

ECE 2400 Computer Systems Programming

Spring 2026

Topic 9: Sorting Algorithms

School of Electrical and Computer Engineering
Cornell University

revision: 2026-03-04-11-49

1	Insertion Sort	3
1.1.	Sorted Insert (Forward)	3
1.2.	Sorted Insert (Reverse)	3
1.3.	In-Place Insertion Sort	4
2	Merge Sort	5
2.1.	Merge	5
2.2.	Merge Sort	6
3	Quick Sort	10
3.1.	Partition	10
3.2.	In-Place Quick Sort	10
4	Comparing Sorting Algorithms	14

-
- We will explore a variety of different kinds of algorithms:
 - **Out-of-Place Algorithms:** Gradually copy elements from input array into a temporary array; by the end the temporary array is sorted; $O(N)$ heap space complexity
 - **In-Place Algorithms:** Keep all elements stored in the input array; use input array for intermediate results; no temporary storage is required; $O(1)$ heap space complexity
 - **Iterative Algorithms:** Use iteration statements to implement an iterative sorting strategy
 - **Recursive Algorithms:** Use recursion to implement a divide-and-conquer sorting strategy
 - For each algorithm, we will ...
 - start by exploring a helper function
 - use this helper function to implement a sorting function
 - For each function, we will use ...
 - cards to build intuition behind algorithm
 - **pseudocode to make algorithm more concrete**
 - complexity analysis

zyBooks The course zyBook also introduces *selection sort*, which is a very simple comparison sort, and *radix sort*, which is a non-comparison sort.

1. Insertion Sort

- sorted_insert helper function (forward and reverse variants)
- Call sorted_insert for every element in input array

1.1. Sorted Insert (Forward)

- Insert new element into sorted array such that array remains sorted
- Search array in the forward direction for correct location
- Once find correct location, insert value and push down rest of array

```
1 sorted_insert_fwd( x, first, last, v ):
2     y = v
3     for i in [first, last]:
4         if y < x[i]:
5             swap( x[i], y )
6     x[last] = y
```

1.2. Sorted Insert (Reverse)

- Insert new element into sorted array such that array remains sorted
- Search array in the reverse direction
- Keep swapping until value is in the correct location

```
1 sorted_insert_rev( x, first, last, v ):
2     x[last] = v
3     for i in last to first:
4         if x[i-1] > x[i]:
5             swap( x[i-1], x[i] )
6     else:
7         break
```

1.3. In-Place Insertion Sort

- Divide input array into sorted and unsorted partitions
- Use sorted insert to **insert** elements from unsorted to sorted partition
- Can use either the forward or reverse version of `sorted_insert`

```
1 insertion_sort_ip( x, n ):
2   for i in 0 to n:
3     sorted_insert_rev( x, 0, i, x[i] )
```

zyBooks The course zyBook includes coding labs to implement insertion sort on an array of integers and an array of strings, and to parameterize insertion sort with a comparison function pointer.

2. Merge Sort

- merge helper function
- Recursively divide array into partitions, merge sorted partitions

2.1. Merge

- Merge two *sorted* input arrays into separate output array
- Ensure output array is also sorted

```
1 merge( dst, a, first0, last0, b, first1, last1 ):
2     size = ( last0 - first0 ) + ( last1 - first1 )
3     assert len(dst) == size
4
5     idx0 = first0
6     idx1 = first1
7
8     for i in 0 to size:
9
10        # done with array a
11        if idx0 == last0:
12            dst[i] = b[idx1]
13            idx1 += 1
14
15        # done with array b
16        elif idx1 == last1:
17            dst[i] = a[idx0]
18            idx0 += 1
19
20        # front of array a is less than front of array b
21        elif a[idx0] < b[idx1]:
22            dst[i] = a[idx0]
23            idx0 += 1
24
25        # front of array b is less than front of array a
26        else:
27            dst[i] = b[idx1]
28            idx1 += 1
```

2.2. Merge Sort

- Recursively partition input array into halves
- Base case is when a partition contains a single element
- After recursive calls return, use merge to **merge** sorted partitions

