```
# section_a_vector_dot.py
import time, random, cProfile, pstats, io, tracemalloc, sys

def gen_data(n, seed=42):
  random.seed(seed)
  x = [random.random() for _ in range(n)]
  y = [random.random() for _ in range(n)]
  return x, y

def vector_add(x, y):
  out = []
  for a, b in zip(x, y):
    out.append(a + b)
  return out

def dot_product(x, y):
  s = 0.0
  for a, b in zip(x, y):
    s += a * b
  return s

def main():
  # Ensure tracemalloc is stopped from any previous runs before starting new profiling
  if tracemalloc.is_tracing():
      tracemalloc.stop()

  # Explicitly ensure sys.setprofile is None before attempting to enable cProfile
  # This is the most robust way to clear any lingering profiler hooks.
  sys.setprofile(None)

  # Ensure any previous cProfile or other sys.setprofile hooks are disabled
  # by creating a temporary profiler and immediately disabling it.
  # This makes the environment clean before enabling the current profiler.
  temp_pr = cProfile.Profile()
  temp_pr.disable() # This sets sys.setprofile(None)

  N = 2_000_000 # adjust based on your machine
  x, y = gen_data(N)

  # --- CPU Profiling with cProfile ---
  pr = cProfile.Profile()
  pr.enable()

  t0 = time.perf_counter()
  s = dot_product(x, y)
  t1 = time.perf_counter()

  t2 = time.perf_counter()
  z = vector_add(x, y)
  t3 = time.perf_counter()

  pr.disable()

  # --- Memory Profiling with tracemalloc ---
  # Since cProfile is now disabled, tracemalloc can run.
  tracemalloc.start()
  # Rerun operations to capture memory usage
  _ = dot_product(x, y) # Assign to _ as result is not needed, just memory
  _ = vector_add(x, y)
  current, peak = tracemalloc.get_traced_memory()
  tracemalloc.stop()

  # Print results
  print(f"Vector length N={N}")
  print(f"Dot product: {s:.6f} | Time: {t1 - t0:.3f} s")
  print(f"Vector add: len(z)={len(z)} | Time: {t3 - t2:.3f} s")
  print(f"Current/Peak memory: {current/1e6:.2f} MB / {peak/1e6:.2f} MB")

  # Profiler output
  s_buf = io.StringIO()
  ps = pstats.Stats(pr, stream=s_buf).sort_stats('cumtime')
  ps.print_stats(15)
  print("\n--- cProfile (Top 15 by cumulative time) ---")
  print(s_buf.getvalue())
```

```
# Call main() to execute the profiling when the cell is run
  main()
```

```
main()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipython-input-451043146.py in <cell line: 0>()
----> 1 main()

/tmp/ipython-input-3558746691.py in main()
     40     # --- CPU Profiling with cProfile ---
     41     pr = cProfile.Profile()
---> 42     pr.enable()
     43 
     44     t0 = time.perf_counter()

ValueError: Another profiling tool is already active
```

Next steps: Explain error

```
# section_a_vector_dot.py
import time, random, cProfile, pstats, io, tracemalloc

def gen_data(n, seed=42):
    random.seed(seed)
    x = [random.random() for _ in range(n)]
    y = [random.random() for _ in range(n)]
    return x, y

def vector_add(x, y):
    out = []
    for a, b in zip(x, y):
        out.append(a + b)
    return out

def dot_product(x, y):
    s = 0.0
    for a, b in zip(x, y):
        s += a * b
    return s

def main():
    N = 2_000_000  # adjust based on your machine
    x, y = gen_data(N)

    tracemalloc.start()
    pr = cProfile.Profile()
    pr.enable()

    t0 = time.perf_counter()
    s = dot_product(x, y)
    t1 = time.perf_counter()

    t2 = time.perf_counter()
    z = vector_add(x, y)
    t3 = time.perf_counter()

    pr.disable()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    # Print results
    print(f"Vector length N={N}")
    print(f"Dot product: {s:.6f} | Time: {t1 - t0:.3f} s")
    print(f"Vector add: len(z)={len(z)} | Time: {t3 - t2:.3f} s")
    print(f"Current/Peak memory: {current/1e6:.2f} MB / {peak/1e6:.2f} MB")

    # Profiler output
    s_buf = io.StringIO()
    ps = pstats.Stats(pr, stream=s_buf).sort_stats('cumtime')
    ps.print_stats(15)
    print("\n--- cProfile (Top 15 by cumulative time) ---")
    print(s_buf.getvalue())

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

