Løsningsforslag – Algoritmer og datastrukturer 20. mai 2008

Oppgave 1

a) byggListe:
Nye elementer settes inn på slutten av lista. Gir arbeidsmengde O(N) for både ArrayList og LinkedList.

skrivListe:
Bruk av get gjør metoden lite effektiv for LinkedList. For å hente et element på en bestemt posisjon i en lenket liste, må lista lista traverseres fram til denne posisjonen, som er en O(N)-operasjon. Totalt gir dette arbeidsmengde $O(N^2)$.

ArrayList: Å indeksere en tabell er en operasjon med arbeidsmengde O(1). Gir arbeidsmengde O(N).

fjernOddetall:
LinkedList: $O(N^2)$: Både get og remove er operasjoner av O(N).
ArrayList: $O(N^2)$: remove er en O(N)-operasjon, siden rekkefølgen i lister skal bevares og etterfølgende elementer må flyttes ett hakk til venstre.

b) Metoden skrivListe kan forbedres ved å bruke en iterator, enten eksplisitt eller implisitt ved hjel av en foreach-løkke. Dette gir arbeidsmengde O(N) for begge listetypene.

```
public static void skrivListe(List<Integer> liste){
    for (int i:liste)
         System.out.print(i + " ");
}
```

Metoden fjernOddetall kan gjøres mer effektiv for LinkedList ved å bruke en iterator. Ved å bruke i iteratorens remove-metode kan vi fjerne elementet når vi befinner oss ved elementet som skal fjernes.

Gir arbeidsmengde O(N) for LinkedList, men fremdeles $O(N^2)$ for ArrayList på grunn av at etterfølgende tabellelementer må flyttes ett hakk til venstre.

```
public static void fjernOddetall(List<Integer> liste){
     Iterator<Integer> itr = liste.iterator();
     while (itr.hasNext())
         if(itr.next() %2 != 0)
            itr.remove();
}
```

Følgende implementasjon er O(N) for både ArrayList og LinkedList:

```
public static void fjernOddetall(List<Integer> liste){
  Integer[] a  = new Integer[liste.size()];
  liste.toArray(a);
  liste.clear();
  for(int k : a) if (k % 2 == 0) liste.add(k);
}
```

Oppgave 2

Posisjonen til elementene i en hashtabell bestemmes ved bruk av en hashfunksjon, som er en funksjon av elementenes verdier/nøkler. En kollisjon er situasjonen som oppstår når to elementer mappes til samme posisjon i tabellen. Begrepene åpen og lukket adressering er knyttet til kollisjonshåndtering.

Lukket adressering: Elementet plasseres alltid på den posisjonen som bestemmes av hashfunksjonen. Hver posisjon i tabellen er en referanse til en lenket liste eller en annen struktur hvor elementene som hører til denne posisjonen lagres.

Åpen adressering: Hvis kollisjon oppstår, plasseres elementet et annet sted i tabellen, på en systematisk måte slik at det er mulig å finne igjen elementene ved søk.

39	38	10	49					28	19

Oppgave 3

a) 1 2 3 4 5 6 7 8 9 10

b)

```
public class QueueStack<T>  implements StackADT<T>{

    private ArrayQueue<T> element_q;

    public QueueStack(){
        element_q = new ArrayQueue<T>();
    }

    public void push(T element) {
        element_q.enqueue(element);
        // Sørger for at det nye elementet kommer først i køen.
        for (int i=0; i<element_q.size()-1; i++)
            element_q.enqueue(element_q.dequeue());
    }

    public T pop()  {
        return element_q.dequeue();
    }

    public T peek() {
        return element_q.first();
    }
```

```
public boolean isEmpty() {
    return element_q.isEmpty();
}

public int size() {
    return element_q.size();
}
}
```