IN2010

Oblig 3

Svein-Gisle Sætre

2021

# Oblig 3

## Deloppgave 1: Korrekthet

Foruten algoritmene Insertion sort og Quicksort har jeg implementert:

### ICantBelieveItCanSort

Dette var en algoritme som ble vist frem i forelesning da den er så lite intuitiv. Herfra og ut vil jeg referere til den forkortet som **ICBICS**. Implementasjonen er derimot meget enkel, som gjør at jeg ikke tror den trenger noen ekstra forklaringer. Ved behov er den uansett ganske nøye forklart på <https://arxiv.org/pdf/2110.01111v1.pdf>.

Algoritmen kjører i da den har to forløkker. For hver i kjører j også gjennom alle elementer. Den gjør også veldig mange unødvendige bytter, som gjør det til en veldig lite effektiv algoritme, men den fungerer.

### Heapsort

Heapsort er en algoritme som vist i forelesning der vi benytter oss av en Maxheap for å kunne sortere elementene.

Algoritmen kjører i .

### Test av algoritmene

For å teste algoritmene har jeg utvidet runAlgsPart1 i Oblig3Runner.java til å lese gjennom filene som blir generert for å sjekke om noen tall er mindre enn det forrige tallet i filen. Dette vil printe en feilmelding og vise tallene som er feilsorterte.

## Deloppgave 2: Sammenligninger, bytter og tid

Sammenligninger er målt ved bruk av hjelpefunksjonene fra pre-koden. Alle sammenligninger i løkker og ellers er gjort ved bruk av funksjonene lt, gt, geq etc.

For bytter prøvde jeg å bruke funksjonen swap, men støtte borti noen indekseringsfeil når jeg brukte den som jeg ikke klarte å finne ut av. Jeg har da heller manuelt inkrementert swaps-variabelen hver gang et bytte utføres.

## Deloppgave 3: Eksperimentér

### I hvilken grad stemmer kjøretiden overens med kjøretidsanalysene (store O) for de ulike algoritmene?

For å se på om algoritmene sin kjøretid stemmer overens med kjøretidsanalysene har jeg satt opp kurver for algoritmene i både nesten sortert og tilfeldig materiale for å se om kurvene har den stigningen som man skulle kunne forvente av deres O-notasjon. Det er noen forskjeller i hvor langt kurvene går da det tok for lang tid å teste noen av algoritmene på de store filene, men man ser likevel tendensene i kurvene jeg har fremstilt.

#### Insertion

Når tallmaterialet nesten er sortert kan Insertion, sett vekk fra noen utliggere, nesten se ut som den er lineær, eller iallfall nærmere enn som algoritmen skal kjøre i, i verste tilfelle. Ved tilfeldig tallmateriale ser man nærmere tendensen til -stigningen man skulle forvente. Ved et nesten sortert tallmateriale trenger ikke Insertion sort å gjøre mange bytter, og kan fort bryte ut av løkken sin som gjør den en del raskere. Ved en sortert liste vil insertion bare gå gjennom listen, så en nesten sortert liste vil altså gå mot jo mer sortert listen er.

Chart, line chart

Description automatically generated

tegnet i wolframalpha, som har en svak kurve oppover i motsetning til den lineære stigning som ikke har noen kurve.

#### Quick sort

Quick sort ser både i nesten sortert og tilfeldig tallmateriale ut som den ligger nærmere enn . Som forklart i forelesning er det veldig sjelden at man kommer ut for det verste tilfellet til Quicksort, så O-notasjonen kan i dette tilfellet være misvisende.

#### I Can't Believe It Can Sort

På ICBICS kan man tydelig se -stigningen i både nesten sortert og tilfeldig tallmateriale. Algoritmen har heller ingen forbedringer ved et nesten sortert materiale, da den vil bytte om på tall unødig mange ganger likevel.

#### Heapsort

Heapsort er og begge kurvene ser ut til å ligne på en slik stigning også.

### Hvordan er antall sammenligninger og antall bytter korrelert med kjøretiden?

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Korrelasjon mellom kjøretid og bytter | | | | | | | |
| Random | | | | NearlySorted | | | |
| Ins | Qck | ICBICS | Heap | Ins | Qck | ICBICS | Heap |
| 0,918628 | 0,999998 | 0,999961 | 1 | 0,998438 | 0,999977 | 0,993701 | 0,999997 |
| Korrelasjon mellom kjøretid og sammenligninger | | | | | | | |
| Random | | | | NearlySorted | | | |
| Ins | Qck | ICBICS | Heap | Ins | Qck | ICBICS | Heap |
| 0,918656 | 0,999996 | 0,999961 | 1 | 0,998426 | 0,999584 | 0,999958 | 0,999998 |

Ved å bruke korrelasjons-funksjonen i Excel ser vi at korrelasjon mellom kjøretid og bytter/sammenligninger nesten er perfekt. Insertion sort har derimot en litt svakere korrelasjon enn de andre algoritmene.

### Hvilke sorteringsalgoritmer utmerker seg positivt når n er veldig liten? Og når n er veldig stor?





Tabellene over ser på total kjøretid for sortering av alle n opp til n. For f. eks n=10 har vi kjøretiden for alle n fra det tomme arrayet opp til 10 n.