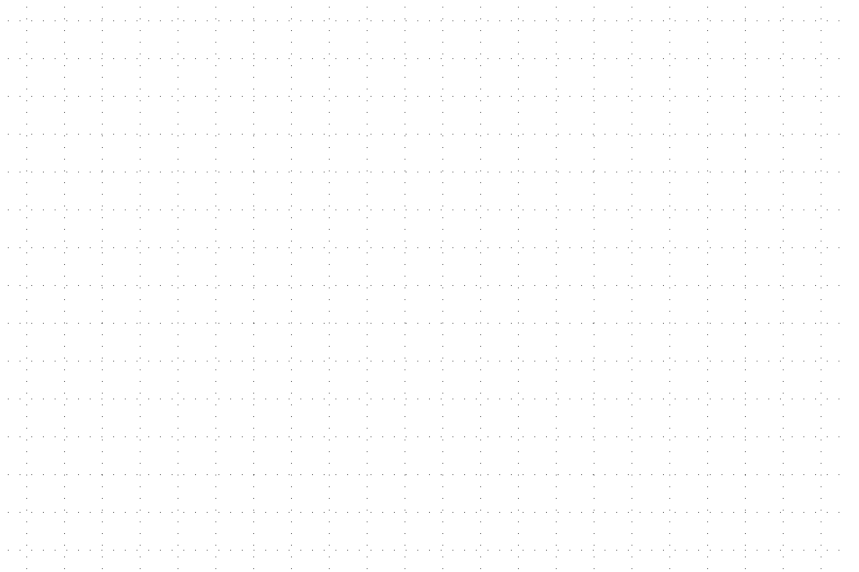
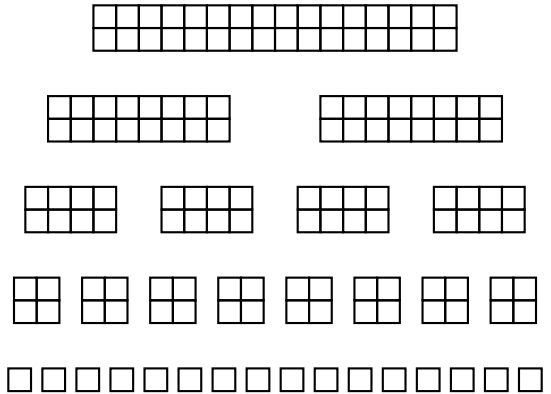
```
1 merge_sort_h( x, first, last ):
2
3   size = last - first
4   if size == 1:
5       return
6
7   mid = ( begin + end ) / 2
8   merge_sort_h( x, first, mid )
9   merge_sort_h( x, mid, last )
10
11  set tmp to an empty array with size elements
12  merge( tmp, x, first, mid, x, mid, last )
13
14  # copy temporary array to input array
15  j = 0
16  for i in first to last:
17      x[i] = tmp[j]
18      j += 1
19
20 merge_sort( x, n ):
21     merge_sort_h( x, 0, n )
```

- Show contents of `x` for each recursive call
- Show contents of `tmp` for each merge

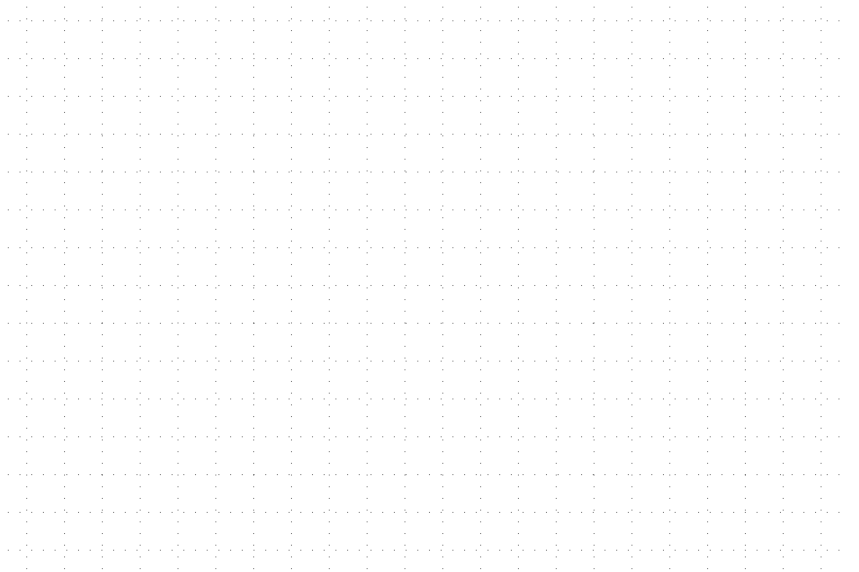
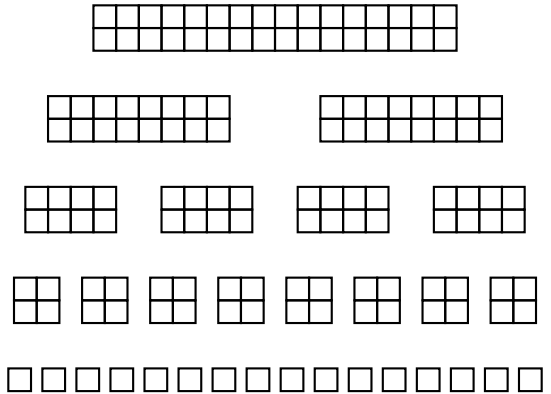
<code>x</code>	14	4	10	15	2	0	13	5	3	7	9	1	8	12	11	6
<code>tmp</code>																