```
Vector length N=2000000
Dot product: 499712.974120 | Time: 0.602 s
Vector add: len(z)=2000000 | Time: 4.734 s
Current/Peak memory: 65.13 MB / 65.13 MB

--- cProfile (Top 15 by cumulative time) ---
         2000019 function calls (2000018 primitive calls) in 5.336 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        5    2.977    0.595    4.455    0.891 {built-in method time.sleep}
  2000000    1.571    0.000    1.571    0.000 {method 'append' of 'list' objects}
      2/1    0.602    0.301    0.571    0.571 /tmp/ipython-input-3909820823.py:16(dot_product)
        1    0.186    0.186    0.280    0.280 /tmp/ipython-input-3909820823.py:10(vector_add)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        6    0.000    0.000    0.000    0.000 {built-in method posix.getppid}
        4    0.000    0.000    0.000    0.000 {built-in method time.perf_counter}
```

```
# section_a_vector_dot.py
import time, random, cProfile, pstats, io, tracemalloc

def gen_data(n, seed=42):
    random.seed(seed)
    x = [random.random() for _ in range(n)]
    y = [random.random() for _ in range(n)]
    return x, y

def vector_add(x, y):
    out = []
    for a, b in zip(x, y):
        out.append(a + b)
    return out

def dot_product(x, y):
    s = 0.0
    for a, b in zip(x, y):
        s += a * b
    return s

def main():
    N = 2_000_000  # adjust based on your machine
    x, y = gen_data(N)

    tracemalloc.start()
    pr = cProfile.Profile()
    pr.enable()

    t0 = time.perf_counter()
    s = dot_product(x, y)
    t1 = time.perf_counter()

    t2 = time.perf_counter()
    z = vector_add(x, y)
    t3 = time.perf_counter()

    pr.disable()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    # Print results
    print(f"Vector length N={N}")
    print(f"Dot product: {s:.6f} | Time: {t1 - t0:.3f} s")
    print(f"Vector add: len(z)={len(z)} | Time: {t3 - t2:.3f} s")
    print(f"Current/Peak memory: {current/1e6:.2f} MB / {peak/1e6:.2f} MB")

    # Profiler output
    s_buf = io.StringIO()
    ps = pstats.Stats(pr, stream=s_buf).sort_stats('cumtime')
    ps.print_stats(15)
    print("\n--- cProfile (Top 15 by cumulative time) ---")
    print(s_buf.getvalue())

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

```
Vector length N=2000000
Dot product: 499712.974120 | Time: 0.602 s
Vector add: len(z)=2000000 | Time: 5.636 s
Current/Peak memory: 65.13 MB / 65.13 MB

--- cProfile (Top 15 by cumulative time) ---
         2000021 function calls (2000020 primitive calls) in 6.238 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        6    3.709    0.618    5.625    0.938 {built-in method time.sleep}
  2000000    1.920    0.000    1.920    0.000 {method 'append' of 'list' objects}
      2/1    0.602    0.301    0.408    0.408 /tmp/ipython-input-425613865.py:16(dot_product)
        1    0.007    0.007    0.011    0.011 /tmp/ipython-input-425613865.py:10(vector_add)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        7    0.000    0.000    0.000    0.000 {built-in method posix.getppid}
        4    0.000    0.000    0.000    0.000 {built-in method time.perf_counter}
```

```
# section_b_matmul.py
import time, random, cProfile, pstats, io, tracemalloc

def gen_matrix(n, seed=123):
    random.seed(seed)
    return [[random.random() for _ in range(n)] for _ in range(n)]

def matmul_naive(A, B):
    n = len(A)
    C = [[0.0]*n for _ in range(n)]
    for i in range(n):
        for k in range(n):
            aik = A[i][k]
            for j in range(n):
                C[i][j] += aik * B[k][j]
    return C

def main():
    n = 150  # raise/lower based on your machine; O(n^3)
    A = gen_matrix(n, seed=1)
    B = gen_matrix(n, seed=2)

    tracemalloc.start()
    pr = cProfile.Profile()
    pr.enable()

    t0 = time.perf_counter()
    C = matmul_naive(A, B)
    t1 = time.perf_counter()

    pr.disable()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    print(f"Matrix size n={n} -> {n}x{n}")
    print(f"Time: {t1 - t0:.3f} s | C[0][0]={C[0][0]:.6f}")
    print(f"Current/Peak memory: {current/1e6:.2f} MB / {peak/1e6:.2f} MB")

    s_buf = io.StringIO()
    ps = pstats.Stats(pr, stream=s_buf).sort_stats('cumtime')
    ps.print_stats(10)
    print("\n--- cProfile (Top 10 by cumulative time) ---")
    print(s_buf.getvalue())

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