Når n er veldig liten ser vi at insertion og litt overraskende ICantBelieveItCanSort utmerker seg positivt. Kjøretiden deres er under halvparten av Quicksort og Heapsort. Når n blir 100 ser vi derimot at dette ikke lenger holder, særlig for ICBICS som fort faller av løpet.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| nearly\_sorted kjøretid | | | | |
|  | Insertion | Quick | ICBICS | Heapsort |
| 10 | 0 | 0 | 1 | 0 |
| 100 | 3 | 5 | 19 | 7 |
| 1 000 | 2 | 26 | 1 098 | 54 |
| 7 868 | 33 | 209 | 167 678 | 537 |
| 10 000 | 43 | 271 |  | 669 |
| 100 000 | 479 | 2 884 |  | 7 894 |
| 312 187 | 1 490 | 9 445 |  | 27 093 |
| 867 029 | **4 595** | **26 327** |  |  |
| 989 661 | **4 779** |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| random kjøretid | | | | |
|  | Insertion | Quick | ICBICS | Heapsort |
| 10 | 0 | 0 | 1 | 0 |
| 100 | 51 | 6 | 14 | 6 |
| 1 000 | 259 | 60 | 669 | 66 |
| 8 244 | 38 054 | 612 | 116 068 | 677 |
| 10 000 | 29 438 | **760** |  | **843** |
| 100 000 |  | 9 349 |  | 11 230 |

Tabellene over ser np på kjøretid for sortering av denne spesifikke n. For f. eks n=10 har vi kjøretiden for sorteringen på størrelsen n.

For tallmaterialet som er nesten sortert ser vi at insertion sort utmerker seg, mens for tilfeldig materiale utmerker Quick sort og Heapsort seg positivt.

### Hvilke sorteringsalgoritmer utmerker seg positivt for de ulike inputfilene?

Når tallmateriet nesten er sortert utmerker Insertion sort seg veldig positivt og har en mye mindre kjøretid enn både Quick sort og Heapsort. Quick sort som har ganske lik kjøretid som Heapsort i tilfeldig tallmateriale viser seg også å være mye raskere enn Heapsort når tallmaterialet nesten er sortert.

### Har du noen overraskende funn å rapportere?

Det mest overraskende er at Insertion sort er så mye bedre enn både Quick sort og Heapsort når tallmaterialet nesten er sortert. Når man ser på korrelasjonen mellom kjøretid og bytter/sammenligninger og antall bytter/sammenligninger i Insertion sort er det derimot ikke så ufattelig. Ved et nesten sortert tallmateriale må Insertion gjøre **langt** færre bytter og sammenligninger enn både Quick sort og Heapsort.

Nesten sortert materiale:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Swaps | Insertion | Quick | ICBICS | Heapsort |
| 10 | 3 | 13 | 16 | 56 |
| 100 | 39 | 137 | 178 | 1 392 |
| 1 000 | 461 | 1 322 | 1 829 | 22 583 |
| 7 868 | 3 576 | 10 312 | 14 445 | 237 391 |
| 10 000 | 4 566 | 13 280 |  | 310 416 |
| 100 000 | 45 929 | 134 144 |  | 3 946 280 |
| 312 187 | **143 919** | **412 880** |  | **13 585 111** |
| 867 029 | **400 703** | **1 156 686** |  |  |
| 989 661 | 457 277 |  |  |  |
| Comparisons | Insertion | Quick | ICBICS | Heapsort |
| 10 | 34 | 119 | 221 | 167 |
| 100 | 376 | 1 750 | 20 201 | 3 270 |
| 1 000 | 3 920 | 24 330 | 2 002 001 | 49 623 |
| 7 868 | 30 754 | 239 520 | 123 826 585 | 510 109 |
| 10 000 | 39 130 | 319 362 |  | 668 599 |
| 100 000 | 391 856 | 3 855 964 |  | 8 363 222 |
| 312 187 | **1 224 397** | **12 983 349** |  | **28 662 599** |
| 867 029 | **3 402 491** | **39 367 869** |  |  |
| 989 661 | 3 883 535 |  |  |  |

Det andre som er overraskende er hvor rask Quick sort er i forhold til Heapsort. Quick sort som er burde intuitivt være treigere enn Heapsort som er , men om vi ser på antall bytter og sammenligninger ser vi at Quick sort trenger å gjøre langt færre enn Heapsort, spesielt bytter. Dette er nok grunnen til at Quick sort holder så nært følge med Heapsort. Som nevnt tidligere er det også veldig sjelden at Quick sort faktisk kjører i verste tilfelle, så mye oftere er den nærmere .

Usortert materiale:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Swaps | Insertion | Quick | ICBICS | Heapsort |
| 10 | 29 | 14 | 33 | 48 |
| 100 | 2 704 | 214 | 2 712 | 1 271 |
| 1 000 | 249 411 | 3 013 | 247 983 | 20 753 |
| 8 244 | 16 940 361 | 30 748 | 16 449 021 | 233 592 |
| 10 000 | 24 925 372 | **37 831** |  | **290 799** |
| 100 000 |  | **457 397** |  | **3 737 865** |
| Comparisons | Insertion | Quick | ICBICS | Heapsort |
| 10 | 83 | 128 | 221 | 157 |
| 100 | 5 700 | 2 393 | 20 201 | 3 115 |
| 1 000 | 501 809 | 36 764 | 2 002 001 | 47 139 |
| 8 244 | 33 905 441 | 400 417 | 135 943 561 | 513 009 |
| 10 000 | 49 880 731 | **492 026** |  | **638 213** |
| 100 000 |  | **6 268 515** |  | **8 044 928** |