

Kommentar til afleveringsopgave 1

Opgave handlede om repræsentation af mængder af tal ved et array udstyret med en tæller. Der skulle implementeres og vurderes effektivitet (ved store-O) for metoderne med `_i` og indsæt ved to forskellige implementationer, én hvor tallene indsættes i den tilfældige rækkefølge de ankommer (dvs. som kald af indsæt) og en anden, hvor arrayet holdes sorteret.

De fleste har forstået så nogenlunde, hvordan koden skulle skrives, men diskussionen af effektivitet er lidt ulden, så derfor vil jeg her opsummere. I opgaveformuleringen var det ikke præciseret, om repræsentationen skulle tillade dubletter, og der er løsninger som har vagt både det ene og det andet.

Her er angivet $O(-)$ for et enkelt kald af de omtale metoder ved forskellige løsninger — når man som angivet benytter binær søgning ved i sorterede arrays og programmerer resten på optimal måde.

Løsningsmetode	Effektivitet af indsæt	Effektivitet af med_i
A. Tilfældig rækkefølge med dubletter	altid $O(1)$	$O(n)$ hvor n er det totale antal gange indsæt er kaldt
B. Tilfældig rækkefølge uden dubletter	$O(m)$ hvor m antallet af forskellige tal, som er indsæt	$O(m)$ hvor m antallet af forskellige tal, som er indsæt
C. Sorteret rækkefølge med dubletter	$O(n)$ hvor n er det totale antal gange indsæt er kaldt	$O(\log n)$ hvor n er det totale antal gange indsæt er kaldt
D. Sorteret rækkefølge uden dubletter	$O(m)$ hvor m antallet af forskellige tal, som er indsæt	$O(\log m)$ hvor m antallet af forskellige tal, som er indsæt

I tilfælde B, indsæt, er man nødt til at udføre et sekventiel gennemløb for at checke om elementet er der i forvejen, før man indsætter; i A kan man nøjes med blot at indsætte det som det næste i arrayet.

Metoder A+B er bedste hvis antallet af indsættelser er stort og antal opslag er meget lille (relativt). Heraf er A bedst i det generelle tilfælde; B er bedre kun bedre end A i det helt særlige tilfælde, det er en meget lille samling tal, som indsættes igen og igen.

Metoder C+D er bedst når antallet af indsættelser er lille i forhold til antallet af kald af `med_i`. Det som koster den lineære faktor ved indsæt for C+D er at man skal forskyde i gennemsnit halvdelen af de kendte elementer i arrayet ved en indsættelse for at få plads til et ny. I modsætning til A+B, synes det ved C+D at kunne betale sig generelt at fjerne dubletter: Det koster kun et tests ekstra i en velstruktureret kode for indsæt at undgå dubletter, og når man vil indsætte et tal som allerede findes (ved D) falder kompleksitetet fra $O(m)$ til $O(\log M)$.

Generelt er det altså A og D som har de bedste egenskaber, og kun i helt særlige tilfælde kan det være at B kan benyttes en fordel.