CS-UY 1134 - Fall 2017 midterm 1

Justin Lin

TOTAL POINTS

45 / 100

QUESTION 1

1 Question 1 **6 / 9**

✓ **- 3 pts** section (II) is wrong

QUESTION 2

2 Question 2 **9 / 12**

✓ **- 3 pts** lst3 is wrong

QUESTION 3

3 Question 3 **8 / 12**

✓ **- 2 pts** used prefix_sum, but logic is correct

✓ **- 2 pts** list iteration is off by one

QUESTION 4

4 Question 4 **8 / 12**

✓ **- 2 pts** sec1: inaccurate explanation - calculated n*n instead of. 1+2+..+n,

✓ **- 2 pts** sec2: inaccurate explanation - did not take resizing of the list in the explanation

QUESTION 5

5 Question 5 **3 / 10**

✓ **- 3 pts** section (i): tree has wrong structure

✓ **- 2 pts** section (i): local costs are wrong

✓ **- 2 pts** section (ii): wrong answer, but consistent with mistake made in section (i)

QUESTION 6

6 Question 6 **10 / 30**

✓ **- 20 pts** Major logic flaws

Were we supposed to infer a loop?

Single letter variable names? Bad code.

QUESTION 7

7 Question 7 **1 / 15**

✓ **- 3 pts** Wrong base

✓ **- 12 pts** Wrong recursive step

\+ **1** Point adjustment

Name: Justin Lin Net ID: Jwl488

Question 1 (9 points)
For each of the following, state if it is true or false (circle your choice):

I. $\sqrt{3n^2 + 4n - 5} = \theta(n)$ TRUE / FALSE

II. $\log_2(n^2) = O(\log_2(n))$ TRUE / FALSE

III. $\log_2(n) = \Omega(\sqrt{n})$ TRUE / FALSE

Question 2 (12 points)

What is printed when the following Python code is executed?

```
lst1 = [1, [2, 3], [4, 5], 6]
lst2 = lst1
lst3 = lst1[ : ]
lst4 = copy.deepcopy(lst1)

lst1[0] = 10
lst1[1][1] = 30
print("lst1 =", lst1)
print("lst2 =", lst2)
print("lst3 =", lst3)
print("lst4 =", lst4)
```

Output:

lst1 = [10, [2,30], [4,5], 6]
lst2 = [10, [2,30], [4,5], 6]
lst3 = [10, [2,30], [4,5], 6]
lst4 = [1, [2,3], [4,5], 6]

Name: justin lin Net ID: jwl488

Question 3 (12 points)

Let *lst* be a list of integers. We define ***prefix_sum(i)*** to be the sum of the first *(i+1)* elements of *lst*. That is: *prefix_sum(i) = lst[0] + lst[1] + … + lst[i]*.

For example, if *lst = [-1, 1, 4, -2, -3]*,

- *prefix_sum(0)* is -1
- *prefix_sum(1)* is 0, since (-1) + 1 = 0
- *prefix_sum(2)* is 4, since (-1) + 1 + 4 = 4
- *prefix_sum(3)* is 2, since (-1) + 1 + 4 + (-2) = 2
- *prefix_sum(4)* is -1, since (-1) + 1 + 4 + (-2) + (-3) = -1

Complete the definition below for the function:

```
def positive_prefix_sum(lst)
```

This function is given a list of integers, `lst`. When called, it creates and returns a list with all the indices of which their prefix-sum is positive.

For example, if `lst = [-1, 1, 4, -2, -3]`, calling `positive_prefix_sum(lst)` should return `[2, 3]`.

Notes:

1. Your implementation should all be in the `return` line. That is, you are not allowed to add lines to this function, define an additional function, etc.
2. **You may want to use the build-in `sum` function.**
3. The indices should come in an ascending order.
4. In this question, don't worry about the runtime of your implementation.

def positive_prefix_sum(lst):

return ~~[x for x in range(1, len(lst)) if (prefix_sum(lst[0:x]) > 0)]~~

[x for x in range(0, len(lst)-1) if prefix_sum>0]

Name: Justin Lin Net ID: jnl488

Question 4 (12 points)
Consider the following two implementations of a function that if given a list, `lst`, create and return a new list containing the elements of `lst` in reverse order.

