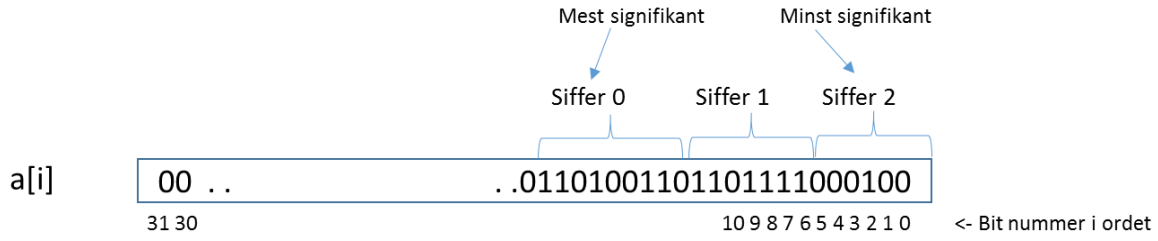
ver. 1.1, am. 11.okt, 2017

Oblig 3 i INF2220 – h2017. Sekvensiell rekursiv VenstreRadix sammenlignet med Kvikksort.

Radix-sortering finnes i to høyst ulike varianter – høyre-radix og venstre-radix for sortering av en heltallsarray : int [] a. Felles for dem begge er at de ser på verdien av ett element av gangen og sorterer det på denne verdien (sammenligner *ikke* to elementer for å sortere). Begge algoritmene deler altså opp de delene av alle tallene som ikke bare er 0, i ett eller flere sifre (et siffer er et antall sammenhengende bit i tallet). Hvor mange bit man bruker i hvert siffer er bare et optimaliseringsspørsmål. HøyreRadix er en iterativ algoritme som går gjennom alle tallene 2 x antall-sifre og starter med det høyre, minst signifikante sifferet først, mens **VenstreRadix** er en rekursiv algoritme **som sorterer en array og starter med å sortere på det første, mest signifikante sifferet,** så på det nest mest signifikante sifferet, …, osv. helt til alle sifrene i tallene er sortert.



VenstreRadix kan sees på som en variant av KvikkSort (som riktignok ble laget senere). I Kvikksort skiller vi på to verdier (store og små). Det gjøres med sammenligninger av alle elementene med en felles verdi (*pivot*). I VenstreRadix skiller vi f.eks. på 256 mulige verdier ved 'bare' å se på verdien av et bestemt siffer i a[i] i hver rekursjon. Riktig implementert er VenstreRadix vesentlig raskere enn KvikkSort, og en del av oblig3 går ut på å sammenligne din implementasjon av VenstreRadix med Javas innbygde sorterings-algoritme Arrays.sort() som bruker Kvikksort.

Et siffer er altså et bestemt antall bit, alt fra 1 til 30, og vi velger selv hvor stort siffer vi sorterer med. Når vi sorterer med flere sifre, behøver ikke alle sifrene være like lange, men for hvert siffer blir selvsagt alle tallene sortert med samme antall bit. Er et siffer *numBit* langt, er de mulige sifferverdiene **: 0 – (2^{numbit} -1)** (f.eks. dersom sifferet er 9 bit langt, er sifferverdiene: 0,1, …, 511). Effektivitetsmessig lønner det seg at det er fra 6 til 10 bit i ett siffer.

I den versjonen av VenstreRadix dere skal programmere, skal data flyttes frem og tilbake mellom to arrayer: a[] og b[]. Når dere starter er alle tallene usortert i a[], og b[] er nullfylt. Hver gang man sorterer på ett siffer, flyttes data (da sortert på dette sifferet) fra den ene arrayen til den andre. Når sorteringen er ferdig, skal det sorterte resultatet ligge i arrayen a[].

Denne versjonen av Radix-sortering som dere skal lage i Oblig3, MultiVRadix, velger selv hvor mange sifre den vil sortere på avhengig av hvor mange bit det er i den største verdien i arrayen a [] (=max) som skal sorteres. Denne max-verdien finner dere først, før sorteringen begynner.

For å få riktig sortering og fart på sorteringen, bør man vurdere følgende punkter:

- Er det den delen man skal sortere, rimelig kort, er InnstikkSort en raskere algoritme for den jobben (og da er sorteringen av denne delen av tallene avsluttet for alle sifrene).
- Når vi rekursivt går nedover med f.eks. med sifre på 8 bit, deler vi opp tallene hver gang i 256 ulike deler, og det er ikke sikkert at rekursjonen går like dypt i hver av disse delene (f.eks. hvis det ikke er noen tall med én av de 256 ulike verdiene eller hvis vi har brukt Innstikksortering, er jo rekursjonen avsluttet nedover den grenen).
- Ett element er sortert.
- Når vi har sorterer ferdig en del av arrayen vil, hvis vi har sortert et *ulike* antall ganger (= sifre), det sorterte resultatet ligge i b[]. Det må da som en siste operasjon denne delen kopieres tilbake til a[].
- For å få en raskest algoritme, må man bare kopiere tilbake fra b[] til a[] de gangene det er nødvendig. Husk at rekursjonen kan gå til ulike dybder i de ulike grenene.
- For å eksperimentere med hastigheten for din løsning bør du ha to konstanter, en som sier hvor kort en del av a[] skal være før du bruker InnstikkSort på den, og en konstant som sier hvor mange bit du i utgangspunktet vil bruke i hvert siffer.

