Oblig 3 – DAT102 – Joachim Leiros

Oppgave 1a

```java
class LeggTil {

    public static void main(String[] args) {
        int tall = 100;
        int sum = leggTil(tall);
        System.out.println("Sum = " + sum);
    }

    public static int leggTil(int nummer) {
        if (nummer != 0)
            return nummer + leggTil(nummer - 1);
        else
            return nummer;
    }
}
```

Utskrift:
Sum = 5050

Oppgave 1b

```java
public class oppgaveB {
    public static void main (String[] args) {
        int n = 9;
        for (int i = 0; i <= n; i++){
            System.out.println("Ledd nummer: " + i + " = " + ledd(i));
        }
    }

    private static int ledd(int n) {
        if (n == 0) {
            return 2;
        }
        if (n == 1) {
            return 5;
        }
        return 5*ledd(n-1) - 6*ledd(n-2) + 2;
    }
```

Utskrift:
Ledd 0 = 2
Ledd 1 = 5
Ledd 2 = 15
Ledd 3 = 47
Ledd 4 = 147
Ledd 5 = 455
Ledd 6 = 1395
Ledd 7 = 4247
Ledd 8 = 12867
Ledd 9 = 38855

Oppgave 1c

```
public class hanoitid {
    public static void main(String[] args) {

        int numDisks = TALL;
        int n = 10;

        TowersOfHanoi towers = new TowersOfHanoi(numDisks);

        long timeElapsed = 0;
        for (int i = 0; i < n; i++) {
            Instant start = Instant.now();
            towers.solve();
            Instant finish = Instant.now();
            long time = Duration.between(start, finish).toMillis();
            timeElapsed += time;
        }

        System.out.println("Antall disker: " + numDisks);

        System.out.println("Antall flytt: " + towers.getTotalMoves()/n);

        System.out.println("Tid: " + timeElapsed/n + "millisekund");
    }
}
```

Utskrift:
Antall disker: 10
Antall flytt: 1023
Tid: 0 millisekund

Antall disker: 28
Antall flytt: 268435455
Tid: 581 millisekund

Antall disker: 32
Antall flytt: 4294967295
Tid: 8361 millisekund

$$\frac{Tid32}{Tid28} = \frac{2^{32}-1}{2^{28}-1} = > \frac{8361\ ms}{581\ ms} = \frac{2^{32}-1}{2^{28}-1} = 14.390 \approx 14$$

Oppgave 2a
Lagt til i leggTil:

```
// Innsett foran noden som p peker på
DobbelNode<T> nyNode = new DobbelNode<>(el);

nyNode.setForrige(p.getForrige());
nyNode.getForrige().setNeste(nyNode);

nyNode.setNeste(p);
p.setForrige(nyNode);
```

Lagt til i fjern:

```
// Tar ut
antall--;
p.getForrige().setNeste(p.getNeste());
p.getNeste().setForrige(p.getForrige());
```

Oppgave 2b

Endret fjern metode.

```
public T fjern(T el) {
    T resultat = null;
    DobbelNode<T> p;

    if ((el.compareTo(foerste.getElement()) <= 0) ||
(el.compareTo(siste.getElement()) >= 0)) {
        System.out.println("Ugyldig verdi. verdi > " + foerste.getElement() +
"verdi < " + siste.getElement());

    } else { // Kun lovlige verdier
        p = finn(el);

        if (p != null) {
            // Tar ut
            antall--;
            p.getForrige().setNeste(p.getNeste());
            p.getNeste().setForrige(p.getForrige());

            // Oppdatere midten
            nyMidten();

            resultat = p.getElement();

        }
```

Oppgave 2c

Metoden nyMidten er O(n) siden algoritmen er lineær ettersom `int midtNR = antall / 2;`. Altså metoden itererer gjennom halvparten av elementene. Vi fjerner konstanter når vi finner O og dermed O=(n).

Dersom vi skal legge til midten for å få O(I)

Oppgave 2d

i) Gjennomsnitt av lineært søk når posisjonen er vilkårlig er 1/n og hver posisjon er like sannsynlig.
1*1/n+2*1/n+3*1/n+.....n*1/n
(1/n+2/n+3/n+....n*1/n)
(1+2+3+4+5....(n-1)+n)/n
sum = 1+2+3+4+5....(n-1)+n
2sum = (n+1)n
sum = n(n+1)/2
n(n+1)/2
(n+1)/2
Gjennomsnitt: n+1/2
I verstefall vil antall søk være n.

ii) Med midtpeker og bare søkning en vei:
Halvparten av gjennomsnittet fra i), verstefall vil være 0.5n.

iii) Halvparten av gjennomstnittet fra ii), verstefall vil være 0.25n.

Oppgave 3a

```
 public static void main(String args[])
   {
      Binaersok ob = new binaersok();
      int arr[] = { 2, 3, 4, 10, 40 };
      int n = arr.length;
      int x = 10;
      int result = ob.binaersok(arr, 0, n - 1, x);
      if (result == -1)
         System.out.println("FINNES_IKKE");
      else
         System.out.println("Fant element: " + result);
   }
}
```

Oppgave 3b

Oppgave 3c

Oppgave 4a
Insertion sort:

n	Antall målinger	Målt tid(snitt)
10000	10	108ms
30000	10	865ms
60000	10	3670ms

Selection sort:

n	Antall målinger	Målt tid(snitt)
10000	10	134ms
30000	10	1032ms
60000	10	4336ms

Bubble sort:

n	Antall målinger	Målt tid(snitt)
1000	10	6ms
3000	10	46ms
6000	10	191ms

Quick sort:

n	Antall målinger	Målt tid(snitt)
10000	10	9ms
30000	10	13ms
60000	10	16ms

Merge sort:

n	Antall målinger	Målt tid(snitt)
10000	10	85ms
30000	10	315ms
60000	10	1044ms

Radix sort:

n	Antall målinger	Målt tid(snitt)
10000	10	207ms
30000	10	269ms
60000	10	403ms

Oppgave 4b

I mine eksempler så er quicksort særdeles raskere enn mergesort. Men slik jeg har forstått det så vil merge sort være raskere ved store datamengder.

Oppgave 4c

Venstre tall

123	398	210	019	528	513	129	294	
0	1	2	3	4	5	6	7	8
019	123	210			528			
	129	294			513			
019	123	129	210	294	528	513		

Høyre tall

123	398	210	019	528	513	129	294		
0	1	2	3	4	5	6	7	8	9
210			123	294				398	019
			513					528	129
210	123	513	294	398	528	019	129		

Midterste tall

123	398	210	019	528	513	129	294		
0	1	2	3	4	5	6	7	8	9
	210	123							398
	019	528							294
	513	129							
210	019	513	123	528	129	398	294		

Vi får dataen sortert tilslutt ved at radix sorteringen kjører gjennom flere ganger, den andre gangen vil sorteringen kjøre gjennom med resultatet av første sortering. Dette kjøres helt til mengden er sortert.

Oppgave 4d