`arrayutil.c0` - utility functions for integer arrays
15-122, Principles of Imperative Computation

```
/* is_in: x in A[lower..upper) */
bool is_in(int x, int[] A, int lower, int upper)
/*@requires 0 <= lower  && lower <= upper && upper <= \length(A); @*/ ;

/* is_sorted: A[lower..upper) SORTED */
bool is_sorted(int[] A, int lower, int upper)
/*@requires 0 <= lower  && lower <= upper && upper <= \length(A); @*/ ;

/* swap(A, i, j) has the effect of switching A[i] and A[j] */
void swap(int[] A, int i, int j)
/*@ requires 0 <= i && i < \length(A) && 0 <= j && j < \length(A); @*/ ;
```

Comparing array segments to a value

x > A[lower..upper) means that, for all i ∈ [lower, upper), we have that x > A[i]

```
/* gt_seg: x > A[lower..upper) */
bool gt_seg(int x, int[] A, int lower, int upper)
/*@requires 0 <= lower && lower <= upper && upper <= \length(A); @*/ ;

/* ge_seg: x >= A[lower..upper) */
bool ge_seg(int x, int[] A, int lower, int upper)
/*@requires 0 <= lower && lower <= upper && upper <= \length(A); @*/ ;

/* lt_seg: x < A[lower,upper) */
bool lt_seg(int x, int[] A, int lower, int upper)
/*@requires 0 <= lower && lower <= upper && upper <= \length(A); @*/ ;

/* le_seg: x <= A[lower..upper) */
bool le_seg(int x, int[] A, int lower, int upper)
/*@requires 0 <= lower && lower <= upper && upper <= \length(A); @*/ ;
```

Comparing two array segments

A[lower1..upper2) > B[lower2..upper2) means that,
for all i ∈ [lower1, upper1) and for all j ∈ [lower2, upper2), we have that A[i] > B[j]

```
/* gt_segs: A[lower1..upper1) > B[lower2..upper2) */
bool gt_segs(int[] A, int lower1, int upper1, int[] B, int lower2, int upper2)
/*@requires 0 <= lower1 && lower1 <= upper1 && upper1 <= \length(A); @*/
/*@requires 0 <= lower2 && lower2 <= upper2 && upper2 <= \length(B); @*/ ;

/* ge_segs: A[lower1..upper1) >= B[lower2..upper2) */
bool ge_segs(int[] A, int lower1, int upper1, int[] B, int lower2, int upper2)
/*@requires 0 <= lower1 && lower1 <= upper1 && upper1 <= \length(A); @*/
/*@requires 0 <= lower2 && lower2 <= upper2 && upper2 <= \length(B); @*/ ;

/* lt_segs: A[lower1..upper1) < B[lower2..upper2) */
bool lt_segs(int[] A, int lower1, int upper1, int[] B, int lower2, int upper2)
/*@requires 0 <= lower1 && lower1 <= upper1 && upper1 <= \length(A); @*/
/*@requires 0 <= lower2 && lower2 <= upper2 && upper2 <= \length(B); @*/ ;

/* le_segs: A[lower1..upper1) <= B[lower2..upper2) */
bool le_segs(int[] A, int lower1, int upper1, int[] B, int lower2, int upper2)
/*@requires 0 <= lower1 && lower1 <= upper1 && upper1 <= \length(A); @*/
/*@requires 0 <= lower2 && lower2 <= upper2 && upper2 <= \length(B); @*/ ;
```