Bucket Sort

Floris Videler

1758374

5-2-2020

Inleiding

Voor het vak high performance programming was de eerste opdracht het maken van een variant op het bucket sort sorteer algoritme en deze analyseren.

Algoritme

Mijn code volgt de pseudo gegeven op canvas. Mijn implantatie verdeelt de lijst eerst onder in twee lijsten, eentje voor de positieve getallen en eentje voor de negatieve getallen. Deze worden daarna apart in mijn recursieve bucket sort functie gegooid.

Analyse

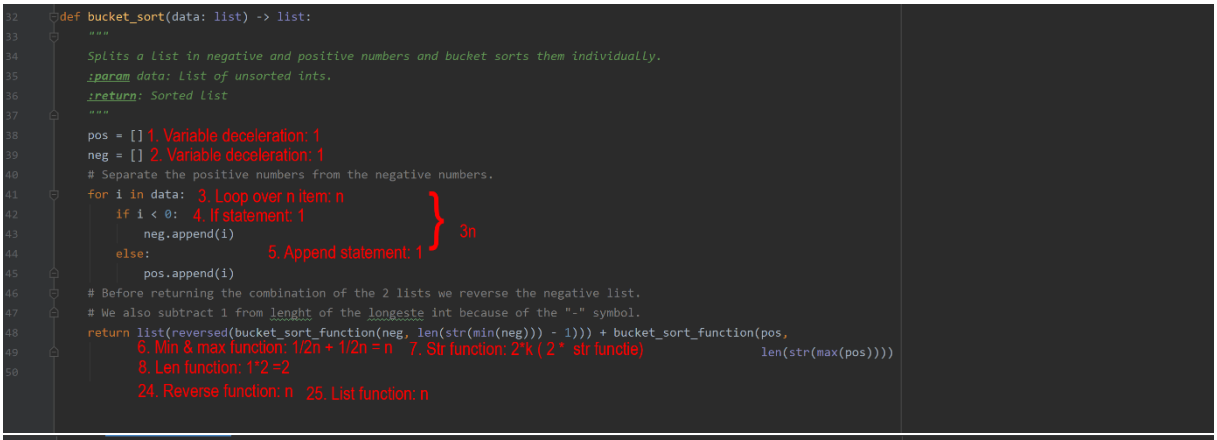
Voor we beginnen met de analyse is het handig om stil te staan bij twee letters en wat deze betekenen:

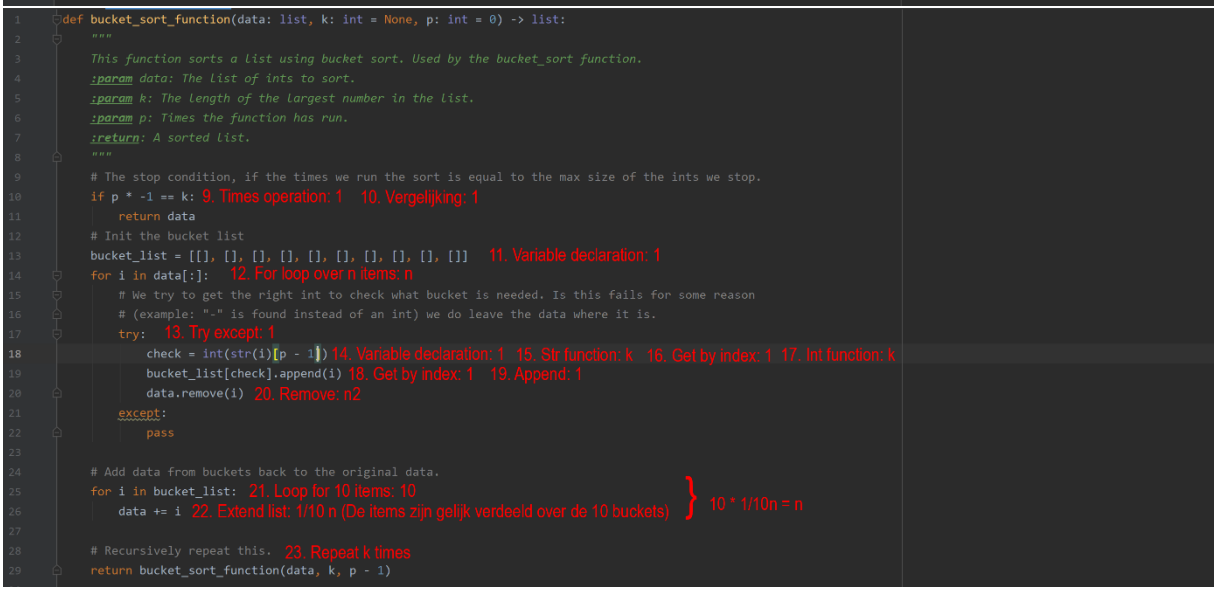
1. *n*: Het aantal elementen in de input lijst.
2. *k*: De lengte van het langste item in de lijst.

