Løsningsforslag teoriøving 4

Du måtte klare 10/16 oppgaver for å få øvingen godkjent. Det beste av to forsøk er tellende. Finner du feil, mangler eller forbedringer, ta gjerne kontkat!

Oppgave 1
Sammenlikningsbasert sortering: $\Omega(n \log n)$ er worst-case for sammenlikningsbasert sortering.

Oppgave 2
Sammenlikningsbasert sortering: Korrekte svar er «S kan ha worst case-kjøretid $\Theta(n^2)$.» og «S kan ha best case-kjøretid $\Theta(n)$». Et eksempel på en sammenlikningsbasert sorteringsalgoritme som oppfyller disse to kriteriene er insertion-sort. Det er viktig å merke seg at $\Omega(n \log n)$ kun gjelder for worst-case og ikke best-case. Videre er ikke Bucket-Sort en sammenlikningsbasert algoritme, og vi kan heller ikke telle forekomster i tabellen for å oppnå kjøretid på $\Theta(n)$ siden dette ikke er sammenlikningbasert.

Oppgave 3
Stabil sortering: [(x=1, y=5), (x=3, y=5), (x=3, y=3), (x=3, y=4)]. Sorter først på x-verdi, ved likhet på x-verdi sorter basert på hvilket element som står først i original listen.

Oppgave 4
Tellesortering: $\Theta(k)$. Fra oppgaveteksten leser vi at $n = \mathrm{O}(k)$. Ergo får vi for tellesortering at $\Theta(n + k) = \Theta(k)$.

Oppgave 5
Tellesortering: «Fordi algoritmen skal være stabil.». Hadde vi gått fra starten av A hadde elementer med lik verdi dukket opp i motsatt rekkefølge av deres rekkefølge i original tabellen.

Oppgave 6
Radix sort: Usant. Lærebokens implementasjon av radix sort sorterer fra minst signifikant siffer til mest signifikante siffer. Her må man vite at minst signifikante siffer er det «bakerste» sifferet i et tall, mens det mest signifikante sifferet er det «første» sifferet i et tall. Det finnes også implementasjoner av radix sort som går andre vegen, da putter man typisk elementene i båser basert på mest signifikante siffer og sorterer hver bås rekursivt.

Oppgave 7
Radix sort: «Merge-Sort». Merge-Sort er stabil og har kjøretid $\Theta(n \log n)$. Radix sort krever stabil hjelpealgoritme, ergo er bokens implementasjon av quicksort utelukket. Videre har insertion sort kjøretid $\mathrm{O}(n^2)$ som er værre enn Merge-Sort. Vi kan heller ikke si noe om sannsynlighetsfordelingen derfor er ikke bucket-sort en passende algoritme. Man kunne tenkte seg å opprette k-bøtter, man ville da ikke trengt insertion sort, siden elementene legges inn i hver bøtte i rekkefølgen de er i tabellen som gis til «bøttesoreringsalgoritmen», men dette kan slå veldig dårlig ut når radiksen k er mye større enn antall elementer.

Oppgave 8
Radix sort: $\Theta(d \cdot n)$. Vi har for Radix-Sort med tellesortering at kjøretiden er $\Theta(d(n + k)) = \Theta(d(n + \log n)) = \Theta(dn)$. Siden $\log n = \mathrm{o}(n)$.

Oppgave 9
Radix sort: 1 er mest signifikante siffer. 4 er minst signifikante siffer. Det er kun plasseringen av sifferene som avgjør dette og ikke verdiene.

Oppgave 10
Bøttesortering: $\Theta(n^2)$. Dette oppstår eksempelvis når alle elementene havner i samme bøtte, og man dermed får samme worst-case som insertion sort.

Oppgave 11
Bøttesortering: $\Theta(n^2)$. Hvis vi kun har tre bøtter så vil vi også i average-case få $\Theta(n^2)$. Ved skuffeprinsippet ser vi at minst én av bøttene må ha $\geq n/3$ elementer og da vil insertion sort få forventet kjøretid på $\Theta(n^2)$ hvis vi antar at alle permutasjoner er like sannsynlige.

Oppgave 12
Median: Det miderste elementet. Siden vi har et odde antall elementer er medianen det miderste elementet, hadde vi hat et partall antall elementer ville medianen vært snittet av de to miderste elementene.

Oppgave 13
Median: Vi kan gjøre dette med worst-case kjøretid $\mathrm{O}(n)$. Dette oppnås ved bruk av select.

Oppgave 14
Randomized select: $\Theta(n)$.

Oppgave 15
Randomized select: $\Theta(n^2)$. Oppstår f.eks når vi skal ha det minste elementet, og kun klarer å luke ut ett og ett element om gangen. Vi får da rekurensen $T(n) = T(n-1) + n$ som har løsning $T(n) = \Theta(n^2)$.

Oppgave 16
Select: $\mathrm{O}(n)$.