Innlevering

Det du skal levere er programkoden **og en rapport** som først viser kjøretider av algoritmen din for n = 100, 1000, 10 000, 100 000, 1 mill. og 10 mill. som du har trukket med nextInt(n) metoden i bibliotekklassen Random. Løsningen skal også inneholde en enkel test på om arrayene er sortert ($a[i-1] \leq a[i]$, i=1,2, …, a.length-1) og den skal gi en feilmelding (se forslag til kode). Denne testen kjøres utenfor tidtakingen. For at dette skal bli en sammenligning med Kvikksort lar du også Arrays.sort (int [] a) sortere samme usorterte array (du må da trekke verdiene i a[] om igjen). I tabellen din skal du altså ha tre rader, en for tidene for hver verdi av n for din VRadixMulti og en for Arrays,sort() og en linje for speedup definert som Tid(KvikkSort)/Tid(VRadixMulti). Kommenter hvorfor VRadixMulti er så mye raskere (i forelesers implementasjon er den i snitt mer enn 4x for alle n).

Tidene du rapporterer (av å kjøre både VenstreRadix og Arrays.sort) skal være medianen av tidene du får ved å kjøre samme sortering 3 eller 5 ganger (hver gang selvsagt med nytt, usortert innhold i a[]). Dette fordi første gang du kjører en Java-metode får du langt høyere tider enn neste gang. Første gang oversettes koden til maskinkode fra bytekoden som er på .class-filen. Kjører man en metode mange ganger, optimaliseres denne maskinkoden ytterligere slik at det da går enda raskere.

Beskriv deretter med egne ord i rapporten om du kan si at VenstreRadix **er en stabil eller ustabil** sorteringsalgoritme og begrunn det.

Tallene som skal sorteres, skal trekkes uniformt mellom 0 ... n-1 (og denne trekkingen holdes utenfor tidtakingen). Tidene du skal levere, beregnes som medianen av minst 3 (eller 5) kjøringer for hver verdi av n.

Obliger i INF2220 innleveres i Devilry. Husk at det sammen med selve koden skal det ligge en rapport med tabell som beskrevet. Du vil ikke få denne obligen godkjent uten denne rapporten med forklaringer på kjøretidene som beskrevet ovenfor. Oblig 3 leveres individuelt og senest innen **onsdag 25. oktober kl. 23.59**.

Appendix A

Kodeskisse for én mulig implementasjon av VRadixMulti -sortering med variabelt antall sifre. Du behøver selvsagt ikke bruke denne hvis du har bedre idéer, ***men det som er fast, er at sorteringen skal gå mellom to arrayer a[] og b[] og at det skal lages en rapport som beskrevet***.

```
/** N.B. Sorterer a[] stigende – antar at: 0 ≤ a[i]) < 2^32 , returnerer tiden i millisek.  */

    final static int NUM_BIT =9; // eller: 6-13 er kanskje best.. finn selv ut hvilken verdi som er raskest
    final static int MIN_NUM = 31; // mellom 16 og 60, kvikksort bruker 47

     double  VRadixMulti(int [] a) {
            long  tt = System.nanoTime();
            int [] b  = new int [a.length];

             // a) finn 'max' verdi i a[]

            // b) bestem numBit = høyeste (mest venstre) bit i 'max' som ==1

            // c) Første kall (rot-kallet) på VenstreRadix med a[], b[] , numBit, og lengden av første siffer

            double tid = (System.nanoTime() -tt)/1000000.0;
            testSort(a);
           return tid;            // returnerer tiden i ms. det tok å sortere a, som  nå er sortert og testet
     } // end VRadixMulti


    // Sorter a[left..right] på siffer med start i bit: leftSortBit, og  med lengde: maskLen  bit,
     void VenstreRadix ( int [] a, int [] b, int left, int right, int leftSortBit, int maskLen){
               int mask = (1<<maskLen]) -1;
               int [] count = new int [mask+1];
                ……. Andre deklarasjoner ……

              // d) count[] =hvor mange det er med de ulike verdiene
              //     av   dette sifret I  a [left..right]

              // e) Tell opp verdiene I count[] slik at count[i] sier hvor i b[] vi skal
                    flytte første element med verdien 'i'  vi sorterer.

             // f) Flytt tallene fra a[] til b[] sorter på dette sifferet I a[left..right] for
                   alle de ulike verdiene for dette sifferet

              // g) Kall  enten innstikkSort eller rekursivt VenstreRadix
              //    på neste siffer (hvis vi ikke er ferdige)  for alle verdiene vi har av nåværende siffer

              //   Vurder når vi. skal kopiere tilbake b[] til a[]  ??

    }// end VenstreRadix

     void testSort(int [] a){
          for (int i = 0; i< a.length-1;i++) {
               if (a[i] > a[i+1]){
                     System.out.println("SorteringsFEIL på: "+
                                 i +" a["+i+"]:"+a[i]+" > a["+(i+1)+"]:"+a[i+1]);
                     return;
               }            }
     }// end testSort
```

// Tiden tas selv av **VenstreRadix.** Du må imidlertid omslutte kallet på **Arrays.sort(a)** med tidtaking,
// Disse tidene for en bestemt verdi av n oppbevarer du i to double – arrayer som du
// så tar innstikkSortering på for å finne medianverdien av hver av dem .

// Midtelementet i en sortert array er medianen.