Finn duplikat 12 poeng

Du er gitt et array `A` med sammenlignbare elementer. Du får vite at `A` inneholder nøyaktig ett duplikat. Altså er alle elementer i `A` unike, bortsett fra *ett* element, som forekommer nøyaktig to ganger.

(a) Anta at du er gitt et *sortert* array `A`, og får vite at x er duplikatet i `A`. Skriv en effektiv prosedyre som skriver ut de to posisjonene som inneholder x. Oppgi kjøretidskompleksiteten på algoritmen.

Input: Et *sortert* array `A` med n sammenlignbare elementer, og et element x som forekommer nøyaktig to ganger
Output: Skriver ut de to posisjonene $0 \leq i < j < n$ hvor `A[i]` $= $ `A[j]` $= x$ forekommer
Procedure `FindSortedIndicesOfDuplicate(A,` x`)`

```
// ···
```

Dette kan løses i $\mathcal{O}(\log(n))$ ved å bruke et binær søk. Binærsøk presentert på forelesning returnerer **`true`**/**`false`**, men kan enkelt modifiseres til å returnere indeksen elementet befinner seg på. I konteksten av denne oppgaven kan vi anta at x er i arrayet, og vi kan la algoritmen være udefinert i tilfellet der x ikke er i arrayet.

Input: Et ordnet array `A` og et element x i `A`
Output: Returner i slik at `A[i]` $= x$
Procedure `BinarySearch(A,` x`)`

```
low ← 0
high ← |A| − 1
while low ≤ high do
    i ← ⌊(low+high)/2⌋
    if A[i] = x then
        return i
    else if A[i] < x then
        low ← i + 1
    else if A[i] > x then
        high ← i − 1
```

Studenten trenger ikke implementere det modifiserte binærsøket, men bør nevne at binærsøket må modifiseres litt.

Med dette kan algoritmen uttrykkes slik:

Procedure `FindSortedIndicesOfDuplicate(A,` x`)`

```
i ← BinarySearch(A, x)
print(i)
if 0 < i and A[i - 1] = x then
    print(i − 1)
if i < n − 1 and A[i + 1] = x then
    print(i + 1)
```

Oppgaven kan også løses ved å løpe gjennom arrayet frem til x, og printe posisjonen og posisjonen etter.

- Opptil 4 poeng for en løsning i logaritmisk tid.
- Opptil 2 poeng for en løsning i lineær tid.
- Det trekkes 1 poeng for feil (eller ikke oppgitt) kjøretidskompleksitet.
- Det trekkes opptil 1 poeng for å ikke nevne at binærsøket må modifiseres.
- Det gis ingen poeng for løsninger over lineær tid.

(b) Anta at du er gitt et *sortert* array `A` (og nå får du ikke oppgitt elementet som er duplisert). Skriv en effektiv prosedyre som skriver ut de to posisjonene til elementet som forekommer to ganger. Oppgi kjøretidskompleksiteten på algoritmen.

```
Input: Et sortert array A med n sammenlignbare elementer
Output: Skriver ut de to posisjonene 0 ≤ i < j < n hvor A[i] = A[j]
Procedure FindSortedDuplicateIndices(A)
    // ···
```

Denne oppgaven løses best på lineær tid, og bør være ganske rett frem.

```
Input: Et sortert array A med n sammenlignbare elementer
Output: Skriver ut de to posisjonene 0 ≤ i < j < n hvor A[i] = A[j]
Procedure FindSortedDuplicateIndices(A)
    for i ← 0 to n − 2 do
        if A[i] = A[i + 1] then
            print(i)
            print(i + 1)
            return
```

- Opptil 3 poeng for en løsning i lineær tid.
- Det trekkes 1 poeng for feil (eller ikke oppgitt) kjøretidskompleksitet.
- Opptil 1 poeng for løsninger over lineær tid.

(c) Anta at du er gitt et array A (nå kan du ikke anta at arrayet er sortert). Skriv en effektiv prosedyre som skriver ut de to posisjonene til elementet som er duplisert. Oppgi kjøretidskompleksiteten på algoritmen.

```
Input: Et array A med sammenlignbare n elementer
Output: Skriver ut de to posisjonene 0 ≤ i < j < n hvor A[i] = A[j]
Procedure FindDuplicateIndices(A)
    // ···
```

Dette kan også løses på (amortisert og forventet) lineær tid ved å bruke et hash map eller et hash set. Studenten kan anta at disse datastrukturene har konstant tid på de relevante operasjonene (altså kreves det ingen bemerkning om amortisert eller forventet kjøretidskompleksitet).

```
Input: Et sortert array A med n sammenlignbare elementer
Output: Skriver ut de to posisjonene 0 ≤ i < j < n hvor A[i] = A[j]
Procedure FindSortedDuplicateIndices(A)
    firstSeen ← empty hash map
    for i ← 0 to n − 1 do
        x ← A[i]
        if x is a key in firstSeen then
            print(firstSeen[x])
            print(i)
            return
        firstSeen[x] ← i
```

Den kan løses i lineæritmisk tid ved å sortere med en god sammenligningsbasert sorteringsalgoritme. Merge sort og Heapsort er gode valg. Quicksort er også et godt valg (men bør også nevnes at den strengt tatt er kvadratisk i verste tilfelle). Bucket- og Radix sort kan ikke brukes her, siden alt vi vet om elementene er at de er sammenlignbare.

Etter sortering kan verdien av duplikatet finnes på lineær tid ved å bruke løsningen fra forrige deloppgave. Deretter kan arrayet løpes over nok en gang, og skrive ut de posisjonene duplikatet befinner seg på.

Oppgaven kan også løses i kvadratisk tid ved å sammenligne hvert element med alle de andre.

- Opptil 5 poeng for en løsning på lineær tid.
- Opptil 4 poeng for en løsning i lineæritmisk tid.
- Opptil 1 poeng for høyere enn lineæritmisk tid.
- Det trekkes 1 poeng for feil (eller ikke oppgitt) kjøretidskompleksitet.