mathematics

MDPI

Article

Interpolation Once Binary Search over a Sorted List

Jun-Lin Lin [1,2,3]

[1] Department of Information Management, Yuan Ze University, Taoyuan 32003, Taiwan; jun@saturn.yzu.edu.tw
[2] Innovation Center for Big Data and Digital Convergence, Taoyuan 32003, Taiwan
[3] Zhen Ding Tech. Group—Yuan Ze University Joint R&D Center for Big Data, Taoyuan 32003, Taiwan

Abstract: Searching over a sorted list is a classical problem in computer science. Binary Search takes at most $\lfloor \log_2 n \rfloor + 1$ tries to find an item in a sorted list of size n. Interpolation Search achieves an average time complexity of $O(\log \log n)$ for uniformly distributed data. Hybrids of Binary Search and Interpolation Search are also available to handle data with unknown distributions. This paper analyzes the computation cost of these methods and shows that interpolation can significantly affect their performance—accordingly, a new method, Interpolation Once Binary Search (IOBS), is proposed. The experimental results show that IOBS outperforms the hybrids of Binary Search and Interpolation Search for nonuniformly distributed data.

Keywords: binary search; interpolation search; interpolated binary search

MSC: 68P10; 68W40

1. Introduction

Searching over a sorted list is a fundamental yet crucial operation in computer science, serving as the backbone for many applications. Several algorithms have been proposed for this operation in the literature [1]. Most of these algorithms share a similar structure: select a pivot element a_p from the sorted list $\left[a_{low}, a_{low+1}, \ldots, a_p, \ldots a_{high-1}, a_{high}\right]$, and if a_p does not match the searching item x, repeat the same process on either $\left[a_{low}, a_{low+1}, \ldots, a_{p-1}\right]$ or $\left[a_{p+1}, \ldots a_{high-1}, a_{high}\right]$, depending on the ordering between a_p and x.

Various algorithms apply different strategies to select the pivot element. Some algorithms determine the pivot element's index without using x or any element in the sorted list. For example, Binary Search [1] chooses the middle element (i.e., $p = \lfloor (low + high)/2 \rfloor$) as the pivot element and reduces the search range by half after each unsuccessful try. On the other hand, Exponential Search [2] starts with a small index for the pivot element, keeps doubling the index until $x < a_p$, and finally applies Binary Search on $\left[a_{\frac{p}{2}+1}, \ldots, a_{p-1}\right]$. Fibonacci Search [3] divides the search range into two parts, with sizes that are consecutive Fibonacci numbers. All three algorithms mentioned above have a time complexity of $O(\log n)$, where n is the number of elements in the sorted list [1].

In contrast, Interpolation Search [4] needs to access the data elements in the sorted list to determine the index of the pivot element. Specifically, it uses linear interpolation to derive the pivot element's index. This algorithm has an average time complexity of $O(\log \log n)$ for uniformly distributed data [5–7]. However, in the worst-case scenario, its time complexity is $O(n)$ for nonuniformly distributed data.

Variants of Interpolation Search have been proposed to alleviate the impact of nonuniformly distributed data [8–13]. For example, Interpolation-Sequential Search uses interpolation to determine the first pivot element and then applies Sequential Search to find the exact location of the search key [8]. Adaptive Search [11] and Interpolated Binary Search [12] are hybrids of Interpolation Search and Binary Search. For ease of exposition, denote the middle element in the sorted list as a_{mid} (used in Binary Search), and the element

Citation: Lin, J.-L. Interpolation Once Binary Search over a Sorted List. *Mathematics* **2024**, *12*, 1394. https://doi.org/10.3390/math12091394

Academic Editors: Jou-Ming Chang, Ling-Ju Hung and Chia-Wei Lee

Received: 12 April 2024
Revised: 30 April 2024
Accepted: 30 April 2024
Published: 2 May 2024

calculated through interpolation as a_{inter} (used in Interpolation Search). Interpolated Binary Search alternately uses a_{inter} and a_{mid} as the pivot elements. Adaptive Search uses a_{inter} as the pivot element, except when the current a_{inter} is not as effective as the current a_{mid} in reducing the searching range, then the new a_{mid} is used as the next pivot element. Although Adaptive Search seems more sophisticated than Interpolated Binary Search, Interpolated Binary Search outperforms Adaptive Search [12].

