Name: Sahil Singh Net ID: SKS9379

NYU, Tandon School of Engineering
CS-1134: Data Structures and Algorithms — Spring 2023

CS-1134 – Midterm Exam

Tuesday, March 7, 2023

i. You have one hour and 15 minutes.

ii. There are 5 questions all together, with 100 points total.

iii. **Do not unstaple the exam or remove any pages**.

iv. Write your Name and NetID at the top of **each page**.

v. **Don't use pencils**, as they don't show up well when scanned.

vi. Write your answers clearly and concisely, in the spaces on the exam. Try to avoid writing near the edge of the page. **YOU MAY NOT USE THE BACKSIDE OF THE EXAM PAPERS**, as they will not be looked at. If you need extra space for an answer, use the **extra page at the end of the exam** and **mark it clearly**, so we can find it when we're grading.

vii. Calculators are not allowed.

viii. Read every question completely before answering it.

ix. For any questions about runtime, show the worst-case asymptotic runtime, using Theta notation.

x. You do not have to do error checking. Assume all inputs to your functions are as described

xi. Cell phones, and any other electronic gadgets must be turned **off**.

xii. Do not talk to any students during the exam. If you truly do not understand what a question is asking, you may raise your hand when one of the CS1134 instructors is in the room.

Name: Sahil Net ID: SKS9379

Question 1 (20 points)

a) What is printed when the following Python code is executed?

```
lst1 = [1, [2, 3]]
lst2 = [1, [2, 3]]
lst3 = lst1 * 2
lst4 = lst1 + lst2
lst3[1][0] = 20
lst3[3][1] = 30
lst4[1][0] = 200
lst4[3][1] = 300

print(lst1)
print(lst2)
print(lst3)
print(lst4)
```

Output:

lst1 = [1, [200, 30]]

lst2 = [1, [2, 300]]

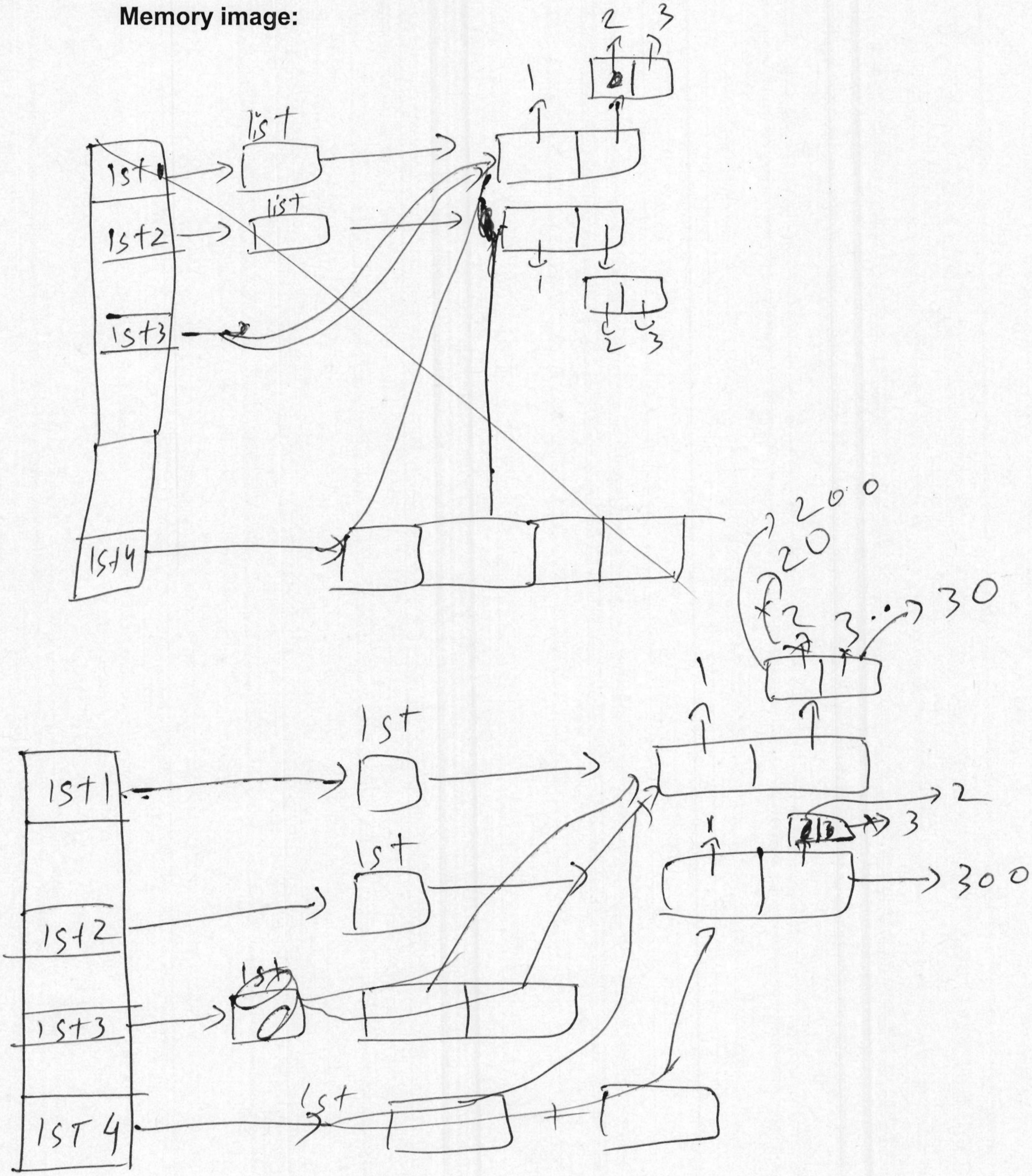
lst3 = [1, [200, 30], 1, [200, 30]]

