

Oblig 1 i INF2440 – v2014.

Finn de 50 største tallene i en stor array

Et problem for internettsøkeprogrammer som Google, Bing og Yahoo, er at et søk kan generere millioner av svar – mer eller mindre relevante for den som søkte. Vedkommende søker har heller aldri muligheter for å se på alt. Hun/han vil se på det mest relevante. Hver side (av de mange millioner treff) kan vi anta har en relevans-score, som er et heltall som er slik at jo større dette tallet er, desto mer relevant er vedkommende side.

Vi skal hjelpe Google til å parallelisere løsningen på dette problemet. Anta at du har n svar og at relevans-scoren ligger lagret i heltallsarrayen $a[0..n-1]$. En triviell sekvensiell løsning i Java er å bruke Javas innebygde sorterings algoritme (`Arrays.sort(int [] a)`), sortere hele arrayen og så plukke ut de 50 største tallene. Men dette tar alt for lang tid. En klart raskere algoritme A2 er følgende:

1. Vi antar først at de 50 tallene er i $a[0..49]$ er de største og innstikksorterer det området i $a[]$ i synkende rekkefølge (du må trivielt skrive om vanlig innstikksortering til å sortere i synkende rekkefølge).
2. Da vet vi at det minste tallet av de 50 første tallene i $a[]$ da ligger i $a[49]$. Dette tallet sammenligner vi etter tur med hvert element i $a[50..n-1]$. Finner vi et element $a[j] > a[49]$, så :
 - a. Bytt $a[49]$ med $a[j]$
 - b. Innstikk-sorter det nye elementet inn i $a[0..48]$ i synkende rekkefølge.
3. Når pkt. 2 er ferdig, ligger de 50 største tallene i $a[0..49]$ og ingen av øvrige tallene er overskrevet eller ødelagt.

Oppgave 1 – sekvensiell løsning:

Implementer den sekvensielle algoritmen A2 ovenfor. Test den for disse ulike verdiene av $n=100, 1000, \dots, 100$ mill ved å lage en array med tilfeldige tall (`java.util.random`), og test at du får riktig svar ved også å sorterer de samme tallene med `Arryas.sort(int [] a)`, og sammenlign dine svar (synkende rekkefølge) med de siste 50 plassene i $a[]$ etter at du har nyttet `Arrays.sort()` for å sjekke at det er riktig

(N.B. for å få samme 'tilfeldige' tall med å trekke tilfeldige tall med klassen `Random` hvis du vil lage samme array flere ganger, må konstruktøren til klassen `Random` få et starttall – f.eks **`Random r = new Random(97361)`** ; Da vil vi få samme tallsekvens når vi sier: **`r.nextInt(n)`** i løkke for å få neste tall mellom 0 og $n-1$).

Skriv en tabell som viser hvilke tid `Arrays.sort` og `Innstikk`-metoden A2 ovenfor bruker for ulike verdier av n ($n=100, 1000, \dots, 100$ mill). Tidene som rapporteres skal være medianen av 9 kall på begge metodene slik det ble vist på forelesningen Uke2. Innlever din kode til A2 og tabellen med kommentarer.

Oppgave 2 – parallell algoritme.

Du skal nå parallellisere A2 så godt du greier med de k kjernene du har på din maskin. Ta f.eks utgangspunkt i parallelliseringen av FinnMax – problemet. Det kan hende at du da står igjen med en liten sekvensiell fase etter at mesteparten av beregningene er gjort (som i FinnMax).

Skriv en tabell som viser hvilke tider sekvensiell og parallell Innstikk-metoden A2 ovenfor bruker for ulike verdier av n (n= 100, 1000,...,100 mill). Tidene som rapporteres her skal også være medianen av 9 kall på begge metodene slik det ble vist på forelesningen Uke2. Innlever din kode til paraA2 og tabellen med kommentarer.

Innlevering

Obliger i INF2440 innleveres i Devilry. Husk at det sammen selve koden på begge punktene skal ligge en liten rapport med tabeller over kjøretidene som beskrevet ovenfor. Oblig 1 leveres individuelt og senest innen fredag 14. feb. kl 23.59.

Tips:

1) Får du en feilmelding når du prøver å kjøre programmet ditt for n = 100 mill at du har for lite hukommelse, kan du starte programmet slik. Da ber du om 4 GB til programmet:

```
> java -Xmx4000m Oblig1 100000000 <+evt andre parametere>
```

2) Kode til innstikksortering (a) hele a[0..49]; b) ett nytt element inn fra a[49])

a) (N.B. **Stigende** rekkefølge – du må selv skrive den om og la den sortere i **synkende** rekkefølge):

```
/** Innstikk-sorterer a[v..h] i stigende rekkefølge */
void insertSort (int [] a, int v, int h) {
    int i, t;

    for ( int k = v ; k < h ; k++) {
        t = a [k+1] ;
        i = k ;
        while ( i >= v && a [i] > t ) {
            a [i +1] = a [i] ;
            i--;
        }
        a[i+1] = t ;
    } // end for k
} // end insertSort
```

b) N.B for å sortere inn et nytt element på plass a[49] trenger du en enklere versjon av koden over fordi de første 49 elementene a[0..48] står jo allerede i synkende, sortert rekkefølge.