This study is motivated by two questions. First, why does Interpolation Search perform poorly over nonuniformly distributed data? Second, why does the seemly more thoughtful strategy for selecting pivot elements in Adaptive Search yield worse performance than the simple turn-taking strategy in Interpolated Binary Search? Answers to these questions uncover the pros and cons of these algorithms. This study aims to improve the existing algorithms for searching within the sorted lists, making this operation more efficient for various applications. We propose a novel algorithm that surpasses both Adaptive Search and Interpolated Binary Search, as demonstrated in the performance study of Section 5.

In this study, we confine the sorted list to an array-like data structure that permits constant-time access to any element within the sorted list. However, more sophisticated data structures have been suggested to enhance the search performance. For example, a data structure called Dynamic Interpolation Search Tree [14] and its variant [15] can support Interpolation Search to achieve $O(\log \log n)$ time complexity for a broad range of data distributions. Searching algorithms can also be used for sorting. For example, Binary Search Sort Algorithm [16] uses Binary Search to add elements to a sorted list incrementally.

The rest of this paper is organized as follows. Section 2 reviews Interpolation Search and discusses its performance bottleneck. Section 3 compares Interpolated Binary Search and Adaptive Search, and explains why the former outperforms the latter. Based on the findings from Sections 2 and 3, Section 4 proposes a new algorithm called Interpolation Once Binary Search. Section 5 shows the experimental results of these algorithms. Discussion and conclusions are given in Sections 6 and 7, respectively.

2. Comparison of Binary Search and Interpolation Search

The algorithms for Binary Search and Interpolation Search are shown in Algorithm 1. To search for x from a sorted list $\left[a_{low}, a_{low+1}, \ldots, a_{high-1}, a_{high}\right]$, Binary Search and Interpolation Search differ only in how the pivot elements are determined. Binary Search uses the middle element as the pivot element a_p, i.e.,

$$p = \left\lfloor \frac{(high + low)}{2} \right\rfloor. \tag{1}$$

Interpolation Search determines the pivot element's index p by assuming that point (p, x) is on the straight line with the endpoints (low, a_{low}) and $\left(high, a_{high}\right)$. Then, p can be derived as follows:

$$p = \left\lfloor \frac{(high - low)\ (x - a_{low})}{a_{high} - a_{low}} \right\rfloor + low. \tag{2}$$

The time complexities of both algorithms have been extensively studied in the literature. As described in Section 1, Binary Search has a time complexity of $O(\log n)$; Interpolation Search has an average-case time complexity of $O(\log \log n)$ if the data elements are uniformly distributed [5,6]. However, the time complexity of Interpolation Search can degrade to $O(n)$ if the data distribution is highly uneven. Notably, the time complexity analysis focuses on the number of iterations within the while-loop (lines 2–6 in Algorithm 1).

Algorithm 1 The Algorithms for Binary Search and Interpolation Search

Input: a sorted list $[a_1, a_2, \ldots, a_n]$ and the searching item x
Output: the index of x in $[a_1, a_2, \ldots, a_n]$, or -1 if not found.
1. $low = 1$ and $high = n$;
2. **While** ($low \leq high$) **do**
3. Calculate p using Equation (1) for Binary Search or Equation (2) for Interpolation Search;
4. **If** ($x = a_p$), **return** p;
5. **Else if** ($x > a_p$), $low = p + 1$;
6. **Else** $high = p - 1$;
7. **Return** -1;

One iteration in the while-loop of Interpolation Search is much slower than that of Binary Search in terms of computational cost and data access cost. First, Equation (1) involves only operations on integers, and a bit-right-shift operation can accomplish the division-by-two operation. In contrast, Equation (2) requires multiplication and division operations on floating-point values, making it much more computation costly than Equation (1). Second, Equation (1) does not need to access any data element in the sorted list, but Equation (2) needs access to two data elements (i.e., a_{high} and a_{low}). That is, one iteration in the while-loop of Binary Search accesses only one data element a_p, but Interpolation Search requires three data elements (i.e., a_p, a_{high} and a_{low}). Notably, with Binary Search, the search range can be reduced by $7/8$ with three accesses of data elements. Thus, interpolation should be avoided for performance reasons unless it's highly effective at reducing the number of iterations. In Section 4, this observation is applied to design the proposed algorithm that minimizes interpolation.