lst4 = [1, [200, 30], 1, [2, 300]]

Name: Sahil Net ID: sks9379

b) Draw the memory image as it evolves when executing of the code above.

Memory image:



Name: Solu Net ID: SKS9779

Question 2 (20 points):
Below is the code we wrote in class for the implementation of `ArrayList`, simulating the built-in `list` type:

```
import ctypes  # provides low-level arrays
def make_array(n):
    return (n * ctypes.py_object)()



class ArrayList:
    def __init__(self):
        self.data_arr = make_array(1)
        self.capacity = 1
        self.n = 0

    def __len__(self):
        return self.n

    def append(self, val):
        if (self.n == self.capacity):
            self.resize(2 * self.capacity)
        self.data_arr[self.n] = val
        self.n += 1

    def resize(self, new_size):
        new_array = make_array(new_size)
        for i in range(self.n):
            new_array[i] = self.data_arr[i]
        self.data_arr = new_array
        self.capacity = new_size

    def __getitem__(self, ind):
        if (not (0 <= ind <= self.n - 1)):
            raise IndexError('invalid index')
        return self.data_arr[ind]

    def __setitem__(self, ind, val):
        if (not (0 <= ind <= self.n - 1)):
            raise IndexError('invalid index')
        self.data_arr[ind] = val

    def __iter__(self):
        for i in range(len(self)):
            yield self.data_arr[i]

    def extend(self, iter_collection):
        for elem in iter_collection:
            self.append(elem)
```

Name: Sahi Net ID: SLS9393

Add the method **def** `insert(self, index, val)` to the `ArrayList` class above. It should allow to insert `val` to position `index` in `self` (shifting the elements from that position one place forward). Your implementation should simulate the behavior of the `insert` method of the (build-in) `list` class.

Note: for simplicity, you should support only positive indices.

```
def insert(self, index, val):
    if (not (0 <= index <= self.n)):
        raise IndexError('invalid index')
```

~~else~~ ~~0 <= index <= self.n~~:
~~self.data_arr.insert(index, val)~~
~~else~~:

else:
self.data_arr.insert(index, val)

~~if index < self.n~~:
~~self.data_arr.insert(index, val)~~

if index == int is True:
if index < self.n:
self.data_arr.insert(index, val)

Name: Sahil Net ID: SKS9379

Question 3 (20 points)
Implement the function:

```
def remove_elemements_at_even_indices(lst)
```

This function gets a list, `lst`. When called, it should **mutate** `lst`, by removing all the elements that are stored at even indices. That is, `lst[0]`, `lst[2]`, `lst[4]`, ... should be removed from `lst`.