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

- Time complexity analysis



- Space complexity analysis



3. Quick Sort

- Use `partition` helper function to recursively partition array

3.1. Partition

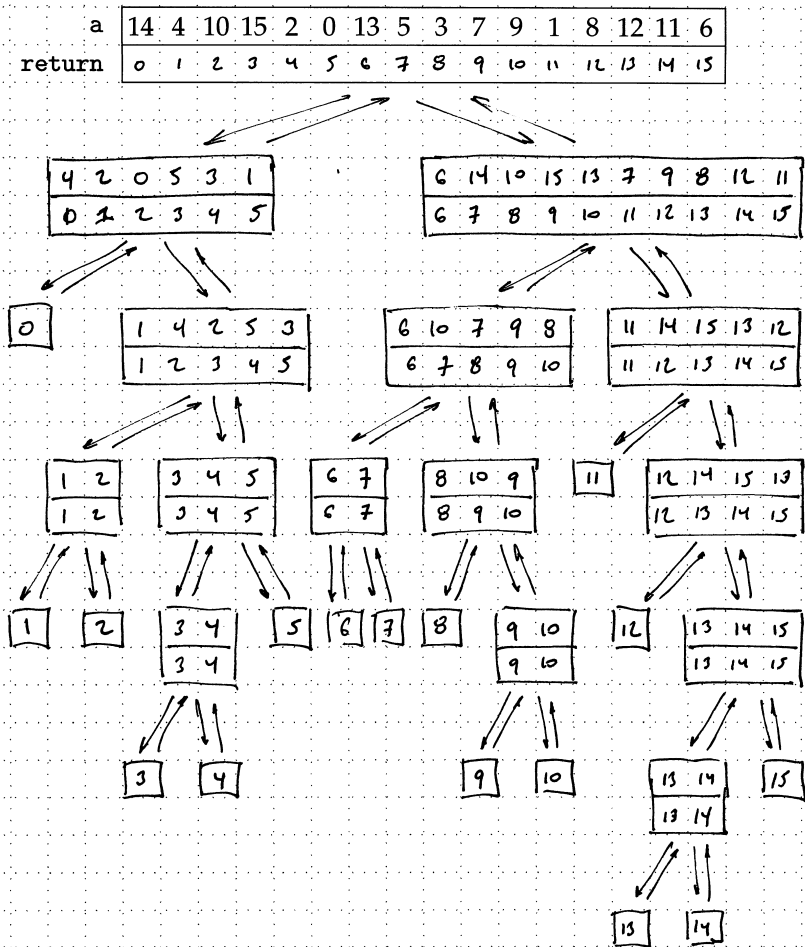
- Choose an element as the *pivot* and *partition* based on pivot
- Move all elements less than the pivot to front of the array
- Move all elements greater than the pivot to end of the array
- Pivot's final location is in between these two partitions

```
1 partition( x, first, last ):  
2     pivot = x[last-1]  
3     idx = first  
4     for i in first to last:  
5         if x[i] <= pivot:  
6             swap( x[i], x[idx] )  
7             idx += 1  
8     return idx-1
```

