Fundamental Algorithmic Techniques III

October 3, 2025

Outline

HeapSort

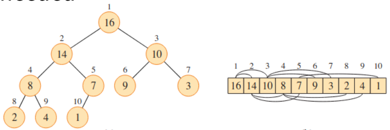
QuickSort

Analysis of sorting Algorithms

HeapSort

Array $\longleftrightarrow$ Complete Binary Tree

sorts in-place — no extra memory needed

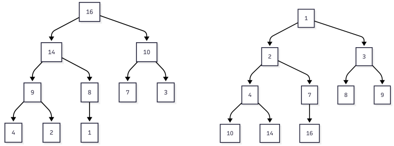


(a) (b)

Root: index 1

- 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:

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

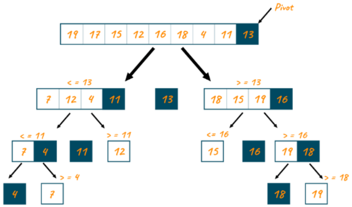
- log(n/2^{level})*operations* **swap**:
  - Exchanges root ($A[0]$) with last element ($A[n-1]$)
  - Reduces heap size by 1
  - $O(1)$ operation

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

MergeSort

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

QuickSort



Quicksort Algorithm

```
function $\textsc{Quicksort}(A, p, r)$
    if $p < r$ then
        $q \leftarrow \textsc{Partition}(A, p, r)$
        $\textsc{Quicksort}(A, p, q - 1)$
        $\textsc{Quicksort}(A, q + 1, r)$
    end if
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 swaping 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 $c_{crit} = c$):

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

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: $T(n) \approx T(n1) + O(n)$, 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. $hs(n) = hs(n-1) + heapify(n) + \mathcal{O}(1)$
2. $hs(1) = \mathcal{O}(1)$
3. heapify(i) $\approx \log i$ (a bit tricky)

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

So finally $hs(n) \leq \mathcal{O}(n \log n)$