Algorithm

09 June 2021 07:07

1. T(1) = 0 .. Last single element -> 0 complexity
2. In-place sorting - constant extra space to perform sorting
3. Stability of sorting algo
   A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted.
4. Time complexity - number of times algo instructions are executed as a function of input number of args
   I
5. Space complexity - memory used by algo as a function of input number of args
   I
6. Asymptotic Analysis, we evaluate the performance of an algorithm in terms of input size
   https://mathworld.wolfram.com/Asymptotic.html
   Algo is said to have complexity of
   O(g(n)) : if f(n) i.e. time-complexity of algo has asymptotic upper bound of c*g(n)
   Θ(g(n)) : if f(n) i.e. time-complexity of algo has asymptotic upper bound of c1*g(n) and lower bound of c2*g(n)
   Ω(g(n)) : if f(n) i.e. time-complexity of algo has asymptotic lower bound of c*g(n)
7. Solving for recurrences
   Substitution
   https://cs.fit.edu/~pkc/classes/writing/hw14/luis.pdf

   Recurrence-Tree
   Draw recurrence tree using recurrence equation & sum all levels (everything)
   https://tildesites.bowdoin.edu/~ltoma/teaching/cs231/fall15/Lectures/02-recurrences/recurrences.pdf

   Master
   https://web.stanford.edu/class/archive/cs/cs161/cs161.1168/lecture3.pdf
   T(n) = a T(n/b) + f(n) ... a >=1 b>1

   If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

   If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.

   If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■
8. Common recurrences

   T(n) = T(n/2) + n --------> O(nlogn)

   - Logarithmic: $\Theta(\log n)$
     - Recurrence: $T(n) = 1 + T(n/2)$
     - Typical example: Recurse on half the input (and throw half away)
     - Variations: $T(n) = 1 + T(99n/100)$
   - Linear: $\Theta(N)$
     - Recurrence: $T(n) = 1 + T(n-1)$
     - Typical example: Single loop
     - Variations: $T(n) = 1 + 2T(n/2), T(n) = n + T(n/2), T(n) = T(n/5) + T(7n/10 + 6) + n$
   - Quadratic: $\Theta(n^2)$
     - Recurrence: $T(n) = n + T(n-1)$
     - Typical example: Nested loops
   - Exponential: $\Theta(2^n)$
     - Recurrence: $T(n) = 2T(n-1)$

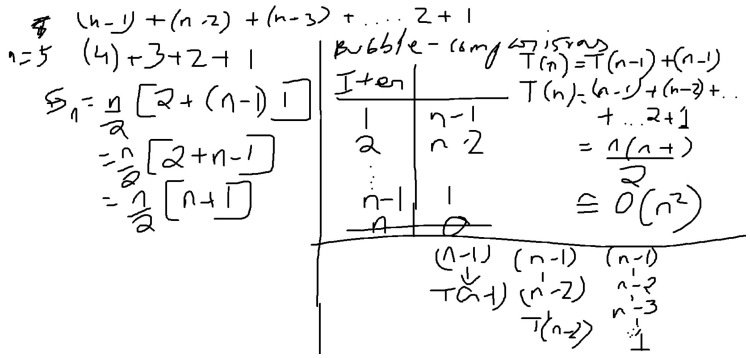
   T(n) = 8T(n/2) + n^2 ---------> O(n3)
9. Summary

	Tcomplexity Unsorted	Tcomplexity Sorted	Space Complexity	Inplace	Stable				
Bubble	O(n2)	O(n) - if optimized	1	Y	Y				
Selection	O(n2)	O(n2)	1	Y	No				
Insertion	O(n2)	O(n)	1	Y	Y				
Quick	O(n2)	O(nlogn)	O(log n)	Y	No				
Merge	O(nlogn)	O(nlogn)	O(n)	No	Y				
Heap	O(nlogn)	O(nlogn)	O(1)	Y	No				

I

10. Bubble

It can be optimized by stopping the algorithm if inner loop didn't cause any swap.