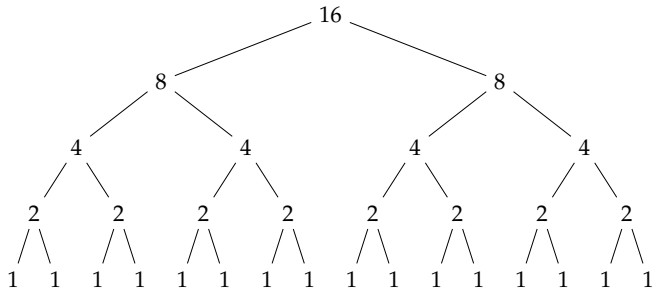
3.2. In-Place Quick Sort

- Recursively partition input array using `partition`
- Base case is when a partition contains a single element

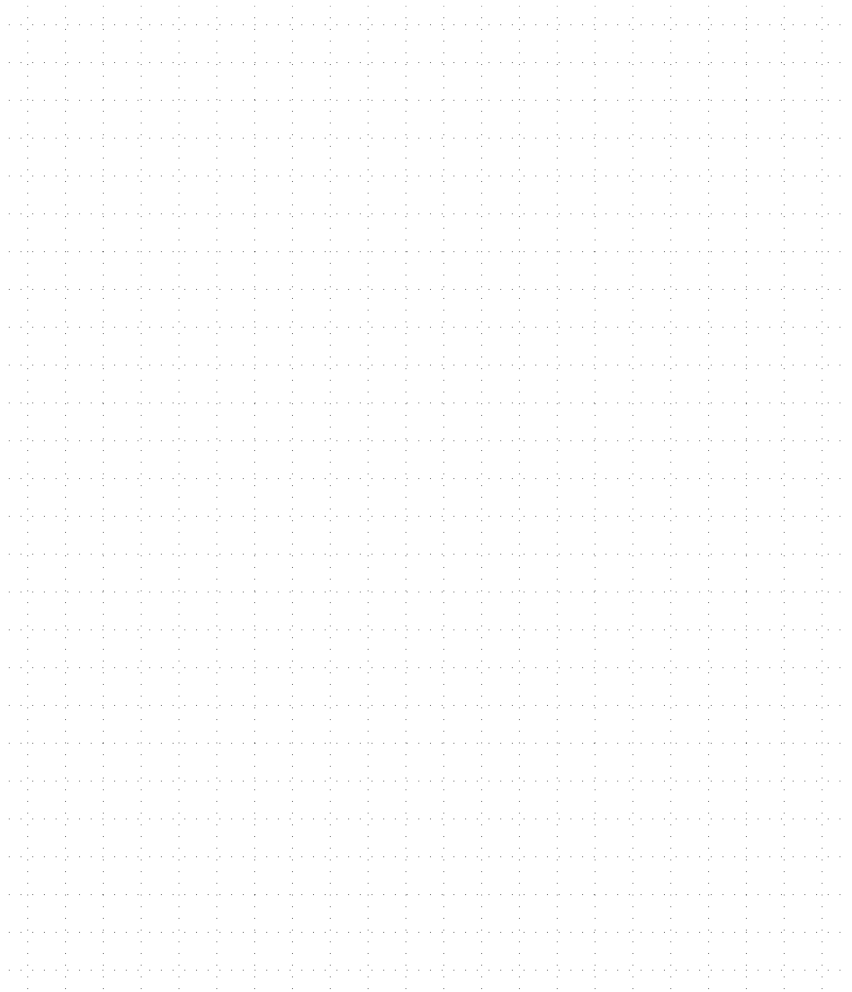
```
1 quick_sort_h( x, first, last ):  
2     size = last - first  
3     if size == 0 or size == 1:  
4         return  
5     p = partition( x, first, last )  
6     quick_sort_h( x, first, p )  
7     quick_sort_h( x, p, last )  
8  
9 quick_sort( x, n ):  
10    quick_sort_h( x, 0, n )
```



- Best-case time complexity analysis



- Worst-case time complexity analysis



4. Comparing Sorting Algorithms

Algorithm	Time Complexity			Comparison Sort?
	Best Case	Worst Case	Avg Case	
insertion (fwd)				
insertion (rev)				
selection				
merge				
quick				

Algorithm	Space Complexity	
	Avg Case Stack	Avg Case Aux Heap
insertion (fwd)		
insertion (rev)		
selection		
merge		
quick		