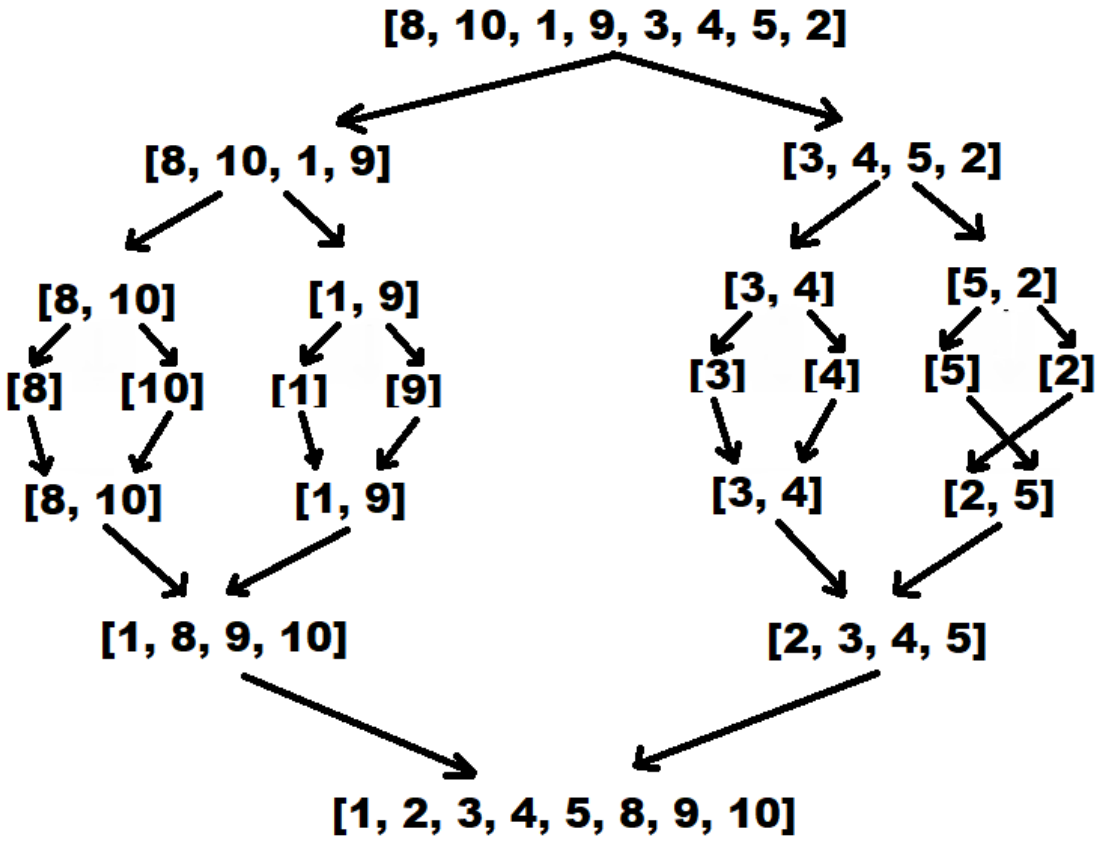
Threaded Merge Sort

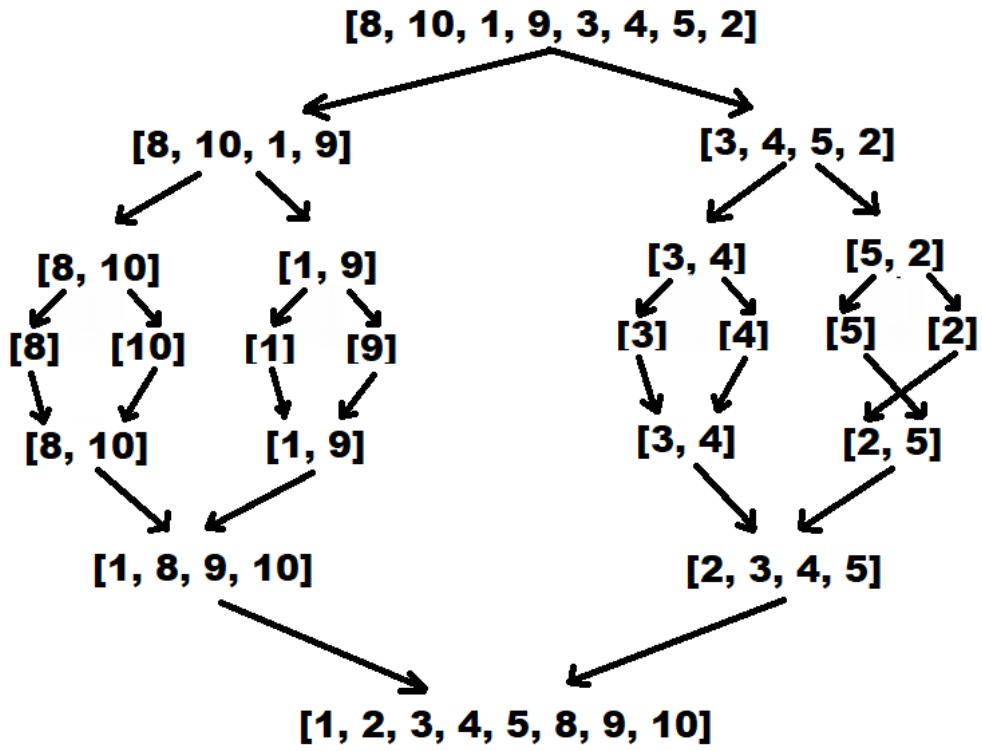


High Performance Programming
Niels Bijl
26/02/2021

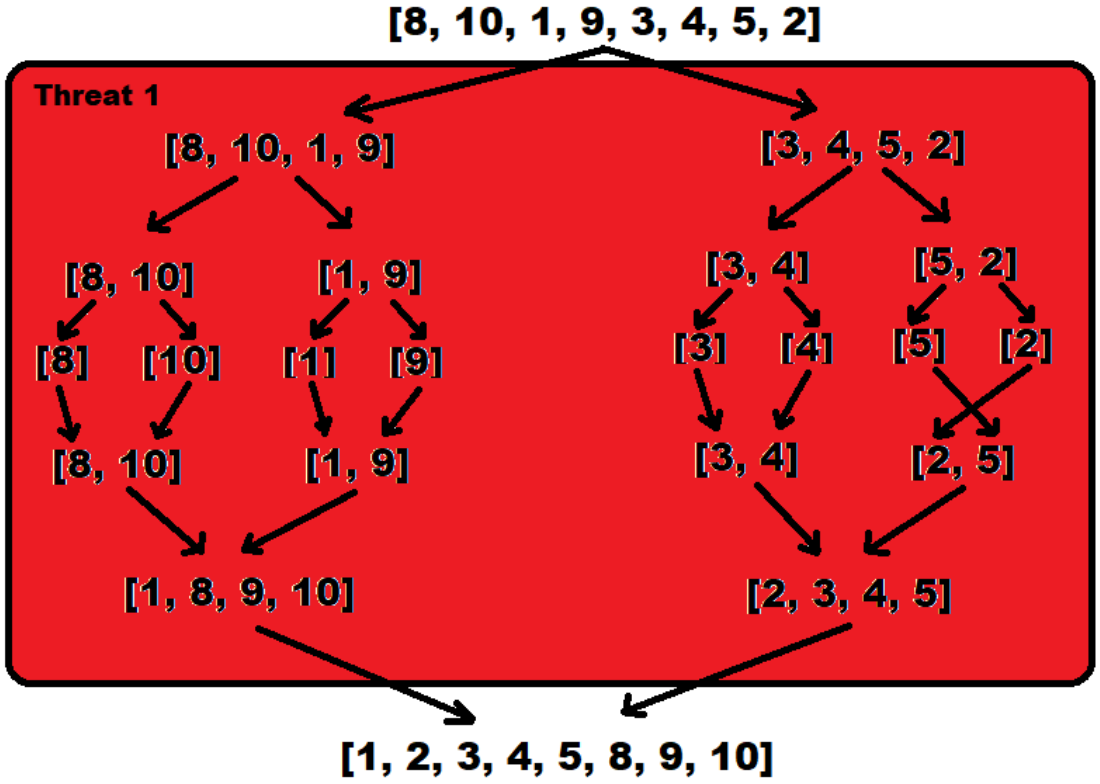
Contents

Ontwerp op papier

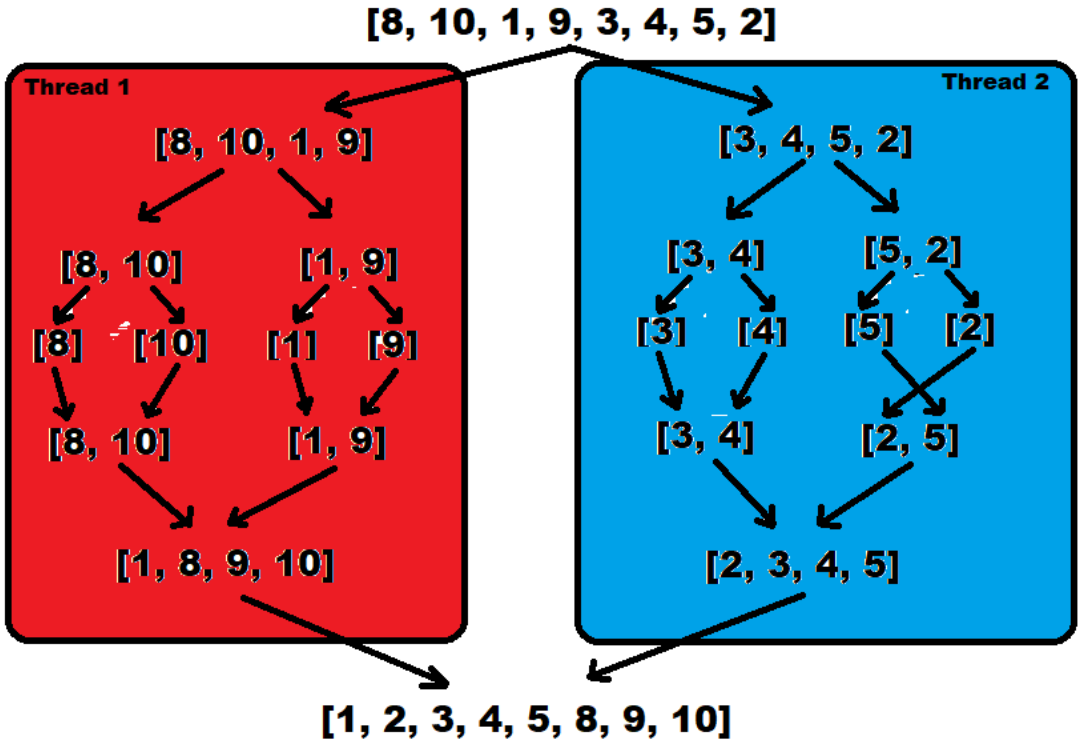
Basis Merge Sort



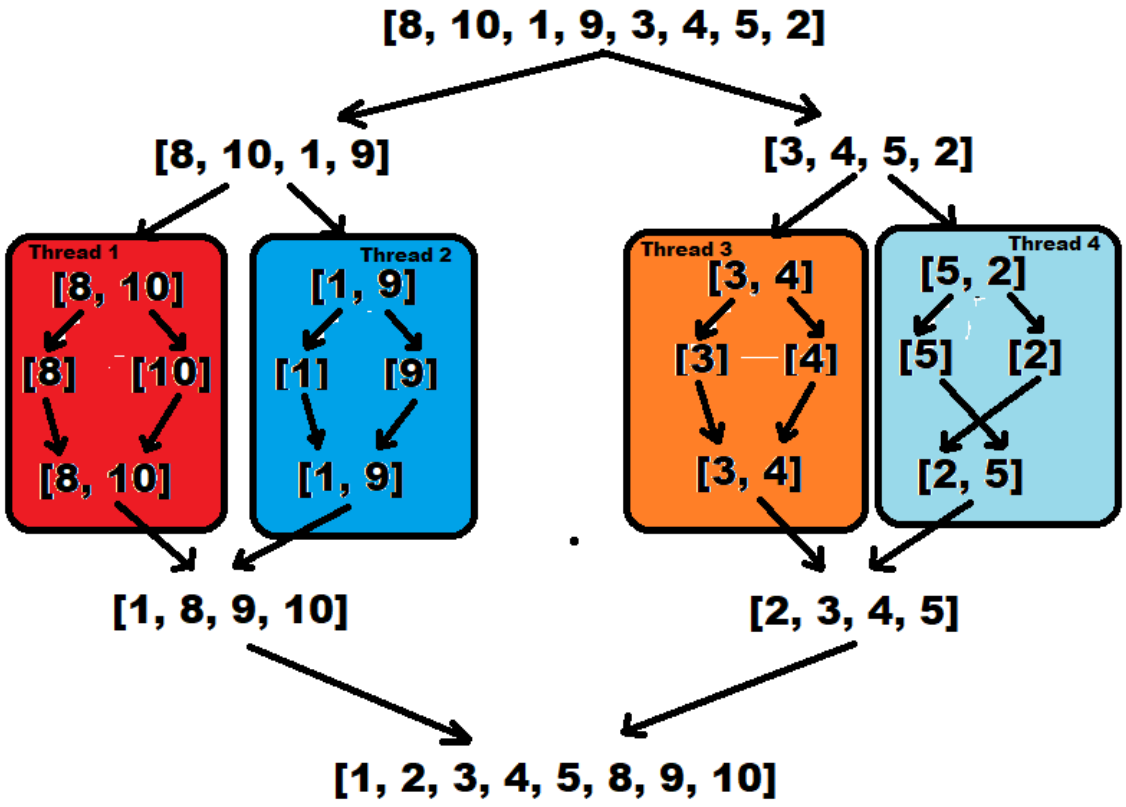
1 thread



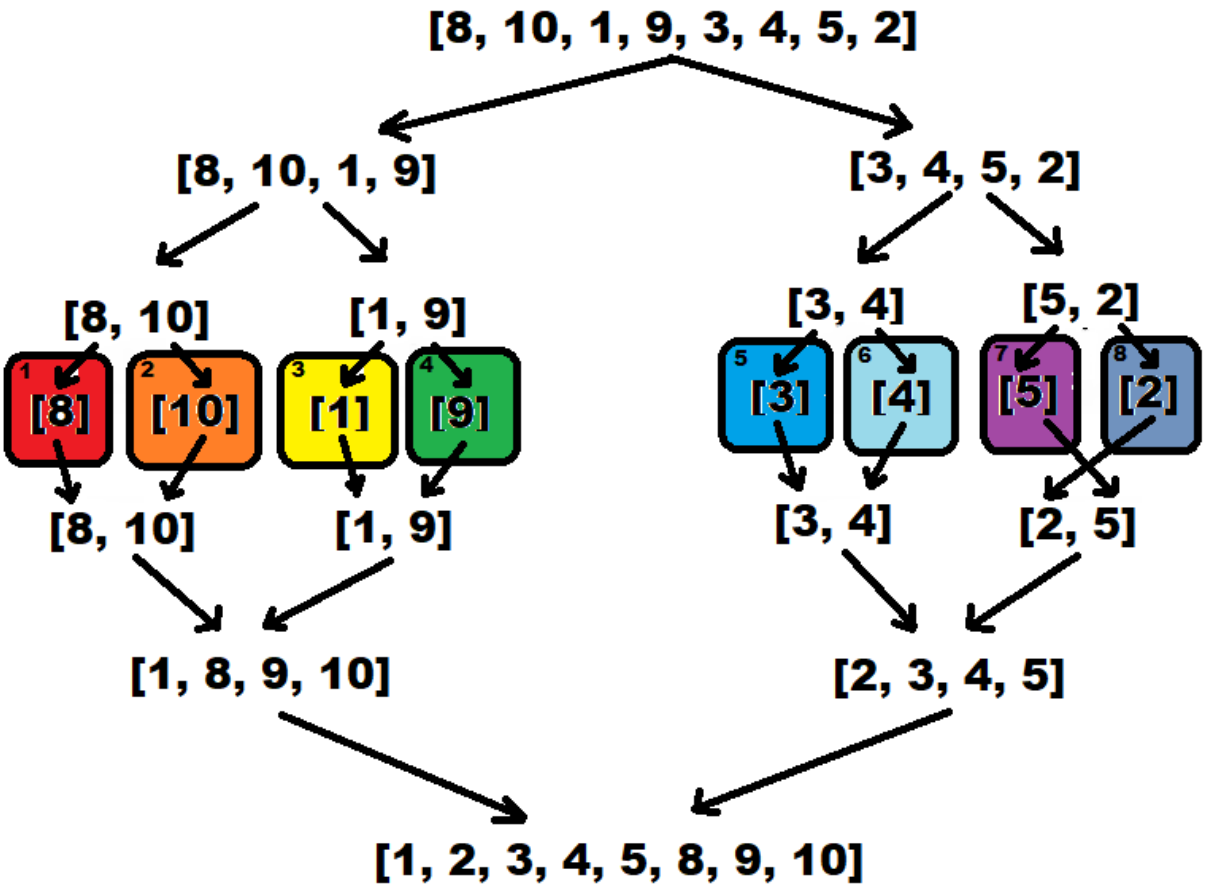
2 threads



4 threads



