Best, worst and average case

From Wikipedia, the free encyclopedia

In computer science, **best**, **worst,** and **average cases** of a given algorithm express what the resource usage is *at least*, *at most* and *on average*, respectively. Usually the resource being considered is running time, i.e. time complexity, but it could also be memory or other resource.

In real-time computing, the worst-case execution time is often of particular concern since it is important to know how much time might be needed *in the worst case* to guarantee that the algorithm will always finish on time.

Average performance and worst-case performance are the most used in algorithm analysis. Less widely found is best-case performance, but it does have uses: for example, where the best cases of individual tasks are known, they can be used to improve the accuracy of an overall worst-case analysis. Computer scientists use probabilistic analysis techniques, especially expected value, to determine expected running times.

The terms are used in other contexts; for example the worst- and best-case outcome of a planned-for epidemic, worst-case temperature to which an electronic circuit element is exposed, etc. Where components of specified tolerance are used, devices must be designed to work properly with the worst-case combination of tolerances and external conditions.

Contents

Best-case performance for algorithm

The term *best-case performance* is used in computer science to describe an algorithm's behavior under optimal conditions. For example, the best case for a simple linear search on a list occurs when the desired element is the first element of the list.

Development and choice of algorithms is rarely based on best-case performance: most academic and commercial enterprises are more interested in improving Average-case complexity and worst-case performance. Algorithms may also be trivially modified to have good best-case running time by hard-coding solutions to a finite set of inputs, making the measure almost meaningless.[1]

Worst-case versus average-case performance

Worst-case performance analysis and average-case performance analysis have some similarities, but in practice usually require different tools and approaches.

Determining what *best input* means is difficult, and often that average input has properties which make it difficult to characterise mathematically (consider, for instance, algorithms that are designed to operate on strings of text). Similarly, even when a sensible description of a particular "average case" (which will probably only be applicable for some uses of the algorithm) is possible, they tend to result in more difficult analysis of equations.

Worst-case analysis has similar problems: it is typically impossible to determine the exact worst-case scenario. Instead, a scenario is considered such that it is at least as bad as the worst case. For example, when analysing an algorithm, it may be possible to find the longest possible path through the algorithm (by considering the maximum number of loops, for instance) even if it is not possible to determine the exact input that would generate this path (indeed, such an input may not exist). This gives a *safe* analysis (the worst case is never underestimated), but one which is *pessimistic*, since there may be no input that would require this path.

Alternatively, a scenario which is thought to be close to (but not necessarily worse than) the real worst case may be considered. This may lead to an *optimistic* result, meaning that the analysis may actually underestimate the true worst case.

In some situations it may be necessary to use a pessimistic analysis in order to guarantee safety. Often however, a pessimistic analysis may be too pessimistic, so an analysis that gets closer to the real value but may be optimistic (perhaps with some known low probability of failure) can be a much more practical approach.

When analyzing algorithms which often take a small time to complete, but periodically require a much larger time, amortized analysis can be used to determine the worst-case running time over a (possibly infinite) series of operations. This **amortized worst-case** cost can be much closer to the average case cost, while still providing a guaranteed upper limit on the running time.

The worst-case analysis is related to the worst-case complexity.[2]

Practical consequences

Many problems with bad worst-case performance have good average-case performance. For problems we want to solve, this is a good thing: we can hope that the particular instances we care about are average. For cryptography, this is very bad: we want typical instances of a cryptographic problem to be hard. Here methods like random self-reducibility can be used for some specific problems to show that the worst case is no harder than the average case, or, equivalently, that the average case is no easier than the worst case.

On the other hand, some algorithms like hash tables have very poor worst case behaviours, but a well written hash table of sufficient size will statistically never give the worst case; the average number of operations performed follows an exponential decay curve, and so the run time of an operation is statistically bounded.

Examples

Sorting algorithms

Algorithm	Data structure	Time complexity:Best	Time complexity:Average	Time complexity:Worst	Space complexity:Worst
Quick sort	Array	O(n log(n))	O(n log(n))	O(n^2)	O(n)
Merge sort	Array	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Heap sort	Array	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Smooth sort	Array	O(n)	O(n log(n))	O(n log(n))	O(1)
Bubble sort	Array	O(n)	O(n^2)	O(n^2)	O(1)
Insertion sort	Array	O(n)	O(n^2)	O(n^2)	O(1)
Selection sort	Array	O(n^2)	O(n^2)	O(n^2)	O(1)

- Insertion sort applied to a list of n elements, assumed to be all different and initially in random order. On average, half the elements in a list A_1 ... A_j are less than elementA_{j+1}, and half are greater. Therefore, the algorithm compares the (j + 1)th element to be inserted on the average with half the already sorted sub-list, so $t_j = j/2$. Working out the resulting average-case running time yields a quadratic function of the input size, just like the worst-case running time.
- Quicksort applied to a list of n elements, again assumed to be all different and initially in random order. This popular sorting algorithm has an average-case performance of O(n log(n)), which contributes to making it a very fast algorithm in practice. But given a worst-case input, its performance degrades to O(n^2). Also, when not implemented with the "shortest first" policy, the worst-case space complexity degrades to O(log(n)).

Data structures

Data structure	Time complexity								Space complexity
	Avg: Indexing	Avg: Search	Avg: Insertion	Avg: Deletion	Worst: Indexing	Worst: Search	Worst: Insertion	Worst: Deletion	Worst
Basic Array	O(1)	O(1)	—	—	O(1)	O(n)	—	—	O(n)
Dynamic array	O(1)	O(n)	O(n)	—	O(1)	O(n)	O(n)	—	O(n)
Singly linked list	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly linked list	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Hash table	—	O(1)	O(1)	O(1)	—	O(n)	O(n)	O(n)	O(n)
Binary search tree	—	O(log (n))	O(log (n))	O(log (n))	—	O(n)	O(n)	O(n)	O(n)
B-tree	—	O(log (n))	O(log (n))	O(log (n))	—	O(log (n))	O(log (n))	O(log (n))	O(n)
Red-black tree	—	O(log (n))	O(log (n))	O(log (n))	—	O(log (n))	O(log (n))	O(log (n))	O(n)
AVL tree	—	O(log (n))	O(log (n))	O(log (n))	—	O(log (n))	O(log (n))	O(log (n))	O(n)

- Linear search on a list of n elements. In the absolute worst case, the search must visit every element once. This happens when the value being searched for is either the last element in the list, or is not in the list. However, on average, assuming the value searched for is in the list and each list element is equally likely to be the value searched for, the search visits only $n/2$ elements.

See also

- Sorting algorithm – an area where there is a great deal of performance analysis of various algorithms.
- Search data structure – any data structure that allows the efficient retrieval of specific items
- Worst-case circuit analysis
- Smoothed analysis
- Interval finite element
- Big O notation

References

1. Introduction to Algorithms (Cormen, Leiserson, Rivest, and Stein) 2001, Chapter 2 "Getting Started".In Best-case complexity, it gives the lower bound on the running time of the algorithm of any instances of input.
2. Worst-case complexity (http://www.fsz.bme.hu/~szirmay/ray6.pdf)

- http://bigocheatsheet.com/
- http://www.algorithmiccomplexity.com/