- This lab will cover dynamic arrays (list) and run-time analysis of list methods, recursion, and sorting.
- It is assumed that you have reviewed chapter 5 and chapter 12 of the textbook. You may want to refer to the text and your lecture notes during the lab as you solve the problems.
- When approaching the problems, think before you code. Doing so is good practice and can help you lay out possible solutions.
- Think of any possible test cases that can potentially cause your solution to fail!
- You must stay for the duration of the lab. If you finish early, you may help other students. If you don't finish by the end of the lab, we recommend you complete it on your own time. Ideally, you should not spend more time than suggested for each problem.
- Your TAs are available to answer questions in the lab, during office hours, and on Piazza.

Vitamins (30 minutes)

---

1. Give the **worst case** (not amortized) asymptotic run-time for each of the following list methods. Write your answer in terms of n, the length of the list. Provide an appropriate summation for multiple calls. (25 minutes)

   Given: `lst = [ 1, 2, 3, 4, … ,n]` and `len(lst)` is `n`.

   What will be the run-time when calling the following for lst?

Method	Single (`1`) Call	Multiple (`n`) Calls `for i in range(n): ...`
`append()`		What will be the total cost if lst = [ ] instead? Will the overall run-time change?
`insert(0, val)`		What will be the total cost if lst = [ ] instead? Will the overall run-time change?

`pop()`		
`pop(0)`		

Optional: Conclude the **amortized (worst case) cost** of a single append( ).

2. Given the following mystery functions: (5 minutes)

- **i.** Replace `mystery` with an appropriate name
- **ii.** Determine the function's worst-case runtime and extra space usage with respect to the input size.

a.
```
def mystery(n):
    lst = []
    for i in range(n):
        lst.insert(i, i)
```

b.
```
def mystery(n):
    for i in range(1, n+1):
        total = sum([num for num in range(i)])
        print(total)
```

c.
```
def mystery(lst):
    lst2 = lst.copy()
    lst2.reverse()
```

```
if (lst == lst2):
    return True
return False
```

3. For each function, identify the sorting algorithm being used and analyze its run-time and extra space usage.

Hint: You may want to draw your list after each iteration to see what's happening. Use a small list input such as [1, 3, 6, 5, 2, 4] to test each function.

Choices: Bubble Sort, Insertion Sort, Selection Sort

a.
```
def sort1(lst):
    for i in range(1, len(lst)):
        curr = lst[i]
        j = i - 1
        while j >= 0 and curr < lst[j]:
            lst[j+1] = lst[j]
            j -= 1
        lst[j+1] = curr
```

b.
```
def sort2(lst):
    for i in range(len(lst)):
        min_ind = i
        for j in range(i+1, len(lst)):
            if lst[min_ind] > lst[j]:
                min_ind = j
        lst[i], lst[min_ind] = lst[min_ind], lst[i]
```

c.
```
def sort3(lst):
    for i in range(len(lst)):
        for j in range(len(lst) - 1, i, -1):
            if lst[j] < lst[j-1]:
                lst[j], lst[j-1] = lst[j-1], lst[j]
```

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code.

Download the **ArrayList.py** file found under Resources/Lectures on NYU Classes

1. Extend the ArrayList class implemented during lecture with the following methods:

a. Implement the `__repr__` method for the ArrayList class, which will allow us to display our ArrayList object like the Python list when calling the print function. The output is a sequence of elements enclosed in [ ] with each element separated by a space and a comma. (10 minutes)

ex) `arr` is an `ArrayList` with `[1, 2, 3]`
→ `print(arr)` outputs `[1, 2, 3]`

Note: Your implementation should create the string in $\Theta(n)$, where n = len(arr).

b. Implement the `__add__` method for the ArrayList class, so that the expression arr1 + arr2 is evaluated to a **new** ArrayList object representing the concatenation of these two lists. (10 minutes)

ex) `arr1` is an `ArrayList` with `[1, 2, 3]`
`arr2` is an `ArrayList` with `[4, 5, 6]`
→ `arr3 = arr1 + arr2`
`arr3` is a new `ArrayList` with `[1, 2, 3, 4, 5, 6]`.

