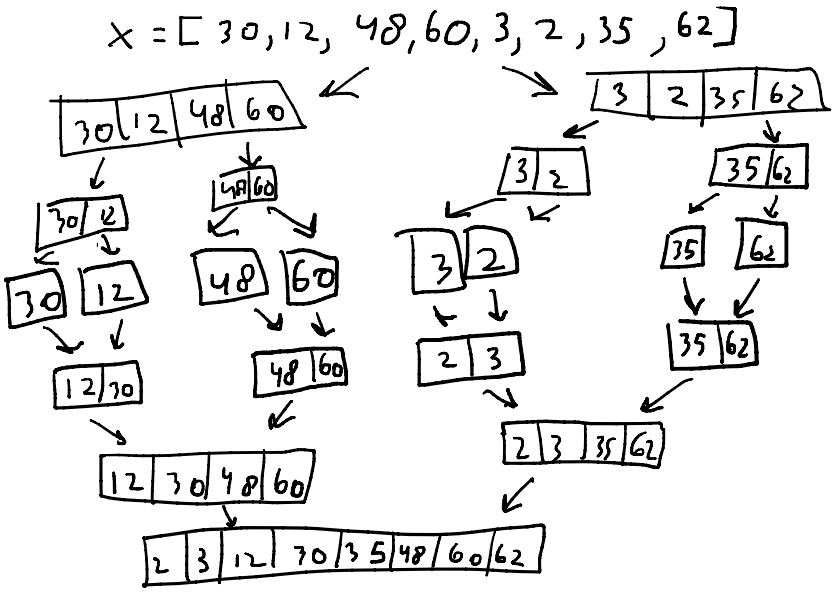
Threaded merge sort

Monday, February 15, 2021 5:05 PM

x = [30, 12, 48, 60, 3, 2, 35, 62]

