Algoritmia y Complejidad (A)

LCCECO 202502

Luis Angel Tórtola Tejeda

Benchmarking: merge sort vs counting sort

Juan Ochaita 202405623

```
J2ANS@Juan MINGW64 ~/OneDrive/Desktop/Algoritmia y complejidad/Algoritmia-y-Complejidad/003-merge-sort-vs-counting-sort (master)
$ python -m pytest benchmarks.py --benchmark-only
==================== test session starts ====================
platform win32 -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\J2ANS\OneDrive\Desktop\Algoritmia y complejidad\Algoritmia-y-Complejidad\003-merge-sort-vs-counting-sort
plugins: benchmark-5.1.0
collected 2 items

benchmarks.py ..                                                                   [100%]


-------------------- benchmark: 2 tests --------------------
Name (time in ms)              Min                Max               Mean            StdDev             Median               IQR            Outliers       OPS            Rounds  Iterations
--------------------
test_counting_sort_small_k    1.1925 (1.0)       1.3943 (1.0)      1.2431 (1.0)      0.0857 (1.0)      1.2032 (1.0)      0.0742 (1.0)         1;1  804.4405 (1.0)       5           1
test_merge_sort_small_k      18.3383 (15.38)    29.0235 (20.82)   24.5519 (19.75)    4.1386 (48.31)   23.9251 (19.88)    5.4187 (73.00)       2;0   40.7301 (0.05)      5           1
--------------------

Legend:
  Outliers: 1 Standard Deviation from Mean; 1.5 IQR (InterQuartile Range) from 1st Quartile and 3rd Quartile.
  OPS: Operations Per Second, computed as 1 / Mean
==================== 2 passed in 0.24s ====================
```

Para los datos con rango pequeño (small_k), counting_sort fue mucho más rápido que merge_sort. Counting_sort tardó en promedio 1,24 milisegundos y realizó 804 operaciones por segundo, mientras que merge_sort tardó 24,55 milisegundos y solo hizo 40 operaciones por segundo. La diferencia se debe a que counting_sort funciona muy bien cuando el rango de los valores es pequeño, mientras que merge_sort depende de comparaciones y es más lento en este caso.

```
J2ANS@Juan MINGW64 ~/OneDrive/Desktop/Algoritmia y complejidad/Algoritmia-y-Complejidad/003-merge-sort-vs-counting-sort (master)
$ python -m pytest benchmarks_big_k.py --benchmark-only
==================== test session starts ====================
platform win32 -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
benchmark: 5.1.0 (defaults: timer=time.perf_counter disable_gc=False min_rounds=5 min_time=0.000005 max_time=1.0 calibration_precision=10 warmup=False warmup_iterations=100000)
rootdir: C:\Users\J2ANS\OneDrive\Desktop\Algoritmia y complejidad\Algoritmia-y-Complejidad\003-merge-sort-vs-counting-sort
plugins: benchmark-5.1.0
collected 2 items

benchmarks_big_k.py ..                                                                   [100%]


-------------------- benchmark: 2 tests --------------------
Name (time in ms)                   Min                  Max                 Mean              StdDev               Median                IQR            Outliers      OPS           Rounds  Iterations
--------------------
test_merge_sort_large_k         17.1227 (1.0)        33.7332 (1.0)        21.5373 (1.0)       6.9458 (1.0)        19.0530 (1.0)        6.4142 (1.0)          1;1  46.4312 (1.0)       5           1
test_counting_sort_large_k   4,988.3443 (291.33)  5,207.7839 (154.38)  5,129.6925 (238.18)   95.1334 (13.70)   5,186.7559 (272.23)   142.6353 (22.24)        1;0   0.1949 (0.00)      5           1
--------------------

Legend:
  Outliers: 1 Standard Deviation from Mean; 1.5 IQR (InterQuartile Range) from 1st Quartile and 3rd Quartile.
  OPS: Operations Per Second, computed as 1 / Mean
==================== 2 passed in 31.23s ====================
```

En esta prueba, merge_sort fue mucho más rápido que counting_sort para los datos con rango grande. Merge_sort tardó en promedio 21,54 milisegundos y realizó 46 operaciones por segundo. Counting_sort tardó más de 5 segundos y solo hizo 0,19 operaciones por segundo. La diferencia se debe a que counting_sort necesita memoria y tiempo proporcional al rango de los valores. Cuando el rango es grande, se vuelve muy lento. Merge_sort no depende del rango, por eso funciona bien en este caso.