Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitenskap
Institutt for datateknologi, elektroteknologi og realfag

DAT102 – ALGORITMER & DATASTRUKTURER

OBLIG 3 – Mengde mm

Gruppemedlemmer:
Sarah Samia Ahsan & Amna Zafar

Oppgave 1
d) Analyse:

boolean inneholder(T element)
- TabellMengde: O(n) i worst case fordi vi må gå gjennom hele arrayet.
- LenketMengde: O(n) i worst case fordi vi må gå gjennom hele listen.

boolean erDelmengdeAv(MengdeADT<T> annenMengde)
- TabellMengde: O(n^2) worst case fordi hver sjekk av elementer involverer en iterasjon over den annen mengde.
- LenketMengde: O(n^2) worst case av samme grunn som TabellMengde.

boolean erLik(MengdeADT<T> annenMengde)
- TabellMengde: O(n^2) worst case siden vi må sammenligne hvert element med elementene i den annen mengde.
- LenketMengde: O(n^2) worst case av samme grunn som TabellMengde.

MengdeADT<T> union(MengdeADT<T> annenMengde)
- TabellMengde: O(n^2) worst case fordi vi må sjekke duplikater fra begge mengder.
- LenketMengde: O(n^2) worst case av samme grunn som TabellMengde.

T fjern(T element)
- TabellMengde: O(n) worst case fordi vi må finne elementet og deretter flytte elementer for å tette hullet.
- LenketMengde: O(n) worst case fordi vi må finne elementet og koble ut noden.

e) Basert på den informasjonen vi fant på nettet er besvarelsen for oppgave 1e) slik;

HashSet:
- Operasjonen `contains(T element)` for `HashSet` kjører i gjennomsnitt på O(1) tid. Dette er fordi elementene lagres basert på en hash-kode, noe som tillater rask oppslagstid. I sjeldne tilfeller med mange hash-kollisjoner kan tiden øke til O(n).

TreeSet:
- Operasjonen `contains(T element)` for `TreeSet` kjører alltid på O(log n) tid fordi elementene er ordnet i et binært søketre, og hvert oppslag reduserer området det må søkes i med halvparten.

TabellMengde og LenketMengde:

- For begge disse implementasjonene er tiden for å sjekke om et element er i mengden (dvs. `contains(T element)`) O(n) i verste fall, fordi det innebærer å se gjennom hvert element i datastrukturen lineært til elementet er funnet.

Sammenligning:
- `HashSet` tilbyr betydelig raskere oppslagstider enn `TabellMengde` og `LenketMengde` under normale omstendigheter. Imidlertid, hvis det skjer en stor mengde hash-kollisjoner, kan ytelsen avta og nærme seg O(n).
- `TreeSet` gir verre oppslagstid enn `HashSet` i gjennomsnitt, men det er mer forutsigbart med en konsekvent O(log n) ytelse, uavhengig av elementenes hash-koder. Det er også raskere enn lineær søketid som er O(n), slik vi ser i `TabellMengde` og `LenketMengde`.
- Dermed, hvis rask oppslagstid er viktig og du kan unngå dårlige hash-kollisjoner, ville `HashSet` være å foretrekke. Hvis du har behov for en ordnet travering av elementene eller en mer forutsigbar ytelse, ville `TreeSet` være et bedre valg.

Oppgave 4

```
Resultater:
Antall treff med HashSet: 951
Tid brukt for søk i HashSet: 4 ms
Antall treff med binærsøk: 951
Tid brukt for binærsøk: 14 ms
Diskusjon:
Vi observerer at søk i HashSet er betydelig raskere enn binærsøk i tabellen.
Dette skyldes at HashSet bruker en hashfunksjon for å organisere verdiene,
mens binærsøk i tabellen krever at tabellen er sortert på forhånd.
Hashfunksjonen gjør søket i HashSet konstant tid O(1),
mens binærsøk i en sortert tabell har en tidskompleksitet på O(log n).
Med større datasett vil forskjellen i ytelse være enda mer merkbar.
```