Tue 02/01/16

Selection sort

```
SelecionSort(A)
 i = findLargestElement(A)
 SWAP(A[i], A[A.length - 1])
 SelectionSort(A[0:A.length-2])
```

```
findLargestElement(A)
 i = 0
 for j=1 to A.length:
  if A[j] > A[i]
   i = j
 return i
```

How many comparisons does this perform for an array of length n?

Performs n-1 comparisons

How about selection sort?

```
T(n) = # comparisons performed by selectionSort()

T(1) = 1

T(n) = n + T(n-1)

T(n-1) = (n-1) + T(n-2)

Hence T(n) = n + (n-1) + T(n-2) = n + (n-1) + (n-2) + T(n-3)
 => n + (n-1) + (n-2) + (n-3) .... 1
```

```
 => n(n-1)/2
```

Merge sort

```
MergeSort(A, B, R)
 nextA = 0; nextB = 0;
 if(nextA < A.length && (nextB >= B.length || A[nextA] <=
B[nextB]))
  R[nextA + nextB] = A[nextA]; nextA++;
 else
  R[nextA + nextB] = B[nextB] nextB++;
```

```
mergeSort(A)
 if A.length <= 1
  return A
 X = mergeSort(A[0 : [A.length / 2 - 1]])
 Y = mergeSort(A[A.length / 2 : A.length - 1)
 R = newArray(A.length)
 MergeSort(X, Y, R)
 return R
```

Number of comparisons in merge w/ output of size n: <= 4

Let T(n) be the most comparisons performed by mergeSort when given an array of size n

```
T(1) = 1
T(n) = 1 + 4n + 2T(n/2) = 2T(n/2) + 4n + 1

T(n/2) = 2T(n/2/2) + 4(n/2) + 1
 => 2T(n/4) + 2n + 1

So now
```

```
T(n) = 2(2T(n/4) + 2n + 1) + 4n + 1
 => 4T(n/4) + 4n + 2 + 4n + 1
 => 4T(n/4) + 8n + 3

T(n/4) = 2T(n/4/2) + 4(n/4) + 1
 => 2T(n/8) + n + 1

So now T(n) = 4(2T(n/8) + n + 1) + 8n + 3
 => 8T(n/8) + 4n + 4 + 8n + 3
 => 8T(n/8) + 12n + 7
```