8 threads


[8, 10, 1, 9, 3, 4, 5, 2]
[8, 10, 1, 9]
[3, 4, 5, 2]
[8, 10]
[1, 9]
[3, 4]
[5, 2]
1
[8]
2
[10]
3
[1]
4
[9]
5
[3]
6
[4]
7
[5]
8
[2]
[8, 10]
[1, 9]
[3, 4]
[2, 5]
[1, 8, 9, 10]
[2, 3, 4, 5]
[1, 2, 3, 4, 5, 8, 9, 10]

Analyse ontwerp

1 thread

Dit is een Single Instruction Single Data (SISD) architecture. De data zijn niet opgesplitst en is gewoon nog de volledige input lijst. Ook worden alle instructies één voor één uitgevoerd.

2 threads

Dit is een Single Instruction Multiple Data (SIMD) architecture. De data worden opgesplitst in twee sub lijsten, deze twee sub lijsten worden beiden tegelijk gesorteerd. De instructies worden nog wel één voor één uitgevoerd.

4 threads

Ook dit is een Single Instruction Multiple Data (SIMD) architecture. De data worden opgesplitst in vier sub lijsten, deze vier sub lijsten worden beiden tegelijk gesorteerd. De instructies worden nog wel één voor één uitgevoerd.

8 threads

Ook dit is een Single Instruction Multiple Data (SIMD) architecture. De data worden opgesplitst in acht sub lijsten, deze acht sub lijsten worden beiden tegelijk gesorteerd. De instructies worden nog wel één voor één uitgevoerd. Omdat de input acht lang is heeft elke sub