}) = 2n-1$

$2n-1$

5. a)

```
def func(lst):
    for i in range(len(lst)):
        if (len(lst) % 2 == 0):
            return
```

O(n) becase len(lst) doesn't change can be odd number

b)

```
def func(lst):
    for i in range(len(lst)):
        if (len(lst) % 2 == 0):
            print("i =", i)
        else:
            return
```

O(n)

6.

Constant	$f(n)$
log	$\log(n)$
Sqrt	$\sqrt{n}$
linear	$f(n)$
linear log	$\sqrt{\log(n)}$
quadratic	$f(n^2)$
cubic	$f(n^3)$
factorial	$f(n!)$

7. a)

```
def func1(n)                          # n =16
        if (n<=1):
              return 0
        else:
            return 10 + func1(n-2)
```

Call Stack ↘

func1(16)^{80}

↓

10+ func1(14)^{70}

↓

10+ func1(12)^{60}

↓

10+ func1(10)^{50}

↓

10+ func1(8)^{40}

↓

10+ func1(6)^{30}

↓

10+ func1(4)^{20}

↓

10+ func1(2)^{10}

↓

10+ func1(0)

↓

0

func1(16)

outputs:

80

n-2

$\Theta(n)$

b)
```
def func2(n):          # n=16
    If (n <= 1):
        return 1
    else:
        return 1 + func2(n//2)
```

$\frac{n}{2}$

$1 + \overset{4}{\text{func2}}(8) \rightarrow 1 + \overset{3}{\text{func2}}(4) \rightarrow 1 + \overset{2}{\text{func2}}(2)$

$\downarrow$

$1 + \overset{1}{\text{func2}}(1)$

$\downarrow$

1

$1+4=5$

output : 5

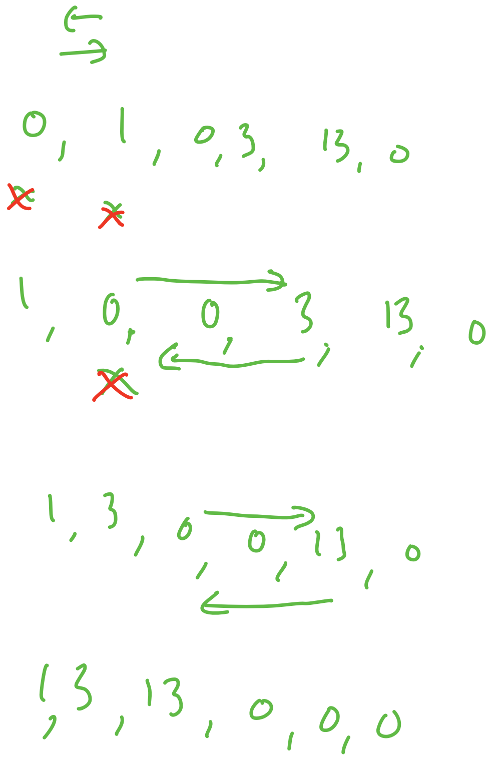
$\Theta(n)$

Coding

1.
```
def reverse_list(lst):
    return [i for i in lst[::-1]]
```

b.
```
def reverse_list(lst, low =None, high =none):
    If low is None:
        low = 0
    If high is None:
        high = len(lst) -1

    While low < high:
        lst[low], lst[high] = lst[high], lst[low]
        low+ =1
        high -= 1
```

```
2.      def  move_zeroes(nums):

             last_zero = 0

             for i  in range(len(nums)):

                    if  nums[i] != 0:

                         nums[i], nums[last_zero] =

                         last_zero += 1       nums[last_zero]
                                              ,
                                              nums[i]
```



```
3. a.  def find_missing(lst):
           for num in range(len(lst) +1):
               if num not in lst:
                   return num

   b.  def find_missing(lst):
           if lst[0] != 0:
               return 0
           for i in range(1, len(lst)):
               if lst[i] - lst[i-1] >1:
                   return lst[i] - 1
           return len(lst)
```

3. C

```
def find_missing(lst):
    total = sum(lst)

    for num in range(len(lst)+1):
        total -= num

    return -total
```

4.

```
def sum_to(n):          #n=3
    if n<0:
        return 0
    else:
        return n + sum_to(n-1)
```

for n=3

3 + sum_to(2) → 2 + sum_to(1)

↙

3+2+1+0=6   1 + sum_to(0)

↙

0 + sum_to(-1) → 0

```
                            srt_lst
5.   def  binary_Search(  , low, high, val):
          If  low > high:
               return  None

          mid = (low + high) // 2

          If    srt_lst[mid]  ==  val
               return  mid

          elif   srt_list[mid]  <  val
               return  binary_Search(srt_lst, val, mid+1,
                                                          high)
          else:
            return  binary_Search(srt_list, val, mid-1, high)
```

6. a.

```
def find_max(lst) :
    if len(lst) == 1 :
        return lst[0]
    prev = find_max(lst[1:])
    if prev > lst[0]:
        return prev
    return lst[0]
```

↗

determine runtime

b.

```
def find_max(lst, low, high):
    if low == high :
        return lst[low]
    curr_max = find_max(lst, low+1, high)
    if curr_max < lst[low]:
        curr_max = lst[low]
    return curr_max
```

7. **Solution on lab**

```
def vc_count(word, low, high):
    if low >= high:
        if word[low] in "aeiouAEIOU":
            return (1, 0)
        return (0, 1)
    else:
        prev = vc_count(word, low+1, high)
        if word[low] in "aeiouAEIOU":
            return (prev[0] + 1, prev[1])
        return (prev[0], prev[1] +1)
```