





WIKIPEDIA
The Free Encyclopedia

[Main page](#)
[Contents](#)
[Featured content](#)
[Current events](#)
[Random article](#)
[Donate to Wikipedia](#)
[Wikipedia store](#)

Interaction
[Help](#)
[About Wikipedia](#)
[Community portal](#)
[Recent changes](#)
[Contact page](#)

Tools
[What links here](#)
[Related changes](#)
[Upload file](#)
[Special pages](#)
[Permanent link](#)
[Page information](#)
[Wikidata item](#)
[Cite this page](#)

Print/export
[Create a book](#)
[Download as PDF](#)
[Printable version](#)


Languages 
[Čeština](#)
[Deutsch](#)
[فارسی](#)
[Русский](#)
[Српски / srpski](#)
[ไทย](#)
[Українська](#)
 [Edit links](#)

[Create account](#) [Log in](#)

Article [Talk](#)

[Read](#) [Edit](#)

[More](#) ▾



Interpolation search

From Wikipedia, the free encyclopedia

Interpolation search (sometimes referred to as **extrapolation search**) is an [algorithm](#) for [searching](#) for a given key value in an indexed array that has been [ordered](#) by the values of the key. It parallels how humans search through a telephone book for a particular name, the key value by which the book's entries are ordered. In each search step it calculates where in the remaining [search space](#) the sought item might be, based on the key values at the bounds of the search space and the value of the sought key, usually via a linear interpolation. The key value actually found at this estimated position is then compared to the key value being sought. If it is not equal, then depending on the comparison, the remaining search space is reduced to the part before or after the estimated position. This method will only work if calculations on the size of differences between key values are sensible.

By comparison, the [binary search](#) always chooses the middle of the remaining search space, discarding one half or the other, again depending on the comparison between the key value found at the estimated position and the key value sought. The remaining search space is reduced to the part before or after the estimated position. The [linear search](#) uses equality only as it compares elements one-by-one from the start, ignoring any sorting.

On average the interpolation search makes about $\log(\log(n))$ comparisons (if the elements are uniformly distributed), where n is the number of elements to be searched. In the worst case (for instance where the numerical values of the keys increase exponentially) it can make up to $O(n)$ comparisons.

In interpolation-sequential search, interpolation is used to find an item near the one being searched for, then [linear search](#) is used to find the exact item.

Contents

[\[hide\]](#)

- [1 Performance](#)
- [2 Adaptation to different datasets](#)
- [3 Book-based searching](#)
- [4 Sample implementation](#)
- [5 See also](#)
- [6 References](#)
- [7 External links](#)

Performance [\[edit\]](#)

Using [big-O notation](#), the performance of the interpolation algorithm on a data set of size N is $O(N)$; however under the assumption of a uniform distribution of the data on the linear scale used for interpolation, the performance can be shown to be $O(\log \log N)$.^{[1][2][3]} However, Dynamic Interpolation Search is possible in $o(\log \log n)$ time using a novel data structure.^[4]

Practical performance of interpolation search depends on whether the reduced number of probes is outweighed by the more complicated calculations needed for each probe. It can be useful for locating a record in a large sorted file on disk, where each probe involves a disk seek and is much slower than the interpolation arithmetic.

Index structures like [B-trees](#) also reduce the number of disk accesses, and are more often used to index on-disk data in part because they can index many types of data and can be updated [online](#). Still, interpolation search may be useful when one is forced to search certain sorted but unindexed on-disk datasets.

Adaptation to different datasets [\[edit\]](#)

When sort keys for a dataset are uniformly distributed numbers, linear interpolation is straightforward to implement and will find an index very near the sought value.

On the other hand, for a phone book sorted by name, the straightforward approach to interpolation search does not apply. The same high-level principles can still apply, though: one can estimate a name's position in the phone book using the relative frequencies of letters in names and use that as a probe location.

Some interpolation search implementations may not work as expected when a run of equal key values exists.

The simplest implementation of interpolation search won't necessarily select the first (or last) element of such a run.

Book-based searching [\[edit\]](#)

The conversion of names in a telephone book to some sort of number clearly will not provide numbers having a uniform distribution (except via immense effort such as sorting the names and calling them name #1, name #2, etc.) and further, it is well known that some names are much more common than others (Smith, Jones,) Similarly with dictionaries, where there are many more words starting with some letters than others. Some publishers go to the effort of preparing marginal annotations or even cutting into the side of the pages to show markers for each letter so that at a glance a segmented interpolation can be performed.

Sample implementation [\[edit\]](#)

The following C++ code example is a simple implementation. At each stage it computes a probe position then as with the binary search, moves either the upper or lower bound in to define a smaller interval containing the sought value. Unlike the binary search which guarantees a halving of the interval's size with each stage, a misled interpolation may reduce/increase the mid index by only one, thus resulting in a worst-case efficiency of $O(n)$.

```
template <typename T>
int interpolation_search(T arr[], int size, T key)
{
    int low = 0;
    int high = size - 1 ;
    int mid ;

    while (arr[high] != arr[low] && key >= arr[low] && key <= arr[high]) {
        mid = low + (key - arr[low]) * ((high - low) / ( arr[high] - arr[low])) ;

        if (arr[mid] < key)
            low = mid + 1 ;
        else if (key < arr[mid])
            high = mid - 1;
        else
            return mid;
    }

    if (key == arr[low])
        return low ;
    else
        return -1;
}
```

Notice that having probed the list at index *mid*, for reasons of loop control administration, this code sets either *high* or *low* to be not *mid* but an adjacent index, which location is then probed during the next iteration. Since an adjacent entry's value will not be much different the interpolation calculation is not much improved by this one step adjustment, at the cost of an additional reference to distant memory such as disc.

Each iteration of the above code requires between five and six comparisons (the extra is due to the repetitions needed to distinguish the three states of < > and = via binary comparisons in the absence of a [three-way comparison](#)) plus some messy arithmetic, while the [binary search algorithm](#) can be written with one comparison per iteration and uses only trivial integer arithmetic. It would thereby search an array of a million elements with no more than twenty comparisons (involving accesses to slow memory where the array elements are stored); to beat that the interpolation search as written above would be allowed no more than three iterations.

See also [\[edit\]](#)

- [Linear search](#)
- [Binary search](#)
- [Exponential search](#)
- [Ternary search](#)
- [Hash table](#)

References [[edit](#)]

- [^] Weiss, Mark Allen (2006). *Data structures and problem solving using Java*, Pearson Addison Wesley
- [^] Armenakis, A. C., Garey, L. E., Gupta, R. D., An adaptation of a root finding method to searching ordered disk files, BIT Numerical Mathematics, Volume 25, Number 4 / December, 1985.
- [^] Sedgewick, Robert (1990), *Algorithms in C*, Addison-Wesley
- [^] Andersson, Ame, and Christer Mattsson. 'Dynamic Interpolation Search in $\alpha(\log \log n)$ Time'. In Automata, Languages and Programming, edited by Andrzej Lingas, Rolf Karlsson, and Svante Carlsson, 700:15–27. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1993. http://dx.doi.org/10.1007/3-540-56939-1_58.

External links [[edit](#)]

- [Interpolation search](#)
- [National Institute of Standards and Technology](#)
- [Interpolation Search - A Log LogN Search](#)

Categories: [Search algorithms](#)

This page was last modified on 30 June 2015, at 14:10.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

