04. Rangering i lineær tid

Forelesning 4

Vi får ofte bedre løsninger ved å styrke krav til input eller svekke krav til output. Sammenligningsbasert sortering er et klassisk eksempel: I verste tilfelle må vi bruke $\lg n!$ sammenligninger, men om vi *antar mer* om elementene eller *bare sorterer noen* av dem så kan vi gjøre det bedre.

Pensum

- ☐ Kap. 8. Sorting in linear time
- ☐ Kap. 9. Medians and order statistics

Læringsmål

- [D$_1$] Forstå hvorfor *sammenlignings-basert sortering* har en *worst-case* på $\Omega(n \lg n)$
- [D$_2$] Vite hva en *stabil sorteringsalgoritme* er
- [D$_3$] Forstå COUNTING-SORT, og hvorfor den er stabil
- [D$_4$] Forstå RADIX-SORT, og hvorfor den trenger en stabil subrutine
- [D$_5$] Forstå BUCKET-SORT
- [D$_6$] Forstå RANDOMIZED-SELECT
- [D$_7$] Kjenne til SELECT; grundig forståelse kreves ikke

📚Sortingsgrensen

Det finnes en nedre grense for hvor raske rangeringsalgoritmer kan være. Denne grensa er på $\Omega(n \lg n)$. Dette er synlig i divide-and-conquer algoritmer som deler opp problemet på en måte som gjør det mulig å forestille seg et rekursjonstre.

A sorting algorithm using at most h comparisons on all inputs corresponds to a tree of height at most h. Such a tree has at most 2^h leaves. On the other hand, each permutation of $1, \ldots, N$ must land at a different leaf, and so there must be at least $N!$ leaves. Putting these together, we deduce that $2^h \geq N!$ and so $h \geq \log_2 N! = \Omega(N \log N)$ (using Stirling's approximation). So every sorting algorithm must use at least $\log_2 N! = \Omega(N \log N)$ comparisons in the worst case (on some inputs it can use less).

https://cs.stackexchange.com/questions/32311/proving-the-lower-bound-of-compares-in-comparison-based-sorting/32312#32312

Likevel finnes det algoritmer som $InsertionSort$ som kjører i $O(n)$ i beste tilfelle, men det er kun på grunn av hvordan algoritmen er skrevet. Dessuten kan vi utnytte sorteringsgrensen, og til og med bryte den ved å skrive smarte algoritmer. Et eksempel på dette er $RandomizedSelect$ som kommer senere. Her er generell tabell for sorteringsgrensa.

	Best	Average	Worst
O	$O(\infty)$	$O(\infty)$	$O(\infty)$
Θ	$\Theta(?)$	$\Theta(?)$	$\Theta(?)$
Ω	$\Omega(n)$	$\Omega(n \lg n)$	$\Omega(n \lg n)$

Det er umulig å si noe om den generelle øvre grensa ettersom algoritmen kan ta uendelig med tid for alt vi vet. Dessuten er det ikke mulig å gå lavere enn n fordi vi må garantere at hele sekvensen behandles.

📚Reduksjonsbevis

Vi kan bevise egenskaper til enkelte problemer ved å bruke de som del-løsning av et annet problem. Dette kalles reduksjonsbevis. Et eksempel på dette er unikhetsproblemet.

Oppgave

Å avgjøre om en tabell er uten duplikater må beviselig ha kjøretid $\Omega(n \lg n)$ i verste tilfelle.

Hva sier dette om sortering?

Bonus: Tenk på både øvre og nedre grenser i begge retninger.

Verdier kan sammenlignes ($a_i \leqslant a_j$), men ikke brukes dem som indekser.

Løsningsskisse

7	3	2	9	3	4	5	1

Sortering: X

1	2	3	3	4	5	7	9

Finn like naboer: $O(n)$

1	2	3	3	4	5	7	9

$$X + O(n) = \Omega(n \lg n)$$
$$X = \Omega(n \lg n)$$

I dette problemet blir vi fortalt at man kan i verste tilfelle finne ut om en tabell har duplikater i $\Omega(n \lg n)$ tid. Siden sortering er mesteparten av arbeidet løsningen kan det umulig ha seg at sortering i verste tilfelle er bedre enn $\Omega(n \lg n)$.

Instans (i)	Løsning (i)
En sekvens, A	Er elementene i A unike?
Reduksjon	Rekonstruksjon
Sortering	Negasjon
Instans (ii)	Løsning (ii)
En sortert sekvens	Har sekvensen like naboer?

> I forelesning 4 presenteres en analogi for reduksjonsbeviset med en låst kiste. Det kan umulig være lettere å åpne kista enn det er å åpne låsen.