Your implementation should allow the following interaction:

```
>>> lst = [10, 20, 30, 40, 50, 60],
>>> remove_elems_at_even_indices(lst)
>>> lst
[20, 40, 60]
```

Implementation requirements:

i. Your function should run in **worst case linear time**. That is, if `lst` is a list with n elements, the call `remove_elemements_at_even_indices(lst)` should run in worst case $\theta(n)$.

ii. Your implementation should **mutate the given list object in-place**, without using any additional data collection.

Note: Write your implementation on the next page.

Name: Sahil Net ID: 9979

```
def remove_elememements_at_even_indices(lst):
```

```
    i=0
    y=len(lst)
    while i<=y:
        if i%2==0:
            lst[i],lst[y]=lst[y],lst[i]
            i+=1
        if i==y:y-=1
```

```
    i=0
    y=len(lst)
    cs=0
    while i<=y:
        if i==y:
            return lst
        if i%2!=1:
            lst[i],lst[y]=lst[y],lst[i]
            i+=1
            y-=1
            cs+=1
    del lst[cs:]
    return lst
```

Answer is on the last page

Name: Gks9379 Net ID: gks9379

Question 4 (20 points)

a) You are given the following function:

```
def func1(lst):
    res = []
    for elem in lst:
        res = [elem] + res
    return res
```

If `lst` is a list of n integers, what is the **worst-case** running time of `func1(lst)`? Give a short explanation of your answer.

$T_1(n) = \theta($ n $)$

len(lst) = n

So, loop is running n times and res is a list in which list(elem) is getting added which amortised its run time is $O(1)$ in this case

So, total complexity is $O(n)$

Name: Sabil Net ID: SKS9379

b) You are given the following function:

```
def func2(lst):
    n = len(lst)
    res = []
    i = 1
    while (i <= n):
        new_item = lst.copy()
        res.append(new_item)
        i *= 2
    return res
```

If `lst` is a list of n integers, what is the worst-case running time of `func2(lst)`? Give a short explanation of your answer.

$T_2(n) = \theta($ n log n $)$

loop is running in 1, 2, 4, 8, 16 ... log n

So, its running is log n

x form. and .append() is amortised

so runtime is O(n). as it is copying the whole list.

So, total run runtime is O(n log n)

Name: Sahil Net ID: SK9379

c) You are given the following function:

```
def func3(lst):
    n = len(lst)
    res = []
    i = 1
    while(i <= n):
        new_item = lst[i-1]
        res.insert(0, new_item)
        i *= 2
    return res
```

If `lst` is a list of n integers, what is the **worst-case** running time of `func3(lst)`? Give a short explanation of your answer.

$T_3(n) = \theta($ nlogn $)$

function is running in the for, 1, 2, 4, 8 which means it's running, ~~on~~ log n times, and res.insert(0, new term) has amortized complexity of O(n) in this case so, total run time is O(nlogn)

Name: Sahil Net ID: SKS99379

Question 5 (20 points)

Give a **recursive** implementation to the following function:

```
def triangle_list(n)
```

This function is given a positive integer n. When calling, `triangle_list(n)`, it should return a list with n elements, where the first element is: `'*'`, the second element is: `'**'`, the third element is: `'***'`, etc.

For example, the call `triangle_list(4)` should return the list:
`['*', '**', '***', '****']`

Implementation requirements:

i. Your function **must be recursive**.
ii. You are **not allowed** to:
   a) Define a helper function.
   b) Add parameters to the function's header line
   c) Set default values to the parameter.
   d) Use global variables

```
def triangle_list(n):
    if lst=[]
        i=0
    if i<n:
        lst.append('*'*i)
        triangle_lst(n-1)
    return lst
    i+=1
```

Name: Sahil Net ID: SKSG7779

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.

```
def remove_elements_at_even_indices(lst):
    i=0
    j=len(lst)
    cs=0
    if i==j:
        return lst
    while i<j:
        if i%2!=1:
            lst[i], lst[i+1] = lst[i+1], lst[i]
            i+1
            cs+1
    del lst(cs:]
    return lst
```