Note: If n_1 is the size of arr1, and n_2 is the size of arr2, then __add__ should run in $\Theta(n_1 + n_2)$

c. Implement the `__iadd__` method for the ArrayList class, so that the expression arr1 += arr2 **mutates** arr1 to contain the concatenation of these two lists. You may remember that this operation produces the same result as the **extend method**.

Your implementation should return *self*, which is the object being mutated. (10 minutes)

ex) `arr1` is an `ArrayList` with `[1, 2, 3]`
`arr2` is an `ArrayList` with `[4, 5, 6]`
→ `arr1 += arr2`
`arr1` is mutated and now has `[1, 2, 3, 4, 5, 6]`.

Note: If n_1 is the size of arr1, and n_2 is the size of arr2, then __iadd__ should run in $\Theta(n_1 + n_2)$. It's not n_2 because we have to take array resizing into account.

d. Modify the `__getitem__` and `__setitem__` methods implemented in class to also support **negative** indices. The position a negative index refers to is the same as in the Python list class. That is -1 is the index of the last element, -2 is the index of the second last, and so on. (20 minutes)

ex) `arr` is an `ArrayList` with `[1, 2, 3]`
→ `print(arr[-1])` outputs `3`
→ `arr[-1] = 5`
`print(arr[-1])` outputs `5` now

Note: Your method should also raise an IndexError in case the index (positive or negative) is out of range.

e. Implement the `__mul__` method for the ArrayList class, so that the expression arr1 * k (where k is a positive integer) creates a **new** ArrayList object, which contains k copies of the elements in arr1. (15 minutes)

ex) `arr1` is an `ArrayList` with `[1, 2, 3]`
→ `arr2 = arr1 * 2`
`arr2` is a new `ArrayList` with `[1, 2, 3, 1, 2, 3]`.

Note: If *n* is the size of arr1 and k is the int, then __mul__ should run in $\Theta(k * n)$.

f. Implement the `__rmul__` method to also allow the expression n * arr1. The behavior of n * arr1 should be equivalent to the behaviour of arr1 * n. (5 minutes)

(You've done this before for the Vector problem in homework 1)

g. Modify the constructor `__init__` to include an option to pass in an iterable collection such as a string and return an ArrayList object containing each element of the collection. Do not account for dictionaries.(10 minutes)

ex) `arr = ArrayList("Python")`
→ `print(arr)` outputs `['P','y','t',h','o','n']`

→ `arr2 = ArrayList(range(10))`
→ `print(arr2)` outputs `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

h. Implement a remove( ) method that will remove the **first** instance of val in the ArrayList. It must be done in $\Theta(n)$ run-time. **You do not need to call resize.**(20 minutes)

ex) `arr` is an `ArrayList` with `[1, 2, 3, 2, 3, 4, 2, 2]`

→ `arr.remove(2)`
→ `print(arr2)` outputs `[1, 3, 2, 3, 4, 2, 2]`

i. Implement a removeAll( ) method that will remove all instances of val in the ArrayList. The implementation should be in-place and maintain the relative order of the other values. It must also be done in $\Theta(n)$ run-time. **You do not need to call resize.**

ex) `arr` is an `ArrayList` with `[1, 2, 3, 2, 3, 4, 2, 2]`

→ `arr.removeAll(2)`
→ `print(arr2)` outputs `[1, 3, 3, 4]`

2. Write a **recursive** function to find the minimum element in a non-empty, **non-sorted** list of numbers. (10 minutes)

ex) if the input list is [13, 9, 16, 3, 4, 2], the function should return 2.

```
def find_min(lst, low, high):
    """
    : lst type: list
    : low, high type: int
    : return type: int
    """
```

3. Write a **recursive** function to find the second minimum element in a **non-sorted** list containing minimum 2 integers. You may also define a *helper function* with additional parameters to help you. If you find this difficult, feel free to solve this **iteratively**. (15 minutes)

ex) if the input list is [13, 9, 16, 3, 4, 2], the function should return 3.

```
def find_second_min(lst):
    """
    : lst type: list
    : low, high type: int #for helper function
    : return type: int
    """
```

Amortized vs Average:

We stated that the average-case run-time of quick select is Θ(n). The average case is different from the amortized case. With amortization, we take the entire run-time cost into account but with the average case, we make an assumption on probability. That is, we assume that the worst-case list input does not happen often.

