Obligatorisk oppgave 3: Sortering

Insertion sort

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

Dette virker som en algoritme som kan være bra når man ikke har mange elementer.

- Tilfeldig: mengden av operasjonene den må utføre blir større jo mere elementer står i feil posisjon, men ikke jo flere elementer det er.
- Sortert etter stigende verdier: best case blir O(n).
- Sortert etter fallende verdier: worst case blir O(n^2).

Quick sort

```
public static void quickSort(int[] array, int start, int end){
   while (start < end){
     int l = partition(array, start, end);
     if (l-1 < end-1){
       quickSort(array, start, l-1);
       start = l + 1;
     } else{
       quickSort(array, l+1, end);
       end = l - 1;
     }
```

```
  }
 }

 public static int partition(int[] array, int start, int end){
  int pivot = array[end];
  int left = start;
  int right = end - 1;
  int temp = array[right];

  while (left <= right){
   while (left <= right && array[left] < pivot){
    left++;
   }
   while (right >= left && array[right] > pivot){
    right--;
   }
   if (left < right){
    temp = array[left];
    array[left] = array[right];
    array[right] = temp;
   }
  }

  temp = array[left];
  array[left] = array[end];
  array[end] = temp;
  return left;
 }
```

Dette er en in-place algoritme, som ikke krever ekstra lagring.

- Tilfeldig: ingen spesifikk form for mønster.
- Sortert etter stigende verdier: hvis pivot-elementet er første eller siste element, får vi en worst case på O(n^2).

- Sortert etter fallende verdier: best case blir når man alltid velger elementet i midten av arrayen som pivoten.

Merge sort

```
public static void mergeSort(int[] array, int start, int end){
  if (start < end){
    int mid = (start + end) / 2;
    mergeSort(array, start, mid);
    mergeSort(array, mid + 1, end);
    merge(array, start, mid, end);
  }
}

 public static void merge(int[] array, int start, int mid, int end){
  int x = start;
  int y = mid + 1;
  int temp = 0;
  int[] tempArray = new int[end - start + 1];

  while (x <= mid && y <= end){
   if (array[x] <= array[y]){
     tempArray[temp] = array[x];
     x++;
     temp++;
   } else{
     tempArray[temp] = array[y];
     y++;
     temp++;
   }
  }
```

```
  while (x <= mid){
    tempArray[temp] = array[x];
    x++;
    temp++;
  }

  while (y <= end){
    tempArray[temp] = array[y];
    y++;
    temp++;
  }

  for (int i = start; i <= end; i++){
    array[i] = tempArray[i - start];
  }
 }
```

Veldig stabil, sett som en «divide and conquer» algoritme.

- Tilfeldig: average case på O(n log n).
- Sortert etter stigende verdier: som quick sort så er best case O(n log n).
- Sortert etter fallende verdier: worst case er også på O(n log n).

1000 elementer	insertion sort	quick sort	merge sort
Tilfeldig	0.003	0.2	0.02
Sortert	0.004	0.07	0.002
Reversert	0.003	0.07	0.003

10 000 elementer	insertion sort	quick sort	merge sort
Tilfeldig	0.2	200	0.2
Sortert	0.2	230	0.03
Reversert	0.2	230	0.03

Insertion sort virker til å være veldig bra når det gjelder tid.