Lecture 7: Sorting Algorithms

CS 2110

February 10, 2026

Today's Learning Outcomes

30. Compare and contrast the *insertion sort*, *merge sort*, and *quicksort* algorithms, discussing aspects such as time/space complexity and stability.

20. Describe the loop invariant of an iterative method involving an array and visualize it using a diagram.

26. Determine the asymptotic time and space complexity of a piece of code involving one or more loops and/or method calls.

29. Determine the number of recursive calls and the maximum depth of the call stack of a recursive method and use these to compute its time and space complexities.

The Importance of Sorting

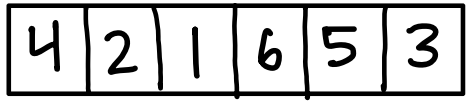
Having sorted data can greatly improve code effiency

Different Sorting Algorithms

Different sorting procedures trade off different desirable properties:

Being familiar with a varied toolbox of sorting algorithms will help you choose the best for a particular scenario.

Insertion Sort



Pre a:

Inv a:

Post a:

Coding Demo: Insertion Sort

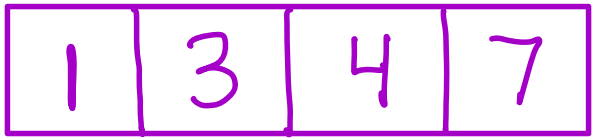
```
/** Sorts `a` using the insertion sort algorithm. */
static void insertionSort(int[] a) {
  /* Loop invariant: a[..i) sorted, a[i..] unchanged. */
  for (int i = 0; i < a.length; i++) {
     insert(a,i);
  }}

/** Inserts `a[i]` into its sorted position in `a[..i)`
   * so `a[..i]` becomes sorted. Requires that
   * `0 <= i < a.length` and `a[..i)` is sorted. */
static void insert(int[] a, int i) {  … }
```

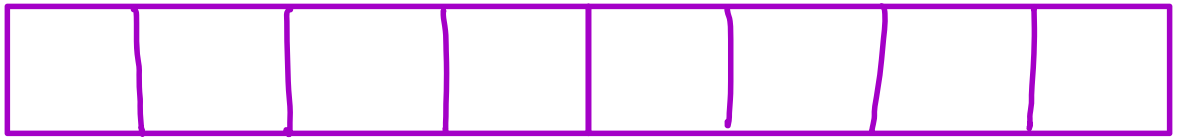
Insertion Sort (Worst-Case) Complexity

```
static void insertionSort(int[] a) {
    for (int i = 0; i < a.length; i++) {
        insert(a,i);
    }}

static void insert(int[] a, int i) {
    int j = i;
    while (j > 0 && a[j - 1] > a[j]) {
        swap(a, j - 1, j); j--;
    }}
```

