Task 1:

```
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     5                                           def res_skimage(imgs):
     6         1         77.0     77.0      0.0      new_size = (imgs[1].shape[0] // 2, imgs[1].shape[1] // 2)
     7
     8         1          5.0      5.0      0.0      def process_image(im):
     9                                                   return resize(im, new_size, anti_aliasing=True)
    10
    11         1        385.0    385.0      0.1      with ThreadPoolExecutor() as executor:
    12         1     339430.0 339430.0     99.1          res_im = list(executor.map(process_image, imgs))
    13
    14         1       2452.0   2452.0      0.7      return np.asarray(res_im)
```

This is the line that was the bottleneck. The highlighted line has been updated so that the bottleneck is now minimized. In the optimized version, the bottleneck is minimized by using multithreading with ThreadPoolExecutor to resize multiple images concurrently, reducing execution time.

Task 2:

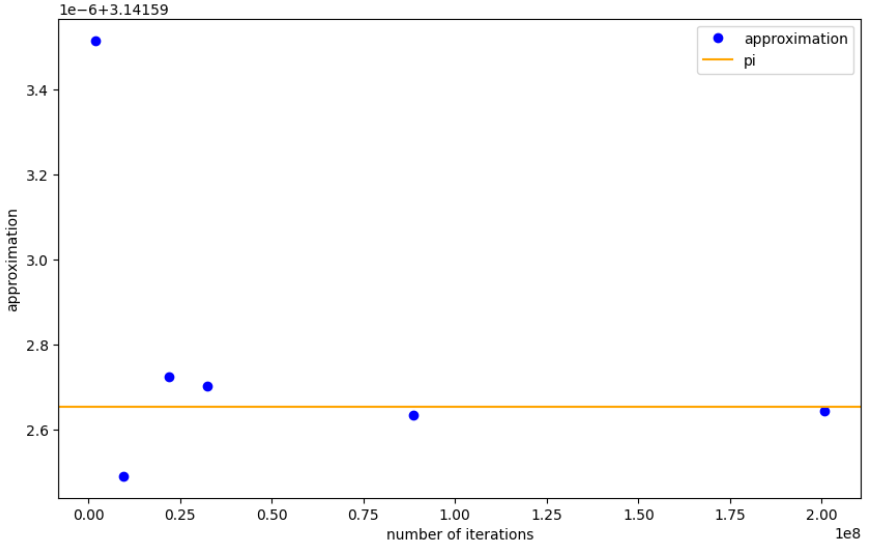
I used multiprocessing because this task is CPU-bound, and multiprocessing is ideal for tasks that require heavy computation (multithreading wouldn't be efficient in this case due to the GIL in Python). The sequential call of the function for each value of N took a total of 68.45 seconds. By using multiprocessing, I was able to reduce this time to 48.15 seconds. That is a speedup of 1.42x.

Task 3:

```python
@jit(nopython=True)
def approximate_pi_optimized(n):
    pi_2 = 1
    nom, den = 2.0, 1.0
    for i in range(n):
        pi_2 *= nom / den
        if i % 2:
            nom += 2
        else:
            den += 2
    return 2 * pi_2
```

By using Numba optimization, I am over 98.26% faster.

Task 4:



I-En Lee, 23365648, di65haji