Sortering

Tema: Sorteringsalgoritmer.

U 1. Avgör för var och en av följande sorteringsalgoritmer om de kan dra fördel av att indata består av en vektor med n lika tal. Det du ska avgöra är om respektive algoritm blir väsentligt effektivare i ett sådant specialfall än vad den normalt är.

a) Insättningssortering
b) Mergesort
c) Heapsort
d) Quicksort, med pivot = median av tre

U 2. Tre personer har behov att sortera indata bestående av n element. De kan välja mellan de sorteringsmetoder som ingått i vår kurs. Det gäller lite olika förutsättningar för deras indata:

1. Person 1 är egentligen bara intresserad av att veta vilka som är de k minsta nycklarna, där k är väsentligt mindre än n.
2. Person 2 har indata som oftast är nästan sorterade från början.
3. Person 3 har inga speciella förutsättningar.

Ge förslag på lämplig sorteringsalgoritm för var och en av de tre personerna. Motivera valen av algoritm.

U 3. Antag att en vektor `int[] v` är fylld med tal. Sortera vektorn genom att använda någon lämplig metod bland Javas färdiga sorteringsmetoder. Vilken sorteringsalgoritm används i den metod du valt? (Se Javas dokumentation på nätet.)

U 4. Effektiva sorteringsalgoritmer används ibland som en slags förbehandling i algoritmer för olika problem. Förklara (i ord, inte programkod) hur sortering kan utnyttjas för att konstruera effektiva algoritmer för följande problem:

a) Ta bort alla dubbletter ur en vektor med n tal.
b) Givet två vektorer med n tal vardera. Ingen av vektorerna innehåller dubbletter, däremot kan tal som finns i ena vektorn också finnas i den andra. Slå samman talen till en vektor av tal så att alla tal som fanns i någon av vektorerna finns med i den nya vektorn och så att den nya vektorn inte innehåller några dubbletter.
c) Givet ett tal `x` och en vektor `m` innehållande n tal. Avgör om det finns två tal i `m` vars summa är `x`. Visa också att om talen är sorterade från början så kan man lösa problemet på linjär tid.

U 5. Implementera följande metod som med liknande teknik som i quicksort effektivt tar reda på det i:te elementet i storleksordning i en vektor a:

```
/** Sök upp det i:te elementet i storleksordning i vektorn a. */
public static <T extends Comparable<? super T>> T findIth(T[] a, int i);
```

Du har tillgång till följande färdigskrivna metod:

```
/** Partitionera elementen i a[first]..a[last]. Den plats pivot-elementet
    hamnar på returneras. Metoden klarar av delvektorer ned till en storlek
    av ett element. */
public static <T extends Comparable<? super T>> int partition (T[] a,
                                                               int first, int last);
```