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

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

1. If `lst` is a list of n integers, what is the worst case running time of `reverse1(lst)`? Give a short explanation of your answer.

Your Answer: $\theta(n^2)$

The while-loop will run n times, ~~on~~ and rev_lst.insert(0, lst[i]) will run also n-time b/c it needs to shift all elements in the list if ~~greater~~ the list is a size larger than 1, making the total run-time $O(n) \cdot O(n) = O(n^2)$

2. If `lst` is a list of n integers, what is the worst case running time of `reverse2(lst)`? Give a short explanation of your answer.

Your Answer: $\theta(n)$

The while-loop will run n times, or $O(n)$, with the body of the loop running $O(1)$ time due to appending to the end of a list, making the total run-time $O(n) \cdot O(1) = O(n)$

Name: Justin Lin Net ID: jwl488

Question 5 (10 points)

Below, you are given a recursive implementation for the function:

```
def min_in_lst(lst, low, high)
```

This function gets the list of integers `lst`, as well as two indices: `low` and `high` (`low ≤ high`), which indicate the range of indices of the elements that should to be considered.

When called, the function would return the **minimum value** out of all of elements in the `low, low+1, …, high` positions.

```
def min_in_lst(lst, low, high):
    if(low == high):
        return lst[low]
    else:
        mid_ind = (low + high) // 2        binary
        min1 = min_in_lst(lst, low, mid_ind)
        min2 = min_in_lst(lst, mid_ind + 1, high)
        if(min1 < min2):
            return min1
        else:
            return min2
```

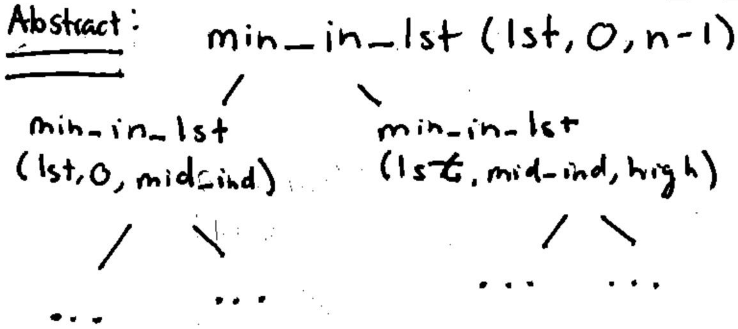
Assuming `lst` is a list of n elements, analyze the worst case running time of the implementation above:

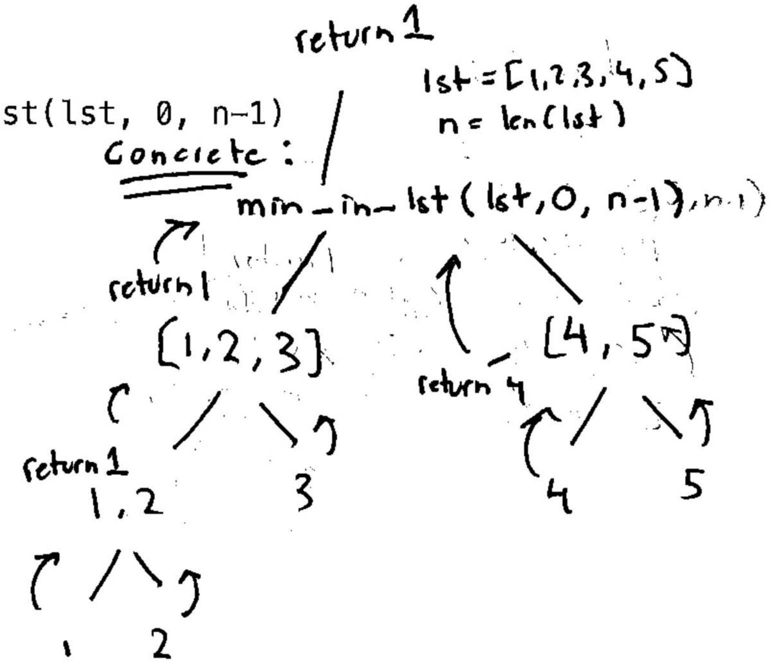
1) Draw the recursion tree that traces the recursive calls of the function.
2) Conclude the total (asymptotic) running time of the function.

Note: Write your answers in the next page.

