DMA — Ugeopgave 4

Helga Rykov Ibsen `<mcv462>`

14. oktober 2021

Del A

1.

Algorithm 1 Insert(S,x)

```
1: c = S.head                                   ▷ Sets the cursor to the first element in the list
2: if x.key ≤ c.key then                        ▷ Should x be 1.element?
3:     x.next = c                               ▷ Let x point at former first
4:     S.head = x                               ▷ x now 1.element
5:     return                                   ▷ stop
6: while c.next ≠ NIL and x.key > c.next.key do   ▷ While current item is larger than x
7:     c ← c.next                               ▷ move cursor to the next element
8: x.next ← c.next                              ▷ Let x point at the next item
9: c.next ← x                                   ▷ Let c point at x
```

2.

Hvis vi skriver antal gange algoritmen laver hvert skridt, så får vi følgende:

Algorithm 2 Insert(S,x)

```
1: c = S.head                                   ▷ 1
2: if x.key ≤ c.key then                        ▷ 1
3:     x.next = c                               ▷ 1
4:     S.head = x                               ▷ 1
5:     return                                   ▷ 1
6: while c.next ≠ NIL and x.key > c.next.key do   ▷ n
7:     c ← c.next                               ▷ n
8: x.next ← c.next                              ▷ 1
9: c.next ← x                                   ▷ 1
```

Hvis vi antager, at det tager en konstant tid c_n at gennemløbe hvert skridt, så

kan vi beregne køretiden $T(n)$ som:

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6 \cdot n + c_7 \cdot n + c_8 + c_9$$

Vi ignorerer alle konstanter og får:

$$T(n) = n + n = 2n = n$$

Vi får altså at $T(n) = \Theta(n)$, hvilket også giver god mening da vi maksimalt besøger hvert element i listen én gang.

3.

Vi fandt ud af at `Insert` har kompleksitet $\Theta(n)$. Hvis vi kalder funktionen `Insert` n gange, hvor n er antal elementer i listen, så er den øvre grænse for hvor mange gange algoritmen kan køre svarer til $T(n) = \frac{n \cdot (n-1)}{2} = \mathcal{O}(n^2)$.

4.

Funktionen `Insert` svarer til funktion `Insertion-Sort`, fordi den gør nogenlunde det samme som `Insertion-Sort` gør ved at tage et element ad gangen fra en usorteret liste og sætte det ind i en anden liste, som er sorteret. Deres worst-case kompleksitet er den samme, hvis `Insertion-Sort` anvendes på en liste sorteret i faldende rækkefølge, nemlig $\mathcal{O}(n^2)$.

5.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
x		17	**14**		9	**1**			2	**4**	93	NIL			40	**16**	53	**10**		

Del B

1.

`B` = listen af lister.

`B.head` = første element i `B`.

`B.head.key` = S_1.

`B.head.key.head` = det første element i S_1.

`B.head.key.head.key` = 1.

2.

Præmissen siger, at der er $\sqrt{n}$ lister, dvs. der er lige så mange lister, som der er elementer i hver liste. Her e.g. er der tre elementer i hver S-liste og der er

tre S-litser — det giver i alt 3^2 eller 9 elementer. Der er ligeledes tre noder i B-listen — en node per S-liste.

Når vi undersøger med `INSERT-FROM-B`, hvor x skal ind hen, kan vi derfor skære problemet op i $\sqrt{n}$.

Vi kan derfor højest komme til at travesere hele B-listen $\sqrt{n}$ gange, og så alle noder i den pågældende underliggende S-liste, som også indeholder $\sqrt{n}$ elementer — det giver altså $2\sqrt{n}$. Men vi kan ignorere 2, da det er en konstant, så det bliver bare $\sqrt{n}$.

Algorithm 3 Insert-From-B(B,x)

```
1: c = B.head
2: while c.next≠ NIL and c.next.key.head.key < x.key do   ▷ As long as the current number is larger than the next list's first number
3:     c ← c.next                                          ▷ move cursor to the next element in B
4: Insert(c.key, x)                                        ▷ Insert x into the list cursor points at
```

3.

Svaret til dette spørgsmål er identisk med svaret til sprøgsmål 3. i del A. Hvis vi har en ikke sorteret liste med n elementer og skal indsætte dem alle i en række sorterede lister $S_1 ... S_k$, så skal funktion Insert kaldes præcis så mange gange som der er elementer n, som skal indsættes. Med andre ord skal vi bare sætte n foran i udregningen, nemlig $T(n) = \mathcal{O}(n\sqrt{n})$.