Majoritetsproblemet

Problem: Til præsidentvalget i Frankrig har cirka 20 millioner vælgere afgivet deres stemme på et antal præsidentkandidater. Afgør om en af kandidaterne har opnået mere end halvdelen af stemmerne, og angiv i givet fald, hvem der er tale om ("vinderen").

Præcisering af inddata: Vælgernes stemmer er lagret i en tabel, `stemme`, således at `stemme[v]` angiver nummeret (≥ 1) på den kandidat, som vælgeren `v` har stemt på.

Præcisering af uddata: Hvis en kandidat har fået mere end halvdelen af stemmerne, så udskriv nummeret på kandidaten. Ellers udskriv "Ingen vinder".

Løsningsmulighed 1

(optælling i bunker)

Algoritme: Opret en tom bunke for hver kandidat. Læg hver stemme i den bunke, der svarer til den kandidat, der er stemt på. Hvis antallet af stemmer i en af bunkerne overstiger halvdelen af de afgivne stemmer, er der fundet en vinder.

Programmering i Java:
Hver bunke repræsenteres ved en tæller.

```
for (k = 1; k <= kandidater; k++)
    tæller[k] = 0;
for (v = 1; v <= vælgere; v++)
    tæller[stemme[v]]++;
vinder = 0;
for (k = 1; k <= kandidater; k++)
    if (tæller[k] > vælgere / 2)
        vinder = k;
if (vinder != 0)
    System.out.println("Vinder: " + vinder);
else
    System.out.println("Ingen vinder");
```

Løsningsmulighed 1

Fordele:

- Simpel at programmere
- Hurtig (køretiden er proportional med antal afgivne stemmer)

Ulemper:

Hvis der er mange kandidater:

- bruges megen ekstra plads til tællere
- hvis tabellen `stemme` indeholder kandidaternes **navne** (i stedet for numre), vanskeliggøres optællingen (algoritmen er med andre ord **ikke** generel)

Løsningsmulighed 2
(sortering)

Algoritme: Sorter stemmesedlerne efter deres kandidatnummer. Gennemløb derefter de sorterede stemmesedler for at afgøre, om en af kandidaterne successivt står på mere end halvdelen af stemmesedlerne.

Programmering i Java:

```
sorter(stemme);
kandidat = tæller = 0;
for (v = 1; v <= vælgere; v++)
    if (stemme[v] == kandidat) {
        if (++tæller > vælgere / 2)
            break;
    } else {
        kandidat = stemme[v];
        tæller = 1;
    }
if (tæller > vælgere / 2)
    System.out.println("Vinder: " + kandidat);
else
     System.out.println("Ingen vinder");
```

Løsningsmulighed 2

Ulemper:

- Sorteringen kan tidsmæssigt være dyr
- Det er et krav, at indholdet af `stemme` kan sorteres, f.eks. indeholder tal eller tekst (algoritmen er stadig **ikke** generel)

Løsningsmulighed 3
(bestemmelse af median)

Algoritme: Find den kandidat, der ville blive den midterste, såfremt stemmesedlerne blev sorteret efter deres kandidatnummer (uden dog at foretage sorteringen). Afgør herefter ved simpel tælling, om kandidaten er vinder.

Programmering i Java:

```
kandidat = median(stemme);
tæller = 0;
for (v = 1; v <= vælgere; v++)
    if (stemme[v] == kandidat)
        tæller++;
if (tæller > vælgere/2)
    System.out.println("Vinder: " + kandidat);
else
    System.out.println("Ingen vinder");
```

Løsningsmulighed 3

Fordel:

- Hurtig. Medianen kan bestemmes i tid, der i **gennemsnit** er proportional med antal afgivne stemmer.
  Med en kompliceret algoritme kan medianen bestemmes i tid, der i **værste tilfælde** er proportional med antal afgivne stemmer.

Ulempe:

- Det er et krav, at indholdet af `stemme` kan sorteres, f.eks. indeholder tal eller tekst (algoritmen er stadig **ikke** generel)

Løsningsmulighed 4
(udtynding, problemreduktion)

Ide: Hvis to stemmer er afgivet på to forskellige kandidater, og de pågældende to stemmesedler fjernes, så vil en eventuel vinder for de oprindelige stemmesedler også være en vinder for de resterende.

NB: Det modsatte er ikke tilfældet. F.eks. har listen (1, 2, 5, 5, 3) ingen vinder, men hvis 1 og 2 fjernes, så bliver 5 vinder blandt de resterende.

Løsningsmulighed 4

Algoritme: Stemmesedlerne gennemløbes i rækkefølge. Hver gang to er forskellige, eliminerer vi begge. Vinderen i den resterende liste bestemmes, hvorefter det kontrolleres, at denne kandidat er vinder i den originale liste.

Men hvad gør vi, hvis to successive stemmer er ens?

Svar: Vedligehold to variabler:

`kandidat`, der udpeger en potentiel vinder, og
`tæller`, der angiver hvor mange gange `kandidat`
ikke har kunnet parres med en anden kandidat

Hvis `tæller` efter gennemløbet er positiv, undersøges ved simpel tælling, om `kandidat` er vinder.

Løsningsmulighed 4

Programmering i Java:

```
tæller = 0;
for (v = 1; v <= vælgere; v++)
    if (tæller == 0) {
        kandidat = stemme[v];
        tæller = 1;
    } else if (stemme[v] == kandidat)
        tæller++;
    else
        tæller--;
if (tæller > 0) {
    tæller = 0;
    for (v = 1; v <= vælgere; v++)
        if (stemme[v] == kandidat)
            tæller++;
}
if (tæller > vælgere / 2)
    System.out.println("Vinder: " + kandidat);
else
    System.out.println("Ingen vinder");
```

Løsningsmulighed 4

Fordele:

- Hurtig. Køretiden er proportional med antal afgivne stemmer
- Kræver næsten ingen ekstra plads
- Let at programmere - og let at bevise
- Algoritmen er generel (forudsætter ikke, at tabelindholdet kan sorteres)

Ulemper:

Ingen

Empirisk undersøgelse af de fire løsningsmuligheder

50,000,000 stemmer afgivet på 255 kandidater

Køretid i sekunder på en 700 MHz Macintosh PowerBook:

1. Optælling i bunker:	1
2. Sortering (quicksort)	24
3. Bestemmelse af median	4
4. Udtynding	2