UNIVERSITETET I BERGEN

Det samfunnsvitenskapelige fakultet

Kjøretidsanalyser og Ekstraoppgaver

Avansert Programmering

Gux006

Kjøretidsanalyser - gux006

Append: O(n)

Kompleksiteten av denne metoden er O(n), da den vil bruke n tid avhengig av størrelsen på listen som legges til.

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

Prepend: O(n)

Kompleksiteten av denne metoden er O(n), da den vil bruke n tid avhengig av størrelsen på listen som legges til.

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

Concat: O(n^2)

Kompleksiteten av denne metoden er O(n^2) da den itererer gjennom n lister med n elementer.

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

Filter: O(n)

Kompleksiteten av denne metoden er O(n) da den vil bruke n tid avhengig av størrelsen på listen som filtreres.

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

}
```

Map: O(n)

Kompleksiteten av denne metoden er O(n) da den vil bruke n tid avhengig av størrelsen på listen som skal mappes.

```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 data = (E)it.next();
        E applied = (E)f.apply(data);
        mappedList.add((U)applied);
    }

    return mappedList;
}
```

Reduce: O(n)
Kompleksiteten av denne metoden er O(n) da den vil bruke n tid avhengig av størrelsen på listen som skal gjennomføre reduce.

```
@Override
public <T> T reduce(T t, BiFunction<T, ? super E, T> f) {
    T finalValue = t;
    Iterator it = this.iterator();
    while(it.hasNext()){
        E data = (E)it.next();
        finalValue = f.apply(finalValue, data);
    }
    return finalValue;
}
```

Ekstraoppgave 3)

Til å begynne med:
Da arbeidet med oppgaven begynte, var ikke testdrevet utvikling noe som appellerte spesielt mye til meg. Det føltes fullstendig bakvendt å begynne arbeidet med å skrive tester før det i det hele tatt fantes noe å teste. Personlig var dette en helt ny måte å produsere kode på, og det tok litt tid å omstille seg til å tenke preventivt. Det oppstod dog en del problemer underveis.

Underveis:
Etterhvert som man ble mer og mer vandt med å skrive tester, begynte ting å falle mer naturlig på plass etterhvert. I begynnelsen var det vrient å tenke rett logisk sett, da man alltid har vært vandt med å skrive tester basert på en kode man har foran seg. Etter en del øvelse, ble det bare mer å mer naturlig å tenke seg frem til hvordan testene skulle bruke de ulike metode for å kunne gjennomføres, og jeg er overbevist om at denne tenkningen gjorde det betraktelig lettere å faktisk skrive metodene etterpå. Dette skyldes at man ble nødt til å tenke seg frem til hvordan man ville at metoden skulle fungere når man skrev testene underveis. Nå skal det ikke sies sikkert at dette var en mer effektiv måte å skrive kode på enn den metoden jeg er vandt med hvor man skriver tester i etterkant for å teste koden man allerede har skrevet, men det er mulig.

Spesielt map, reduce og filter var vrien å skrive tester på før jeg hadde implementert metodene, da disse metodene tok i bruk parameter av typer jeg var fremmed for (Function, Comparator, Predicate). Det var heller ikke alt for mange konkrete eksempler å finne hvor disse ble brukt på samme måte som ønsket implementert på min LinkedList.

Konklusjon:

Til tross for at denne utviklingsmetoden fremstod som en bratt motbakke og var seig i starten av prosjektet, føles det som en lærerik prosess post-mortem. Det å tråkke litt utenfor komfortsonen kan i dette tilfellet ikke ansees som annet enn positivt, og det ga mestringsfølelse når metodene begynte å fungere i samspill og flere og flere av testene passerte.