Name: justin lin Net ID: jw1488

i. Draw the **recursion tree** for `min_in_lst(lst, 0, n−1)`





ii. **Conclude the running time** of `min_in_lst(lst, 0, n−1)`

Your Answer: $\theta(\ 2^n\ )$

Calculations:

The run-time will get exponentially longer by a factor of two each time, with each node doing the work of O(1), meaning $O(2^L) \cdot O(1) = O(2^n)$

Name: ~~gal~~ justin lin Net ID: jwl 418

Question 6 (30 points)
Implement the function:

```
def remove_all_evens(lst)
```

This function gets a list of positive integers, `lst`. When called, it should remove all the even numbers from `lst`, and keep only the odd ones.
Note: The relative order of the odd numbers that are left in `lst` at the end, doesn't matter.

For example, if `lst = [2, 3, 5, 2, 16, 13]`, after calling `remove_all_evens(lst)`, `lst` could be the following 3-element list: `[13, 5, 3]`.

Implementation requirements:

1. Your implementation should be in-place.
2. At the end, your list will contain only the odd numbers.
3. Your function should run in **worst case linear time**. That is, if there are n items in `lst`, calling `remove_all_evens(lst)` will run in $\theta(n)$.
4. For the memory used, in addition to `lst`, you are allowed to use only $\theta(1)$ memory. That is, for example, you could **not** use an additional non-constant sized list. But, you could have a few variables.

Note: Write your implementation on the next page.

Logic

2, 3, 5, 2, 16, 13
↑
a, b

2, 3, 5, 2, 16, 13
↑ ↑
a b

3, 2, 5, 2, 16, 13
a b

3, 5, 2, 2, 16, 13
a b

[illegible]
a b

if b = ~~even~~ odd swap
b += 1
a += 1

if b = even
b += 1

[illegible]

Name: justin lin Net ID: jwl488

```
def remove_all_evens(lst):
    a=0
    b=0
    counter_even =0
    if lst[b]%2==1:
        lst[a],lst[b] = lst[b],lst[a]
        a = a+1
        b = b+1
        counter_even = counter_even+1
    if lst[b]%2==0:
        b = b+1
    if b == (len(lst)-1):
        for i in range(0,counter_even):
            lst.pop()
    return lst
```

Name: justin lin Net ID: jnl488

Question 7 (15 points)

Give a **recursive** implementation for the function:

```
def is_sorted(lst, low, high)
```

This function is given a list of numbers, `lst`, as well as two indices: `low` and `high` (`low ≤ high`), which indicate the range of the indices for the elements that should to be considered.
When called, the function should determine if the elements that are placed at the `low, low+1, …, high` positions, are in an (ascending) sorted order. That is, it should return **True** if-and-only-if `lst[low] ≤ lst[low+1] ≤ … ≤ lst[high]`

For example, if `lst = [1, 3, 6, 8, 12, 15, 31]`,
the call `is_sorted(lst, 0, 6)`, will return **True**.

Implementation requirements:
Your function should run in **worst case linear time**. That is, if n is the size of the range `low, low+1, …, high`, calling `is_sorted(lst, low, high)` will run in $\theta(n)$.

Note: Write your implementation on the next page.

Logic

[1, 3]   6, 8, 12, 15, 31

1, 3, 6, 8, 12, 15, 31

1   1   6   6

low+2

low ~~+2~~: 2

return [0 : 2] + [2 :]

lst[0] < lst[1]   [0 : 2]   [3 :]

return

Name: justin lin Net ID: jnl 488

```
def is_sorted(lst, low, high):
    if len(lst ==2):
        return lst[0] < lst[1]                      len(lst) > 2
    if len(lst[low: low +2]) == 2 and high-low>=n
        return is_sorted(lst[low:low +2], low, high) +
                 is_sorted(
                 lst[low +2: high], 0, high-low)
    elif len(lst) == 3:
        return (lst[0] < lst[1] and lst[1] < lst[2])
    else:
        return                                      len(lst) == 2
    if len(lst[low: low+2]) ==2 and
        return is_sorted(lst[low:low+2], low, high)
```

no space

your solution doesn't deal with ranges of n>3

Name: ______________________ Net ID: ________________

EXTRA PAGE IF NEEDED

Note question numbers of any questions or part of questions that you are answering here.
Also, write "ANSWER IS ON LAST PAGE" near the space provided for the answer.