



## Uitwerkingen

1. Eerst representeren we de getallen in het getalsysteem met basis 4. Dit levert de volgende rij:

22 13 33 01 11 23

We sorteren vervolgens de tweede component (stabiel). Dit levert:

01 11 22 13 33 23

En vervolgens sorteren we de eerste component (stabiel). Dit levert:

01 11 13 22 23 33

2. (Dit is ongeveer opgave R-4.15/R-4.16.)

De ruimtecomplexiteit van bucket sort is  $\mathcal{O}(n + N)$ ; bucket sort is niet in-place. (Bijvoorbeeld voor het sorteren van het lijstje met twee elementen  $[99, 0]$  wordt een array ter grootte 100 aangemaakt.)

3. (Dit is ongeveer opgave C-4.14 uit het boek.)

De vraag is om een rij elementen in  $[0, n^2 - 1]$  ter lengte  $n$  te sorteren met tijdscomplexiteit  $\mathcal{O}(n)$ .

De eerste cruciale observatie is dat we de getallen in  $[0, n^2 - 1]$  in het getalsysteem met basis  $n$  kunnen representeren met lengte ten hoogste 2.

Voorbeeld:  $n = 3$ . De getallen  $[0, 8]$  in het getalsysteem met basis 3 zijn:

00 01 02  
10 11 12  
20 21 22

Voorbeeld:  $n = 10$ . De getallen  $[0, 99]$  in het getalsysteem met basis 10 zijn:

00 01 ... 09  
10 11 ... 19  
20 21 ...  
⋮

Nu voor algemene  $n \in \mathbb{N}$ . De getallen  $[0, n^2 - 1]$  in het getsysteem met basis  $n$  zijn:

$$\begin{array}{rcccc}
 00 & 01 & \dots & 0(n-1) \\
 10 & 11 & \dots & 1(n-1) \\
 20 & 21 & \dots & \\
 \vdots & & & \\
 & & & (n-1)(n-1)
 \end{array}$$

Terug naar het oorsponkelijke probleem. We representeren de getallen in  $[0, n^2 - 1]$  in het getsysteem met basis  $n$ . We sorteren de rij met radix-sort. De complexiteit van radix-sort is in het algemeen  $O(d(n + N))$ , met  $d$  het aantal dimensies,  $n$  de lengte van de te sorteren rij, en  $N$  een bovengrens op de te sorteren getallen. Voor onze nieuwe representatie van het sorteerprobleem geldt  $d = 2$  en  $N = n$ . Dus we kunnen de rij sorteren met tijdscomplexiteit  $O(4n)$  oftewel  $O(n)$ .

#### 4. Informele beschrijving:

Input:  $n$  tupels ter lengte  $k$ , dus elk van de vorm  $(p_{i1}, \dots, p_{ik})$ , met elke  $p_{ij}$  in het interval  $[0, N]$ .

Stop eerst alles in bucket  $B_{? \dots ?}$  met annotatie  $k$  keer een vraagteken. Dan sorteren we op de eerste component: een element uit  $B_{? \dots ?}$  van de vorm  $(i, \dots)$  wordt toegevoegd aan bucket  $B_{i? \dots ?}$ , nu met  $k - 1$  keer een vraagteken. Elke bucket  $B_{i? \dots ?}$  met twee of meer elementen wordt vervolgens gesorteerd door een element van de vorm  $(i, j, \dots)$  toe te voegen aan bucket  $B_{ij? \dots ?}$  nu met  $k - 2$  vraagtekens. Etcetera.

In het beste geval, als alle eerste componenten verschillend zijn, is dit algoritme in  $O(n + N)$ . In het slechtste geval, als de verschillen pas in de laatste dimensei  $d$  zitten, is dit algoritme in  $O(d(n + N))$ .

