**Análise Temporal - Projeto 3**

**Docente:** João Dias

**Discente:** Diogo Fonseca nº79858

**Comparação do Recursive Sort com o Quick Sort**

|  |  |  |  |
| --- | --- | --- | --- |
| n | Recursive Sort (ns) | Quick Sort (ns) | Nº testes |
| 10 | 2171 | 2015 | 100000 |
| 15 | 2546 | 2281 | 100000 |
| 25 | 3593 | 3242 | 100000 |
| 35 | 3867 | 3554 | 100000 |
| 37 | 4648 | 4648 | 100000 |
| 40 | 4277 | 4843 | 100000 |
| 50 | 5375 | 5781 | 100000 |
| 100 | 8125 | 11562 | 100000 |
| 500 | 60937 | 79687 | 100000 |
| 1000 | 162500 | 165625 | 100000 |
| 2000 | 346875 | 426562 | 100000 |
| 5000 | 765625 | 1281250 | 100000 |
| 8000 | 1296875 | 1828125 | 1000 |
| 10000 | 2000000 | 2562500 | 1000 |
| 12000 | 2671875 | 3546875 | 1000 |
| 1000000 | 356562500 | 595156250 | 100 |

Tabela 1. Tempos médios em relação ao tamanho da array

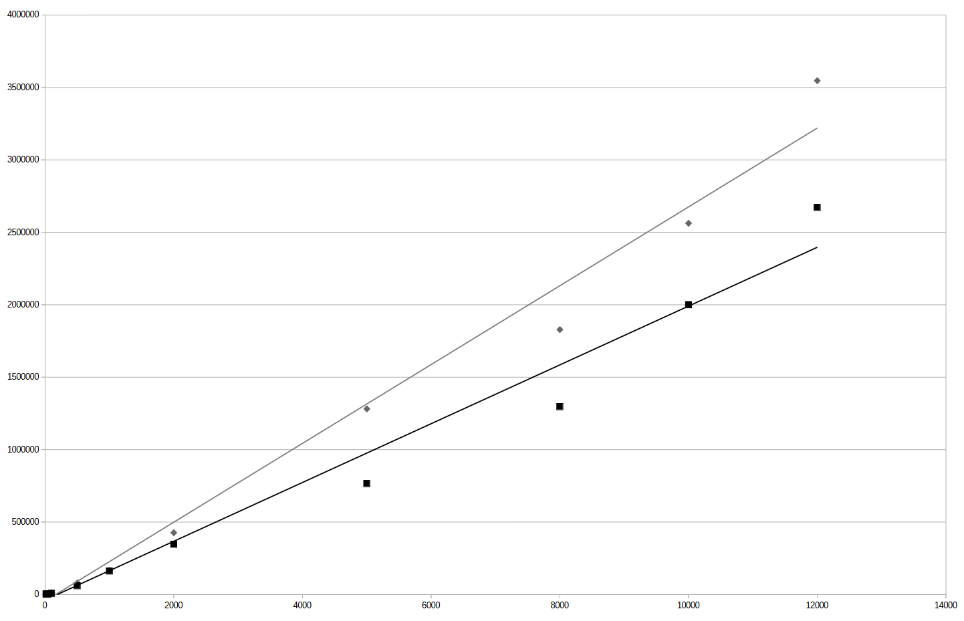


Gráfico I. Tempos médios (Quick Sort – cinzento) (Recursive Sort – Preto)

Para cada entrada na tabela 1, n sendo o tamanho do array, foram realizados ~100000 testes ao qual o tempo médio de execução é a média geométrica dos tempos.

É evidenciado pelos dados em cima que para quase qualquer caso o Recursive Sort é melhor que o Quick Sort (dado que o recursive sort só ordena strings, e sacrifica muita memória para o fazer). Para arrays menores que 37 o quick sorte é em média mais rápido que o Recursive Sort.

Observando o gráfico I também é fácil deduzir que (a partir de ~37) o recursive sort é bastante mais rápido que o quick sort (e a diferença só aumenta).

Os arrays gerados são arrays onde cada string são caracteres alfanuméricos (minúsculos e maiúsculos) os quais variam aleatóriamente de tamanho entre 1 e 1000 caracteres.

**Complexidade temporal do Recursive Sort**

|  |  |  |
| --- | --- | --- |
| n | Tempo Médio de Execução (ms) | Razão |
| 250 | 0.5 | 0 |
| 500 | 0.5 | 1.0 |
| 1000 | 0.5 | 1.0 |
| 2000 | 0.5 | 1.0 |
| 4000 | 2.1 | 4.0 |
| 8000 | 2.6 | 1.2 |
| 16000 | 2.1 | 0.8 |
| 32000 | 7.8 | 3.8 |
| 64000 | 18.2 | 2.3 |
| 128000 | 37.0 | 2.0 |
| 256000 | 85.9 | 2.3 |
| 512000 | 174.0 | 2.0 |
| 1024000 | 362.5 | 2.1 |
| 2048000 | 899.4 | 2.5 |
| 4096000 | 2682.3 | 3.0 |
| 8192000 | 5577.1 | 2.1 |

