IN2010 – obligatorisk oppgave 3

Beskrive eventuelle valg du har tatt og utfordringer du har møtt i arbeidet med å konvertere fra algoritme til Java-implementasjon.

Det mest utfordrende i disse implementasjonene var å ikke få IndexOutOfBoundsException. Fikk veldig ofte problemer med index, men etter å ha testet en del verdier så gikk det til slutt. Den mest utfordrende var quick-sort på grunn av den vanskelige tankegangen om å dele opp listen i mindre deler og kjøre det rekursivt videre. Valgte å sette pivot lik det første elementet i array, selv om det kanskje er mer optimalt å bruke medianen. Bucket-sort var også ganske utfordrende med tanke på en liten utfyllende algoritmebeskrivelsen i boka og valg av måte å lagre tallene på i buckets. Valgte også å sette antall elementer i array som array.length() som gjør at jeg ikke kan ha høyere verdier i array enn lengden til array i bucket-sort. Selection-sort og insertion-sort var ganske greie da det var lett å bare følge beskrivelsene som var veldig rett opp og ned.

Implementasjoner

Insertion-sort

```
29    public void insertionAlgorithm(int[] array){
30      for(int i = 2; i < array.length; i++){
31        int numb = array[i];
32        int j = i;
33        while(j > 1 && array[j-1] > numb){
34          array[j] = array[j-1];
35          j = j - 1;
36        }
37        array[j-1] = numb;
38      }
39      this.insertion_sort = array;
40    }
```

Selection-sort

```java
public void selectionAlgorithm(int[] array){
  for(int i = 0; i < array.length - 1; i++){
    int small = i;
    for(int j = i + 1; j < array.length; j++){
      if(array[j] < array[small]){
        small = j;
      }
    }
    //Bytter rekkefølge på verdier.
    int temp = array[small];
    array[small] = array[i];
    array[i] = temp;
  }
  this.selection_sort = array;
}
```

Quick-sort

```java
public void quickAlgorithm(int[] array, int low, int high){
  int index;
  if(low < high){
    index = partition(array, low, high);
    quickAlgorithm(array, low, index - 1);
    quickAlgorithm(array, index + 1, high);
  }
}
public int partition(int[] array, int low, int high){
  int pivot = array[low];
  int numb = low + 1;
  for(int i = numb; i <= high; i++){
    if(array[i] < pivot){
      if(i != numb){
      int temp = array[numb];
      array[numb] = array[i];
      array[i] = temp;
      }
      numb++;
    }
  }
  //Om ikke i <= high:
  array[low] = array[numb - 1];
  array[numb - 1] = pivot;
  return low;
}
```

Bucket-sort

```
87    public void bucketAlgorithm(int[] array){
88      int[] bucket = new int[array.length];
89      for(int i: array){
90        bucket[i] = 0;
91      }
92      int count = 0;
93      for(int i: array){
94        bucket[array[count++]]++;
95      }
96      int index = 0;
97      for(int i = 0; i < bucket.length; i++){
98        for(int j = 0; j < bucket[i]; j++){
99          array[index++] = i;
100       }
101     }
102      this.bucket_sort = array;
103   }
```

Beskrive for hver type input (tilfeldig/sortert/reversert) om det fremkommer noe spesielt mønster, og eventuelt hvilket.

Gi en forklaring på disse mønstrene ut fra hvordan algoritmen fungerer.

Mønster for insertion-sort:

*** Random list ***	*** Sorted list ***	*** Reversed list ***
4 2 7 4 9 6 0 3 7 0	0 1 2 3 4 5 6 7 8 9	9 8 7 6 5 4 3 2 1 0
2 4 7 4 9 6 0 3 7 0	0 1 2 3 4 5 6 7 8 9	8 9 7 6 5 4 3 2 1 0
2 4 7 4 9 6 0 3 7 0	0 1 2 3 4 5 6 7 8 9	7 8 9 6 5 4 3 2 1 0
2 4 4 7 9 6 0 3 7 0	0 1 2 3 4 5 6 7 8 9	6 7 8 9 5 4 3 2 1 0
2 4 4 7 9 6 0 3 7 0	0 1 2 3 4 5 6 7 8 9	5 6 7 8 9 4 3 2 1 0
2 4 4 6 7 9 0 3 7 0	0 1 2 3 4 5 6 7 8 9	4 5 6 7 8 9 3 2 1 0
0 2 4 4 6 7 9 3 7 0	0 1 2 3 4 5 6 7 8 9	3 4 5 6 7 8 9 2 1 0
0 2 3 4 4 6 7 9 7 0	0 1 2 3 4 5 6 7 8 9	2 3 4 5 6 7 8 9 1 0
0 2 3 4 4 6 7 7 9 0	0 1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9 0
0 0 2 3 4 4 6 7 7 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9

