Flest forekomster 8 poeng

STRATEGI

Input: Et array `A` med n positive heltall
Output: Tallet som forekommer flest ganger i `A`

```
1  Procedure MostFrequent1(A)
2      H ← new HashMap()
3      for x in A do
4          H.put(x, 0)
5      m ← A[0]
6      for x in A do
7          f ← H.get(x)
8          H.put(x, f + 1)
9          if H.get(x) > H.get(m) then
10             m ← x
11     return m
```

STRATEGI

Input: Et array `A` med n positive heltall
Output: Tallet som forekommer flest ganger i `A`

```
1  Procedure MostFrequent2(A)
2      k ← maximum element in A
3      B ← array of size k + 1
4      for i ← 0 to k do
5          B[i] ← 0
6      m ← A[0]
7      for x in A do
8          B[x] ← B[x] + 1
9          if B[x] > B[m] then
10             m ← x
11     return m
```

Ovenfor ser du to (korrekte) algoritmer som begge tar et array `A` med positive heltall som input, og returner tallet som forekommer flest ganger. Drøft fordeler og ulemper ved de to ulike strategiene. Du bør oppgi både kjøretidskompleksiteten i *verste* tilfelle, og hva som er den *forventede* kjøretidskompleksiteten, med hensyn til n og k. Den bør begrunnes *kort*. Du bør også oppgi hvilke situasjoner **Strategi 1** bør foretrekkes over **Strategi 2**, og vice versa, med hensyn til konkret kjøretid. Gjøre rede for eventuelle antagelser du gjør underveis.

Dette er en oppgave som går ut på å gjøre rimelige antagelser og anvende dem til å si noe fornuftig om hvilken fremgangsmåte man bør bruke. Dette er den oppgaven fra eksamenssettet hvor sensoren er nødt til å anvende mest skjønn i poengsettingen. Tanken bak oppgaven er å få studenten til å reflektere over hvilken fremgangsmåte man bør velge i lys av kjøretidskarakteristikkene til de to strategiene. Det bør legges vekt på at teksten er godt skrevet (dette betyr ikke at det skal trekkes poeng for enkle skrivefeil, men at teksten må være tydelig og enkel å forstå; en fin tommelfingerregel er at en setning man ikke forstår etter to gjennomlesninger ignoreres). Under lister vi noen refleksjoner vi ønsker å se.

Strategi 1 benytter seg av et hashmap, som i verste tilfelle har lineærtid på innsetting og oppslag. Den gjør $O(n)$ innsettinger og oppslag i H, som tilsammen utgjør $O(n^2)$ kjøretidskompleksitet i verste tilfelle. (Merk at vi her antar et hashmap implementert med linear probing eller separate chaining, slik vi har sett i kurset, og ikke for eksempel Java sine hashmaps, som bruker separate chaining, hvor hver «bøtte» er et rød-svart tre og gir logaritmisk tid på alle operasjoner i verste tilfelle.) Hashmaps har derimot *forventet* konstant tid på innsetting og oppslag, og dermed får $O(n)$ forventet kjøretidskompleksitet.

Strategi 2 bruker kun arrayer, som har konstant tid på alle operasjoner. I motsetning til **Strategi 1** finner vi det største elementet k, og lager et array `B` med plass til k elementer, og initialiserer det med verdien 0. Å finne det største elementet er i $O(n)$, og opprettelsen og initialiseringen av `B` er i $O(k)$. Den siste løkken er også i $O(n)$. Tilsammen gir dette $O(n + k)$ i både verste og forventet kjøretidskompleksitet (fordi vi ikke benytter oss av hashing).

Den mest vesentlige forskjellen mellom **Strategi 1** og **Strategi 2** er at **Strategi 2** er treg dersom k er stor. Men hvis dette *ikke* er tilfelle (som for eksempel kan bety at $k < n$) så vil **Strategi 2** være mer effektiv enn **Strategi 1**, fordi en hashing har en del høyere konstantfaktorer som følge av rehashing og kollisjonshåndtering. I tillegg er det alltid en (svært liten) sjanse for at vi får mange kollisjoner, som gjenspeiles i analysen for verste tilfelle.

Merk: Det var en liten feil i **Strategi 2**, der vi lot størrelsen på `B` må være k, snarere enn $k + 1$.