

Dat102 Oblig 3

Thobias K. Høivik

March 11, 2025

Uke 10 d) Tidskompleksitetsanalyse

Vi skal analysere tidskompleksiteten for følgende metoder for **TabellMengde** (array-basert mengde) og **LenketMengde** (lenket liste-basert mengde). Vi bruker *O-notasjon* for å uttrykke kjøretiden, og tar med både beste og verste tilfelle der det er forskjell.

i. boolean inneholder(T element)

- *Sjekk om et element finnes i mengden*

TabellMengde (Array-basert mengde):

- **Beste tilfelle:** Elementet finnes i første posisjon, som tar konstant tid, $O(1)$.
- **Verste tilfelle:** Elementet er på siste posisjon eller ikke i mengden i det hele tatt, og vi må iterere gjennom hele arrayet, som tar lineær tid, $O(n)$.

Så, for **TabellMengde**, er tidskompleksiteten:

Beste tilfelle : $O(1)$, Verste tilfelle : $O(n)$

LenketMengde (Lenket liste-basert mengde):

- **Beste tilfelle:** Elementet er i hodet av listen, og tar konstant tid, $O(1)$.
- **Verste tilfelle:** Elementet er på slutten av listen eller ikke i listen, og vi må traversere hele listen, som tar lineær tid, $O(n)$.

Så, for **LenketMengde**, er tidskompleksiteten:

Beste tilfelle : $O(1)$, Verste tilfelle : $O(n)$

ii. boolean erDelmengdeAv(MengdeADT<T> annenMengde)

- *Sjekk om mengden er en delmengde av en annen mengde*

TabellMengde (Array-basert mengde):

- **Beste tilfelle:** Hvis første element ikke finnes i den andre mengden, kan vi umiddelbart returnere `false`, som tar $O(1)$.
- **Verste tilfelle:** Vi må sjekke hvert element i den nåværende mengden for å se om det finnes i den andre mengden. For hvert element i den nåværende mengden, sjekker vi om det finnes i den andre mengden, som tar $O(n)$ for hvert av de n elementene, som gir en total tidskompleksitet på $O(n^2)$.

Så, for `TabellMengde`, er tidskompleksiteten:

$$\text{Beste tilfelle : } O(1), \quad \text{Verste tilfelle : } O(n^2)$$

LenketMengde (Lenket liste-basert mengde):

- **Beste tilfelle:** Hvis første element ikke finnes i den andre mengden, kan vi umiddelbart returnere `false`, som tar $O(1)$.
- **Verste tilfelle:** Vi må sjekke hvert element i den nåværende mengden, som innebærer å sjekke om hvert element er i den andre mengden. Dette krever å traversere begge listene, og derfor tar det $O(n^2)$ i verste tilfelle, der n er antall elementer.

Så, for `LenketMengde`, er tidskompleksiteten:

$$\text{Beste tilfelle : } O(1), \quad \text{Verste tilfelle : } O(n^2)$$

iii. boolean `erLik(MengdeADT<T> annenMengde)`

- *Sjekk om mengden er lik en annen mengde*

TabellMengde (Array-basert mengde):

For å se om to mengder er like, matematisk, innebærer å sjekke om de to mengdene er delmengder av hverandre. Så hvis de har forskjellig kardinalitet vil de ikke være like. Desverre ville dette involvert å se gjennom begge mengdene før vi gjør noe mer så vår metode kaller bare hver mengde sin `delMengdeAv()` metode siden vi ikke kan spare noe meningsful tid.

- **Beste tilfelle:** `erDelMengdeAv` $\rightarrow O(1)$. Vi kaller metoden to ganger som gir oss, men dette simplifiseres til $O(1)$.
- **Verste tilfelle:** Igjen for vi det dobbelte av `erDelMengdeAv`, men $2n^2$ simplifiseres til n^2 så vi sitter igjen med $O(n^2)$.

