Lab 4: Abstract Data Type Solution

4) Additional Tasks

Graphing your results

1. Implement *time_queue_lists*, a modified version of your timing experiment function that returns a tuple containing three lists:

   - A list of queue sizes it tried
   - A list of the corresponding times to run enqueue for each queue size
   - A list of the corresponding times to run dequeue for each queue size

   Note that each of your lists should have the same length.

```
        ...
        def time_queue_lists() -> Tuple[List[int], List[float], List[float]]:
            """Run timing experiments for Queue.enqueue and Queue.dequeue.

            Return lists storing the results of the experiments.  See the lab
            handout for further details.
            """
            queue_sizes = [10000, 20000, 40000, 80000, 160000]
            enqueue_time_list = []
            dequeue_time_list = []

            trials = 200

            for queue_size in queue_sizes:
                queues = _setup_queues(queue_size, trials)

                time = 0
                for queue in queues:
                    time += timeit('queue.enqueue(1)', number=1, globals=locals())

                print(f'enqueue: Queue size {queue_size:>7}, time {time}')
```

```
                enqueue_time_list.append(time)

            for queue_size in queue_sizes:
                queues = _setup_queues(queue_size, trials)

                time = 0
                for queue in queues:
                    time += timeit('queue.dequeue()', number=1,
globals=locals())

                print(f'dequeue: Queue size {queue_size:>7}, time {
time}')
                dequeue_time_list.append(time)

            return (queue_sizes, enqueue_time_list, dequeue_time_list
)

        ...

```

Listing 1: task_4_q1_part_1_solution.py

2. To actually plot the data, we'll use the Python library *matplotlib*, which is an extremely powerful and popular library for plotting all sorts of data.

   If you're on a Teaching Lab machine, you already have this library installed.

   If you're on your own machine, you should have already installed this library by following the CSC148 Software Guide. (Look for the section on installing Python libraries.)

   Add the statement *import matplotlib.pyplot as plt* to the top of *timequeue.py*, and make sure you can still run your file without error.

```
    ...
    import matplotlib.pyplot as plt
    ...

```

Listing 2: task_4_q1_part_2_solution.py

Note:

- If *matplotlib* is missing, install by typing *pip3 install matplotlib* in terminal or windows command line.

3. To get a basic 2-D plot of your timing data, work your way through the first part of this guide (Links to an external site.). (Ignore all of the references to "numpy", which is

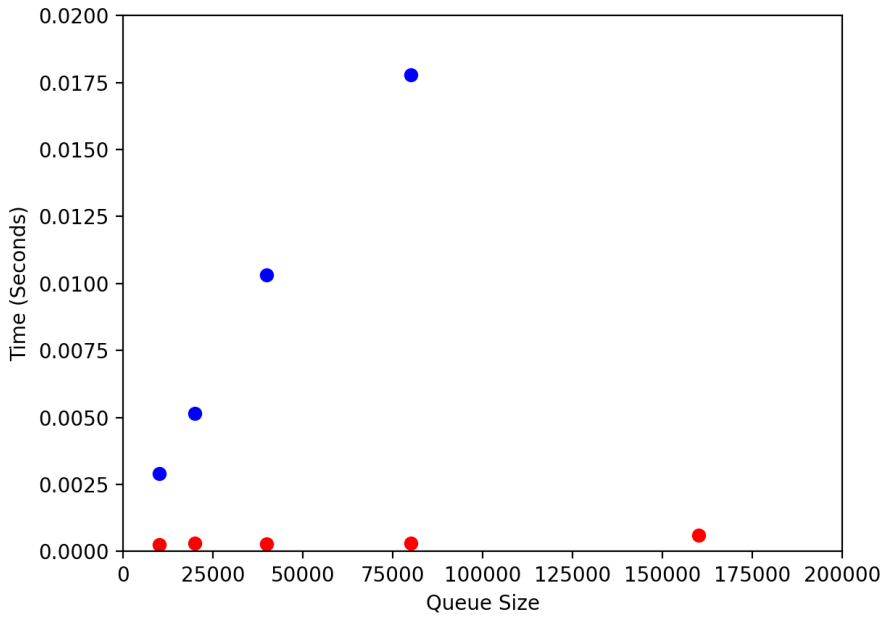
another Python library we aren't using in this course. Also ignore the other sections after the first one; the whole tutorial is pretty long!)

You can use an x-axis range of 0-200000 and a y-axis range of 0-0.02 (feel free to adjust the y-axis depending on how long the experiments take to run on your computer).

```
    if __name__ == '__main__':
        ...
        plt.plot(queue_sizes, enqueue_time_list, 'ro')
        plt.plot(queue_sizes, dequeue_time_list, 'bo')
        plt.xlim([0,200000])
        plt.ylim([0,0.02])
        plt.xlabel('Queue Size')
        plt.ylabel('Time (Seconds)')
        plt.show()

```

Listing 3: task_4_q1_part_3_solution.py



4. If you still have time, explore! There's lots of customization you can do with *matplotlib* to make your graphs really pretty.

Undo and redo

```
    from mystack import Stack

    current_string = ''
```

```
    undo_stack = Stack()
    redo_stack = Stack()

    print('Please type one of the following commands:')
    print('REDO - redo undoed action')
    print('UNDO - show previously registered string')
    print('ADD - Add new string')
    print('EXIT - exit program')

    # 1. Prompt string
    while True:
        print('Current string: {}'.format(current_string))
        print(undo_stack._items)
        response = input('>>> ')

        # 2. If user types 'EXIT', then exit program
        if response == 'EXIT':
            break
        # 3. If user types 'UNDO',
        elif response == 'UNDO':
            #   3.1 If stack for undo is empty, return nothing
            if undo_stack.is_empty():
                continue

            #   3.2 If stack for undo not empty, then pop an item,
push a copy to
            #   redo and display value to user
            redo_stack.push(current_string)
            undo_popped_val = undo_stack.pop()
            current_string = undo_popped_val

        # 4. If user types 'REDO'
        elif response == 'REDO':
            #   4.1 If stack for redo is empty, return nothing
            if redo_stack.is_empty():
                continue

            #   4.2 If stack for redo not empty, then pop an item,
push a copy to
            #   undo and display value to user
            undo_stack.push(current_string)
            redo_popped_val = redo_stack.pop()
            current_string = redo_popped_val

        else:
            undo_stack.push(current_string)
            current_string = response
            redo_stack = Stack()

```

Listing 4: task_4_q1_part_3_solution.py

When a user types new string after typing 'REDO', then the current list of stack items for redo should become clear.