*Чеботарев Георгий*

В данном отчете представлены результаты bench-тестов для некоторых из сортировок:

## Insertion Sort

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Avgt | Cnt | Score | Error | Units |
| 10 | 5 | 9,537 | 0,426 | ns/op |
| 100 | 5 | 42,706 | 20,261 | ns/op |
| 1000 | 5 | 1599,966 | 58,843 | ns/op |
| 10000 | 5 | 15911,473 | 251,854 | ns/op |
| 50000 | 5 | 79636,731 | 3237,705 | ns/op |
| 100000 | 5 | 159029,609 | 2254,972 | ns/op |

Результат считаю не показательным и странным. Как и все ниже следующие, ошибки найти так и не удалось, объяснить не квадратичное поведение не могу.

## Сортировка вставками + бин.поиск + сдвиги

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Avgt | Cnt | Score | Error | Units |
| 1 | 5 | 24,615 | 0,799 | ns/op |
| 10 | 5 | 345,127 | 314,315 | ns/op |
| 100 | 5 | 7641,417 | 10119,525 | ns/op |
| 1000 | 5 | 165567,074 | 26804,488 | ns/op |
| 10000 | 5 | 2406041,125 | 319516,321 | ns/op |
| 50000 | 5 | 14065929,835 | 1052558,014 | ns/op |
| 100000 | 5 | 33593862,233 | 10050769,399 | ns/op |

## 3.Сортировка Шелла

Результаты тестов быстродействия для сортировки Шелла представлены ниже:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Avgt | Cnt | Score | Error | Units |
| 1 | 5 | 3,940 | 0,358 | ns/op |
| 10 | 5 | 52,476 | 2,661 | ns/op |
| 100 | 5 | 368,277 | 29,259 | ns/op |
| 1000 | 5 | 6948,796 | 3686,802 | ns/op |
| 10000 | 5 | 278212,599 | 45475,124 | ns/op |
| 50000 | 5 | 1675769,447 | 407829,786 | ns/op |
| 100000 | 5 | 3821649,148 | 440812,371 | ns/op |

## 4. Sort Merge

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Avgt | Cnt | Score | Error | Units |
| 10 | 5 | 429,392 | 48,404 | ns/op |
| 100 | 5 | 6136,544 | 293,769 | ns/op |
| 1000 | 5 | 67669,693 | 2212,620 | ns/op |
| 10000 | 5 | 765937,304 | 37735,191 | ns/op |
| 50000 | 5 | 4123693,997 | 2529794,082 | ns/op |
| 100000 | 5 | 8358592,979 | 2278144,360 | ns/op |

Аналогичная ситуация, сортировка со сложностью n\*log(n) выполняется хуже, чем квадратичная сортировка вставками. Бред. Графики не привожу, т.к. при таких результатах они совершенно не показательны.

## 5. Merge sort без дополнительной памяти

Тесты в данном алгоритме оказались очень кстати. Они помогли обнаружить ошибку (при сортировке 100 чисел, все были отсортированы, кроме 82, оно было после 91 и перед 92, что странно))). Но ошибка была обнаружена и исправлена.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Avgt | Cnt | Score | Error | Units |
| 10 | 5 | 154,963 | 7,339 | ns/op |
| 100 | 5 | 2393,429 | 213,475 | ns/op |
| 1000 | 5 | 183413 | 19509,336 | ns/op |
| 10000 | 5 | 12960788,118 | 946498,443 | ns/op |
| 50000 | 5 | 414991737,000 | 7738259,391 | ns/op |
| 100000 | 5 | 1755384617,200 | 24904246,503 | ns/op |

Если довериться результатам, получается, что сортировка без дополнительной памяти выигрывает у обычной сортировки слиянием при массивах маленьких размеров (массив размером до 5 тыс). Данная сортировка поддается замирению с трудом, т.к. содержит 150 строк кода и 6 методов. Java машина не способна качественно оптимизировать этот код.

## 6. Quick Sort - фиксированный выбор опорного элемента

Quick sort имеет квадратичную сходимость. Результаты аналогичные сортировки вставками. Квадратность не наблюдается, наблюдается линейность.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Avgt | Cnt | Score | Error | Units |
| 1 | 5 | 3,783 | 0,070 | ns/op |
| 10 | 5 | 68,658 | 5,348 | ns/op |
| 100 | 5 | 965,144 | 75,523 | ns/op |
| 1000 | 5 | 10978,812 | 372,532 | ns/op |
| 10000 | 5 | 158727,807 | 12616,930 | ns/op |
| 50000 | 5 | 884750,896 | 16500,568 | ns/op |
| 100000 | 5 | 1770151,795 | 47496,230 | ns/op |

Тесты в порядке.

## 7. Quick Sort + random для опорного + разделение на три части

Результаты чуть на рандомных данных примерно такие же.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Avgt | Cnt | Score | Error | Units |
| 1 | 5 | 4,718 | 2,522 | ns/op |
| 10 | 5 | 94,817 | 19,030 | ns/op |
| 100 | 5 | 1050,736 | 166,693 | ns/op |
| 1000 | 5 | 10978,812 | 4093,996 | ns/op |
| 10000 | 5 | 16112,807 | 16811,171 | ns/op |
| 50000 | 5 | 1177553,925 | 154955,594 | ns/op |
| 100000 | 5 | 2085300,996 | 235298,242 | ns/op |

## 8. K-ая порядковая статистика в среднем O(n)

В данном случае выполнялся поиск среднего элемента в массиве.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Avgt | Cnt | Score | Error | Units |
| 1 | 5 | 3,776 | 0,201 | ns/op |
| 10 | 5 | 220,447 | 51,912 | ns/op |
| 100 | 5 | 550,102 | 102,251 | ns/op |
| 1000 | 5 | 1513,028 | 76,208 | ns/op |
| 10000 | 5 | 9122,842 | 419,721 | ns/op |
| 50000 | 5 | 42593,306 | 1936,329 | ns/op |
| 100000 | 5 | 75260,572 | 3471,115 | ns/op |

## 9. K-ая порядковая статистика в среднем O(n)

В худшем случае процедура partition возвращает левую или правую границу. В таком случае характеристика составит O(n^2). Для достижения этой цели был использован алгоритм из задачи «Анти-Quick Sort»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Avgt | Cnt | Score | Error | Units |
| 1 | 5 | 4,313 | 2,835 | ns/op |
| 10 | 5 | 218,839 | 31,057 | ns/op |
| 100 | 5 | 731,654 | 645,084 | ns/op |
| 1000 | 5 | 1507,318 | 46,127 | ns/op |
| 10000 | 5 | 8984,362 | 1009,842 | ns/op |
| 50000 | 5 | 44182,516 | 15644,572 | ns/op |
| 100000 | 5 | 80073,438 | 6856,971 | ns/op |

Ожидалось квадратичное увеличение времени выполнения, но нет. Возможно это обусловлено скоростью работы ЦП. Если k-ая порядковая статистика находиться за 1 квант-времени процессора и в том и в другом случае будут одинаковые результаты. Другого логического объяснения я найти не могу.

**Вывод:**

Технология интересна и однозначна полезна. Однако практический результат не совпадает с теоретическим. Это странно.