Tabela 2. Ensaios de razão dobrada do recursive sort

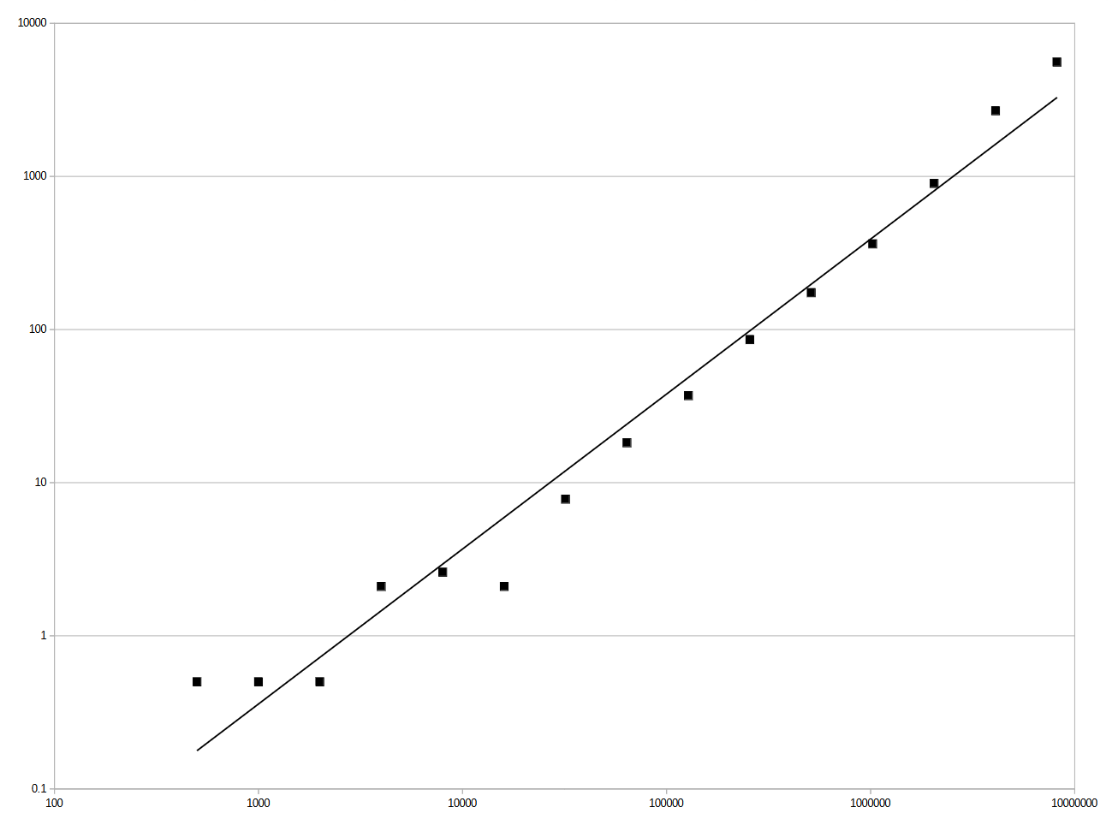


Gráfico I. Tempos médios (Quick Sort – cinzento) (Recursive Sort – Preto)

Gráfico II. (Escala log) Ensaios de razão dobrada do recursive sort

Para cada entrada na tabela 2 foram realizados 30 testes aos quais o tempo médio de execução é a média geométrica dos tempos dos testes.

Os exemplos gerados para os ensaios de razão dobrada são iguais aos da comparação anterior: entre 1 e 1000 caracteres alfanuméricos para cada elemento do array de tamanho n.

Observando quer a tabela 2 podemos concluir que a razão flutua por volta do 2, logicamente querendo dizer que a sua complexidade temporal assintótica é linear (. Isto também pode ser facilmente deduzido ao observar o gráfico II (escala log), onde se pode confirmar uma linha que descreve os pontos obtidos pelo ensaio (sendo então linear).

A complexidade assintótica consegue também ser deduzida através de uma análise ao algoritmo ao invés de pelos testes empíricos: O custo de introduzir algo no seu respetivo balde é constante, já que estes estão numa array. O custo de ordenar os baldes que têm menor que um determinado número de elementos também é constante (o caso base do algoritmo), pois o insertion sort vai estar sempre a trabalhar no seu máximo com o mesmo número de elementos. Na primeira iteração, cada string vai ser colocada num balde, sendo necessário passar por todos os elementos do array, ou n. Na segunda iteração a soma de todos os elementos de todos os baldes da segunda iteração vai também dar n supondo que as strings são todas do mesmo tamanho, logo as divisões nos baldes consequentes também vão ser feitas iterando por um total de n strings. Isto dá um somatório de n operações de cópia de um valor para um array: onde é a **média** do tamanho das strings. Esta expressão pode ser simplificada em . Se assumirmos que é o tamanho **máximo** das strings, e que estas demonstram uma distribuição uniforme, então a complexidade média seria .

Por esta lógica o pior caso para n será quanto maior for o valor do tamanho médio das strings. Se as strings forem todas do valor máximo estamos à espera que no pior caso seja duas vezes mais lento que no caso médio.

**Pior caso do Recursive Sort**

Como deduzido analiticamente no parágrafo anterior…