```
Matrix size n=150 -> 150x150
Time: 1.718 s | C[0][0]=37.256203
Current/Peak memory: 0.76 MB / 0.76 MB

--- cProfile (Top 10 by cumulative time) ---
         503 function calls (493 primitive calls) in 1.706 seconds

   Ordered by: cumulative time
   List reduced from 113 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    1.005    1.005    1.005    1.005 {built-in method time.sleep}
        1    0.521    0.521    0.521    0.521 /tmp/ipython-input-2637978816.py:8(matmul_naive)
       13    0.031    0.002    0.031    0.002 /usr/local/lib/python3.12/dist-packages/zmq/sugar/socket.py:632(send)
        1    0.000    0.000    0.026    0.026 /usr/local/lib/python3.12/dist-packages/ipykernel/iostream.py:219(<lambda>)
        1    0.000    0.000    0.026    0.026 /usr/local/lib/python3.12/dist-packages/ipykernel/iostream.py:221(_really_send)
        1    0.000    0.000    0.026    0.026 /usr/local/lib/python3.12/dist-packages/zmq/sugar/socket.py:709(send_multipart)
        4    0.000    0.000    0.001    0.000 /usr/lib/python3.12/asyncio/events.py:86(_run)
        4    0.000    0.000    0.001    0.000 {method 'run' of '_contextvars.Context' objects}
        2    0.000    0.000    0.001    0.000 /usr/local/lib/python3.12/dist-packages/tornado/ioloop.py:750(_run_callback)
      3/1    0.000    0.000    0.000    0.000 /usr/lib/python3.12/asyncio/base_events.py:1922(_run_once)
```

```python
# section_c_conv2d.py
import time, cProfile, pstats, io, tracemalloc

def gen_grid(h, w):
    return [[(i*j) % 255 / 255.0 for j in range(w)] for i in range(h)]

def conv2d(grid, kernel):
    H, W = len(grid), len(grid[0])
    kh, kw = len(kernel), len(kernel[0])
    rh, rw = kh//2, kw//2
    out = [[0.0]*W for _ in range(H)]
    for i in range(rh, H-rh):
        for j in range(rw, W-rw):
            acc = 0.0
            for di in range(-rh, rh+1):
                for dj in range(-rw, rw+1):
                    acc += grid[i+di][j+dj] * kernel[di+rh][dj+rw]
            out[i][j] = acc
    return out

def make_uniform_kernel(size=5):
    val = 1.0 / (size*size)
    return [[val]*size for _ in range(size)]

def main():
    H, W = 256, 256  # adjust based on machine
    grid = gen_grid(H, W)
    kernel = make_uniform_kernel(5)

    tracemalloc.start()
    pr = cProfile.Profile()
    pr.enable()

    t0 = time.perf_counter()
    out = conv2d(grid, kernel)
    t1 = time.perf_counter()

    pr.disable()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    print(f"Grid: {H}x{W} | Kernel: 5x5 | Time: {t1 - t0:.3f} s")
    print(f"Sample out[128][128]={out[128][128]:.6f}")
    print(f"Current/Peak memory: {current/1e6:.2f} MB / {peak/1e6:.2f} MB")

    s_buf = io.StringIO()
    ps = pstats.Stats(pr, stream=s_buf).sort_stats('cumtime')
    ps.print_stats(10)
    print("\n--- cProfile (Top 10 by cumulative time) ---")
    print(s_buf.getvalue())

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

```
Grid: 256x256 | Kernel: 5x5 | Time: 3.014 s
Sample out[128][128]=0.490980
Current/Peak memory: 2.07 MB / 2.07 MB

--- cProfile (Top 10 by cumulative time) ---
         14 function calls (13 primitive calls) in 3.014 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        2    2.010    1.005    2.010    1.005 {built-in method time.sleep}
      2/1    1.003    0.502    0.710    0.710 /tmp/ipython-input-3230325479.py:7(conv2d)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        3    0.000    0.000    0.000    0.000 {built-in method posix.getppid}
        2    0.000    0.000    0.000    0.000 {built-in method time.perf_counter}
        4    0.000    0.000    0.000    0.000 {built-in method builtins.len}
```

```python
# section_d_montecarlo_pi.py
import time, random, cProfile, pstats, io, tracemalloc

def estimate_pi(n_samples, seed=2025):
    random.seed(seed)
    inside = 0
    for _ in range(n_samples):
        x = random.random()
        y = random.random()
        if x*x + y*y <= 1.0:
            inside += 1
    return 4.0 * inside / n_samples

def main():
    N = 2_000_000  # adjust for your machine
    tracemalloc.start()
    pr = cProfile.Profile()
    pr.enable()

    t0 = time.perf_counter()
    pi_est = estimate_pi(N)
    t1 = time.perf_counter()

    pr.disable()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    print(f"N={N} -> pi ≈ {pi_est:.6f} | Time: {t1 - t0:.3f} s")
    print(f"Current/Peak memory: {current/1e6:.2f} MB / {peak/1e6:.2f} MB")

    s_buf = io.StringIO()
    ps = pstats.Stats(pr, stream=s_buf).sort_stats('cumtime')
    ps.print_stats(10)
    print("\n--- cProfile (Top 10 by cumulative time) ---")
    print(s_buf.getvalue())

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