lijst maar één getal, de enige instructie die hieruit wordt gevoerd is het returnen van het getal. Dit heeft helemaal geen nut en is extreem inefficiënt!

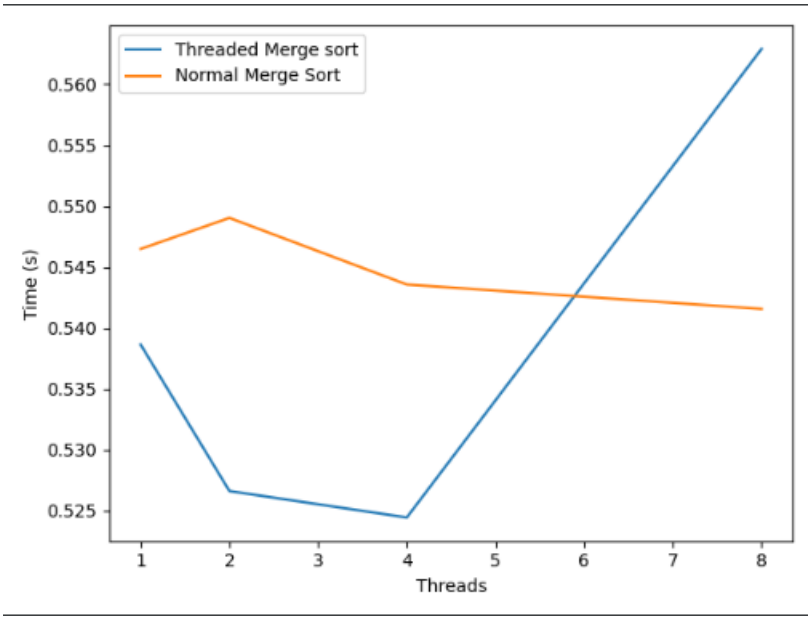
De complexiteit van de normale merge sort is O(n log n). Door het verdelen van de data over 2 threads zou de complexiteit O((n log n)/2) moeten zijn. Omdat je bijna al het werk parallel aan het doen bent.