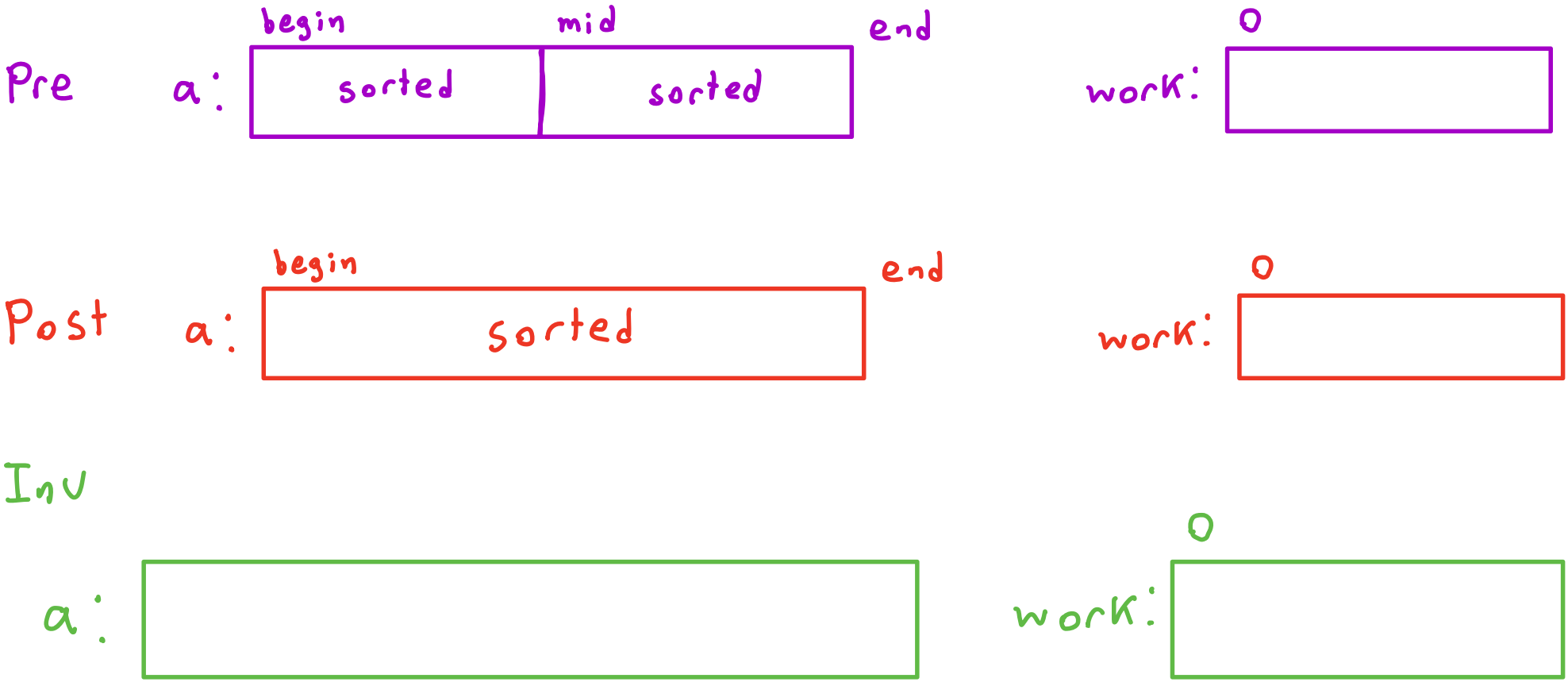
Adaptivity and Stability

Merge Sort





The `merge()` Invariant

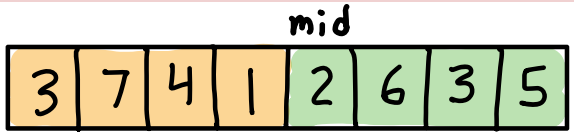


Coding Demo: The `merge()` Method

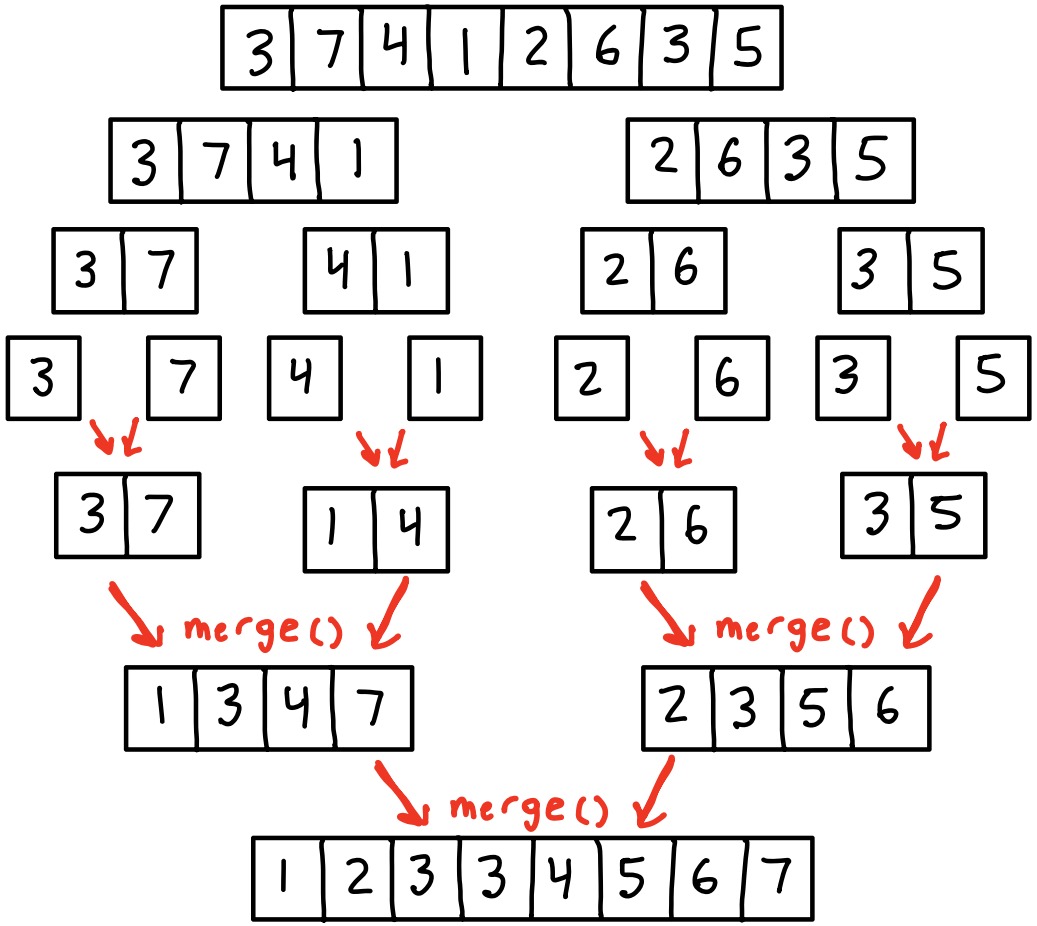
Merge Sort

```
/** Sorts the entries of `a` using the merge sort algorithm. */
static void mergeSort(int[] a) {
  mergeSortRecursive(a, 0, a.length);
}

/** Recursively sorts `a[begin..end)` using the merge sort algorithm. */
static void mergeSortRecursive(int[] a, int begin, int end) {




}
```

Visualizing Merge Sort



Merge Sort Runtime Analysis



Merge Sort Space Complexity

Merge Sort: Other Considerations

Quicksort

Coding Demo: `quicksort()`

Quicksort Complexity Analysis

Quicksort: Other Considerations

Sorting Summary

Algorithm	Worst-Case Time Complexity	Expected Time Complexity	Best-Case Time Complexity	Space Complexity	Stable?	Adaptive?
Insertion Sort	$O(N^2)$	$O(N^2)$	$O(N)$	O(1)	Yes	Yes
Merge Sort	$O(N \log N)$	$O(N \log N)$	$O(N \log N)$	$O(N)$	Yes	No
Quicksort	$O(N^2)$	$O(N \log N)$	$O(N \log N)$	$O(\log N)$*	No	No