UNIVERSITETET I BERGEN

Det samfunnsvitenskapelige fakultet

Obligatorisk oppgave 2 - Kjøretidsanalyser og rapport

Thomas Sebastian Rognes (rut005)

Deloppgave 6)

append: O(n)

```
@Override
public void append(IList<? extends E> list) {
    for (E elem : list) {
        this.add((E) list.remove());
    }
}
```

Denne metoden legger til alle elementene i den angitte listen på slutten av listen. Kompleksiteten til denne metoden er O(n) ettersom kjøretiden kommer an på lengden av arrayet.

prepend = O(n)

```
@Override
public void prepend(IList<? extends E> list) {
    for (E elem : list) {
        this.put((E) list.remove());
    }
}
```

Denne metoden legger til alle elementene i den angitte listen på begynnelsen av listen. Kompleksiteten til denne metoden er O(n) ettersom kjøretiden kommer an på lengden av arrayet.

concat: O(n^2)

```java
@Override
public IList<E> concat(IList<? extends E>... lists){
    IList<E> returnList = new LinkedList<>();
    for(IList<? extends E> list : lists){
        while (!list.isEmpty()){
            returnList.add(list.remove());
        }
    }
    return returnList;
}
```

Denne metoden slår sammen flere lister. Kompleksiteten til denne metoden er O(n^2) ettersom denne metoden tar inn flere lister og vi må derfor bruke en "dobbel" for-loop når vi itererer gjennom hver liste for å legge den til i returnlist. Derfor er kjøretiden avhengig av størrelsen på listene og antall lister.

Deloppgave 12)

filter = O(n)

```java
@Override
public void filter(java.util.function.Predicate<? super E> filter) {
    Node node = head;
    while (node != null) {
        if (filter.test(node.getData())){
            Object data = node.getData();
            System.out.println(data);
            this.remove(data);
        }
        node = node.getNext();
    }
}
```

Denne metoden fjerner elementer fra listen som svarer til et predikat. Kompleksiteten til denne metoden er O(n) ettersom kjøretiden kommer an på lengden av arrayet.

map = O(n)

```java
@Override
public <U> IList<U> map(Function<? super E, ? extends U> f) {
    IList<U> mappedList = new LinkedList<U>();
    Iterator it = this.iterator();

    while(it.hasNext()){
        E input = (E)it.next();
        E executed = (E)f.apply(input);
        System.out.println(executed.getClass());
        mappedList.add((U)executed);

    }
    return mappedList;
}
```

Denne metoden kjører en funksjon over hvert element i listen og returnerer en liste over elementene som funksjonen returnerer. Kompleksiteten til denne metoden er O(n) ettersom kjøretiden kommer an på lengden av arrayet.

reduce = O(n)

```java
@Override
public <T> T reduce(T t, BiFunction<T, ? super E, T> f) {
    T value = t;
    Iterator myIterator = this.iterator();
    while (myIterator.hasNext()){
        E input = (E) myIterator.next();
        E executed = (E) f.apply(value, input);
        value = (T) executed;
    }
    return value;
}
```

Denne metoden slår sammen alle elementene i listen ved hjelp av en kombinasjon funksjon og returnerer den akkumulerte verdien av sammenslåingen. Kompleksiteten til denne metoden er O(n) ettersom kjøretiden kommer an på lengden av arrayet.

Ekstraoppgave 3)

Før jeg angrep oppgaven så bestemte jeg meg for å følge oppgavene slavisk. Jeg var litt usikker på hvordan det ville bli ettersom jeg er vant til å gjøre implementasjonen før jeg skriver tester.
Det gikk veldig tregt i starten og jeg brukte unødvendig mye tid på å skrive testene, noe som føltes ut som en lite effektiv måte å utføre oppgaven på. Det som var ubehagelig var å finne ut hvordan man skulle skrive testene slik at implementasjonen ble riktig. Blant annet var det vanskelig å skrive gode nok tester som dekker det viktigste i metodene som skulle bli testet. Når jeg ble ferdig med de første testene så gikk implementasjonen så å si knirkefritt. Testene som jeg laget ble bestått og motivasjonen gikk også opp. Jeg opplevde også at det var lettere å skrive testene når jeg hadde allerede implementert et par metoder. Da ble det enklere å forestille seg hvordan testene og implementasjonen skulle bli. Jeg skrev implementasjonen på map, filter og reduce før jeg skrev egne tester ettersom det var opprettet tester i vedlegg som skulle kjøres på disse metodene. Jeg fikk god hjelp av de skrevne testene for å skjønne hvordan jeg skulle løse disse metodene.
Denne utviklingsmetoden synes jeg fungerer godt ettersom den gjør det enklere å forstå hvordan en kan implementere metodene på en effektiv måte. Når jeg skrev testene så opplevde jeg at jeg måtte tenke som om metodene allerede var implementert og at jeg skulle bruke programmet. I starten brukte jeg mye tid på å forstå hvordan dette skulle utføres. Alt i alt synes jeg denne utviklingsmetoden var spennende, og personlig så var det en helt ny måte å programmere på.