

Análise do MergeSort

Para essa análise, usei o algoritmo corrigido (Merge.java), que disponibilizei no moodle, e o BetterGenerator.java, um gerador de strings aleatórias. Usei o gerador para gerar N strings de 10 caracteres pertencentes ao alfabeto (abcdefghijklmnopqrstuvwxyz).

Usei valores de N dobrados a cada rodagem do programa (Doubling Method), como mostra a imagem:

```
~/Documents/Curso/CM/2º Período/Computação II/EPs/EP 3> java-introcs BetterGenerator 10000 | java-introcs Merge
5 milliseconds
~/Documents/Curso/CM/2º Período/Computação II/EPs/EP 3> java-introcs BetterGenerator 20000 | java-introcs Merge
11 milliseconds
~/Documents/Curso/CM/2º Período/Computação II/EPs/EP 3> java-introcs BetterGenerator 40000 | java-introcs Merge
14 milliseconds
~/Documents/Curso/CM/2º Período/Computação II/EPs/EP 3> java-introcs BetterGenerator 80000 | java-introcs Merge
27 milliseconds
~/Documents/Curso/CM/2º Período/Computação II/EPs/EP 3> java-introcs BetterGenerator 160000 | java-introcs Merge
59 milliseconds
~/Documents/Curso/CM/2º Período/Computação II/EPs/EP 3> java-introcs BetterGenerator 320000 | java-introcs Merge
121 milliseconds
~/Documents/Curso/CM/2º Período/Computação II/EPs/EP 3> java-introcs BetterGenerator 640000 | java-introcs Merge
271 milliseconds
~/Documents/Curso/CM/2º Período/Computação II/EPs/EP 3> java-introcs BetterGenerator 1280000 | java-introcs Merge
586 milliseconds
~/Documents/Curso/CM/2º Período/Computação II/EPs/EP 3> java-introcs BetterGenerator 2560000 | java-introcs Merge
1344 milliseconds
~/Documents/Curso/CM/2º Período/Computação II/EPs/EP 3> java-introcs BetterGenerator 5120000 | java-introcs Merge
3039 milliseconds
~/Documents/Curso/CM/2º Período/Computação II/EPs/EP 3> java-introcs BetterGenerator 10240000 | java-introcs Merge
6847 milliseconds
```

Podemos fazer uma tabela com esses dados.

N	T_N (milisegundos)	$T_N/T_{N/2}$
10000	5	
20000	11	2.2
40000	14	1.27
80000	27	1.93
160000	59	2.18
320000	121	2.05

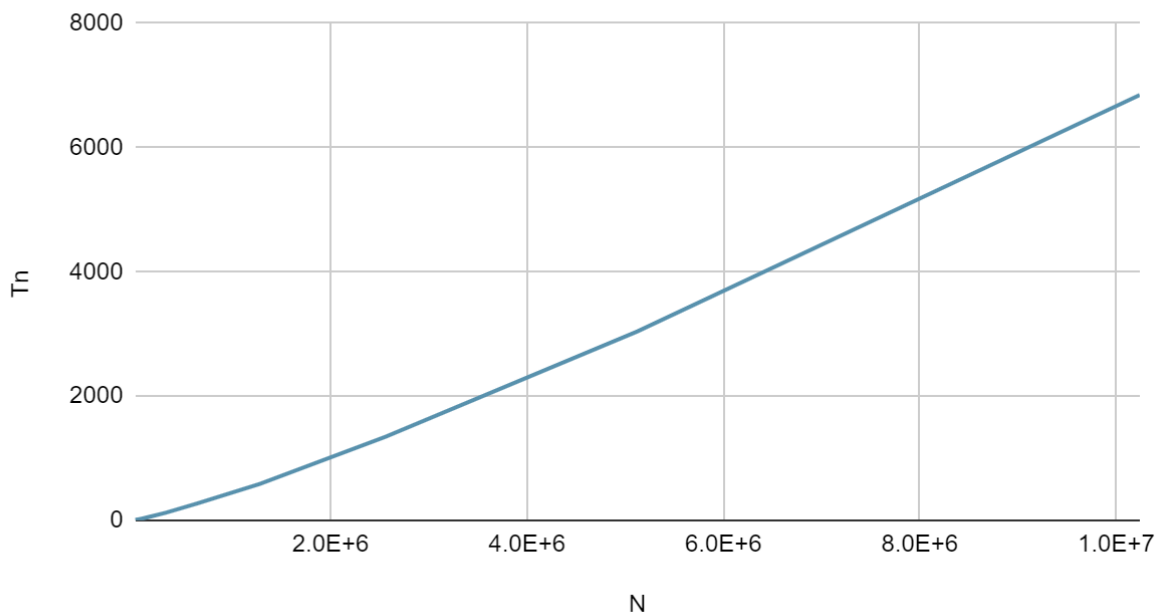
N	T_N (milisegundos)	$T_N/T_{N/2}$
640000	271	2.16
1280000	586	2.16
2560000	1344	2.29
5120000	3039	2.26
10240000	6847	2.25

Suponhamos que o tempo de execução do programa seja $T_N \sim a N^b$, onde a é uma constante. Logo, $T_N/T_{N/2} = a N^b / a (N/2)^b = 2^b$. Portanto, quanto maior N , mais $T_N/T_{N/2}$ se aproxima de 2^b .

Da tabela, podemos perceber que $T_N/T_{N/2}$ se aproxima de 2.3, logo $2^b=2.3$, $b = \log_2 2.3$. Isso implica em crescimento de ordem $N \log N$.

Podemos fazer um gráfico $T_N \cdot N$ a partir da tabela, que corresponde a um crescimento linear bem próximo do linear, como esperado de um crescimento de ordem $N \log N$:

T_N versus N



Além disso, o algoritmo é estável, já que ele separa o vetor em partes menores, ordena-as e une-as, sem mudar, portanto, a ordem de elementos iguais do input.