Avdeling for informasjonsteknologi, Høgskolen i Østfold
Løsningsforslag
Eksamen i ITF20006 Algoritmer og datastrukturer
03.01.2019

Oppgave 1

A: O(n)
Løkka går n/2 ganger. Totalt n/2 = O(n)

B: $O(n^2)$
Ytre løkka går n ganger, indre 2*i ganger. Totalt $O(n^2)$

C: $O(n^2 \log(n))$
Ytre løkke går n^2 ganger. Indre løkke dobler inkrementet i hvert gjennomløp, går $\log(i^2) = 2*\log(i)$ ganger. Totalt $O(n^2 \log n)$

D: O(n)
Parameter reduseres med 1, dette vil gi n kall.

E: O(log n)
Repetert halvering, som i f.eks. binærsøk.

F: O(n)
Halverer parameteren n, men kaller to ganger som i f.eks. flettesortering. Kallstrukturen blir som et perfekt balansert binærtre som har høyde log(n), der det er én node for hvert rekursivt kall. Antall noder i et slik tre (og antallet rekursive kall) er O(n).

Oppgave 2

A:
1. Preorder : 15 7 6 3 9 12 19 18 22
2. Inorder : 3 6 7 9 12 15 18 19 22
3. Postorder: 3 6 12 9 7 18 22 19 15

B: Ja, dette er et AVL-tre. Ingen av nodene har større høydeforskjell enn 1 på venstre og høyre subtre.

C: 15 7 19 6 9 18 22 3 8 12 21 5

D: 12 7 19 6 9 18 22 3 (evt: 18 7 19 6 9 22 3 12)

Oppgave 3

A:
```
int antallBlader(binærNode rot)
{
 if (rot == null)
     return 0;
 if (rot.venstre == null && rot.høyre == null)
     return 1;
 return antallBlader(rot.venstre) + antallBlader(rot.høyre);
}
```

B:
```
void speil(binærNode rot)
{
 if (rot != null)
 {
     binærNode tmp = rot.venstre;
     rot.venstre = rot.høyre;
     rot.høyre = tmp;
     speil(rot.venstre);
     speil(rot.høyre);
 }
}
```

C:
```
void speil(binærNode rot)
{
 if (rot == null)
     return;
 Queue Q = new Queue();
 Q.enqueue(rot);
 // Går gjennom treet nivå for nivå, og bytter venstre og høyre
 // barn
 while (!Q.isEmpty())
     {
      // Tar ut første node i køen
      binærNode denne = Q.dequeue();
      // Bytter venstre og høyre barn
      binærNode tmp = denne.venstre;
      denne.venstre = denne.høyre;
      denne.høyre = tmp;
      // Legger barna i kø
      if (denne.venstre != null)
          Q.enqueue(denne.venstre);
      if (denne.høyre != null)
          Q.enqueue(denne.høyre);
     }
}
```

Oppgave 4

A: 1. Lineær probing:

```
 0: 9679
 1: 4371
 2: 1989
 3: 1323
 4: 6173
 5: 4344
 6:
 7:
 8:
 9: 4199
```

2. Kvadratisk probing:

```
 0: 9679
 1: 4371
 2:
 3: 1323
```

```
  4: 6173
  5: 4344
  6:
  7:
  8: 1989
  9: 4199
```

3. Hashing med kjeding:

```
  0:
  1: 4371
  2:
  3: 6173 1323
  4: 4344
  5:
  6:
  7:
  8:
  9: 1989 9679 4199
```

4. Last come, first served

```
  0: 9679
  1: 4199
  2: 4371
  3: 6173
  4: 4344
  5: 1323
  6:
  7:
  8:
  9: 1989
```

5. Robin Hood hashing

```
  0: 9679
  1: 1989
  2: 4371
  3: 1323
  4: 6173
  5: 4344
  6:
  7:
  8:
  9: 4199
```

B: 1. En verdi som skal fjernes kan ikke uten videre tas ut av hastabellen, fordi vi da risikerer å ta vekk et ledd i kjeden av probes. I stedet *merkes* verdien som skal fjernes på en eller annen måte (f.eks. ved å bruke en boolsk variabel). Selve fjerningen og "gjenbruk" av de ledige plassene i tabellen, gjøres først når det er nødvendig med en rehashing

2. Finner listen som verdien som skal fjernes ligger i, ved å beregne hashindeksen. Deretter gjøres vanlig fjerning av verdien fra denne lenkede listen.

C: 1. LCFS vil fungere fordi vi følger en og samme kjede med probes, slik at vi i hvert steg vet på hvilken indeks vi finner neste element som evt. skal flyttes.

2. RH vil ikke fungere. Hvis vi skal flytte et element som allerede befinner seg i tabellen, vet vi ikke sikkert hvor neste probe befinner seg.