In het bestand “*analyze.py*” kan een programmatische analyse gevonden worden die de onderstaande resultaten onderbouwd.

Tijdscomplexiteit

Het analyseren van zelf geschreven code was een stuk lastiger dan gedacht. Hier onder staan twee afbeeldingen, de eerste is de functie is ook wel de “verdeel” functie. De tweede afbeelding is van het algoritme zelf.





Hier onder heb ik complexiteit van iedere stap uitgeschreven:

1 + 1 + 3n + n + 2k + 2 + 1 + k (1 + 1 + n (1 + 1 + k + 1 + k + 1 + 1 + n) + n) + n + n

Vereenvoudigd:

1 + 1 + 6n + 2k + 2 + 1 + k (1 + 1 + 2n (1 + 1 + 2k + 1 +1 + 1 + n))

Weghalen van alle constanten:

$n + k + k\ (\ n\ (\ k + n\ ))$

Haakjes weghalen:

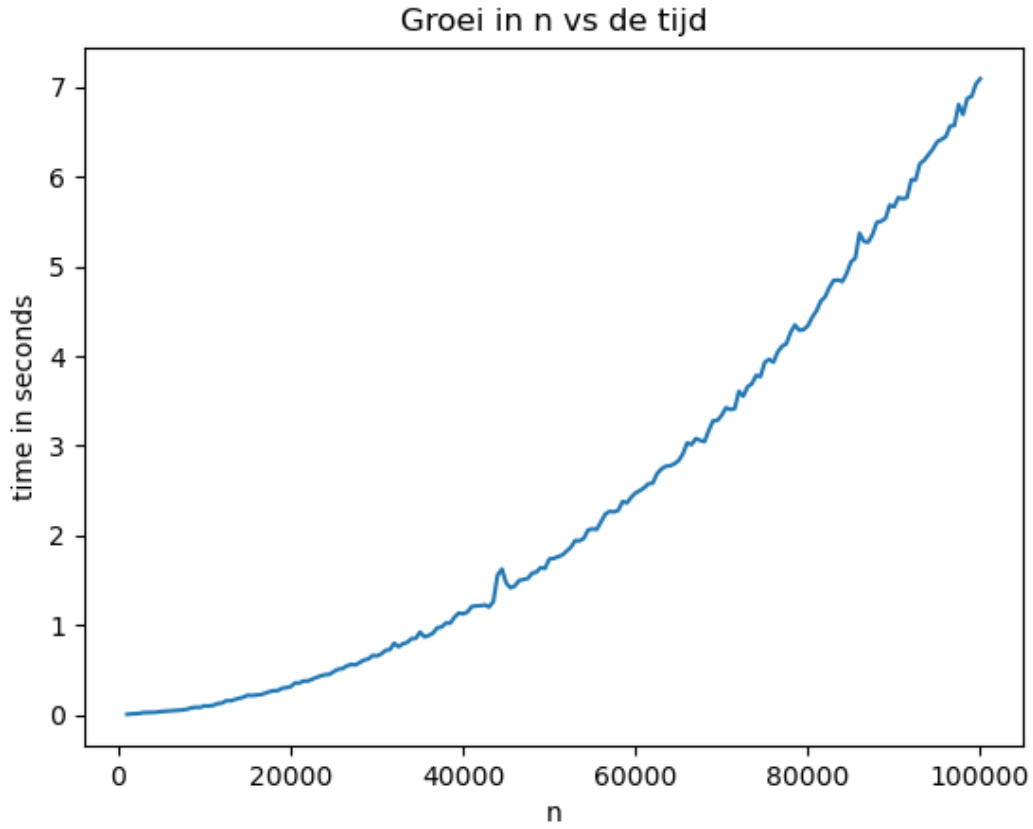
$n + k + k\ (\ n\ (\ k + n\ ))$

$n + k + k\ (\ nk + n^2)$

$n + k + k^2n + kn^2$

Als we hier de dominante term uit willen halen lopen we tegen een probleem aan. k^2n kan groter zijn dan kn^2 afhankelijk van de grote van k en n. Deze kunnen beide groeien tot oneindig. Als er ik toch één moet kiezen, zou ik kiezen voor kn^2. Dit omdat n vaak relevanter bij het bepalen van tijdcomplexiteit en de big O notatie. De big O notatie zou in dit geval dan zijn: $O(kn^2)$.

De onderstaande grafiek bevestigd ook dat de complexiteit inderdaad wel eens exponentieel zou kunnen zijn.



Wat natuurlijk ook effect kan hebben op de snelheid van een algoritme is de input: zijn er dubbele getallen? Is de lijst al gesorteerd? Etc. In het bestand “analyze.py” heb ik hier onderzoek naar gedaan. In de onderstaande afbeelding is hier een korte samenvatting van te zien.

```
Each metric is calculated with a population of 100 and a list length of 75000
Type                          Avg      Min      Max        Std
------------------------  -------  -------  -------  ---------
Bucket sort normal        3.18761  3.02368  3.50576  0.0574777
Bucket sort sorted list   3.18882  3.08348  3.39032  0.0452626
bucket sort reversed list 3.18486  3.04703  3.31774  0.0472281
bucket sort unique values 3.00869  2.91375  3.22659  0.0451142
```

In de tabel wordt voor ieder geval het volgende getoond:

- Gemiddelde
- Kortste tijd
- Langste tijd
- En de standaardafwijking

