

Fundamental Algorithmic Techniques III

October 10, 2025



Outline

HeapSort

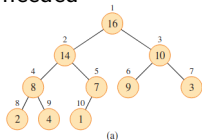
QuickSort

Analysis of sorting Algorithms

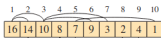
HeapSort

Array \longleftrightarrow Complete Binary Tree

sorts in-place — no extra memory needed



(a)



(b)

Root: index 1

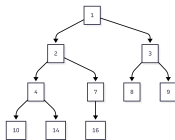
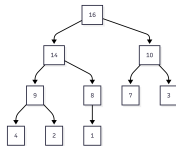
■ $\text{Parent}(i) \rightarrow \left\lfloor \frac{i}{2} \right\rfloor$

■ $\text{Left}(i) \rightarrow 2i$

■ $\text{Right}(i) \rightarrow 2i + 1$

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:

- Restores max-heap property after root removal
- Compares parent with children \rightarrow swaps if needed
- Recurses upward

■ $\log(n/2^{\text{level}})$ operations **swap:**

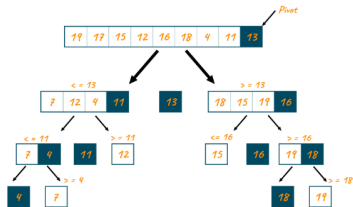
- Exchanges root ($A[0]$) with last element ($A[n - 1]$)
- Reduces heap size by 1
- $O(1)$ operation

example: 3,7,1,8,2,5,9,4,6

MergeSort

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

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**

Problem Space Reduction

space of permutations for array v of size n :

$$\approx n!$$

Idea: Reduce the permutation space with astute parallelised transformations!

Heuristics for Merge Sort: each transformation swapping any two neighbouring elements so that $v_i < v_{i+1}$ reduces possible permutation space by a factor 2.

There are $\approx \log_2 n$ such steps with $\leq n$ operations.

And so $\mathcal{O}(n \cdot \log n)$.

Analysis of Merge Sort

Simplest analysis for Sorting algorithms!

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

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

And so applying master theorem (balanced $c_{crit} = c$):

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

Analysis of Quick Sort

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

with $1 \leq r \leq n$ index of max/min.

Analysis:

- balanced: $T(n) \approx 2T(n) + \mathcal{O}(n)$, and so $\mathcal{O}(n \cdot \log(n))$.
- unbalanced: $T(n) \approx T(n1) + \mathcal{O}(n)$, and so $\mathcal{O}(n^2)$.
- average would be close to balanced: $\mathcal{O}(n \cdot \log(n))$,

Improved pivots: random or best of three (low, middle, up)

Analysis of Heap Sort

Master Theorem doesn't apply!

- Does not solve subproblem of same size
- **general form:** $T(n) = 2T(n-1) + f(n)$, not $a \cdot T(n/b)$!

Instead:

- 1 $hs(n) = hs(n-1) + \text{heapify}(n) + \mathcal{O}(1)$
- 2 $hs(1) = \mathcal{O}(1)$
- 3 $\text{heapify}(i) \approx \mathcal{O}(\log i)$ (a bit tricky)

$$hs(n) = \mathcal{O}(1) + \sum_{i=2}^n [\text{heapify}(i) + \mathcal{O}(1)] = \mathcal{O}(1) + \sum_{i=2}^n \mathcal{O}(\log i),$$

By Stirling's approximation: $\sum_{k=1}^n \log k = \log(n!) = \Theta(n \log n)$