

Fundamental Algorithmic Techniques IV

February 5, 2026

Outline

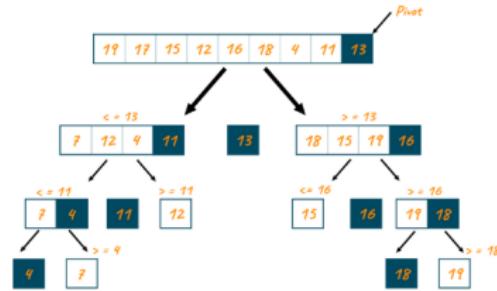
Quick Sort

Merge Sort

Heap Sort

Analysis of Sorting Algorithms

QuickSort



Quicksort Algorithm

```
1: function QUICKSORT( $A, p, r$ )
2:   if  $p < r$  then
3:      $q \leftarrow \text{PARTITION}(A, p, r)$ 
4:     QUICKSORT( $A, p, q - 1$ )
5:     QUICKSORT( $A, q + 1, r$ )
6:   end if
7: end function
```

MergeSort Algorithm

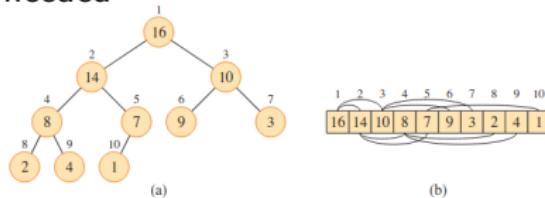
```
1: function MERGESORT( $A, p, r$ )
2:   if  $p < r$  then
3:      $q \leftarrow \lfloor (p + r)/2 \rfloor$ 
4:     MERGESORT( $A, p, q$ )
5:     MERGESORT( $A, q + 1, r$ )
6:     MERGE( $A, p, q, r$ )
7:   end if
8: end function

9: function MERGE( $A, p, q, r$ )
10:    $L \leftarrow A[p..q], R \leftarrow A[q + 1..r]$ 
11:    $i \leftarrow j \leftarrow 1$ 
12:   for  $k \leftarrow p$  to  $r$  do
13:     if  $j > |R|$  or ( $i \leq |L|$  and  $L[i] \leq R[j]$ ) then
14:        $A[k] \leftarrow L[i]; i \leftarrow i + 1$ 
15:     else
16:        $A[k] \leftarrow R[j]; j \leftarrow j + 1$ 
17:     end if
18:   end for
19: end function
```

HeapSort

Array \longleftrightarrow Complete Binary Tree

sorts in-place — no extra memory needed

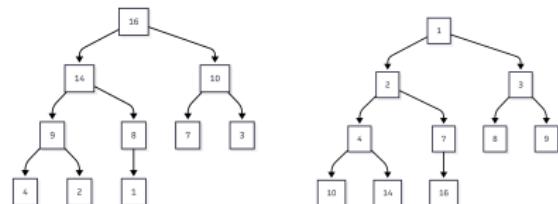


Root: index 1

- Parent(i) $\rightarrow \left\lfloor \frac{i}{2} \right\rfloor$
- Left(i) $\rightarrow 2i$
- Right(i) $\rightarrow 2i + 1$

Goal: Sorting

[1, 2, 3, 4, 7, 8, 9, 10, 14, 16] or
[16, 14, 10, 9, 8, 7, 4, 3, 2, 1]



2 operations on Tree:

- heapify or max/min heap
- swap

quick video link

Core Operations in Heapsort

heapify (max-heapify):

- Restores max-heap property after root removal
- Compares parent with children → swaps if needed
- Recurses **downward** toward leaves
- Takes $O(\log n)$ time (at most $2\lfloor \log_2 n \rfloor$ comparisons)

swap:

- Exchanges root ($A[0]$) with last element ($A[n - 1]$)
- Reduces heap size by 1
- $O(1)$ operation

Example: [3, 7, 1, 8, 2, 5, 9, 4, 6]

Analysis of Merge Sort

Simplest analysis for sorting algorithms!

$$T(n) = 2T(n/2) + \mathcal{O}(n)$$

- 2 subproblems of size $n/2$, $c_{\text{crit}} = \log_2 2 = 1$
- Work $f(n) = \mathcal{O}(n)$, $c = 1$

Applying the Master Theorem (balanced case $c_{\text{crit}} = c$):

$$T(n) = \Theta(n^{c_{\text{crit}}} \log n) = \Theta(n \log n)$$

Analysis of Quick Sort

$$T(n) = T(r-1) + T(n-r) + \mathcal{O}(n),$$

where $1 \leq r \leq n$ is the pivot position after partitioning.

Analysis:

- **Balanced:** $T(n) \approx 2T(n/2) + \mathcal{O}(n) \Rightarrow \mathcal{O}(n \log n)$
- **Unbalanced:** $T(n) \approx T(n-1) + \mathcal{O}(n) \Rightarrow \mathcal{O}(n^2)$
- **Average case:** Close to balanced $\Rightarrow \mathcal{O}(n \log n)$

Improved pivots: random selection or median-of-three (first, middle, last)

Analysis of Heap Sort

Master Theorem doesn't apply!

- Subproblems have **different sizes** (not n/b)
- General form: $T(n) = T(n - 1) + f(n)$, not $a \cdot T(n/b)$

Instead:

- 1 Sorting phase: $hs(n) = hs(n - 1) + \text{heapify}(n) + \mathcal{O}(1)$
- 2 Base: $hs(1) = \mathcal{O}(1)$
- 3 $\text{heapify}(i) = \mathcal{O}(\log i)$ (height of subtree)

$$hs(n) = \mathcal{O}(1) + \sum_{i=2}^n \mathcal{O}(\log i) = \mathcal{O}\left(\sum_{i=1}^n \log i\right) = \mathcal{O}(\log n!) = \Theta(n \log n)$$

(using $\log n! = \Theta(n \log n)$ via Stirling's approximation)

Problem Space Reduction

Space of permutations for array v of size n :

$$\approx n!$$

Idea: Reduce permutation space via divide-and-conquer transformations.

MergeSort heuristic: Merging two sorted subarrays of size $n/2$ reduces uncertainty by combining two independent orderings. With $\approx \log_2 n$ levels of recursion and $O(n)$ work per level:

$$O(n \log n)$$