Voorbeeld: sorteer  $(1, 3, 2), (1, 3, 3), (1, 6, 9), (2, 3, 4)(5, 6, 10)$ . Eerst dimensie 1:

$$\begin{array}{l}
 (1, 3, 2), (1, 3, 3), (1, 6, 9) \\
 (2, 3, 4) \\
 (5, 6, 10)
 \end{array}$$

Dan dimensie 2:

$$\begin{array}{l}
 (1, 3, 2), (1, 3, 3) \\
 (1, 6, 9) \\
 (2, 3, 4) \\
 (5, 6, 10)
 \end{array}$$

Dan dimensie 3:

(1, 3, 2)  
(1, 3, 3)  
(1, 6, 9)  
(2, 3, 4)  
(5.6.10)

Dus output: (1, 3, 2), (1, 3, 3), (1, 6, 9), (2, 3, 4), (5, 6, 10).

5. (Dit is ongeveer opgave C-4.16.)

We gebruiken bucket-sort met  $N$  buckets. We initialiseren elke bucket met de lege lijst.

Eerst lopen we door  $S_1, \dots, S_k$  heen om alle elementen een label te geven, namelijk label  $i$  als het element in  $S_i$  zit. Dit is in  $\mathcal{O}(n)$ .

Dan lopen we door  $S_1, \dots, S_k$  heen om de elementen in de buckets te stoppen, namelijk een element  $n$  komt in bucket  $n$  (zo'n element heeft nu dus een extra label om aan te geven uit welke  $S_i$  het afkomstig is. Dit is in  $\mathcal{O}(n)$ .

Vervolgens lopen we door de bucket-array heen. Als bucket  $i$  het element  $i$  met label  $j$  bevat, voeg je  $i$  achteraan toe in het outputrijtje  $S_j$ . Dit is in  $\mathcal{O}(N)$ .

Totaal dus in  $\mathcal{O}(n + N)$ .

Of: Als  $n$  voorkomt in  $S_i$ , dan voegen we aan de lijst in bucket  $n$  het element  $i$  toe. Voor het uitlezen van het resultaat: we lopen door de bucket heen. We bekijken voor elke  $n$  de bucket  $n$ ; elk element  $i$  in de lijst daar leidt tot het toevoegen van  $n$  achteraan in de output-rij voor  $S_i$ .

6. Idee: Doe stabiele bucket sort op  $m$ , dan  $l$ , dan  $k$ . Met  $d$  dimensies: begin bij dimensie  $d$  met stabiele bucket sort, dan downto dimensie 1.

7. Voor wat betreft de worst-case tijdscomplexiteit van quick-select: bij steeds maar weer een slechte pivot, die de rij splitst in een lege rij, en eentje met bijna de lengte van de oorspronkelijke rij, krijgen we  $n$  lagen in de quickSelect-boom. Het werk per laag is van orde  $n$ . Totaal is het algoritme dan in  $\mathcal{O}(n^2)$ .

Of met recurrente betrekking:

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-1) + n - 1 + n \\ &= 1 + 2 + \dots + (n-1) + n \end{aligned}$$

Dit levert  $T(n) \in \mathcal{O}(n^2)$ .

Deze grens wordt ook echt bereikt bij een worst-case input zoals bijvoorbeeld zoek 1 in rijtje [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] met als pivot steeds de laatste.

Overigens: best-case tijdscomplexiteit van quick select is in  $\mathcal{O}(n)$ : we vinden direct na de eerste keer partitioneren wat we zoeken.

8. Verwachte tijdscomplexiteit van quick-select, zoals in het boek p247. We schrijven  $T(n)$  voor de verwachte running time van quick-select bij een input ter lengte  $n$ . Net als bij de analyse van randomized quick-sort noemen we een pivot *goed* als de recursieve call een input heeft ter lengte  $\frac{3}{4}$  van de lengte van de input ervoor. De kans op een goede pivot is  $\frac{1}{2}$ . We hebben dus gemiddeld twee recursieve calls nodig voor we weer een goede call hebben. Met  $b$  als constante factor voor de overhead bij elke call levert dit:

