Complexiteit 2025 - Huiswerk 1

a. (i) rijtje $3, 2, 3, 4, 1$:

$i = 1$; $j = 3$; 2 vergelijkingen
$i = 2$: $j = 6$; 5 vergelijkingen
$i = 3$: $j = 3$; 2 vergelijkingen
$i = 4$: $j = 2$; 1 vergelijking

Totaal dus 10 arrayvergelijkingen

(ii) rijtje $3, 2, 3, 1, 1$:

$i = 1$; $j = 3$; 2 vergelijkingen
$i = 2$: $j = 5$; 4 vergelijkingen
$i = 3$: $j = 3$; 2 vergelijkingen
$i = 4$: $j = 6$; 5 vergelijkingen

Totaal dus 13 arrayvergelijkingen

b. (α) (meest linker) kleinste waarde op positie $< n$
Voor de kleinste waarde $A[\ell]$ met $\ell < n$ geldt dat deze sowieso met alle array-elementen $A[j], j = 1, \ldots, n$ vergeleken zal worden, want `kleiner` blijft gelijk aan 0. Dus alleen al voor dat element worden n arrayvergelijkingen gedaan.
(β) kleinste waarde is uniek en staat op plek n
Merk op dat er altijd (voor $i = 1, \ldots, n-1$) een element $A[j]$ bestaat dat kleiner is (namelijk $A[n]$ of een eerder element), en dat dus `gevonden` nooit op True zal worden gezet. Kortom, `gevonden` blijft altijd False, dus de buitenste while wordt voor elke $i = 1, \ldots, n-1$ uitgevoerd. Voor elke i wordt de vergelijking op regel (5) ten minste één keer gedaan, aangezien direct na regel (3) $j < n$ en `kleiner` = `0` geldt: dus minstens $n - 1$ arrayvergelijkingen.
De arrayvergelijking uit (5) wordt voor $i = 1$ zelfs altijd ten minste 2 keer gedaan, aangezien $A[1] < A[1]$ natuurlijk False is en de twee condities uit regel (4) voor $j = 2$ vervolgens beide True zijn. Dat betekent dat de arrayvergelijking ook in dit geval minstens $n - 1 + 1 = n$ keer wordt uitgevoerd.

Alternatief: de kleinste waarde is uniek, dus er is minstens één arrayelement dat de op een na kleinste waarde bevat. Voor die een na kleinste waarde alleen al doet het algoritme n arrayvergelijkingen, omdat het pas een kleiner element tegenkomt als $j = n$. Dus het algoritme doet altijd ten minste n arrayvergelijkingen.

c. We schatten het aantal keer dat een regel gebeurt af op het aantal keer dat de arrayvergelijking op regel (5) gebeurt.

$\#(12), \#(13) \leq \#(11) = \#(1) = 1 \leq \#(5)$, want $\#(5) \geq n \geq 1$.
$\#(6) \leq \#(5)$, $\#(7) = \#(5)$.
De eerste test in regel (2) is hooguit $n - 1$ keer True, en wordt dus maximaal n keer gedaan; voor de tweede test geldt dat die hooguit $n - 1$ keer wordt gedaan, namelijk niet meer als de eerste test False is. Derhalve is $\#(2) \leq n \leq \#(5)$, waar we de dubbele test voor het gemak als één test hebben geteld.

Per doorgang door de buitenste while geldt:
$\#(5) \geq 1$, want direct na regel (3) is ($j \leq n$ **and** `kleiner` $=$ `0`) True,
en dan is $\#(9) \leq \#(8) = \#(3) = \#(10) = 1 \leq \#(5)$.
Dus ook in totaal geldt: $\#(3), \#(8), \#(9), \#(10) \leq \#(5)$.

Nu nog regel (4).
Per doorgang door de buitenste while geldt:
De test ($j \leq n$ **and** `kleiner` $=$ `0`) uit regel (4) wordt hooguit één keer vaker gedaan dan de test uit regel (5). (Immers, als die test True is wordt de arrayvergelijking gedaan; en als die test False is gaan we naar de volgende i, dus de volgende doorgang, of stopt het hele algoritme.) Dat betekent - want er zijn maximaal $n-1$ doorgangen - dat in totaal $\#(4) \leq \#(5)+n-1 \leq 2 \cdot \#(5)$, dus in orde van grootte wordt de test uit regel (4) hooguit even vaak gedaan als regel (5).

d. (i) Stel dat de (meest linker) kleinste waarde uit A op een positie $1 < i_0 < n$ staat. Dan doet het algoritme voor $i = i_0$ precies n arrayvergelijkingen, en stopt daarna meteen. Echter voor $1 \leq i < i_0$ doet het algoritme nog minstens 2 vergelijkingen, want voor $i = 1$ moeten al minstens 2 vergelijkingen gedaan worden. Dus in totaal altijd $\geq n + 2$ arrayvergelijkingen.
(ii) De kleinste waarde staat op positie[1] $1 < i_0 < n$ en $A[i_0] < A[i_0 - 1] < \cdots < A[3] < A[2] < A[1]$. De doorgang met $i = i_0$ levert precies n arrayvergelijkingen op en daarna stopt het algoritme. Voor $i = 1, \ldots, i_0 - 1$ doet het algoritme $i + 1$ vergelijkingen, omdat pas op positie $i + 1$ een element staat dat kleiner is dan $A[i]$. In totaal worden dus $2 + 3 + 4 + \cdots + i_0 - 1 + i_0 + n = 1 + 2 + 3 + 4 + \cdots + i_0 - 1 + i_0 + n - 1 = \frac{1}{2} i_0(i_0 + 1) + n - 1$ arrayvergelijkingen gedaan.