Op basis van deze gegevens kunnen we de conclusie trekken dat geen speciaal geval echt effect heeft op de performance van het algoritme. Het verschil dat gezien wordt is zeker geen groot verschil en kan ook toeval worden genoemd.

Ruimtecomplexiteit

Over het algemeen wordt er gezegd dat ruimtecomplexiteit minder belangrijk is dan tijd complexiteit. We hebben hier ook een stuk minder uitleg over gekregen. In mijn algoritme heb ik een ruimtecomplexiteit bepaald van $O(kn)$. Dit omdat in het slechtste geval alle elementen in de lijst k keer worden ingedeeld in buckets. Dit zou betekenen dat er k keer nieuwe buckets volledig gevuld zouden moeten worden met n items.

Conclusie en discussie

Terug kijkend om mijn uit uitwerking en analyse hiervan zijn er een aantal punten die genoemd moeten worden:

1. Mijn uitwerking is niet de beste: Ik heb mijn beste gedaan om dit algoritme zo efficiënt mogelijk te maken en ongetwijfeld is er ook een efficiëntere manier. Dit kan dus leiden tot een onjuiste analyse voor bucket sort. Het is dus belangrijk te onthouden dat deze analyse is gemaakt voor mijn versie van het algoritme.
2. Er is zoveel mogelijk meegenomen in het bepalen van de big O: In de les was er al besproken wat er wel en niet meegenomen moet worden in het bepalen van de big O. Tellen variable assignments mee? Tellen normale Python functies mee? Hier is niet één definitief antwoord op. Voor mijn analyse heb ik dit wel allemaal meegenomen. Dit heb ik gedaan om de analyse zo precies mogelijk te laten kloppen met de code die er staat. Een goed voorbeeld hier van is de remove functie voor een Python lijst, deze heeft een complexiteit van $O(n)$. Deze functie in een for loop gebruiken zou de complexiteit al snel verhogen.
3. Hoe bruikbaar is deze implementatie? Mijn specifieke implementatie van deze variatie op bucket sort is zeker niet het beste sorteer algoritme. Er zijn veel algoritmes met een betere tijd en ruimtecomplexiteit. Wel vind ik het belangrijk

om te noemen dat ondanks mijn uitwerking niet de snelste of de beste is, ik wel anders ben gaan kijken naar hoe je algoritmes bouwt en welke aanpak je gebruikt. Het opzoeken van complexiteiten van Python functies heeft mij veel kennis gegeven over wat voor effect input heeft op de uitvoer tijd.