Time Complexity comparison of Sorting Algorithms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Data Structure** | **Time Complexity** | | |
|  |  | **Best** | **Average** | **Worst** |
| [Quicksort](http://scanftree.com/Data_Structure/Quick-Sort) | Array | O(n log(n)) | O(n log(n)) | O(n^2) |
| [Mergesort](http://scanftree.com/Data_Structure/Merge-sort) | Array | O(n log(n)) | O(n log(n)) | O(n log(n)) |
| [Heapsort](http://scanftree.com/Data_Structure/Heap-sort) | Array | O(n log(n)) | O(n log(n)) | O(n log(n)) |
| [Bubble Sort](http://scanftree.com/Data_Structure/bubble-sort) | Array | O(n) | O(n^2) | O(n^2) |
| [Insertion Sort](http://scanftree.com/Data_Structure/Insertion-sort) | Array | O(n) | O(n^2) | O(n^2) |
| [Select Sort](http://en.wikipedia.org/wiki/Selection_sort) | Array | O(n^2) | O(n^2) | O(n^2) |
| [Bucket Sort](http://scanftree.com/Data_Structure/bucket-Sort) | Array | O(n+k) | O(n+k) | O(n^2) |
| [Radix Sort](http://scanftree.com/Data_Structure/radix-sort) | Array | O(nk) | O(nk) | O(nk) |

Space Complexity comparison of Sorting Algorithms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Data Structure** | **Worst Case Auxiliary Space Complexity** | | |
|  |  |  | Worst case space complexity |  |
| [Quicksort](http://scanftree.com/Data_Structure/Quick-Sort) | Array | O(n) |  |  |
| [Mergesort](http://scanftree.com/Data_Structure/Merge-sort)/bucket sort | Array | O(n) | /O(n\*K) |  |
| [Heapsort](http://scanftree.com/Data_Structure/Heap-sort)/Radix sort | Array | O(1) | /O(K N) |  |
| [Bubble Sort](http://scanftree.com/Data_Structure/bubble-sort) | Array | O(1) |  |  |
| [Insertion Sort](http://scanftree.com/Data_Structure/Insertion-sort) | Array | O(1) | O(n) total, O(1) auxiliary |  |
| [Select Sort](http://en.wikipedia.org/wiki/Selection_sort) | Array | O(1) |  |  |
| [Bucket Sort](http://scanftree.com/Data_Structure/bucket-Sort) | Array | O(nk) |  |  |
| [Radix Sort](http://scanftree.com/Data_Structure/radix-sort) | Array | O(n+k) |  |  |

What *Operations* Should We Count?

* Must do a detailed counting of *all* executed operations.
* Estimate number of primitive operations that RAM model (basic microprocessor) must do to run program:
  + –  Basic operation: arithmetic/logical operations count as 1 operation. • even though adds, multiplies, and divides don’t (usually) cost the same
  + –  Assignment: counts as 1 operation.  
    • operation count of righthand side of expression is determined separately.
  + –  Loop: number of operations per iteration \* number of loop iterations.
  + –  Method invocation: number of operations executed in the invoked method.

• ignore costs of building stack frames, ...

* + –  Recursion: same as method invocation, but harder to reason about.
  + –  Ignoring garbage collection, memory hierarchy (caches), ...

Big-Oh rules

If is f(n) a polynomial of degree d, then f(n) is O(nd), i.e.,

1. Drop lower-order terms

2. Drop constant factors

Use the smallest possible class of functions

Say “2n is O(n)” instead of “2n is O(n2)” Use the simplest expression of the class

Say “3n + 5 is O(n)” instead of “3n + 5 is O(3n)”

Asymptotic Algorithm analysis

The asymptotic analysis of an algorithm determines the running time in big-Oh notation

To perform the asymptotic analysis

 We find the worst-case number of primitive operations executed as a function of the input size



 We express this function with big-Oh notation Example:

 We determine that algorithm arrayMax executes at most 7n −1 primitive operations

 We say that algorithm arrayMax “runs in O(n) time”



Since constant factors and lower-order terms are eventually dropped anyhow, we can disregard them when counting primitive operations