IN2010
Oblig 3

3.

```
public static void selectionSort(int[] array){
        int n = array.length;
        for(int i = 0; i < n-1; i++){
          int minIndex = i;
          for(int j = i+1; j < n; j++){
            if(array[j]  < array[minIndex]){
              minIndex = j;
            }
          }
          int temp = array[minIndex];
          array[minIndex] = array[i];
          array[i] = temp;
        }
      }

 public static void insertionSort(int[] array){
             int n = array.length;
             for(int i=1; i < n; i++){
               int key = array[i];
               int j = i-1;
               while(j >= 0 && array[j] > key){
                 array[j+1] = array[j];
                 j = j-1;
               }
               array[j+1] = key;
             }
           }

 public static void quickSort( int[] array, int low, int high){
   int pivot = array[(low+(high-low)/2)];
   int i = low, j = high;
   while(i <= j){
     while(array[i] < pivot){ i++; }
     while(array[j] > pivot){ j--; }
     if(i <= j){
       swap(array, i, j);
       i++;
       j--;
     }
   }
   if(low < j){ quickSort(array, low, j); }
   if(high > i){ quickSort(array, i, high); }
 }
public static void swap(int array[], int x, int y){
        int temp = array[x];
        array[x] = array[y];
        array[y] = temp;
        //skrivUt(array);
          }

 public static void bucketSort(int[] array, int maxValue) {

            int[] bucket = new int[maxValue+1];
            for (int i = 0; i < bucket.length; i++) {
              bucket[i]=0;
            }
            for (int i = 0; i < array.length; i++) {
              bucket[array[i]]++;
            }
            int outPosition = 0;
            for (int i = 0; i < bucket.length; i++) {
              for (int j = 0; j < bucket[i]; j++) {array[outPosition++] = i; }
            }
          }
```

Jeg vurderte å ha en return in[] på hver av algoritmene slik at de returnerte ett nytt array objekt som var sortert. Men, da jeg kom til bucketSort skjønte jeg at det ikke ville vungere så lett på den fordi den er rekursiv. Så, for å holde et fast system på algoritmene mine valgte jeg å ikke ha returnverdi. Jeg lager istedefor en array for hver av testene I main metoden min. Ellers har jeg forsøkt å gjøre koden så minimal som mulig.

4.

Selection Sort:
selectionSort algoritmen fungerer som den skal. Den sorterer det tilfeldige arrayet, det lar den stigende arrayentvære som det er, og stokker om på det synkende arrayet og den tilfeldige. Måten den fungerer på er at den går gjennom arrayet, finner det minste tallet, og bytter det med det tallet som er forrerst Så går den gjennom alle tallende uten om det første, finner det minste, og bytter det med det tallet som er nest forrerst. Slik fortsetter den til den er er gjennom hele listen. På det stigende arrayet vill den gjøre samme prosess men plasere tallet på samme plass som det allerede står. Den kommer ikke til å variere i hastighet ut ifra rekkefølgen fordi den alltid må gjennom hele listen, noe som gjør algoritmen ikke er helt optimal.

Insertion Sort.
Denne fungerer også som den skal. Den sorterer alle de ulke arreysa. Insertion sort deler på et hvis opp arrayet i to deler. Den første delen er den sorterte delen, og den andre delen er den usorterte resten. For hvert element i arrayet så tar den det første tallet i den usorterte delen og plasserer det inn på riktig plass i den sorterte listen. Slik gjør den helt til alle elementene er i den sorterte delen. Denne algoritmen er ikke like konsekvent som den første. Her varierer tiden på hvor godt inputen er sortert fra før. Den sorterte listen går rett gjennom listen uten forandringer. Den tilfeldige varierer. Og den synkende har absolutt treigest kjøretid av de 3 forskjellige fordi den må iterere gjennom hele den sorterte delen for hvert element som skal sorteres.

Quick Sort:
Quick sort metoden finner først et pivot som blir mitten. Så sorterer den slik at alle tall som er mindre enn pivot elementet blir flyttet over før tallet, mens tallene høyere enn pivot blir sendt etter pivot. Så gjør den det samme rekursivt med de to sidene av pivot og slik fortsetter den. Når jeg sorterer det sitgende arrayet går den gjennom alle tallene i arrayet en gang for å sjekke at det er riktig rekkefølge. Dette er det raskeste quick sort kan være. På synkende array og tilfeldig har den en veldig konsekvent tidsbruk. Dette varierer mere på størrelsen på listen enn på rekkefølgen av tallene tror jeg.

Bucket sort:

Bucket sort er også veldig konsekvent på de ulike arraysa. Den bruker like lang tid både på random, synkende og stigende array.

5.

Jeg trodde egentlig at quick sort skulle være den raskeste generelt. Men det viste seg at insertion sort tar var raskere i omtrent alle tilfeller. Jeg hadde heller ikke trodd at arrays.sort skulle være så rask som den var. Den viser seg å være raskst i allerde sorterte lister og raskere enn insertion på kortere lister. Jeg føler ikke big O notasjon stemmer helt. Selvom selection sort har samme avrage og worst case så er insertion mye raskere. Dette kan selvsagt ha noe å gjøre med hvordan jeg implementerte den.

Farge på tekst er raskeste/tregeste mellom algoritmene.
Farge på boks er raskeste/tregeste mellom antall elementer.

Elementer ->		1000	5000	10000	50000	1000000	500000
Algorymer V	Average case V						
Selection Sort	$O(n^2)$						
Random		0.5936188	2.9432202	9.7824937	232.2031579	949.6245415	23683.7484
Stigende		0.1316821	2.4115822	9.9794641	114.4041562	477.4908113	23120.994175
Synkende		0.2790757	1.9669224	7.2817934	146.3382011	714.1847827	25696.151564
Insertion Sort	$O(n^2)$						
Random		0.0063872	0.0250446	0.0489171	0.2358927	0.639451	1.0525586
Stigende		0.0060961	0.0238421	0.0471226	0.2459304	0.4030219	1.2596294
Synkende		0.0062381	0.0242749	0.0468226	0.4180978	0.2328832	0.0393351
Quick Sort	O(n log(n))						
Random		0.0493025	0.1249891	0.1816358	2.8169483	3.0817617	8.3953074
Stigende		0.0390346	0.0326694	0.0728458	1.6290034	0.5144677	2.0189802
Synkende		0.0258898	0.032481	0.0703621	1.4776067	0.514392	1.5469047
Bucket Sort							
Random	O(n+k)	0.0147371	0.0578251	0.1150385	0.8179619	1.7326118	2.532879
Stigende		0.0101509	0.0493161	0.1011838	0.5450267	0.5131426	3.205591
Synkende		0.0133778	0.0501772	0.0671539	0.5516114	0.4425877	0.6174096
Array Sort							
Random		0.1113794	0.2920047	0.6765024	1.8344996	5.3668425	16.9055122
Stigende		0.0045254	0.0164303	0.0307938	0.2097275	0.2986401	0.6746797
Synkende		0.0052726	0.0242272	0.0477389	0.3545769	0.4769265	1.0163921