$$T(n) \leq 2bn + T\left(\frac{3}{4}n\right)$$

We werken deze recurrente betekking verder uit:

$$\begin{aligned} T(n) &= 2bn + T\left(\frac{3}{4}n\right) \\ &= 2bn + 2b\frac{3}{4}n + T\left(\frac{3^2}{4^2}n\right) \\ &= 2bn + 2b\frac{3}{4}n + 2b\frac{3^2}{4^2}n + T\left(\frac{3^3}{4^3}n\right) \\ &= \dots \\ &= T\left(\frac{3^i}{4^i}n\right) + 2b\frac{3^{i-1}}{4^{i-1}}n + \dots + 2b\frac{3^2}{4^2}n + 2b\frac{3}{4}n + 2bn \end{aligned}$$

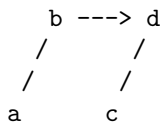
De basiswaarde wordt bereikt bij  $\left(\frac{3}{4}\right)^i n = 1$ , dus  $i = \log_{\frac{4}{3}} n$ . Dus

$$T(n) = \sum_{i=0}^{i=\lceil \log_{\frac{4}{3}} n \rceil} \left(\frac{3}{4}\right)^i$$

Met toepassen van stelling 1.12 uit het boek volgt nu (deze stap hoef je op het tentamen niet zonder hints te kunnen uitvoeren) dat  $T(n) \in \mathcal{O}(n)$ .

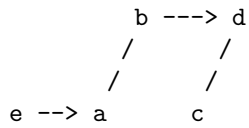
9. Extra opgave (van Knuth, ter illustratie van de ondergrens van  $n \log(n)$  voor comparison-based sorteren): sorteer 5 elementen, zeg  $a, b, c, d, e$  met ten hoogste 7 vergelijkingen. NB:  $\lceil \log(5!) \rceil = 7$ .

Eerst bepalen we met twee vergelijkingen de volgorde van  $a, b$ , en  $c, d$ . Zeg dat dit oplevert  $a < b$  en  $c < d$ . Met een derde vergelijking bepalen we de volgorde van  $b, d$ , zeg dat dit oplevert  $b < d$ . Dan zijn we in de volgende situatie, met pijltje of naar boven voor  $<$ :

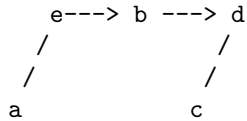


Vervolgens vergelijken we  $e, b$ . Als  $e < b$  dan vergelijken we  $a, e$ . Als  $b < e$  dan vergelijken we  $e, d$ . Nu zitten we totaal op 5 vergelijkingen, en er zijn nu 4 mogelijkheden:

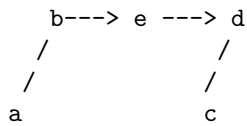
- (1) Als  $e < b$  en  $e < a$  dan:



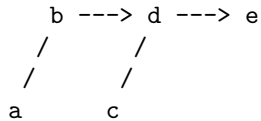
(2) Als  $e < b$  en  $a < e$  dan:



(3) Als  $b < e$  en  $e < d$  dan:



(4) Als  $b < e$  en  $d < e$  dan:



We mogen nog twee vergelijkingen gebruiken. Dit is in alle vier de gevallen genoeg om de plaats van  $c$  in het rijtje te bepalen. Voorbeeld van het eerste geval: Vergelijk  $c, a$ . Als  $c < a$ , vergelijk dan  $c, e$ . Als  $a < c$ , vergelijk dan  $c, b$ . Dit levert vier gevallen:

Als  $c < a$  en  $c < e$  dan:

$$c < e < a < b < d$$

Als  $c < a$  en  $e < c$  dan:

$$e < c < a < b < d$$

Als  $a < c$  en  $c < b$  dan:

$$e < a < c < b < d$$

Als  $a < c$  en  $b < c$  dan:

$$e < a < b < c < d$$

De gevallen (2), (3), (4) hierboven kunnen ook met twee vergelijkingen op analoge manier opgelost worden.