3. Comparison of Interpolated Binary Search and Adaptive Search

In practice, the distribution of data elements in the sorted list is often unknown or does not follow a specific pattern. This uncertainty makes it difficult to determine whether Binary Search or Interpolation Search is the most suitable for a given problem. To address this, hybrid algorithms that combine aspects of both Binary Search and Interpolation Search have been proposed. In this section, we describe and compare two such algorithms, Interpolated Binary Search (IBS) and Adaptive Search (AS). Additionally, we derive critical insights for designing a more effective algorithm.

IBS employs Equations (1) and (2) alternately to decide the pivot elements. A concise version of the original algorithm [12] is depicted in Algorithm 2.

Adaptive Search (AS) employs a more sophisticated strategy than IBS does to switch between Binary Search style (i.e., Equation (1)) and Interpolation Search style (i.e., Equation (2)). Each iteration within the while-loop of Algorithm 3 involves one or two probes for the pivot elements. The first probe follows Interpolation Search style (line 3) and the second probe (lines 4–9) follows Binary Search style. The second probe is added only when the first probe is ineffective in reducing the search range by more than half.

Both IBS and AS have the same time complexity. If the data elements in the sorted list are distributed uniformly, both algorithms will have an average time complexity of $O(\log \log n)$, which is the same as Interpolation Search. Even if the data elements are not uniformly distributed, the worst time complexity of both algorithms will still be $O(\log n)$, which is the same as Binary Search.

However, AS is more sophisticated than IBS, and a detailed comparison between IBS and AS from the computational cost perspective can be found in Section 5 of Reference [12]. The key here is whether the added sophistication can help reduce the number of iterations of the while-loop in AS. Unfortunately, there is no evidence to support the two heuristics of AS: (1) an ineffective Interpolation Search probe should follow by a Binary Search probe, and (2) an effective Interpolation Search probe should follow by another Interpolation Search probe. Experimental results from Reference [12] and Section 5 also show that the simple IBS outperforms the sophisticated AS. Therefore, unless the added sophistication to

a search method effectively reduces the number of iterations, it might negatively impact the performance.

Algorithm 2 The Algorithm for Interpolated Binary Search.

Input: a sorted list $[a_1, a_2, \ldots, a_n]$ and the searching item x
Output: the index of x in $[a_1, a_2, \ldots, a_n]$, or -1 if not found.
1. $low = 1$, $high = n$, and $binaryTurn$ = **False**;
2. **While** ($low \leq high$) **do**
3. **If** ($binaryTurn$), calculate p using Equation (1); // a Binary Search probe
4. **Else** calculate p using Equation (2); // an Interpolation Search probe
5. **If** ($x = a_p$), **return** p;
6. **Else if** ($x > a_p$), $low = p + 1$;
7. **Else** $high = p - 1$;
8. $binaryTurn$ = **not** ($binaryTurn$); // change turn
9. **Return** -1;

Algorithm 3 The Algorithm for Adaptive Search

Input: a sorted list $[a_1, a_2, \ldots, a_n]$ and the searching item x
Output: the index of x in $[a_1, a_2, \ldots, a_n]$, or -1 if not found.
1. $low = 1$, $high = n$;
2. **While** ($low < high$) **do**
3. Calculate p using Equation (2); // an Interpolation Search probe
4. **If** $a_p > x$ and $(p - low) > \left\lfloor \frac{high-low}{2} \right\rfloor$,
5. $high = p - 1$;
6. Calculate p using Equation (1); // insert a Binary Search probe
7. **Else If** $a_p < x$ and $(high - p) > \left\lfloor \frac{high-low}{2} \right\rfloor$, then
8. $low = p + 1$;
9. Calculate p using Equation (1); // insert a Binary Search probe
10. **If** $a_p > x$,
11. $high = p - 1$;
12. **Else If** $a_p < x$
13. $low = p + 1$;
14. **Else**
15. **Return** p;
16. **Return** -1;

