In2010

Oppgave 2)

a)

**Algorithm:** legger til et element bakerst i køen

**Input:** En kø K og en node V

**Output:** En kø K med kardinalitet |K|+1

**Procedure** Push\_back(K,V)

**If** K.forste = null **then**

K.forste←V

K.siste←V

lengde ← lengde+1

**else**

i←0

temp←K.forste

**while** i<lengde-1 **do**

temp←temp.neste

i←i+1

temp.neste←V

lengde ← lengde+1

**Algorithm:** legger til et element først i køen

**Input:** En kø K og en node V

**Output:** En kø K med kardinalitet |K|+1

**Procedure** Push\_front(K,V)

**If** K.forste = null **then**

K.forste←V

K.siste←V

lengde ← lengde+1

**else**

V.neste←K.forste

K.forste←V

lengde ← lengde+1

**Algorithm:** legger til et element i midten av køen

**Input:** En kø K og en node V

**Output:** En kø K med kardinalitet |K|+1

**Procedure** Push\_middle(K,V)

**If** K.forste = null **then**

K.forste←V

K.siste←V

lengde ← lengde+1

**else**

pos←((lengde+1)/2)

i←1

temp←K.forste

**while** i<pos **do**

temp←temp.neste

i←i+1

V.neste ←temp.neste

temp.neste←V

lengde ← lengde+1

**Algorithm:** printer det i-te elementet i køen

**Input:** En kø K og en posisjon P

**Output:** En node i posisjonen P

**Procedure** get(K,P)

**If** P <= lengde-1 **then**

i←0

temp←K.forste

**while** i<pos **do**

temp←temp.neste

i←i+1

**return** temp

**return** null

c)

public void push\_back(Noden x) bruker i verste tilfelle O(n)

public void push\_front(Noden x) bruker i verste tilfelle O(1)

public void push\_middle(Noden x) bruker i verste tilfelle O(n)

public Noden get(int pos) bruker i værste verste tilfelle O(n)

d)

Dersom vi vet hva N maksimalt kan være vil kunne oppgi verste tilfelle som en konstant.

Dermed vil algoritmen utføres på konstant tid. Da vil vi få O(1) for alle algoritmene (funksjonene).

Oppgave 3)

a)

**Algorithm:** Finner veien katten må for å komme til bunnen

**Input:**

**Output:** Verdien på grenene katten må gå

**Procedure** FinnVei()

print(kattpos)

**while** true **do**

print(kattGren)

sistegren← kattGren

**for**(nokkel ∈ tre.keySet()) **do**

**If** Arrays.asList(tre.get(nokkel)).contains(kattGren) **then**

kattGren ← nokkel;

break

**If** sistegren.equals(kattGren)**then**

break

Oppgave 4)

a)

**Algorithm:** Printer ut et balansert søketre

**Input:** En liste L

**Output:** Verdiene for et balansert søketre

**Procedure** FinnBarn(L)

**If** L. length= 0 **then**

**else**

i←L. length/2

print(L [i])

Object[] Test1 ←Arrays.copyOfRange(L,0,i);

Object[] Test2 ←Arrays.copyOfRange(L,0,L. length);

FinnBarn(Test1)

FinnBarn(Test2)

b)

**Algorithm:** Printer ut et balansert søketre

**Input:** En liste L

**Output:** Verdiene for et balansert søketre

**Procedure** FinnBarn(L)

**If** L. size== 0 **then**

**Else if** L. size== 1 **then**

print(List.poll())

**Else**

i←L. length/2

listeix← new PriorityQueue<>()

ix←0

s←0

While ix<i do

S ← List.poll()

listeIx.offer(S)

ix←ix+1

S ←List.poll()

print(S)

FinnBarn(listeIx)

FinnBarn(List)