Sorting Algorithims

Ryan Hallock

September 1, 2023

1 Merge Sort

Merge sort is considered the conquer and divide sorting method. It gets slightly complicated to implement with non even array sizes.

1.1 Steps

Divide It divides the array down into two equal sized sub-arrays. It keeps doing this until the the array has one.

Conquer The sub arrays are sorted individually.

Merge The sorted sub arrays are then merged back into the final array. The smaller element is placed into an array.

2 Quicksort

Quicksort also uses the conquer and divide strategy but is better on space complexity. It uses pivots to split the array and uses recursion to sort. Not stable, can change equal elements.

2.1 Steps

Pivot A pivot could be anywhere in the table, most of the time it is the in the middle of the array.

Splitting the Array We then split the array based of where the pivot is.

Recursion We then go back to step pivot to split the array down like MergeSort.

Combine Using the stack the array returned after the recursion is the sorted array.

3 Heapsort

Heap Sort is not known to be stable, meaning it can change equal elements. It uses a binary heap to sort the array, to efficently sort the array.

3.1 Steps

Building the Heap The first step is to build the heap. The binary heap is just a binary tree. When creating the heap the parent node is greater than or equal to its child nodes. Creating a tree pattern.

Using the Heap So we created the heap, to use the heap, usually the top element on max heap is the root element. Then we the last child and swap with it. Removing the root from the tree.

Repeat We keep rebuilding the heap again until we have our sorted array, when the heap has only one element left.

4 Big O Notation

O notation is used to denote complexity. It compared running complexity to input size. $O(f(n))$ is the formula. The more flat on a graph n is the better it is, most of the time. Big O notation does not take in account time, only time complexity. So if you have a 100ms time to start with a O(log n), it would take longer to run than a 1ms O(n!) with a 1ms startup time, with small sample size.

5 Overall Time Complexity

1. Merge Sort - $O(n\ log\ n)$
2. Quicksort - $O(n\ log\ n)$ on average, worst case $O(n^2)$
3. Heapsort - $O(n\ log\ n)$

6 Overall Space Complexity

1. Merge Sort - $O(n)$
2. Quicksort - $O(log\ n)$
3. Heapsort - $O(1)$

7 What One?

You should use the best one for your usecase. Out of all of these only Merge Sort is stable, so if you care about the equal elements, you cant use the other two. If you care about memory use, you should use Heapsort as it is inline, or have alot of data. If you care about space and time complexity (best case), you should use Quicksort.