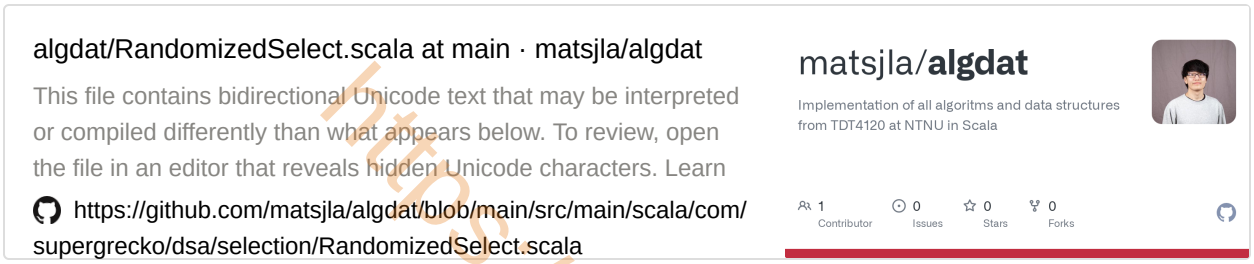
💻Randomized Select

Randomized Select er en divide & conquer algoritme som effektivt fungerer som "QuickSort som BinarySearch". Randomized Select finner elementet på i-ende indeks i lista. Vi bruker faktum at $Partition$ subprosedyren til $QuickSort$ plasserer pivotelementet på riktig plass, og sørger for at elementene til vestre er lavere, og

elementene til høyre er høyere. På denne måten kan vi splitte lista igjen og finne elementet på en gitt indeks.

- Best case: $O(n)$ av rekurrensen $2n - 1$
- Average case: $O(n)$
- Worst case: $O(n^2)$ hvis pivotelementet er alene, slik som i $QuickSort$. Lite sannsynlig til å skje med randomisering.

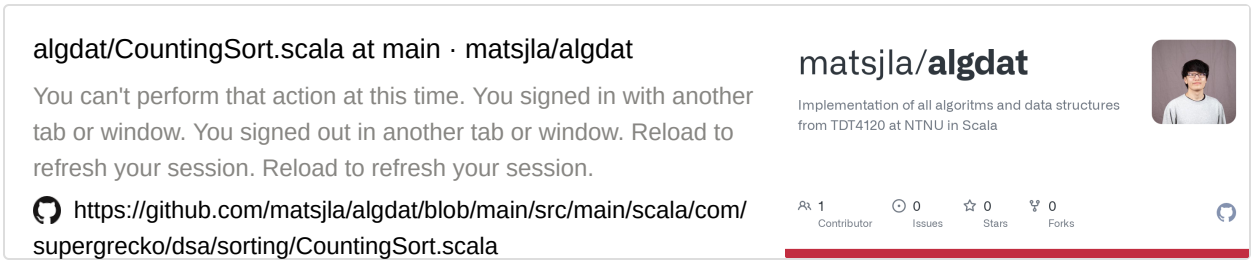
I tillegg finnes en annen select algoritme som heter $Select$. Denne er mer kompleks og grundig forståelse er ikke et krav i pensum.



💻Counting Sort

Counting Sort er en stabil sorteringsalgoritme som bryter sorteringsgrensen ved å gjøre en antakelse at ingen elementer er høyere enn en gitt verdi k. Deretter lager den en liste som holder styr på hvilken indeks ethvert element i lista skal være i den endelige lista.

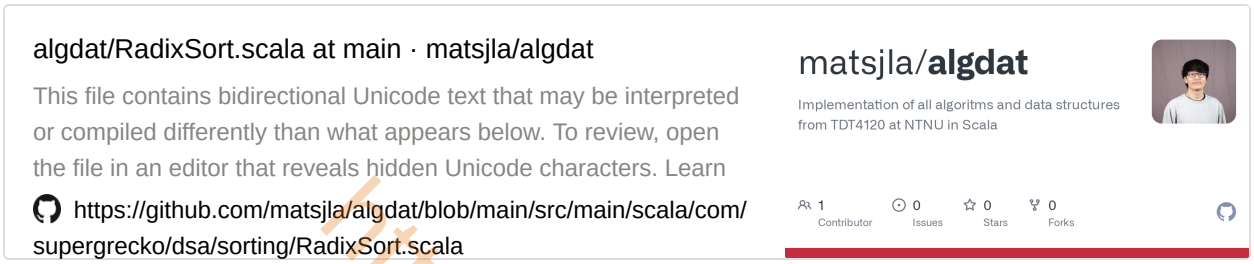
- Best case: $\Theta(n + k)$
- Worst case: $\Theta(n + k)$
- Average case: $\Theta(n + k)$



💻Radix Sort

Radix Sort er en stabil sorteringsalgoritme som bryter sorteringsgrensen ved å gjøre en antaklese at ingen elementer har flere enn d siffer. Deretter bruker den en stabil sorteringsalgoritme som $CountingSort$ til å sortere etter hvert siffer.

- Best case: $\Theta(d(n+k)$
- Worst case: $\Theta(d(n+k)$
- Average case: $\Theta(d(n+k)$



💻Bucket Sort

Bucket Sort er en sorteringsalgoritme som bryter sorteringsgrensen ved å anta at alle verdier er flyttall mellom $[0, 1)$ på en uniform distribusjon. Den deler elementene opp i n bøtter, og siden distribusjonen skal være uniform kommer det ikke til å være mange tall i hver bøtte. Dermed slår den sammen alle listene. Bucket Sort er stabil hvis sorteringsalgoritmen den bruker er stabil.

- Best case: $O(n)$
- Worst case: $O(n^2)$
- Average case: $O(n)$