Så, for `TabellMengde`, er tidskompleksiteten:

$$\text{Beste tilfelle : } O(1), \quad \text{Verste tilfelle : } O(n^2)$$

LenketMengde (Lenket liste-basert mengde):

For akkurat samme grunner som i `TabellMengde` seksjonen får vi

LenketMengde:

$$\text{Beste tilfelle : } O(1), \quad \text{Verste tilfelle : } O(n^2)$$

iv. MengdeADT<T> union(MengdeADT<T> annenMengde)

- *Union av to mengder*

TabellMengde (Array-basert mengde):

- **Beste tilfelle:** Hvis begge mengdene er tomme, lager vi og returnerer vi bare en mengde som tar $O(1)$.
- **Verste tilfelle:** Vi må traversere begge mengdene. I verste tilfelle, hvis begge mengdene har n elementer, tar det $O(n)$ for å traversere begge mengdene og $O(n)$ for å kopiere resultatet til en ny array. Så, tidskompleksiteten er $O(n)$.

Så, for **TabellMengde**, er tidskompleksiteten:

$$\text{Beste tilfelle : } O(1), \quad \text{Verste tilfelle : } O(n)$$

LenketMengde (Lenket liste-basert mengde):

- **Beste tilfelle:** Samme igjen viss begge mengdene er tomme: $O(1)$.
- **Verste tilfelle:** Vi må traversere begge lenkede listene og sette inn alle elementene fra den andre listen i den første. Dette krever å besøke alle elementene i begge listene, og tar derfor $O(n)$.

Så, for **LenketMengde**, er tidskompleksiteten:

$$\text{Beste tilfelle : } O(1), \quad \text{Verste tilfelle : } O(n)$$

v. T fjern(T element)

- *Fjern et element fra mengden*

TabellMengde (Array-basert mengde):

- **Beste tilfelle:** Elementet er på første posisjon, og vi kan bare bytte dette elementet med det siste og fjerne det siste. Dette tar $O(1)$. Alternativt er mengden tom og det tar fortsatt $O(1)$.
- **Verste tilfelle:** Elementet er på siste posisjon eller finnes ikke i det hele tatt, og vi må traversere hele arrayet før vi kan fjerne elementet. Dette tar $O(n)$.

Så, for **TabellMengde**, er tidskompleksiteten:

$$\text{Beste tilfelle : } O(1), \quad \text{Verste tilfelle : } O(n)$$

LenketMengde (Lenket liste-basert mengde):

- **Beste tilfelle:** Elementet er i hodet av listen eller hodet finnes ikke, og vi kan bare oppdatere pekerne/gjøre ingenting. Dette tar $O(1)$.
- **Verste tilfelle:** Vi må traversere hele listen for å finne elementet, som tar $O(n)$.

Så, for **LenketMengde**, er tidskompleksiteten:

$$\text{Beste tilfelle : } O(1), \quad \text{Verste tilfelle : } O(n)$$

Oppsummering av tidskompleksiteter

Metode	Tabell Beste	Tabell Verste	Lenket Beste	Lenket Verste
i. inneholder()	$O(1)$	$O(n)$	$O(1)$	$O(n)$
ii. erDelmengdeAv()	$O(1)$	$O(n^2)$	$O(1)$	$O(n^2)$
iii. erLik()	$O(1)$	$O(n^2)$	$O(1)$	$O(n^2)$
iv. union()	$O(1)$	$O(n)$	$O(1)$	$O(n)$
v. fjern()	$O(1)$	$O(n)$	$O(1)$	$O(n)$