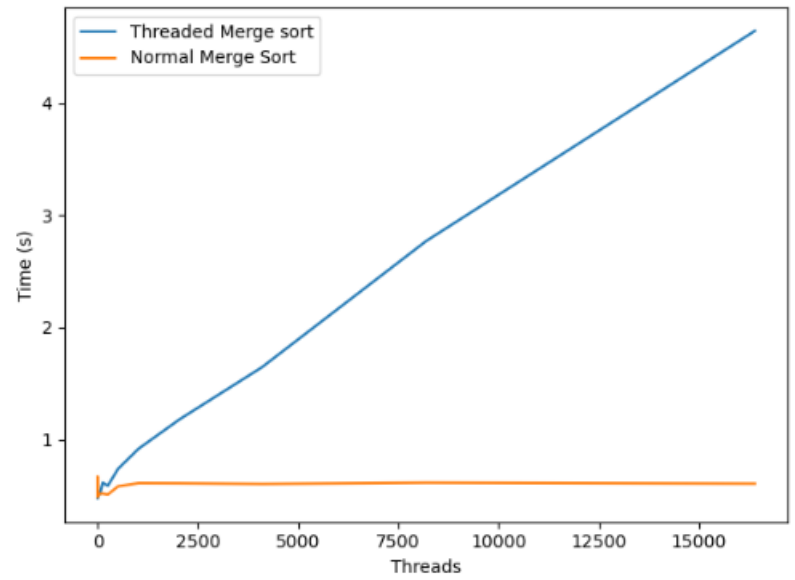
De complexiteit van de Multi threaded merge sort is O((n log n)/threads). De originele complexiteit gedeeld door het aantal threads.

Omdat python door de Global Interpreter Lock de threads alsnog 1 voor 1 uitvoert, wordt de complexiteit juist complexer, door het onnodig aanmaken van de threads. En is de complexiteit dus niet O((n log n)/threads). Wat de complexiteit dan wel is, is afhankelijk van de complexiteit van het maken van threads in een pool. Maar het verhogen van het aantal threads verlengt de run-time enorm.

Het zou kunnen zijn dat de complexiteit O((n log n)*threads) wordt.