Algoritmen oppfører seg på en forventet måte. Den har flere linjer der det ser ut som den ikke gjør noe som helst, men det som skjer er at den finner at det neste tallet er større enn

det den er på og om den ikke er det så bytter den om tallene. En god beskrivelse på dette er at den tar hvert enkelt tall og finner et som er større som den tar med seg videre og dytter de små tallene bak i køen (noe man ser tydelig på reversed list. Det som er dumt med denne algoritmen er at man må gå gjennom listen unødvendig mange ganger, og dette vises på resultatet spesielt når det blir mange elementer i lista.

Mønster for selection-sort:

*** Random list ***	*** Sorted list ***	*** Reversed list ***
6 2 8 6 5 3 6 4 4 7	0 1 2 3 4 5 6 7 8 9	9 8 7 6 5 4 3 2 1 0
2 6 8 6 5 3 6 4 4 7	0 1 2 3 4 5 6 7 8 9	0 8 7 6 5 4 3 2 1 9
2 3 8 6 5 6 6 4 4 7	0 1 2 3 4 5 6 7 8 9	0 1 7 6 5 4 3 2 8 9
2 3 4 6 5 6 6 8 4 7	0 1 2 3 4 5 6 7 8 9	0 1 2 6 5 4 3 7 8 9
2 3 4 4 5 6 6 8 6 7	0 1 2 3 4 5 6 7 8 9	0 1 2 3 5 4 6 7 8 9
2 3 4 4 5 6 6 8 6 7	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9
2 3 4 4 5 6 6 8 6 7	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9
2 3 4 4 5 6 6 8 6 7	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9
2 3 4 4 5 6 6 6 8 7	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9

For selection-sort finner den for hver iterasjon det minste tallet og legger det i stigende rekkefølge. Dette fungerer spesielt godt på reversed list med tanke på at den bytter de minste tallene med de tallene som ligger på den plassen de skal på, som er de høyeste tallene, og dermed blir listen korrekt ved få iterasjoner.

Mønster for bucket-sort:

*** Random list ***	*** Sorted list ***	*** Reversed list ***
0 8 8 1 3 6 0 7 5 6	0 1 2 3 4 5 6 7 8 9	9 8 7 6 5 4 3 2 1 0
0 0 8 1 3 6 0 7 5 6	0 1 2 3 4 5 6 7 8 9	0 8 7 6 5 4 3 2 1 0
0 0 1 1 3 6 0 7 5 6	0 1 2 3 4 5 6 7 8 9	0 1 7 6 5 4 3 2 1 0
0 0 1 1 3 6 0 7 5 6	0 1 2 3 4 5 6 7 8 9	0 1 2 6 5 4 3 2 1 0
0 0 1 3 3 6 0 7 5 6	0 1 2 3 4 5 6 7 8 9	0 1 2 3 5 4 3 2 1 0
0 0 1 3 3 6 0 7 5 6	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 4 3 2 1 0
0 0 1 3 5 6 0 7 5 6	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 3 2 1 0
0 0 1 3 5 6 6 7 5 6	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 2 1 0
0 0 1 3 5 6 6 7 5 6	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 1 0
0 0 1 3 5 6 6 7 8 8	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 0
0 0 1 3 5 6 6 7 8 8	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9

I bucket-sort lager jeg buckets med masse 0 tall. Finner de riktig index til alle tall for at de skal bli sortert riktig. Bytter ut alle tall fra array med den riktige rekkefølgen i bucket. Litt vanskelig å forstå hva som skjer bare ved å se på printen, men for hver iterasjon så legger den til tall som tilhører den gruppen med tall i sortert rekkefølge. Man kan se, spesielt i reversed list, at den jobber seg fra starten i arrayen og utover til alt er sortert. Problemet er hvis en av tallene har en verdi større enn lengden til listen, og det kunne vært lurt å ha maks antall tall som en parameter i metoden for å forsikre om at det ikke blir feil.