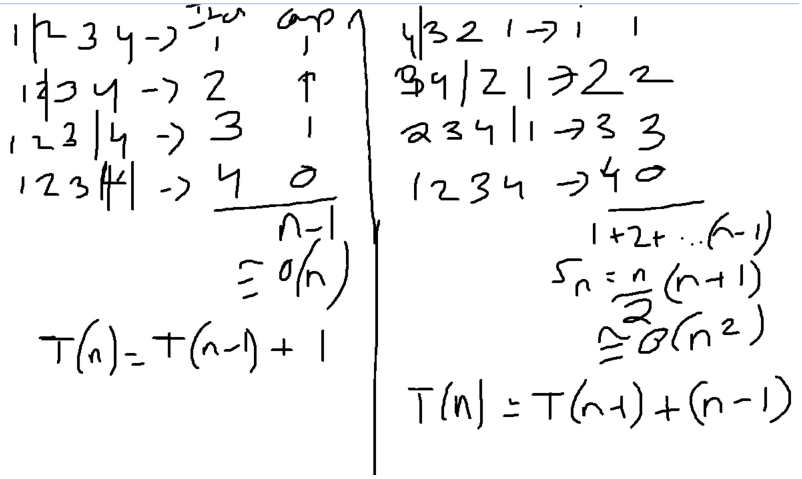


11. Selection

Time complexity analysis same as bubble
Stable selection sort -

12. Insertion

I

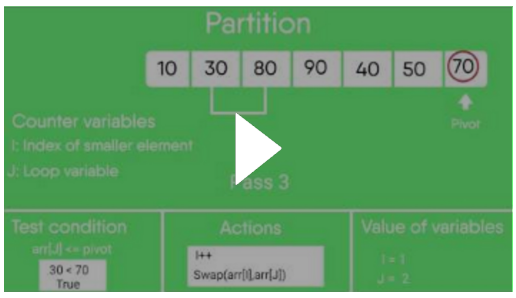


13. Quick

Pivots can be

1. Last
2. Median
3. Random

Quick Sort | GeeksforGeeks



https://tildesites.bowdoin.edu/~ltoma/teaching/cs231/fall15/Lectures/04-quicksort/quicksort.pdf

Median of the list is chosen as pivot
T(1) = 0 .. Last single element -> 0 complexity

The complexity of best-case Quick Sort is:

$$T(N)=2T(N/2)+N-1 \quad (4.1)$$
$$T(N)=2[2T(N/4)+N/2-1]+N-1 \quad (4.2)$$
$$T(N)=4[2T(N/8)+N/4-1]+2N-3 \quad (4.3)$$
$$T(N)=8T(N/8)+N+N+N-4-2-1 \quad (4.4)$$

$$T(N) = 2^k T(\frac{N}{2^k}) + kN - (2^k - 1) \quad (4.5)$$

$$T(1) = 0 \quad (4.6)$$

$$2^k = N \quad (4.7)$$

$$k = \log_2 N \quad (4.8)$$

$$T(N) = N \log_2 N - N + 1 \quad (4.9)$$

4.2.2 Worst Case

Quick Sort's best case is when the chosen pivot is either the largest or smallest element of the list. When this happens, one of the two sublists will be empty, so Quick Sort is only called on one list during the ***Sort*** step.

$$T(N) = T(N-1) + N - 1 \quad (4.10)$$

$$T(N) = T(N-2) + N - 1 + N - 2 \quad (4.11)$$

$$T(N) = T(N-3) + 3N - 1 - 2 - 3 \quad (4.12)$$

$$T(N) = T(1) + \sum_{i=0}^{N-1} (N-i) \quad (4.13)$$

$$T(N) = \frac{N(N-1)}{2} \quad (4.14)$$

14. Merge-Sort

- Let $T(n)$ denote the worst-case running time of mergesort on an array of n elements. We have:

$$\begin{aligned} T(n) &= c_1 + T(n/2) + T(n/2) + c_2 n \\ &= 2T(n/2) + (c_1 + c_2 n) \end{aligned}$$

- Simplified, $T(n) = 2T(n/2) + \Theta(n)$
- We can see that the recurrence has solution $\Theta(n \log_2 n)$ by looking at the **recursion tree**: the total number of levels in the recursion tree is $\log_2 n + 1$ and each level costs linear time (more below).

15. Heap

Better attributes of merge (space complexity lesser) and insertion (O(nlogn) > O(n2))
https://tildesites.bowdoin.edu/~ltoma/teaching/cs231/fall15/Lectures/03-heaps/heaps.pdf
Heap Sort + Step by Step Coding + Time and Space Complexity + Priority Queue - CS Lecture Ep 6

```
BUILD-MAX-HEAP:
        for i = length/2; i >= 1; i--
                MAX-HEAPIFY(i)
```

```
HEAP-SORT(A):
        BUILD-MAX-HEAP(A)          O(n)
        for i = length; i > 1; i--  O(n)
            swap(A[1], A[i])        O(1)
            A.heap_size--;          O(1)
            MAX-HEAPIFY(A, 1)       O(lgn)
```

Time complexity: O(n*lgn)

Time complexity:

- **Best case: O(nlogn),**
- **Worst case: O(nlogn),**
- **Average case: O(nlogn).**

Space complexity: O(1)

16. DFS
    Depth-first search in 4 minutes
17. BFS
    Breadth-first search in 4 minutes
18. D
19. D
20. D
21. D
22. D
23. D
24. D
25. d

I