Vraag 4 - Algoritme analyse Oplossing.py

Een algoritme ontwerpen/maken is voor velen niet zo'n probleem. Echter is het efficient ontwerpen van een algoritme een heel ander verhaal. In deze PDF zal ik kort toelichten hoe ik een algoritme analyseer (volgens het boek Algoritmen en Datastructuren – Veerle Fack).

De code

```
1  dictionary = open("../../dictionary.txt", "r")
2  tupleWithWords = tuple(dictionary)
3  alphabeticalWords = 0
4  for line in tupleWithWords:
5      wordCount = 0
6      while True:
7          if wordCount < len(line[:-2]):
8              if ord(line[wordCount]) > ord(line[wordCount + 1]):
9                  break
10             wordCount = wordCount + 1
11         else:
12             alphabeticalWords = alphabeticalWords + 1
13             break
14 print(alphabeticalWords)
```

Wat we nu gaan doen is het kijken naar de totale uitvoeringstijd van dit programma. Dit doen we door te kijken hoevaak een statement uitgevoerd wordt voor input N. Hieronder in een tabel weergegeven:

Statement	Aantal keer uitgevoerd
1	1
2	1
3	1
4	n
5	n
6	nk
7	nk
8	nk
9	n
10	nk
11	n
12	n
13	n
14	1

Dit kunnen we herschrijven naar functie T.

T(n) = 4 + 6n + 4nk.

Hierbij is ‘n’ de input en ‘k’ de lengte van het woord in karakters. Aangezien K steeds varieert, doe ik het middels een worst-average-best case scenario. Dit geeft dan ook direct weer hoe mijn algoritme in optimale toestand werkt en slechte toestand (gebaseerd op de dictionary).
Big-O Notatie: O(n*k)
Voor meer informatie: http://www.leepoint.net/notes-java/algorithms/big-oh/bigoh.html

Worst case scenario:
K = 27 (langste woord in de dictionary)

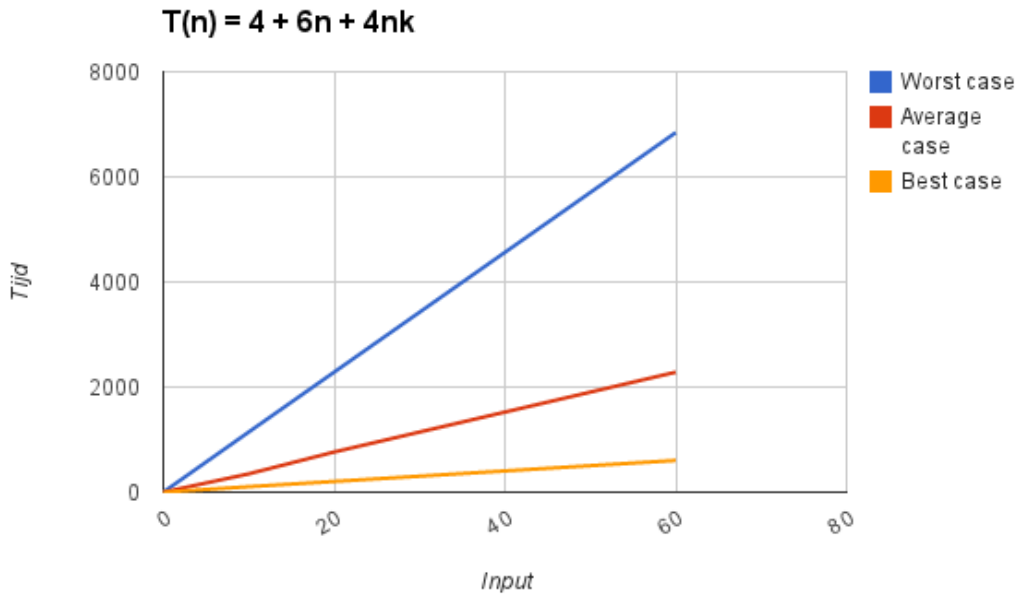
Average case scenario:
Gemiddeld woord in de dictionary is: 8.087530378428422 = 8 karakters.
K = 8

Best case scenario:
K = 1

Hier kunnen we dus 3 formules uit halen:

T(n) = 4 + 6n + 4n*27
T(n) = 4 + 6n + 4n*8
T(n) = 4 + 6n + 4n*1

Als we dit in een grafiek zetten, krijgen we het volgende:



Let op: Tijd is hierbij niet de tijd in sec, minuten, uren o.i.d. Het is enkel een aanwijzing om aan te geven hoe snel het algoritme is in verhouding tot. Oftewel, de worst case scenario (blauw) is dus 11x zo langzaam dan de best case scenario. Echter zijn dit beide uitschieters (1 karakter of 27 karakters).

Op dit moment zegt deze analyse nog niet zo veel. De kracht van een analyse van algoritmen komt pas zodra je twee of meer algoritmen hebt (liefst in pseudocode). Hierdoor kun je in een vroeg stadium al testen welk algoritme uiteindelijk het snelst is/best toepasbaar.

De uitdaging
De uitdaging is dan ook:

"Creeer een ander algoritme dat sneller is dan deze. Analyse mag je zelf doen, eventueel wil ik hem ook voor je analyseren."