```
N=2000000 -> pi ≈ 3.142150 | Time: 4.967 s
Current/Peak memory: 0.03 MB / 0.03 MB

--- cProfile (Top 10 by cumulative time) ---
         4000591 function calls (4000584 primitive calls) in 4.966 seconds

   Ordered by: cumulative time
   List reduced from 117 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      5/4    0.000    0.000    4.965    1.241 /usr/lib/python3.12/asyncio/base_events.py:1922(_run_once)
        4    0.009    0.002    4.953    1.238 /usr/lib/python3.12/selectors.py:451(select)
        4    3.398    0.850    4.021    1.005 {built-in method time.sleep}
  4000000    0.769    0.000    0.769    0.000 {method 'random' of '_random.Random' objects}
        1    0.607    0.607    0.720    0.720 /tmp/ipython-input-3913647630.py:4(estimate_pi)
        5    0.000    0.000    0.012    0.002 /usr/lib/python3.12/asyncio/events.py:86(_run)
        5    0.000    0.000    0.012    0.002 {method 'run' of '_contextvars.Context' objects}
        2    0.000    0.000    0.011    0.006 /usr/local/lib/python3.12/dist-packages/zmq/eventloop/zmqstream.py:574(_handle_eve
        1    0.000    0.000    0.011    0.011 /usr/local/lib/python3.12/dist-packages/tornado/platform/asyncio.py:206(_handle_ev
        2    0.000    0.000    0.011    0.006 /usr/local/lib/python3.12/dist-packages/zmq/eventloop/zmqstream.py:615(_handle_rec
# section_e_pairwise.py
import time, random, math, cProfile, pstats, io, tracemalloc

def gen_points(n, seed=7):
    random.seed(seed)
    return [(random.random(), random.random()) for _ in range(n)]

def pairwise_potential(points, eps=1e-6):
    n = len(points)
    pot = [0.0]*n
    for i in range(n):
        xi, yi = points[i]
        acc = 0.0
        for j in range(n):
            if i == j:
                continue
            xj, yj = points[j]
            dx, dy = xi - xj, yi - yj
            r = math.sqrt(dx*dx + dy*dy) + eps
            acc += 1.0 / r
        pot[i] = acc
    return pot

def main():
    N = 800  # adjust based on machine; O(N^2) interactions
    pts = gen_points(N)

    tracemalloc.start()
    pr = cProfile.Profile()
    pr.enable()

    t0 = time.perf_counter()
    pot = pairwise_potential(pts)
    t1 = time.perf_counter()

    pr.disable()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    print(f"N={N} -> computed potentials | Time: {t1 - t0:.3f} s")
    print(f"Sample pot[0]={pot[0]:.6f}, pot[N//2]={pot[N//2]:.6f}")
    print(f"Current/Peak memory: {current/1e6:.2f} MB / {peak/1e6:.2f} MB")

    s_buf = io.StringIO()
    ps = pstats.Stats(pr, stream=s_buf).sort_stats('cumtime')
    ps.print_stats(10)
    print("\n--- cProfile (Top 10 by cumulative time) ---")
    print(s_buf.getvalue())

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

```
N=800 -> computed potentials | Time: 5.093 s
Sample pot[0]=2390.396335, pot[N//2]=2337.388513
Current/Peak memory: 0.03 MB / 0.03 MB

--- cProfile (Top 10 by cumulative time) ---
         639215 function calls (639214 primitive calls) in 5.093 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        4    3.389    0.847    4.024    1.006 {built-in method time.sleep}
   639200    0.794    0.000    0.794    0.000 {built-in method math.sqrt}
      2/1    0.910    0.455    0.603    0.603 /tmp/ipython-input-2231473572.py:8(pairwise_potential)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        5    0.000    0.000    0.000    0.000 {built-in method posix.getppid}
        2    0.000    0.000    0.000    0.000 {built-in method time.perf_counter}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
```