Mediankø 12 poeng

En *mediankø* er en abstrakt datatype for en samling som støtter følgende operasjoner:

- `insert`(Q, x) – plasserer et element x på køen
- `removeMedian`(Q) – fjerner og returnerer *median*-elementet fra køen

Medianen er det «midterste» elmentet i samlingen, dersom elementene ordnes fra minst til størst. Dersom vi setter inn tallene $8, 3, 1, 5$ og 7 i en mediankø, så vil:

- første kall på `removeMedian` gi 5
- andre kall på `removeMedian` gi 3
- tredje kall på `removeMedian` gi 7
- fjerde kall på `removeMedian` gi 1
- femte kall på `removeMedian` gi 8

Merk at vi velger det minste av de to midterste elementene når køen inneholder et partall antall elementer, som vist i eksempelet ovenfor.

(a) Hvis vi setter inn tallene $4, 1, 8, 3, 2$ på en mediankø, hva vil de tre første kallene på `removeMedian` gi? Oppgi svaret som en kommaseparert liste med tall.

> 1 poeng for $3, 2, 4$.

(b) Forklar *kort* hvordan du vil representere en mediankø.

> 1 poeng for en forståelig forklaring. Må ses i sammenheng med neste deloppgave.

(c) Forklar hvordan du ville ha implementert operasjonene `insert` og `removeMedian`. Du kan bruke pseudokode som en del av forklaringen din, men det er ikke et krav. Alle datastrukturer nevnt i pensum er tilgjengelige for deg.

Maksimalt 6 poeng.

Det finnes mange måter å gå løs på dette problemet. En naiv løsning er å sette elementene inn i en usortert struktur (liste eller dynamisk array), og sortere med en effektiv sorteringsalgoritme for å finne medianen ved behov. Maksimalt 2 poeng for slike løsninger.

En løsning som bruker sortert innsetting i en liste får lineær tid på innsetting, og det vil være mulig å holde styr på hvor medianen er, slik at den kan finnes i konstant tid. Hvis man holder styr på hvor midten er, og bruker en dobbeltlenket liste, så kan man fjerne det minste elementet i konstant tid. Maksimalt 4 poeng for en slik løsning.

Den beste løsningen (vi har funnet) er å bruke to prioritetskøer (eller binære heaps); en max-heap og en min-heap, som lar oss få logaritmisk tid på begge operasjoner. Vi lar $Q.\texttt{small}$ være en max-heap og $Q.\texttt{large}$ være en min-heap. Hvis vi sørger for at $Q.\texttt{small}$ inneholder den minste halvparten av elementene og $Q.\texttt{large}$ inneholder den største halvparten, vil toppen av $Q.\texttt{small}$ alltid være medianen.

Merk at denne løsningen, som involverer å holde to datastrukturer like store, ligner på oppgaven Teque, som er blitt gitt som en innleveringsoppgave.

```
1  Procedure balance(Q)
2  |  if |Q.large| + 1 < |Q.small| then
3  |  |  x ← removeMax(Q.small)
4  |  |  insert(Q.large, x)
5  |  if |Q.small| < |Q.large| then
6  |  |  x ← removeMin(Q.large)
7  |  |  insert(Q.small, x)
8
9  Procedure insert(Q, x)
10 |  if Q.small is empty or x < peek(Q.small) then
11 |  |  insert(Q.small, x)
12 |  else
13 |  |  insert(Q.large, x)
14 |  balance(Q)
15
16 Procedure removeMedian(Q)
17 |  x ← removeMax(Q.small)
18 |  balance(Q)
19 |  return x
```

For full uttelling må forklaringen være dekkende nok til at man kan implementere algoritmene med utgangspunkt i forklaringen.

(d) Basert på forklaringen din i (c), hva kan du si om kjøretidskompleksiteten til `insert` og `removeMedian`?

Maksimalt 2 poeng, der det gis 1 poeng for hver prosedyre. For løsningen ovenfor er begge prosedyrer i $\mathcal{O}(\log(n))$.

(e) Hvis vi tar n elementer fra en mediankø ved hjelp av `removeMedian`, og setter dem inn i et binært søketre, én etter én, hvordan vil det resulterende binære søketreet se ut? Forklar kort.

Maksimalt 2 poeng.

I venstre subtre fra roten vil alle noder kun ha venstre barn, og i høyre subtre fra roten vil alle noder kun ha høyre barn. Det vil altså se slik ut:

