Vitamins (35 minutes)

1. What is the output of the following code?

```
s = ArrayStack()
i = 2

s.push(1)
s.push(2)
s.push(4)
s.push(8)
#stack looks like [1,2,4,8]
i += s.top()
s.push(i)
[1,2,4,8,10]
s.pop()
s.pop()
#stack looks like [1,2,4]
print(i)#i = 10
print(s.top()) #s.top() = 4
```

2. Trace the following function with different list inputs. Describe what the function does, and give a meaningful name to the function:

```
def nothing(lst):

    s = ArrayStack()
    for i in range(len(lst)):
      s.push(lst.pop())

    for i in range(len(s)):
      lst.append(s.pop())
```

3. Trace the following function, which takes in a stack of integers. Describe what the function does, and give a meaningful name to the function:

```
def stack_min(s):
        if len(s) == 1:
           return s.top()
        else:
           val = s.pop()
           result = stack_min(s)

           if val < result:
                result = val
           s.push(val)
           return result
```

Coding (See CS1134 Lab8 Solutions.py)

In this section, it is strongly recommended that you solve the problem on paper before writing code. Note that you should not access the underlying list in the ArrayStack implementation. **Treat it as a black box and only use the len, is_empty, push, top, and pop methods.**

Download the **ArrayQueue.py**, **ArrayStack.py** & **ArrayList.py** files under Resources/Labs on NYU Classes

Note: import the class like so → `from ArrayStack import *`

1. Write a **recursive function** that takes in a Stack of integers and returns the sum of all values in the stack. Do not use any helper functions or change the function signature. Note that the stack should be restored to its original state if you pop from the stack.

   ex) s contains [1, -14, 5, 6, -7, 9, 10, -5, -8] from top → bottom.
   stack_sum(s) returns -3

```
def stack_sum(s):
    """
```

```
    : s type: ArrayStack
    : return type: int
    """
```

Hint: See how the stack is restored in the code snippet from vitamins question 3.

2. Create an **iterative function** that evaluates a valid prefix string expression. You may only use **one ArrayStack** as an additional data structure to the given setup. Do not use any helper functions or change the function signature.

   In addition, each character is separated by one white space and numbers may have more than one digit. Therefore, we will use the split function to create a new list of each substring of the string separated by a white space. You may assume all numbers will be positive.

   ex) exp_str is " - + * 16 5 * 8 4 20"
   eval_prefix(exp_str) returns = 92

```
def eval_prefix(exp_str):
    """
    : exp type: str
    : return type: int
    """
    exp_lst = exp_str.split( )
```

   Hint:
   To check if a string contains digits, use `.isdigit( )`.
   To check if a string is an operator, you may want to do `if char in "-+/*"` similarly to how you checked for vowels.

   As you parse the expression lst, think about when you would push/pop the operator or number to/from the stack. Try to trace this execution on paper first before writing your code.

   Test your code with the various prefix expressions from the Vitamins q4.

3. Write an **iterative functio**n that flattens a nested list while retaining the left to right ordering of its values using one **ArrayStack** and its defined methods. That is, you should not directly access the underlying array in the implementation. Do not use any helper functions or change the function signature.
(30 minutes)

   In addition, do not create any other data structure other than the ArrayStack.

   ex) lst = [[[[0]]], [1, 2], 3, [4, [5, 6, [7]], 8], 9]
   flatten_list(lst)
   print(lst) → lst = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
def flatten_list(lst):
    """
    : lst type: list
    : return type: None
    """
    s = ArrayStack()
```

   Hint: You may want to traverse the list from the end for 2 reasons: pop and append has an amortized cost of O(1) and a stack reverses the collection order because of FIFO.

4. Write an **iterative function** that will sort a stack of unsorted integers. You are **only allowed to use another stack** and no other additional data structure for your solution. Do not use any helper functions or change the function signature.

   A sorted stack is determined if its values are in ascending order from top to bottom. Smallest on top, largest on bottom.

   What is the worst case run-time of your solution?

```
def stack_sort(s):
    """
    : input_str type: ArrayStack
    : return type: None
    """

    #use this to help with sorting

    helper_stack = ArrayStack( )
```