Mønster for quick-sort:

*** Random list ***	*** Sorted list ***	*** Reversed list ***
5 3 2 2 8 9 6 6 5 7	0 1 2 3 4 5 6 7 8 9	9 8 7 6 5 4 3 2 1 0
2 3 2 5 8 9 6 6 5 7	0 1 2 3 4 5 6 7 8 9	0 8 7 6 5 4 3 2 1 9
2 3 2 5 8 9 6 6 5 7	0 1 2 3 4 5 6 7 8 9	0 8 7 6 5 4 3 2 1 9
2 2 3 5 8 6 6 5 7 9	0 1 2 3 4 5 6 7 8 9	0 1 7 6 5 4 3 2 8 9
2 2 3 5 7 6 6 5 8 9	0 1 2 3 4 5 6 7 8 9	0 1 7 6 5 4 3 2 8 9
2 2 3 5 5 6 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 6 5 4 3 7 8 9
2 2 3 5 5 6 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 6 5 4 3 7 8 9
2 2 3 5 5 6 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 5 4 6 7 8 9
	0 1 2 3 4 5 6 7 8 9	0 1 2 3 5 4 6 7 8 9
	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9

Man kan se tydelig på reversed list at jeg har valgt pivot som første element. Alle tallene som er mindre enn 9 har blitt lagt på venstre side av tallet. Dette gjør at ”listene” blir skjevt fordelt og det vil ta mye lenger tid enn om jeg hadde valgt et tall i midten(4 eller 5) og fått ca. like mange tall på hver side. Implementasjonen er uansett ganske rask når det er få tall i listen sånn som her. Den tar det største elementet og bytter det med det minste og fortsetter med det innover listen til den har blitt sortert.

Test-resultater

1000 elementer	Insertion	Selection	Bucket	Quick	Arrays.sort
Random	3,308863	4,029762	0,272574	1,03408	2,353463
Sorted	12,405606	0,387716	0,140381	3,753255	0,055516
Reversed	11,676291	0,477805	0,156786	3,260771	0,042571
5000 elementer	Insertion	Selection	Bucket	Quick	Arrays.sort
Random	20,882887	14,650105	0,704154	2,276512	2,292895
Sorted	13,94068	5,72913	0,459758	10,021124	0,173195
Reversed	14,273505	3,40259	0,488255	10,276411	0,1661
10000 elementer	Insertion	Selection	Bucket	Quick	Arrays.sort
Random	28,734789	25,528797	1,036201	3,900673	2,64311
Sorted	34,343467	15,409632	1,086639	35,737633	0,290675
Reversed	36,381312	13,584691	1,173462	64,165108	0,375296
50000 elementer	Insertion	Selection	Bucket	Quick	Arrays.sort
Random	377,924317	343,162432	3,933734	stackOverflow	5,097635
Sorted	668,168874	332,566847	1,846089	stackOverflow	3,747348
Reversed	672,687923	329,060066	1,51758	stackOverflow	0,621643
100000 elementer	Insertion	Selection	Bucket	Quick	Arrays.sort
Random	1326,567439	1828,518545	6,934227	stackOverflow	5,814698
Sorted	2679,518395	1437,441301	8,321465	stackOverflow	3,179766
Reversed	2746,960083	1454,228373	0,6703	stackOverflow	0,256525
500000 elementer	Insertion	Selection	Bucket	Quick	Arrays.sort
Random	40362,17533	55446,02038	40,172114	stackOverflow	14,081968
Sorted	80695,67373	40131,13173	12,299667	stackOverflow	0,971554
Reversed	111333,6681	49975,90722	17,709963	stackOverflow	6,565485

