Mergesort

Merge sort is a divide and conquer algorithm. Think of it in terms of 3 steps -

1. The divide step computes the midpoint of each of the sub-arrays. Each of this step just takes O(1) time.
2. The conquer step recursively sorts two subarrays of n/2 (for even n) elements each.
3. The merge step merges n elements which takes O(n) time.

Now, for steps 1 and 3 i.e. between O(1) and O(n), O(n) is higher. Let's consider steps 1 and 3 take O(n) time in total. Say it is cn for some constant c.

How many times are these steps executed?

For this, look at the tree below - for each level from top to bottom Level 2 calls merge method on 2 sub-arrays of length n/2 each. The complexity here is 2 * (cn/2) = cn Level 3 calls merge method on 4 sub-arrays of length n/4 each. The complexity here is 4 * (cn/4) = cn and so on ...

Now, the height of this tree is (logn + 1) for a given n. Thus the overall complexity is (logn + 1)*(cn). That is O(nlogn) for the merge sort algorithm.

Merge takes linear time. Height of the 'tree' is logn. Merge function merges the arrays logn times. Hence, nlogn.

Andrew Silverman · Follow

Principal PM, Azure Hardware Architecture, Microsoft · 1y

Merge sort breaks down the n-sized input into the smallest possible groups (item size 1) and then merges two of those groups together in sorted order. The time to perform the merge is linear with the combined size of all of the groups, so after this first merge pass where each group only contains one item, you have consumed time that is directly proportional to n.

However, you must now repeat the merge operation with the resulting groups, which are now of size 2. This takes another n to complete for all of the groups, so we've now done 2n total time. So how many times do we have to repeat this merge operation to wind up with a single list that contains the entire original data set in sorted order? Well, the lists get twice as large in each pass, so we must perform log n passes to get back to having a single list. Log n * n per pass gives us O(n log n).

Quicksort

Sorting is a way of arranging items in a systematic manner. Quicksort is the widely used sorting algorithm that makes **n log n** comparisons in average case for sorting an array of n elements. It is a faster and highly efficient sorting algorithm. This algorithm follows the divide and conquer approach. Divide and conquer is a technique of breaking down the algorithms into subproblems, then solving the subproblems, and combining the results back together to solve the original problem.

Divide: In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element.

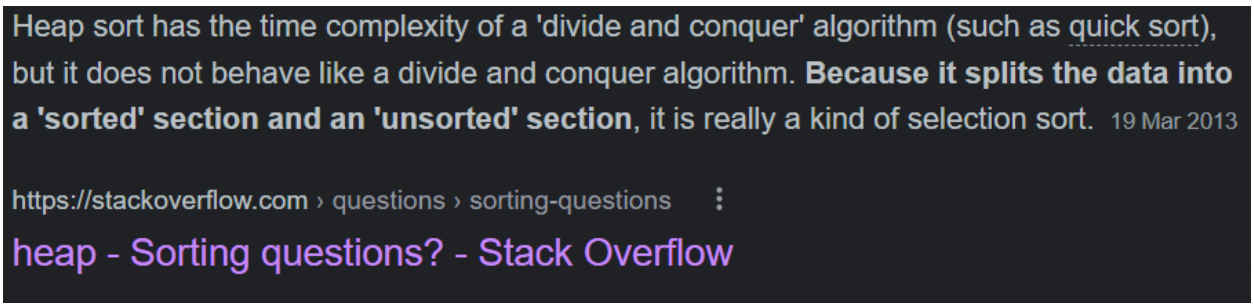
Conquer: Recursively, sort two subarrays with Quicksort.

Combine: Combine the already sorted array.

Quicksort picks an element as pivot, and then it partitions the given array around the picked pivot element. In quick sort, a large array is divided into two arrays in which one holds values that are smaller than the specified value (Pivot), and another array holds the values that are greater than the pivot.

After that, left and right sub-arrays are also partitioned using the same approach. It will continue until the single element remains in the sub-array.

Heapsort



Complexity of Heapsort

Heapsort has a running time of $O(n \log n)$.

Building the max-heap from the unsorted list requires $O(n)$ calls to the `max_heapify` function, each of which takes $O(\log n)$ time. Thus, the running time of `build_heap` is $O(n \log n)$.

Note: While it is true that `build_heap` has a running time of $O(n \log n)$, a tighter bound of $O(n)$ can be proved by analyzing the height of the tree where `max_heapify` is called. However, this does not change the overall running time of heapsort, and since the explanation of this is quite involved, it has been omitted.

Heapsort has a running time of $O(n \log n)$ since the call to `build_heap` takes $O(n \log n)$ time, and each of the $O(n)$ calls to `max_heapify` takes $O(\log n)$ time.

Heapsort has a worst- and average-case running time of $O(n \log n)$ like mergesort, but heapsort uses $O(1)$ auxiliary space (since it is an in-place sort) while mergesort takes up $O(n)$ auxiliary space, so if memory concerns are an issue, heapsort might be a good, fast choice for a sorting algorithm. Quicksort has an average-case running time of $O(n \log n)$ but has notoriously better constant factors, making quicksort faster than other $O(n \log n)$-time sorting algorithms. However, quicksort has a worst-case running time of $O(n^2)$ and a worst-case space complexity of $O(\log n)$, so if it is very important to have a fast worst-case running time and efficient space usage, heapsort is the best option. Note, though, that heapsort is slower than quicksort on average in most cases.

Pros and Cons

Pros

- Time-efficient with time complexity of $O(n \log n)$
- Less memory usage
- Consistent performance: it performs equally well in best-, average-, and worst-case scenarios

Cons

- Unstable sort
- External sorting not possible with heapsort

Comparisons

Is insertion better than merge sort?

Insertion Sort is preferred for fewer elements. It becomes fast when data is already sorted or nearly sorted because it skips the sorted values. Efficiency: Considering average time complexity of both algorithm we can say that Merge Sort is efficient in terms of time and Insertion Sort is efficient in terms of space. 8 Feb 2022

What is the difference between insertion and selection sort?

Insertion sort is a simple sorting algorithm that builds the final sorted list by transferring one element at a time. Selection sort, in contrast, is a simple sorting algorithm that repeatedly searches remaining items to find the smallest element and moves it to the correct location. 10 May 2019

Stable sorting algorithms preserve the relative order of equal elements, while unstable sorting algorithms don't. In other words, stable sorting maintains the position of two equals elements relative to one another. 23 Jun 2022

What is stable sorting algorithm with example?

Some examples of stable algorithms are **Merge Sort, Insertion Sort, Bubble Sort and Binary Tree Sort**. While, QuickSort, Heap Sort, and Selection sort are the unstable sorting algorithm. If you remember, Collections. sort() method from Java Collection framework uses iterative merge sort which is a stable algorithm.

28 Jun 2017

What is stable and unstable sort give examples?

Some examples of stable algorithms are **Merge Sort, Insertion Sort, Bubble Sort, and Binary Tree Sort**. While, QuickSort, Heap Sort, and Selection sort are unstable sorting algorithms. If you remember, the Collections. sort() method from the Java Collection framework uses iterative merge sort which is a stable algorithm.

What is stability in sorting algorithms?

Stable sorting algorithms **maintain the relative order of records with equal keys (i.e. values)**. That is, a sorting algorithm is stable if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list.