1 thread :
main

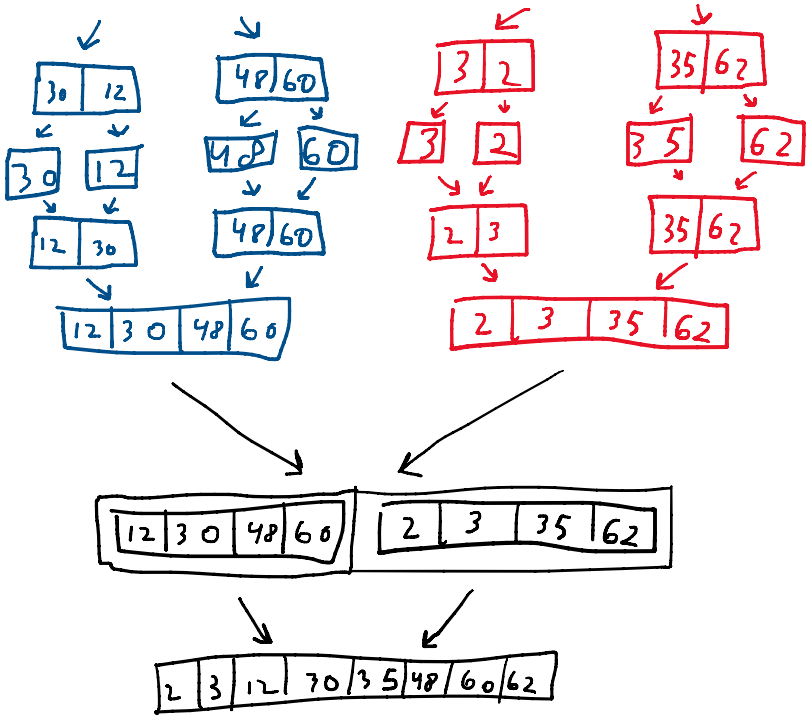


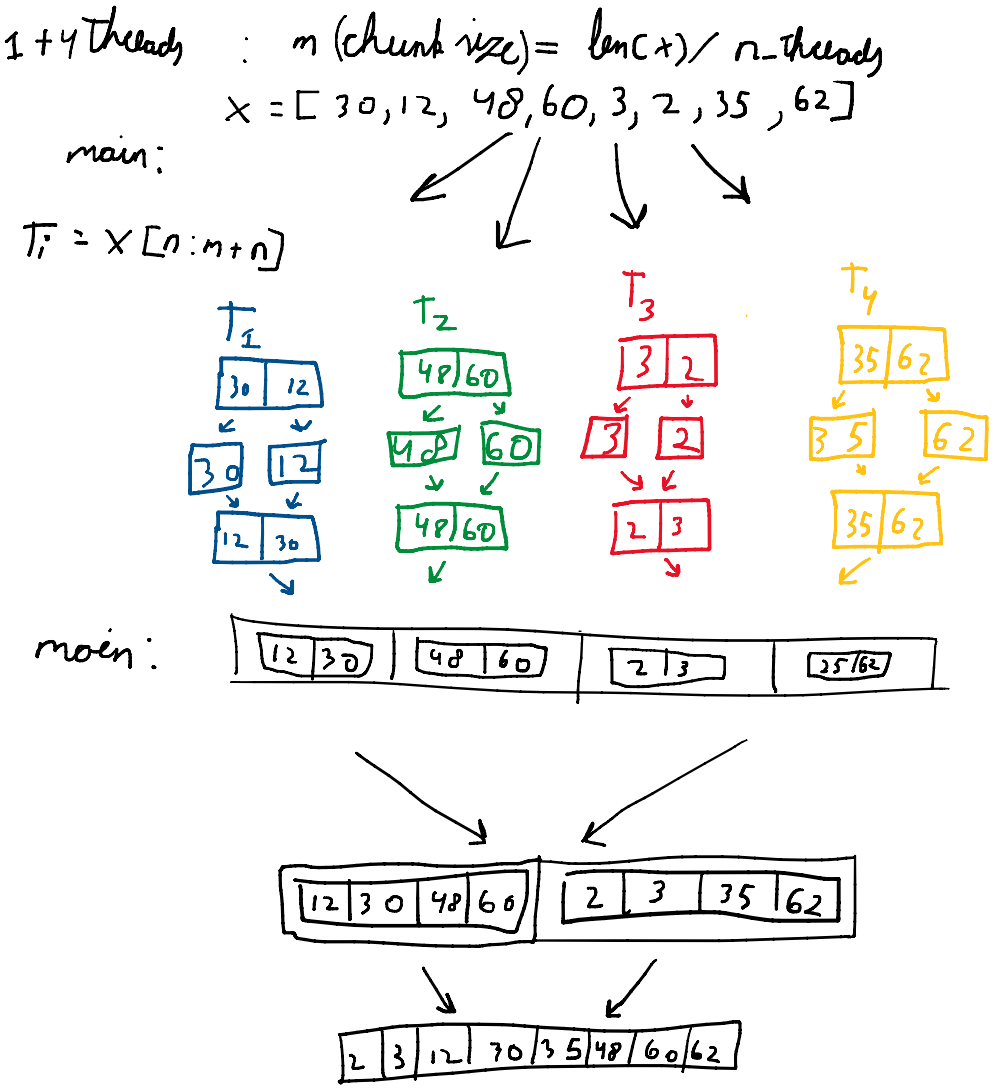
---

1 + 2 threads : n (chunk size) = len(x) / n_threads

x = [30, 12, 48, 60, 3, 2, 35, 62]

main





Stap 2:
Het grote nadeel van deze aanpak is dat wanneer alle (zeg 4) threads klaar zijn alle losse gesorteerde lijsten nog samen gebracht moeten worden.
Dit is iets wat normaal ook moet gebeuren maar dit wordt zoals het programma nu is overgelaten aan de main thread.
Dit zou op te lossen zijn door het aantal gebruikte threads te halveren door .join() te gebruiken.

Het algorithme is wel iets sneller bij het gebruiken van 4 threads, dit verschil is het best zichtbaar bij een lijst van 100.000 lang.
Een verklaring hiervan is dat het algorithme erg memory intensief is. (Threads zijn sequentieel in tegenstelling tot Processes)

Stap 3:
De communicatie overhead is simpel gezegd: 2 * aantalThreads
Dit komt omdat alleen data van en naar de threads extra stappen zijn.
Het splitten is een actie die al zou gebeuren.

Stap 5:
De Global Interpreter Lock heb ik in mijn code niet gebruikt.
Als ik het samen voegen van alle uitvoeren per thread niet naar de main thread had verplaatst maar ook laten samen voegen door 1 thread (respectievelijk) had ik de Lock kunnen gebruiken om te wachten op de uitvoer van een van de andere threads om zo recursief door te gaan.
Zo wordt wanneer alles klaar is het aantal threads door 2 gedaan, dit is efficienter dan alle losse uitvoeren naar de main thread te verplaatsen.