1. ***Schrijf een implementatie van het bovenstaande Bucket Sort algoritme voor het sorteren van gehele getallen (x ∈ Z) .***

Ik heb het algorithme geïmplementeerd, ik heb comments in mijn code toegevoegd maar hier is ter verduidelijking nog meer uitleg van mijn code:

De bucket\_sort methode heeft als parameter ‘data’ die een lijst van getallen kan zijn. Vervolgens maak ik hier 2 lijsten van, 1 met alle positieve getallen en 1 met alle negatieve getallen. Vervolgens sorteer ik de 2 lijsten als volgt:

* Ten eerste return ik als de list leeg is (dit is de aangeraden manier volgens de docs en daarnaast ook de snelste[[1]](#footnote-1))
* Dan bereken ik k die de langste lengte van een nummer in de lijst moet voorstellen
* Daarna bereken ik hoeveel decimale punten er zijn in het langste nummer
* Vervolgens loop ik voor 0 tot en met k - 1 en doe het volgende hier in:
  + Ik maak een nieuwe bucket aan met 10 rijen en met elk een lege list als kolom
  + Ik loop langs elke nummer en doe dit:
    - Ik verkrijg de digit die we willen hebben
    - Ik gebruik deze digit om te indexeren en het nummer toe te voegen aan de bucket
  + Vervolgens zet ik de data naar een list met alle bucket getallen in volgorde om voor bereid te zijn op de volgende loop

Ten slot reverse ik de negatieven getallen en pak ik de negatie daarvan waarbij ik deze lijst vervolgens voor de positives list toevoegt. Daarna zet ik de meegegeven data list naar deze gesorteerde list.

1. ***Benader zo precies mogelijk de de tijds- en ruimtecomplexiteit van je algoritme. (Dus niet m.b.v. asymptotische notatie)***
2. (# 2n^2 + 9n + 16)
3. def bucket\_sort(data):
4. postives, negatives = [x *for* x *in* data *if* x >= 0], [-x *for* x *in* data *if* x < 0] # 1
5. sort\_list(postives), sort\_list(negatives) totaal = # 2n^2 + 8n + 14
6. negatives = list(map(lambda x: -x, reversed(negatives))) # n
7. data[:] = negatives + positives # 1
8. (# n^2 + 4n + 7)
9. def sort\_list(data):
10. *if* not data: # 1
11. *return*
12. k = len(max(map(str, data), key=len)) # n
13. max\_decimal = len(str(min(data)).split('.')[-1]) # n
15. *for* n *in* range(k): # n
16. bucket = [[] *for* y *in* range(10)] # 1
17. *for* number *in* data: # n^2
18. digit = get\_digit(number, max\_decimal, n) totaal = # 4
19. bucket[digit].append(number) # 1
20. data[:] = sum(bucket, []) # n
21. (# 4)
22. def get\_digit(number, max\_decimal, n):
23. *if* type(number) != int: # 1
24. whole\_number\_length = len(str(number).split('.')[0]) + 1 # 1
25. number\_padded = str(number).ljust(whole\_number\_length + max\_decimal, '0') # 1
26. number = int(str(number\_padded).replace('.', '')) # 1
27. *return* number // 10 \*\* n % 10 # 1

Totaal = “2n^2 + 9n + 16” en als we de constante weghalen wordt het “2n^2 + 9n” we rekenen de constante niet mee omdat hoe meer je richting de oneindigheid gaat hoe minder die constante eraan doet.

1. ***Bepaal, net als eerder, op basis van een test en door analyse, de Big O waarde van Bucket Sort.***

De n met de grootste factor is n^2 dit houdt in dat het algoritme een big O notatie heeft van O(n^2)

1. ***Extra: Maak je BucketSort implementatie ook geschikt voor het sorteren van getallen met cijfers achter de komma (x ∈ R )***

Mijn implementatie voldoet hier aan. En ik heb test cases geschreven om dit aan te tonen.

1. ***Extra: Laat bij je analyse zien dat je onderzoek hebt gedaan over wat wel en niet mee te nemen in je analyse (is een variable assignment een stap? hoeveel ruimte kost je datastructuur?)***

Ik heb met veel dingen rekening moeten houden zoals de operatie de we gebruiken op een list, inserten of appenden als voorbeeld. Van dit voorbeeld kwam de conclusie[[2]](#footnote-2) uit dat appenden sneller was dus ben ik voor deze implementatie gegaan. Een variable assignment is een constante die we weg kunnen laten bij de big O notatie. De ruimte die mijn applicatie gebruikt gaat als volgt:

* 2 lists `positives` en `negatives`
* Int waardes voor `k`, `max\_decimal`, `digit`, `whole\_number\_length` en `number`
* 2d list voor de bucket
* En een string voor `number\_padded`

Laten we bij deze berekening van n = 1000 uit gaan waarbij n het aantal elementen is in de meegegeven lijst.

De 2 list kunnen uit alle elementen bestaan of geen een dus laten we een gemiddelde van n / 2 nemen. Integers en floats nemen beide 8 bytes in als je het bij een lijst toevoegt en de lijst object zelf is 72 bytes. Dus 72 + 500 x 8 bytes = 4072 bytes

Er zijn 5 int een alleen staande int waarde is 24 bytes dus 5 x 24 = 120 bytes

Voor de 2d list zijn het 10 lists met daarin nogmaals 10 lists dus 20 x 72 = 1440 bytes

En ten slot de string die is 37 bytes plus 1 byte voor elke karakter, laten we gemiddeld van 10 karakters uit gaan dus 32 + 10 = 42 bytes.

Alles bij elkaar is dus 4072 + 120 + 1440 + 42 = 5674 bytes

*De waardes zijn berekent aan de hand van de bron[[3]](#footnote-3).*

1. ***Extra: Overleg met je docent als je andere ideeën hebt of deze kunnen gelden als extra voor een hoger cijfer***

Ik heb test cases toegevoegd om mijn validiteit van mijn implementatie te onderbouwen. Bij deze test cases pak ik veel nummers met verschillende waardes zodat het niet geluk is dat een test slaat maar we het met enige zekerheid kunnen zeggen.

1. How do I check if a list is empty? (2008, 10 september). Stack Overflow. <https://stackoverflow.com/questions/53513/how-do-i-check-if-a-list-is-empty> [↑](#footnote-ref-1)
2. H. (2019, 27 juli). Performance difference between append and insert in Python. Step-by-step Data Science. <https://h1ros.github.io/posts/performance-difference-between-append-and-insert-in-python/> [↑](#footnote-ref-2)
3. Sayfan, G. (2016, 24 maart). Understand How Much Memory Your Python Objects Use. Code Envato Tuts+. [https://code.tutsplus.com/tutorials/understand-how-much-memory-your-python-objects-use--cms-25609#:%7E:text=Measuring%20the%20Memory%20of%20Python%20Objects&text=Interesting.,An%20integer%20takes%2024%20bytes.&text=Hmm%E2%80%A6%20a%20float%20takes%2024%20bytes%20as%20well.](https://code.tutsplus.com/tutorials/understand-how-much-memory-your-python-objects-use--cms-25609#:~:text=Measuring%20the%20Memory%20of%20Python%20Objects&text=Interesting.,An%20integer%20takes%2024%20bytes.&text=Hmm…%20a%20float%20takes%2024%20bytes%20as%20well.) [↑](#footnote-ref-3)