Uke 10 e) HashSet & TreeSet vs. Tabell & Lenket

Generellt når vi gjorde praktiske hastighetstester mellom Lenket- og Tabellmengde vs. HashSet, som vi bruket i JavaSetToMengde implementasjonen, observerte vi en hastighetsforskjell på to størrelsesordener; en enorm forskjell. Når det gjelder sammenligning mellom Lenket- og Tabellmengde observerte vi i våre tester at tabellmengde som oftest var litt raskere. Med HashSet og TreeSet står det på nettet at HashSet som oftest har bedre kompleksitet, gjerne $O(1)$ når TreeSet er $O()$. Den store fordelen for TreeSet er vistnok er sortert og kan da traverseres effektivt. TreeSet har også noen ekstra metoder som egner seg til noen bruk-sområder.

```

02:14:54 2025-03-09 1 2 ~/Code/dat102/Oblig3

[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ mengde-adt ---
Comparing times for inneholder
TabellMengde inneholder: 4471068806 ns
LenketMengde inneholder: 8231526761 ns
HashSetMengde inneholder: 2897691 ns

Comparing times for erDelmengdeAv
TabellMengde erDelmengdeAv: 4288300 ns
LenketMengde erDelmengdeAv: 2493952 ns
HashSetMengde erDelmengdeAv: 5744575 ns

Comparing times for erLik
TabellMengde erLik: 3257034602 ns
LenketMengde erLik: 6915381394 ns
HashSetMengde erLik: 7574542 ns

Comparing times for union
TabellMengde union: 6450403454 ns
LenketMengde union: 13717795196 ns
HashSetMengde union: 12820513 ns

Comparing times for fjern
TabellMengde fjern: 1493436145 ns
LenketMengde fjern: 6067215806 ns
HashSetMengde fjern: 4753961 ns

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:03 min
[INFO] Finished at: 2025-03-09T02:13:50+01:00
[INFO] -----

^ .../Oblig3 main ?

```

Figure 1: Kjøretider

Uke 10 f)



```
02:27:19 2025-03-09 1 2 3 ~/Code/dat102/Oblig3
~/Oblig3 main ?
> mvn clean compile exec:java
[INFO] Scanning for projects...
[INFO] -----< com.uke10:mengde-adt >-----
[INFO] Building mengde-adt 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO] --- clean:3.2.0:clean (default-clean) @ mengde-adt ---
[INFO] Deleting /home/thobias/Code/dat102/Oblig3/target
[INFO] --- resources:3.3.1:resources (default-resources) @ mengde-adt ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/thobias/Code/dat102/Oblig3/src/main/resources
[INFO] --- compiler:3.8.1:compile (default-compile) @ mengde-adt ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 6 source files to /home/thobias/Code/dat102/Oblig3/target/classes
[INFO] --- exec:3.1.0:java (default-cli) @ mengde-adt ---
Match score between Arne and Bjorn: 0.2
Match score between Arne and Charlotte: -0.3333333333333333
Match score between Bjorn and Charlotte: -0.7142857142857143
Best match: Arne and Bjorn
Match score for Arne with themselves: 1.0
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.891 s
[INFO] Finished at: 2025-03-09T02:26:55+01:00
[INFO] -----
~/Oblig3 main ?
>
```

Figure 2: Match

Uke 11



```
13:37:47 2025-03-11 2 3 4 ~/Code/dat102/Oblig3

Binærsøk søketid: 3183735 ns
Funne tal i tabellen: 991

^A ~/Oblig3 P main ?
> java src/main/java/uke11/Main.java
HashSet søketid: 589900 ns
Funne tal i HashSet: 976
Binærsøk søketid: 3128271 ns
Funne tal i tabellen: 976

^A ~/Oblig3 P main ?
> java src/main/java/uke11/Main.java
HashSet søketid: 2256309 ns
Funne tal i HashSet: 10000
Binærsøk søketid: 5655108 ns
Funne tal i tabellen: 10000

^A ~/Oblig3 P main ?
^C
^A ~/Oblig3 P main ?
> java src/main/java/uke11/Main.java
HashSet søketid: 2072732 ns
Funne tal i HashSet: 10000
Binærsøk søketid: 9788171 ns
Funne tal i tabellen: 10000

^A ~/Oblig3 P main ?
> java src/main/java/uke11/Main.java
HashSet søketid: 592957 ns
Funne tal i HashSet: 976
Binærsøk søketid: 3895074 ns
Funne tal i tabellen: 976

^A ~/Oblig3 P main ?
> 5;10u
```

Figure 3: Kjøretider

Vi observerer at HashSet er 5-6 ganger raskere for verdiene vi testet. Med en god hashing funksjon er tidskompleksiteten for å hente ut et gitt element $O(1)$ vs. binærsøk som er $O(\log n)$.