Queens College of CUNY
Department of Computer Science
Programming Languages
(CSCI 316)
Winter 2026

Assignment #10
"Concurrency"

Due: January 18, 2026

Introduction:

In this assignment, we consider "concurrency", the ability to run "simultaneous" execution of multiple tasks / threads.

You are given code for a mult-threaded Merge Sort, and asked to make some modifications.

After Merge Sort works, repeat the exercise for Quick Sort and 3-Way Merge Sort

Question: does the muli-threading make it slower or faster? Is there a "depth" that is optimal?

Submissions:

In the Google form, please submit:

- Assignmen10.py (source code in Python or another language of your choice)
- Assignment10.txt (console output from your program execution)

Tasks:

[1] Create a new Python program Assignment10.py (or one in another language of your

[2] Start with this code below:

```
from concurrent.futures import ThreadPoolExecutor

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result

def mergesort(arr, max_depth=8):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = arr[:mid]
    right = arr[mid:]
    if max_depth > 0:
        with ThreadPoolExecutor(max_workers=2) as executor:
            left_future = executor.submit(mergesort, left, max_depth-1
            right_future = executor.submit(mergesort, right, max_depth-1)
            left_sorted = left_future.result()
            right_sorted = right_future.result()
    else:
        left_sorted = mergesort(left, 0)
        right_sorted = mergesort(right, 0)
    return merge(left_sorted, right_sorted)

def main():
    data = [38, 27, 43, 3, 9, 82, 10, 5, 1, 99, 17]
    sorted_data = mergesort(data)
    print(sorted_data) # comment after initial testing

if __name__ == "__main__":
    main()
```

[3] Modify the code to use a random list of size n instead of the hard-coded list

[4] Add and call this code to show the threads

```
import threading

def show_threads():
    threads = threading.enumerate()
    print(f"Active threads: {len(threads)}")
    for t in threads:
        print(f"Name={t.name}, Ident={t.ident}, Alive={t.is_alive()}, Daemon={t.daemon}")
```

[5] Modify the code to time it - get the time() before and after and subtract. (When timing, disable the prints which will skew the timing results.)

[6] Compare the performance for difference "depths", including 0 (= No threading)

[7] Repeat this entire exercise for Quick Sort