Sorting by Divide and Conquer

- Recall the abstract class *ASorter* in the handout.

```
public final void sort(int[] A, int lo, int hi)
{
  if (lo < hi) {
    int s = split(A, lo, hi);
    sort(A, lo, s-1);
    sort(A, s, hi);
    join(A, lo, s, hi);
  }
}

public abstract int split(int[] A, int lo, int hi);

public abstract void join(int[] A, int lo, int s, int hi);
```

Selection Sort

- Selection Sort is a *hard-split*, *easy-join* method.
  - `split()` divides the array into two partitions of size 1 and $n-1$.
    * The smallest element in the array is swapped with the element in the smaller partition.

```
public int split(int[] A, int lo, int hi)
{
  int s = lo;
  int i = lo + 1;

  // Invariant: A[s] <= A[lo:i-1].
  // Scan A to find minimum:
  while (i <= hi) {
    if (A[i] < A[s])
      s = i;
    i++; // Invariant is maintained.
  } // On loop exit: i = hi + 1;
    // also invariant still holds;
    // this makes A[s] the minimum of A[lo:hi].

  // Swapping A[lo] with A[s]:
  int temp = A[lo]; A[lo] = A[s]; A[s] = temp;

  return lo + 1;
}
```

  - `join()` does nothing!

- Selection Sort take $O(n^2)$ steps.

Insertion Sort

- Insertion Sort is an *easy-split, hard-join* method.
  - `split()` is trivial.

```
public int split(int[] A, int lo, int hi)
{
  return hi;
}
```

  - `join()` starts with a sorted array and finds the correct position in which to insert the new element.

```
public void join(int[] A, int lo, int s, int hi)
{
  int j = hi; // remember s == hi.
  int key = A[hi];

  // Invariant: A[lo:j-1] and A[j+1:hi] are sorted and
  // key < all elements of A[j+1:hi].

  // Shifts elements of A[lo:j-1] that are greater than key
  // to the "right" to make room for key.
  while (lo < j && key < A[j-1]) {
    A[j] = A[j-1];
    j = j - 1;     // invariant is maintained.
  } // On loop exit: j = lo or A[j-1] <= key.
    // Also invariant is still true.
  A[j] = key;
  // A[lo:hi] is sorted because:
  // A[lo:j-1] is sorted, A[j-1] <= key == A[j] < A[j+1:hi],
  // and A[j+1:hi] is sorted.
}
```

- Insertion Sort take $O(n^2)$ steps.