Threaded vs normal

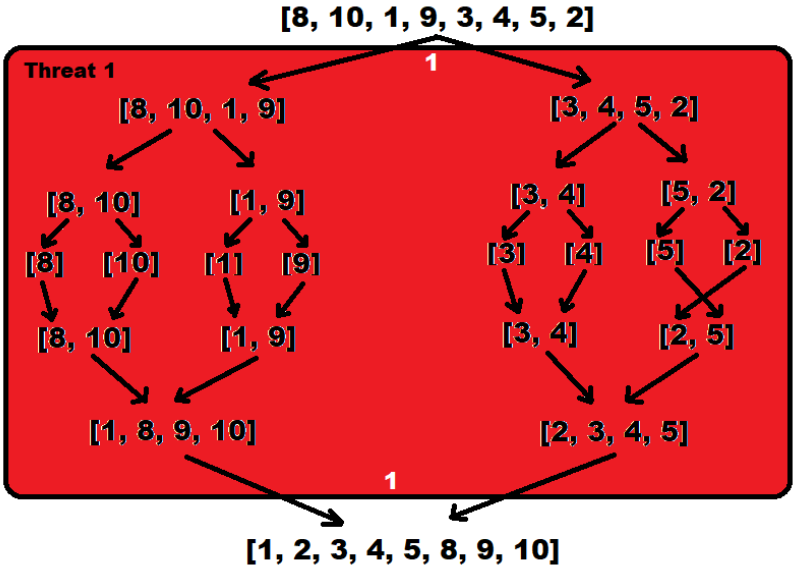


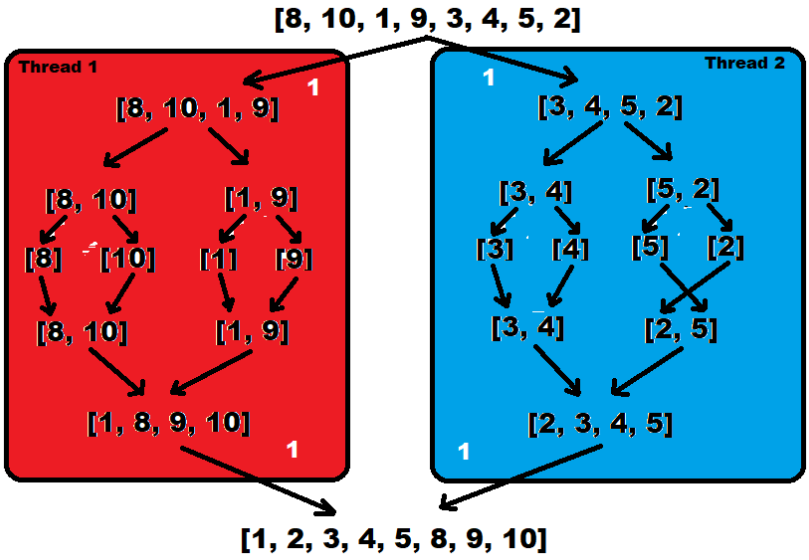


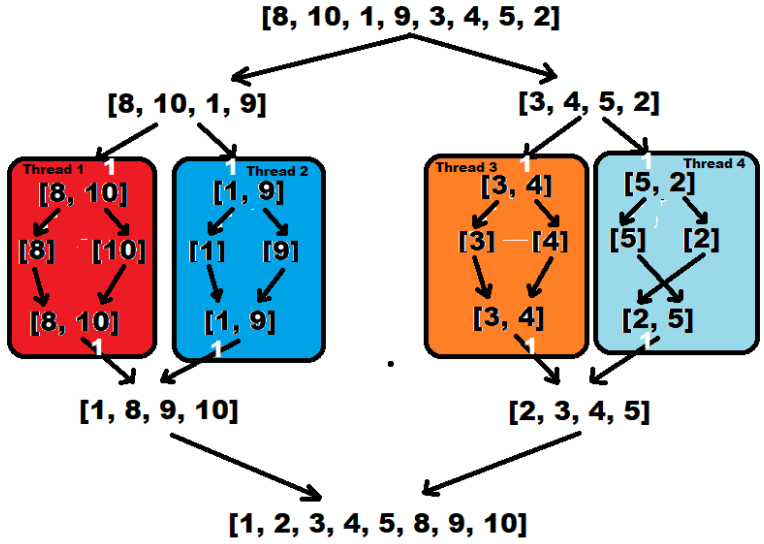
Run-time complexiteit Threaded Merge Sort vs Normal Merge Sort. Lijst van 100000 items.

Hier is te zien dat 1, 2 en 4 threads sneller zijn dan 0 threads. Bij 8 threads begint het echt slomer te worden dan zonder threads. Dit geldt alleen bij een lijst van 100000 items. Dit is waarschijnljk te verklaren omdat I/O-bound instructies wel parallel worden uitgevoerd en dit dus een positief effect geeft bij 1, 2 en 4 threads.

Communicatie overhead







Hierboven is de communicatie geteld (de witte 1'tjes) voor de implementatie van 1, 2 en 4 threads. De communicatie is tussen het main thread en de extra threads. Bij 1 extra thread is de communicatie 2, want de mainthread geeft de volledige lijst mee aan de extra thread. De extra thread geeft hem daarna weer gesorteerd terug aan de mainthread.

