

Sorting Algorithms Summary

In-place: means modifying list to be sorted [as opposed to returning new sorted list]. **Stable:** means the order of elements with equal values is preserved.

Lower bound on Comparion-based Sorting

For a comparison-based algorithm, the expected number of comparions is $\Omega(n \cdot \log(n))$.

Merge Sort

Sort list by merging sorted sub-lists, reduces total number of comparisons needed.

1. Divide list into equally sized halves until 1 element per sub-list
2. Sort sub-list [recursively]
3. Merge sub-list using comparator

- Comparison-based
- **Not in-place**
- **Stable**
- **Runs in $O(n \cdot \log(n))$ time**

// performs at most $n \cdot \log(n)$ comparisons

Heap Sort

Sort by traversing down a heap tree.

1. Create a heap from list

2. Swap first and last nodes [swap in list too]
 - first node is root
 - last node is smallest leaf
3. Heapify [make sure heap property is preserved]
4. Repeat steps 2–4 until no more elements to sort

- Comparison-based
- **In-place**
- **Not stable**
- **Runs in $O(n \log(n))$ time**

// performs at most $2n \cdot \log(n) + O(n)$ comparisons

Quick Sort

Sort using a randomly selected value as partition point, sorting sub-lists. Since random selection, might choose worst value [ideally middle value].

1. Randomly select value
2. Add all values less than to left sub-list, otherwise add to right sub-list
3. Repeat until 1 element in sub-list

- Comparison-based
- **In-place**
- **Not stable**
- **Runs in $O(n \log(n))$ [expected] time**

// performs at most $2n \cdot \log(n) + O(n)$

Comparison-based Algorithms

| | Comparisons | In-place | Stable |
|------------|--------------------------------|----------|--------|
| Merge Sort | $n \cdot \log(n)$ [worst-case] | no | yes |

| | | | |
|------------|---|-----|----|
| Heap Sort | $1.38n \cdot \log(n) + O(n)$ [expected] | yes | no |
| Quick Sort | $2n \cdot \log(n) + O(n)$ [worst-case] | yes | no |