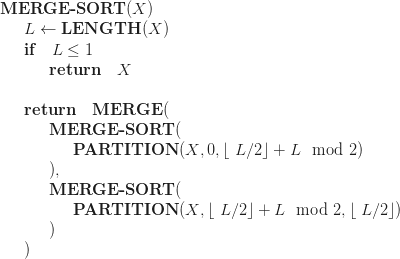
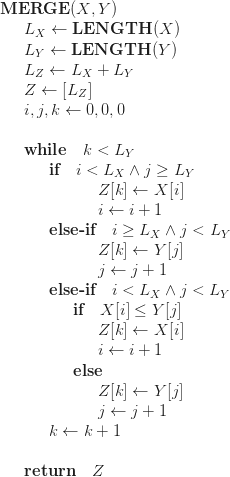
**Parallel Mergesort**

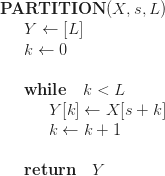
documentaţie

Acest document prezintă evaluarea performanţei algoritmului de Mergesort implementat în limbajul Java, folosind Threads pentru a-l rula concurent.

**Specificaţii**

Fiind dat un șir de valori, numerele sunt sortate folosind o strategie de Divide and Conquer, care partiţioneaza șirul inițial în două partiţii de dimensiuni (aproape) egale, urmând ca la fiecare pas recursiv cele două partiţii la rândul lor să fie împărţite în alte două de dimensiuni (aproape) egale, iar când acestea au dimensiunea 2, se vor păstra sortate, urmând ca la revenirea din recursie să se facă procedura de *merge* a celor două porţiuni sortate.





Time complexity: O(NlogN)

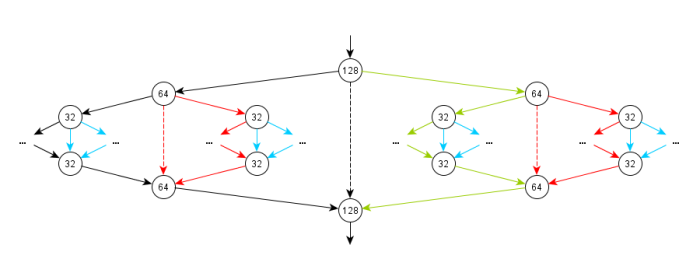
Space complexity: O(N)

[1]

**Abordarea paralelă**

Implementarea paralelă, folosind Threads, se folosește de faptul că porţiunea de *Divide* poate fi usor paralelizată, sortând o partiţie în thread-ul curent, și cealaltă partiție într-un alt thread. În momentul în care cel de-al doilea thread și-a terminat treaba, apare operațiunea de *join* a celor două thread-uri, și cele două porțiuni sunt supuse operației de *merge*.

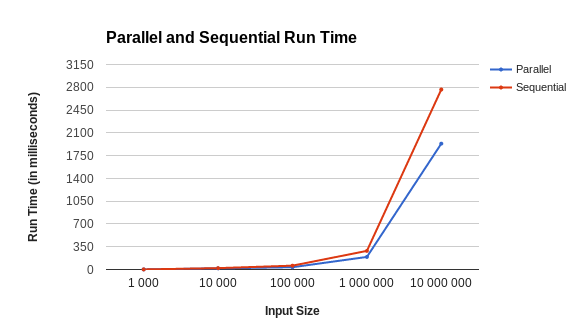
Ilustrăm procesul descris mai sus printr-o imagine.

[2]

Menționăm că în cazul versiunii paralele, complexitatea timp a devenit O(N), considerându-se că instrucțiunile executate în cele două thread-uri se execută simultan, fiind privită ca timp liniar.

**Performance analysis**

|  |  |  |
| --- | --- | --- |
| **Input Size** | **Parallel Run Time** (ms) | **Sequential Run Time** (ms) |
| 1 000 | 1 | 1 |
| 10 000 | 17 | 21 |
| 100 000 | 35 | 59 |
| 1 000 000 | 193 | 286 |
| 10 000 000 | 1934 | 2766 |



**Bibliografie**

[1] <https://en.wikipedia.org/wiki/Merge_sort>

[2] <https://antimatroid.wordpress.com/2012/12/01/parallel-merge-sort-in-java/>