e. (i) De kleinste waarde is uniek en staat op plek n. Dat betekent dat `gevonden` voor $i = 1, 2, \ldots, n - 1$ False blijft en de buitenste while dus voor elke i wordt gedaan.
Het minimale aantal arrayvergelijkingen komt dus voor d.e.s.d.a. voor elke i het minimale aantal arrayvergelijkingen wordt gedaan. Voor $i = 1$ is dat 2, voor $i = 2, \ldots, n-1$ is dat 1; dit zou samen n worden. Echter, dit aantal komt alleen voor als geldt: $A[1] > A[2]$ en $A[1] < A[i]$ voor elke $i = 2, \ldots, n - 1$. Dus in het bijzonder zou moeten gelden dat $A[1] > A[2]$ en $A[1] < A[2]$, hetgeen onmogelijk is. Conclusie: n vergelijkingen is niet haalbaar.
Opmerking: voor $n = 2$ is n vergelijkingen wel haalbaar, namelijk dan wordt de buitenste while alleen voor $i = 1$ gedaan en doet het algoritme altijd 2 arrayvergelijkingen, onafhankelijk van de waardes van $A[1]$ en $A[2]$.
(ii) We bekijken nu wat het best case aantal arrayvergelijkingen dan wél is.
Het beste geval komt voor d.e.s.d.a. `kleiner` zo snel mogelijk > 0 wordt voor elke i, dus als je steeds zo snel mogelijk een waarde $< A[i]$ tegenkomt; bij voorkeur meteen.
Stel dat de op een na kleinste waarde op positie $1 \leq k < n$ staat (en misschien nog vaker voorkomt rechts van k). Dan geldt dat $A[k] < A[1]$, $A[k] < A[2]$, $\ldots$, $A[k] < A[k-1]$ en $A[k] \leq A[k+1]$, $A[k] \leq A[k+2]$, $\ldots$, $A[k] \leq A[n-1]$ en $A[k] > A[n]$. Voor $i = k$ worden in elk geval n vergelijkingen gedaan omdat $A[n]$ de eerste (en enige) waarde is die kleiner is dan $A[k]$. Voor elk volgend voorkomen van $A[k]$ doe je ook $n > 2$ arrayvergelijkingen. Voor elke andere $A[i]$ is dat minstens 1 (of 2 als $i = 1$) en maximaal $k < n$, dus echt minder dan als $A[i]$ gelijk is aan $A[k]$, de op een na kleinste waarde. In het gunstigste geval komt $A[k]$ dus maar 1 keer voor.
We gaan er nu vanuit dat $A[k]$ uniek is. Onderscheid dan de gevallen $k = 1$ en $k > 1$. Als $k = 1$ doe je minimaal $n + 1 + 1 + \cdots + 1 = n + n - 2 = 2n - 2$ arrayvergelijkingen. Dit aantal wordt ook gehaald, want voor elke $i = 2, \ldots, n - 1$ is meteen $A[1] < A[i]$. In alle andere gevallen doe je sowieso al ten minste 2 $(i = 1) + n - 3 + n = 2n - 1$ arrayvergelijkingen. Kortom, in

[1] omdat i in het algoritme een lopende variabele is, noemen we de eerste plek vanaf links waar de kleinste waarde staat even i_0 in plaats van i

het beste geval worden $2n-2$ arrayvergelijkingen gedaan, en dat komt voor d.e.s.d.a. $A[1]$ de unieke op een na kleinste waarde is.

f. De kleinste waarde is uniek en staat op plek n. Dat betekent dat `gevonden` voor $i = 1, 2, \ldots, n-1$ False blijft en de buitenste while dus voor elke i wordt gedaan.
Het slechtste geval komt voor d.e.s.d.a. `kleiner` voor elke $i = 1, \ldots, n-1$ steeds zo lang mogelijk nul blijft, dat wil zeggen d.e.s.d.a. voor elke i: $A[i] \leq A[j]$ voor $j = 1, \ldots, n-1$. Dit betekent dat $A[1] \leq A[2], A[3], \ldots, A[n-1]$ en $A[2] \leq A[1], A[2], \ldots, A[n-1]$ en $\cdots$ en $A[n-1] \leq A[1], A[2], \ldots, A[n-1]$, en dat geldt d.e.s.d.a. $A[1] = A[2] = A[3] = \cdots = A[n-2] = A[n-1]$. In dit geval worden voor elke $i = 1, \ldots, n-1$ precies n vergelijkingen gedaan, dus in totaal $n(n-1)$ arrayvergelijkingen.
Voorbeeld: $6, 6, 6, 6, 6, 6, 6, 2$.