Analysis of Run-time (worst, average) and Extra Space

Sorting Algorithm	**Worst Case Run-Time**	**Average Case Run-Time**	**Extra Space**
Bubble Sort	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlog(n))	O(nlog(n))	O(n) temp list
Quick Sort	O(n^2)	O(nlog(n))	O(log(n)) number of pivots from recursive calls on the call stack
Selection Quick Select	O(n^2)	O(n) only recurse on one side	O(log(n)) recursive O(1) non-recursive because you can reuse the same pivot variable in an iteration. More complicated to write

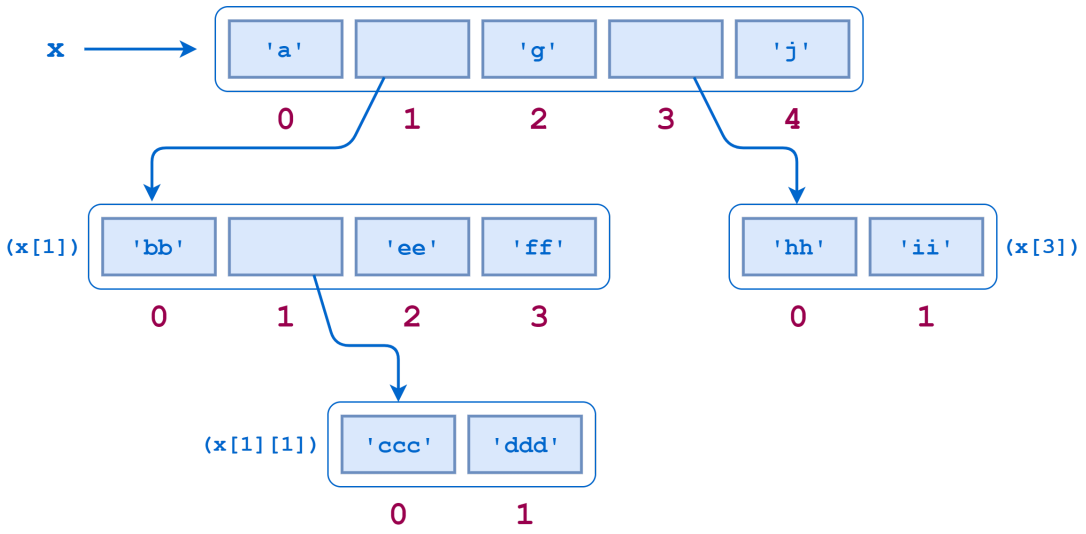
Nested Lists:

A nested list of integers is a list that stores integers in some hierarchy. The list can contain integers and other nested lists of integers. An example of a nested list of integers is [[1, 2], 3, [4, [5, 6, [7], 8]], [[[[9]]]]].

To check the type of an object, use the isinstance function.

ex) lst = [1, 2, 3, 4]
if isinstance(lst, list): #returns True
if isinstance(lst, int): #returns False

Here is a visual representation of a nested list, x:



4. Write a **recursive** function that reverses the order of values of each list in the hierarchy.

(30 minutes)

ex) If lst = [[1, 2], 3, [4, [5, 6, [7], 8]], [[[[9]]]]], deep_reverse(lst) should modify it so that it now has [[[[[9]]]] , [[8, [7], 6, 5], 4], 3, [2, 1]],

---

```
def deep_reverse(lst):
      """
      : lst type: list
      : output type: None
      """
```

The next question is OPTIONAL but we recommend you try it on your own time for better practice with nested lists.

Optional:

5. Write a recursive generator function that takes in a nested list, and yields each integer of the list, from left to right. Note that you do not need to flatten the list. **(20 minutes)**

```
def yield_flattened(lst):
      """
      : lst type: list
      : yield type: int
      """

def print_flattened(lst):
      print("[" + ",".join(str(num) for num in yield_flattened(lst))+
"]")
```

If lst = [[1, 2], 3, [4, [5, 6, [7], 8]], [[[[9]]]]]

print_flattened(lst) should output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

Note:

yield value will only yield values from **one** function call. To yield values from another **recursive** call, you should have for elem in recursive_function(...): yield elem, or use yield from recursive_function(...).

ex) You can shorten for i in range(n): yield i → yield from range(n)

```
def sample(n):                          def sample(n):
    for  i in range(n):        →            yield from range(n)
        yield i
```