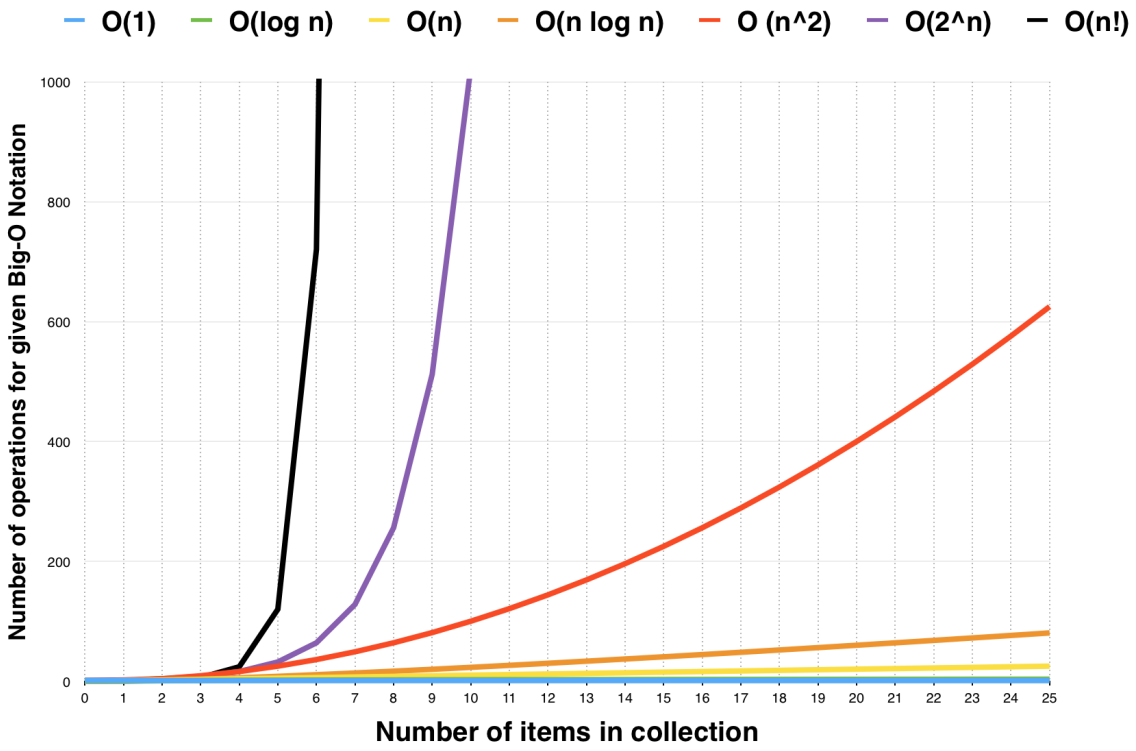
(Måleenhet i millisekunder)

Er det noen av resultatene i tabellen som overrasker deg?

Sammenlign faktisk kjøretid med forventet kjøretid basert på O-notasjon for hver av algoritmene. Stemmer teorien med praksis?

Det som overrasker meg:

- At selection-sort ikke gjorde det raskere på reversed enn det den gjorde. Fikk litt problemer når elementene økte.
- At ikke quick-sort var raskere enn de fleste andre algoritmene, og at den ikke klarte å håndtere flere elementer enn 10 000. (Dette har selvfølgelig noe med at jeg valgte pivot som det første elementet i listen, og at det ikke er quick-sort med innstikksortering). Quick-sort hadde nok vært raskest i flere tilfeller om listen hadde vært delt i likt antall elementer.
- At bucket-sort var helt klart den beste algoritmen jeg har laget for de testene jeg gjorde.



Source: https://medium.freecodecamp.org/my-first-foray-into-technology-c5b6e83fe8f1 (12.10.2015)

Tid kompleksitet

Insertion:	Selection:	Bucket:	Quick:
best = O(n)	best = O(n^2)	best = O(n + N)	best = O(n log n)
worst = O(n^2)	worst = O(n^2)	worst = O(n^2)	worst = O(n^2)

Jeg er overrasket over at insertion-sort gjorde det så mye dårligere enn selection-sort med tanke på at best-case tid kompleksiteten til insertion er bedre enn best-case for selection, mens worst-case er den samme.

Med tanke på kompleksiteten til bucket, er det forståelig at det er den raskeste og den beste løsningen på i oppgaven.

Grunnen til at quick-sort ikke gjør det bra er på grunn av den dårlige plass kompleksiteten. Når elementene øker drastisk blir plutselig tregere enn til og med insertion og selection.