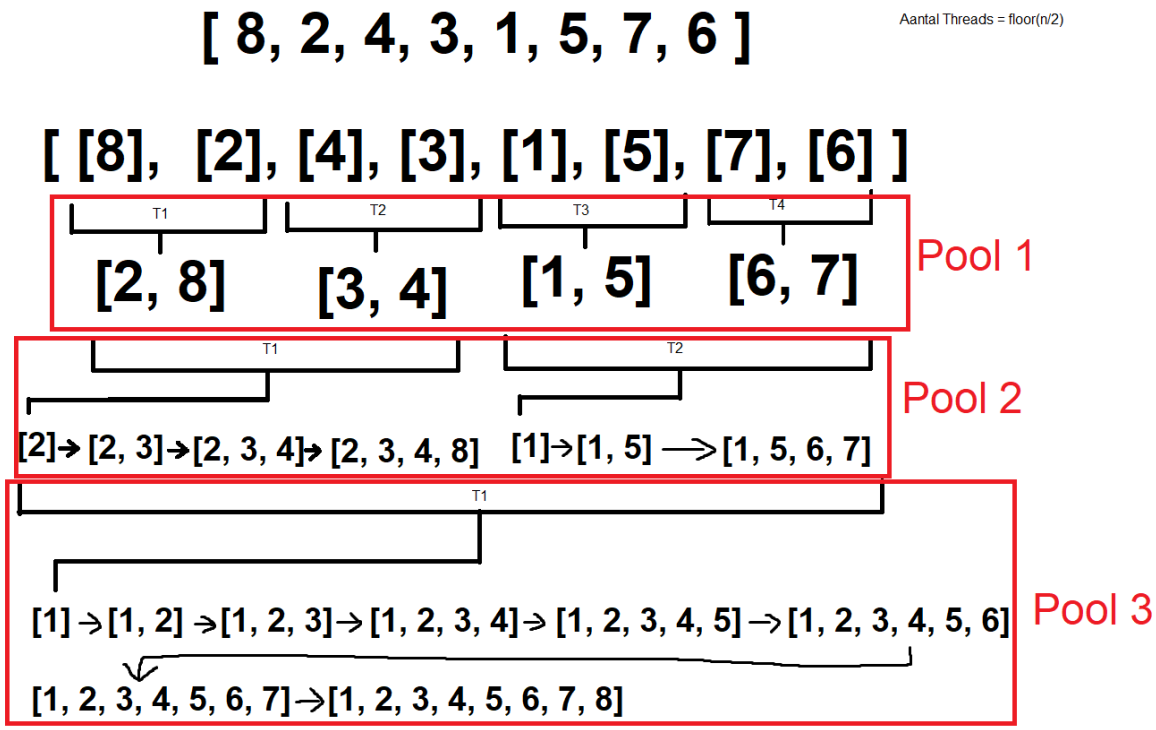
Toelichting

Ontwerp



Mijn ontwerp werkt als volgt:

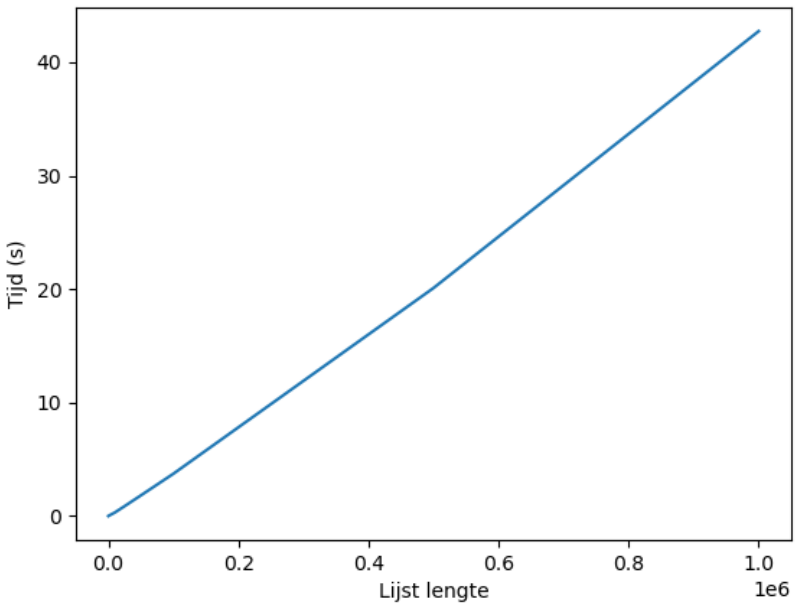
Er wordt een lijst gemaakt van n lang, met willekeurige waarden. Vervolgens wordt elk element in een lijst gezet, omdat mijn functie werkt met lijsten als argumenten, zodat deze functie flexibeler met lijsten kan werken die verschillende groottes hebben.

Als een lijst een oneven aantal elementen heeft, wordt het laatste element bewaard, en op het einde nog gemerged met de eerste, al gemergde, lijst. Dit gebeurt op de main thread.

Elke worker voert een functie uit die elke waarde uit een gesorteerde lijst vergelijkt en bij elkaar toevoegt.

Dit proces wordt herhaald totdat de lijst een lengte heeft van 1. Er worden steeds minder threads gebruik, omdat er minder lijsten te mergen zijn.

Tijdsanalyse



Deze grafiek laat zien hoelang mijn implementatie doet over het sorteren van een lijst met n elementen. Op de x-as staat de lengte van de lijst en op de y-as staat de tijdsduur. Ik heb mijn algoritme getest met de volgende waarden voor n: 10, 100, 1000, 10.000, 25.000, 50.000, 100.000, 500.000, en 1.000.000. Zoals te zien is, neemt de lijn vrijwel lineair toe.

Er is best veel tijdswinst doordat we dit multithreaded doen, des te meer waardes we hebben, hoe meer tijdswinst, omdat we meer processen parallel kunnen draaien. Natuurlijk zit hier een limiet aan; het aantal cores/threads van je processor.

Big O

Merge sort heeft een Big O notatie van O(log(n)), omdat het aantal merges met de helft afneemt.

Dus bij een lijst met 8 elementen:

Eerste iteratie: 4 merges
Tweede iteratie: 2 merges
Derde iteratie: 1 merge