# Complexiteit

## Uitwerking opgave 3 (ongeordend lineair zoeken)

**a.** Het is duidelijk dat de test $A[index] \neq X$ (test (2.2)) essentieel is voor het algoritme. We tonen nu (misschien enigszins overdreven in dit geval) aan dat deze operatie maatgevend is voor de complexiteit.
Duidelijk: $\#(1) = 1 = \#(5)$; $\#(6) \leq 1$; $\#(8) \leq 1$; $\#(3) \leq \#(2.2)$;
Verder: als test (2.1) True oplevert, wordt test (2.2) ook gedaan, als deze False is (index > n) niet, maar dan is het algoritme meteen afgelopen. Dus $\#(2.1) = \#(2.2) + 1$. Omdat $n \geq 1$ wordt (2.2) ten minste 1 keer gedaan (*), waarmee (2.1) in orde van grootte (hooguit) even vaak als (2.2) wordt uitgevoerd. Immers: $\#(2.1) = \#(2.2) + 1 \leq 2 \cdot \#(2.2)$. En overigens zijn vanwege (*) ook $\#(1)$ , $\#(5)$, $\#(6)$ en $\#(8) \leq \#(2.2)$.
**b.** Best case: komt voor dan en slechts dan als (d.e.s.d.a.) meteen, voor $index = 1$, de while-loop stopt, dus d.e.s.d.a. $A[1] = X$.
**c.** Worst case komt voor als $A[index] \neq X$ zo vaak mogelijk (en dat is $n$ keer) wordt uitgevoerd, en dat komt voor d.e.s.d.a. $A[index] \neq X$ True is voor alle $index < n$ en de test $A[n] \neq X$ dus nog plaatsvindt. De uitslag maakt dan niet uit, want het algoritme stopt daarna sowieso. Dus worst case is $n$ sleutelvergelijkingen ( $A[index] \neq X$) en dit komt voor $\Longleftrightarrow$ $X$ staat op plek $n$ of $X$ komt niet voor.
**d.** Stel dat de kans dat $X$ in het array voorkomt $q$ is, dan is de kans dat $X$ niet voorkomt $1-q$. Verder nemen we aan dat alle elementen verschillend zijn en veronderstellen we dat als $X$ in $A$ voorkomt, de kans dat $X$ op plek $i$ staat voor elke $i$ even groot is. Die kans is dan dus $\frac{1}{n}$.
- als $X$ niet in $A$ voorkomt doet het algoritme altijd $n$ sleutelvergelijkingen.
- als $X$ op positie $i$ in het array staat, kost het precies $i$ vergelijkingen om dit te ontdekken.
Dat betekent dat het, als gegeven is dat $X$ ergens in het array voorkomt, gemiddeld $\frac{1}{n} \cdot 1 + \frac{1}{n} \cdot 2 + \frac{1}{n} \cdot 3 + \cdots + \frac{1}{n} \cdot n = \frac{1}{n} \cdot (1 + 2 + 3 + \cdots + n) = \frac{1}{n} \cdot \frac{1}{2} \cdot n \cdot (n+1) = \frac{1}{2} \cdot (n+1)$ vergelijkingen kost.
- als we beide gevallen samen nemen is het gemiddeld aantal vergelijkingen dus

$$q \cdot \frac{1}{2} \cdot (n+1) + (1-q) \cdot n$$

.
**e.** We moeten bewijzen: *elk* algoritme voor het zoekprobleem dat is gebaseerd op het doen van sleutelvergelijkingen doet in de worst case ten minste $n$ sleutelvergelijkingen. Zo'n algoritme werkt voor elk mogelijk invoerarray, en geeft daarop –uiteraard– het correcte antwoord.
We bewijzen de bewering via een bewijs uit het ongerijmde.
Dus: veronderstel dat de stelling niet waar is; dan is er dus een algoritme voor het zoekprobleem dat in de worst case hooguit $n-1$ vergelijkingen doet, en dus doet dat algoritme voor *elke* invoer $n-1$ vergelijkingen of minder. We laten zien dat dit tot een tegenspraak leidt door 2 invoeren te geven waarop het algoritme ten onrechte hetzelfde antwoord geeft.

Bekijk als invoer (noem die even invoer1) een ongesorteerd array $A$, bestaande uit $n$ verschillende waardes, en een $X$ die niet in het array zit. Het algoritme zal op deze invoer

dus antwoord $-1$ geven (hetgeen betekent dat $X$ niet in $A$ zit). Aangezien dit algoritme volgens de aanname hooguit $n-1$ sleutelvergelijkingen doet, zal ten minste één arrayelement niet zijn bekeken; zeg dat dit $A[j]$ is. Bekijk nu een invoerarray $B$ dat precies gelijk is aan invoer1, behalve dat nu op plek $j$ wel $X$ staat: $B[i] = A[i]$ voor alle $i \neq j$ en $B[j] = X$. Laat nu het algoritme los op $B$. Op deze invoer doet het algoritme precies hetzelfde als op invoer1. (Extra uitleg: aangezien $A[j]$ nooit bekeken werd en alle andere $B[i]$'s gelijk zijn aan de $A[i]$'s en $X$ hetzelfde is als voorheen, zijn de antwoorden op de gestelde vragen (sleutelvergelijkingen) hetzelfde en worden ook dezelfde sleutelvergelijkingen gedaan als voor invoer1[1]. ) Het algoritme zal dus ook op deze invoer antwoord $-1$ geven. Dat is in dit geval fout, omdat $X$ wel degelijk in $B$ voorkomt. Dit is in tegenspraak met de correctheid van ons algoritme. Ergo, de aanname was verkeerd, en dus geldt: elk algoritme voor het zoekprobleem doet in de worst case ten minste $n$ sleutelvergelijkingen. QED.

Samengevat, als we aannemen dat er een algoritme bestaat dat altijd $\leq n-1$ sleutelvergelijkingen doet, kunnen we altijd twee verschillende invoeren construeren waarop het algoritme ten onrechte hetzelfde antwoord geeft. Aangezien dit in tegenspraak is met het feit dat het algoritme voor elke invoer correct werkt, was de aanname verkeerd. En klaar.

Opmerking: we hebben in het bewijs gebruikt dat we helemaal niets weten over het array. Je kunt alleen informatie over een arrayelement krijgen door dat element rechtstreeks te vergelijken met $X$. Als je een element niet vergeleken hebt weet je er ook niets van. Het bewijs gaat bijvoorbeeld niet op voor arrays waarvan we weten dat ze gesorteerd zijn, dus waarvan we de ordening kennen. In dat geval hoef je een array-element $A[\ell]$ niet direct met $X$ vergeleken te hebben om te kunnen concluderen dat $A[\ell] \neq X$. Voor willekeurige arrays is dat wel het geval.

[1]Stel het algoritme vraagt of $B[i] = X$; dat is net als bij invoer1 ($= A$) niet het geval; dus gaat het algoritme op $B$ precies hetzelfde verder als op $A$; dit gaat herhaald zo door. Het algoritme zal in het bijzonder dan ook nooit $B[j]$ bekijken