Name: Rutvi shah
Matriculation:23159043
Idm :to20raje

Task1:

```
Total time: 0.0622488 s
File: C:\Users\rutvi\PycharmProjects\pythonProject16\main.py
Function: res_skimage at line 9

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     9                                           def res_skimage(imgs):
    10         1         35.0     35.0      0.0      new_size = (imgs[1].shape[0] // 2, imgs[1].shape[1] // 2)

    11         1          3.0      3.0      0.0      res_im = []
    12
    13       281       1127.0      5.6      0.2      for im in imgs:
    14       280     618549.0   3092.7     99.4          image_resized = resize(im, new_size, anti_aliasing=True)
    15       280       1185.0      5.9      0.2          res_im.append(image_resized)
    16
    17         1       1589.0   1589.0      0.3      return np.asarray(res_im)
```

The bottleneck was in this line. The highlighted line has been updated, minimizing the bottleneck

Task 2:

The multi-processing is faster than multi-threading it is CPU bound task which is running in parallel for calculating the approximating value of pi. Here multiprocessing has various CPU cores when compared to the multi-threading which is a single core. The reason for choosing multiprocessing over multithreading in this case is that Python has the Global Interpreter Lock (GIL), which prevents multiple native threads from executing Python byte codes at once. This means that multithreading is not as effective for CPU-bound tasks in Python. Multiprocessing, on the other hand, creates separate processes with their own interpreter and memory space, avoiding the GIL limitation. Huge time reduction by 61.76%

```
Sequential results: [3.1415935153733052, 3.1415927247955033, 3.1415927021077117, 3.141592635888531, 3.1415924919219385, 3.141592645761164]
Parallel results: [3.1415935153733052, 3.1415927247955033, 3.1415927021077117, 3.141592635888531, 3.1415924919219385, 3.141592645761164]

Sequential execution time: 34.034602642059326 seconds
Parallel execution time: 21.166608572006226 seconds

Speedup with multiprocessing: 1.61x
```

Task3:

```
1 usage
@jit(nopython=True)
def approximate_pi_numba(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 jit I am over 8676% faster

Task: 4