Communicatie = aantal extra threads * 2

De communicatie heeft geen correlatie met de grootte van de lijst

Extra: Global Interpreter Lock

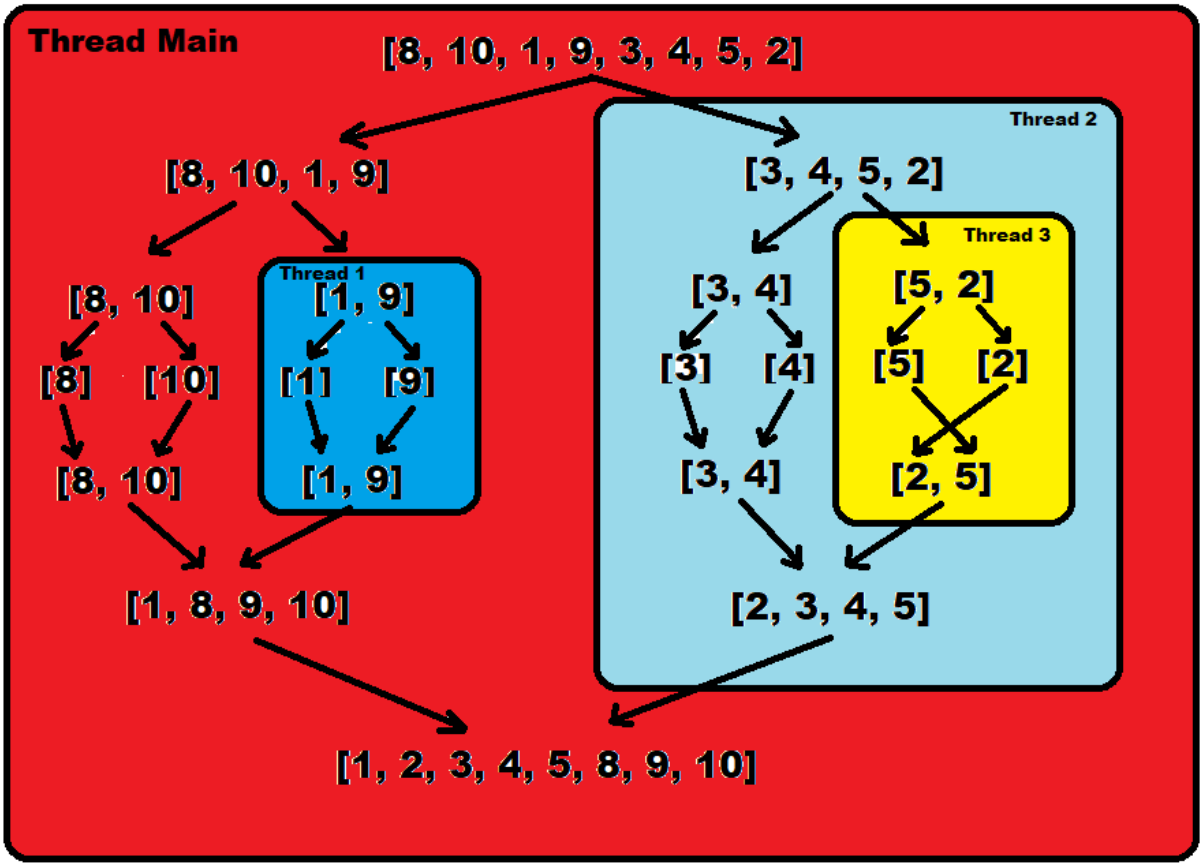
De Global Interpreter Lock zorgt ervoor dat er maar 1 thread tegelijk de controllen hebben over de Python interpreter. Dit resulteert erin dat er maar 1 thread tegelijk uitgevoerd kan worden. Dit wordt ook wel gezien als een van de slechtste feature van Python.

Een anekdoten hiervoor is stel je hebt 100 developers in een bedrijf en je hebt maar 1 koffiekop, bijna alle developers zijn dan aan het wachten op de koffiekop in plaats van het programmeren. De koffiekop is hierin de python interpreter.

Bij multiprocessing heb je hier geen last van, want elk proces krijgt zijn eigen interpreter en memory space.

Bij I/O-bound programma's heeft de Global Interpreter Lock geen negatief effect. Omdat deze geen gebruik maken van de python interpreter. Deze programma's kunnen dus wel parallel uitgevoerd worden.

Extra: ontwerp op papier worker threads



De communicatie overhead is hier 6. Dus communicatie = threads * 2. Dit is dus dezelfde communicatie overhead als bij de thread pool. Alleen hier is het aantal threads afhankelijk van het aantal elementen in je lijst. Er worden hier 3 threads aangemaakt. Een thread kan ook een thread aanmaken als die nog taken te verdelen heeft. Als een thread een lijst groter van 2 elementen binnenkrijgt maakt die nog een thread aan.