1 Interpreteren van code

Oefening 1-1: Bepaal de complexiteit van de functie `main()` in onderstaande code. Hoe kan die verbeterd worden?

```
from random import *

def f(lengte):
  s = [0]*lengte
  for i in range(0, len(s)):
    m = randint(0,lengte)
    j = i
    p = j
    while j > 0:
      if m < s[j-1]:
        s[j] = s[j-1]
        p = j-1
      j-=1
    s[p] = m
  print(s)

def main():
  invoer = int(input("Invoer?"))
  f(invoer)

main()
```

De functie `f` maakt een gesorteerde lijst van willekeurige getallen aan tussen $\{0, \ldots,$ `lengte-1`$\}$. Binnen de `while` lus wordt elke element van de reeds gegenereerde lijst, vergeleken met het nieuw willekeurig gegenereerde element `m`. Indien het nieuwe element kleiner is worden het element van de gegenereerde lijst waarmee vergeleken werd een plaats opgeschoven in de lijst.

In het worst case scenario worden de elementen in omgekeerde volgorde gegenereerd: $[N-1, \ldots, 0]$ met N de `lengte` die werd ingevoerd. Zo zal het aantal iteraties bij de `while` de volgde reeks zijn: $0, 1, 2, \ldots, N-1$. In het slechtste geval bekijken we dus $\frac{N(N-1)}{2}$ keer of `m < s[j-1]`. Een element nemen uit een lijst is van constante complexiteit. Zo wordt de complexiteit van de functie gelijk zijn aan $O(N^2)$.

Door gebruik te maken van de ingebouwde `sort` functie van python die een complexiteit heeft van $O(N \log(N))$ kunnen we de complexiteit tot deze orde krijgen met behulp van volgende code.

```
from random import *

def f(lengte):
  s = []
  for i in range(0, len(s)):
```

```
      s.append(randint(0,lengte))
    s.sort()
    print(s)

  def main():
    invoer = int(input("Invoer?"))
    f(invoer)

  main()
```

Oefening 1-2: Wat doet de functie `gorec`? Wat is de output van `main()`? Wat is de tijdscomplexiteit? Kan je een betere versie schrijven?

```
  def gorec(r,s,mm):
    if s > len(r):
      return mm
    for t in range(len(r)):
      drs = 0
      n = 0
      for x in range(t,len(r),s):
        if r[x] > 0:
          drs += r[x]
          n += 1
      if n > 0:
        m = drs/n
        if m > mm:
          mm = m
    return gorec(r,s+1,mm)

  def main():
    r = [1,6,12,18,24,30,36,42]
    print("r = %50s : %10.2f" %(r, gorec(r,1,0)))
    r = [42,36,30,24,18,12,6,1]
    print("r = %50s : %10.2f" %(r, gorec(r,1,0)))
    r = [1,-12,24,42,-36,30,-18,6]
    print("r = %50s : %10.2f" %(r, gorec(r,1,0)))
```

De functie `gorec` berekent het gemiddelde van deellijsten, van de elementen in de deellijst die groter zijn dan 0. De deellijst van de lijst wordt bepaald door de start index `t` en de stapgrootte `s`. Binnen de functie wordt er gelopen over de start index `t` $\in 0, \ldots,$ `len(r)-1`. Voor elke `t` wordt het gemiddelde berekend. Met een stap grootte van 1 zal de functie dus al zeker kwadratische complexiteit hebben: $O(N^2)$ met N de lengte van lijst `r`.

Recursief wordt deze stapgrootte vergroot, tot bij het laatste recursiestap voor het basisgeval de deellijsten slechts één element bevatten. Aangezien in deze stap het gemiddelde van elk element afzonderlijk berekent wordt, zal de functie uiteindelijk het grootste element teruggeven met als ondergrens de waarde `mm`.

Zoals hierboven vermeld, heeft de eerste recursiestap complexiteit van orde $O(N^2)$. Aangezien het werk voor elke recursie stap kleiner wordt, schrijven we de complexiteit voor de recursie stap als $O(NM)$. Hierbij M afhangt van het recursieniveau l en de lengte van de lijst N: $M = \frac{N}{l}$.

De functie zal recursief N keer opgeroepen worden, dus zal de complexiteit van de functie gelijk zijn aan:

$$C(N) = \sum_{l=1}^{N} O(\frac{N^2}{l}) = O(N^2 \log(N))$$

Dezelfde functionaliteit kan geschreven worden in $O(N)$.

```
1  def gorec(r, mm):
2    r.append(mm)
3    return max(r)
4
5  def main():
6    r = [1,6,12,18,24,30,36,42]
7    print("r = %50s : %10.2f" %(r, gorec(r, 0)))
8    r = [42,36,30,24,18,12,6,1]
9    print("r = %50s : %10.2f" %(r, gorec(r, 0)))
10   r = [1,-12,24,42,-36,30,-18,6]
11   print("r = %50s : %10.2f" %(r, gorec(r, 0)))
```