

Sorting

20.1 Five Sorting Algorithms

Here is a too brief summary of 5 sorting algorithms. (Needs to be extended!)

Insertion Sort. Add elements one at a time from right, maintaining sorted list. (E.g. using an array.) Running time: $O(n)$ – $O(n^2)$

Shellsort. Given increment sequence $h_i = n/2^i$: in phase i , do insertion sort on array split into every h_i th element. Running time: $O(n \log n)$ – $O(n^2)$

Heap Sort. buildHeap, then repeatedly deleteMin. Running time: $\Theta(n \log n)$.

MergeSort.

1. Arbitrarily split
2. Call MergeSort on each half
3. Merge two sorted halves

Running time: $O(n \log n)$. Extra space needed, but sequential access enough.

QuickSort.

1. Pick pivot (e.g. first element or middle-of-three)
2. Partition array w.r.t. pivot
3. Call QuickSort on each piece

Running time: $O(n \log n)$ – $O(n^2)$

20.2 Lower Bound for Sorting

This is the idea behind the lower bound for time needed for sorting. Given an array, sorting entails determining the rank (1 to n) of every element. There are $n!$ possibilities for the list of ranks. Each operation (such as a comparison) reduces the number of possibilities by at best a factor of 2. So we need at least $\log_2(n!)$ steps to produce the one correct possibility. (The code can be thought of as a binary decision tree.) A mathematical fact is that $\log_2(n!)$ is approximately $n \log_2 n$.