4. Interpolation Once Binary Search—The Proposed Method

This section proposes a hybrid of Interpolation Search and Binary Search, namely Interpolation Once Binary Search (IOBS). First, IOBS avoids any logic in switching between Interpolation Search and Binary Search since there is no clear evidence supporting that such an arrangement benefits performance, as discussed in Section 3. Second, IOBS reduces the number of interpolations to the extreme to mitigate the computational cost, as discussed in Section 2.

The algorithm for IOBS is shown in Algorithm 4. Given a sorted list $[a_1, a_2, \ldots, a_n]$ and the searching item x, IOBS returns the index i of a_i if $a_i = x$, and -1 otherwise. IOBS uses Interpolation Search to determine the first pivot element (line 2). Then, all subsequent probes follow Binary Search (line 7). The time complexity of IOBS is $O(\log n)$, the same as Binary Search.

Algorithm 4 The Algorithm for Interpolation Once Binary Search

Input: a sorted list $[a_1, a_2, \ldots, a_n]$ and the searching item x
Output: the index of x in $[a_1, a_2, \ldots, a_n]$, or -1 if not found.
1. $low = 1$, $high = n$;
2. Calculate p using Equation (2); // Interpolation Search style
3. **Do**
4. **If** $(x = a_p)$, **return** p;
5. **Else if** $(x > a_p)$, $low = p + 1$;
6. **Else** $high = p - 1$;
7. Calculate p using Equation (1); // Binary Search style
8. **While** $(low \leq high)$;
9. **Return** -1;

5. Performance Study

For performance comparison, we adopted the source code for AS, Interpolation Search, and Binary Search from [17] and implemented IBS and IOBS from scratch. We also adopted test dataset generation in the original source code. All experiments were conducted on a PC with JRE1.8, 64-bit Windows 10 OS, Intel® Core™ i7-7700 processor, and 8 GBs of RAM.

The test datasets consist of ordered instances of Java double-precision floating-point values that are randomly generated. These values are distributed uniformly, normally or exponentially. The size of the test datasets is measured by the number of instances present, which ranges from 5×10^3 to 5×10^5 for small datasets and from 10^6 to 10^8 for large datasets.

Thirty test datasets were generated for each dataset size and data distribution. We utilized all instances in each dataset as search keys and calculated the average running time per instance. The process was repeated across the 30 test sets with the same size and data distribution, and the average running time was reported. The above experiment closely resembles the one described in [12], with two key differences. First, they randomly selected 1000 instances as the search keys, irrespective of the dataset's size, and repeated this process 1000 times to mitigate the sampling bias. In contrast, we used all instances as the search keys to avoid sampling bias. Second, they used only one dataset for each dataset size and data distribution, but we used 30 datasets to minimize dataset bias.

5.1. Test Results over Uniformly Distributed Datasets

Figures 1 and 2 show the experimental results for small and large datasets with uniform distribution. The results reflect that interpolation is highly effective at narrowing the searching range and reducing the number of probes for uniformly distributed datasets. Consequently, Binary Search and Interpolation Search yield the worst and the best performance, respectively.

According to Algorithms 2 and 3, AS uses a higher percentage of Interpolation Search probes than IBS does, because the former inserts a Binary Search probe only after an ineffective Interpolation Search probe, but the latter inserts a Binary Search probe after every Interpolation Search probe. However, the performance results show that IBS outperforms AS. As discussed in Section 3, AS needs to evaluate the effectiveness of each Interpolation Search probe, making it slower than IBS.

IOBS achieves the second-best result over small datasets. However, it yields the second-worst result over large datasets, as the benefit of interpolation strengthens in Interpolation Search, AS, and IBS for large datasets.

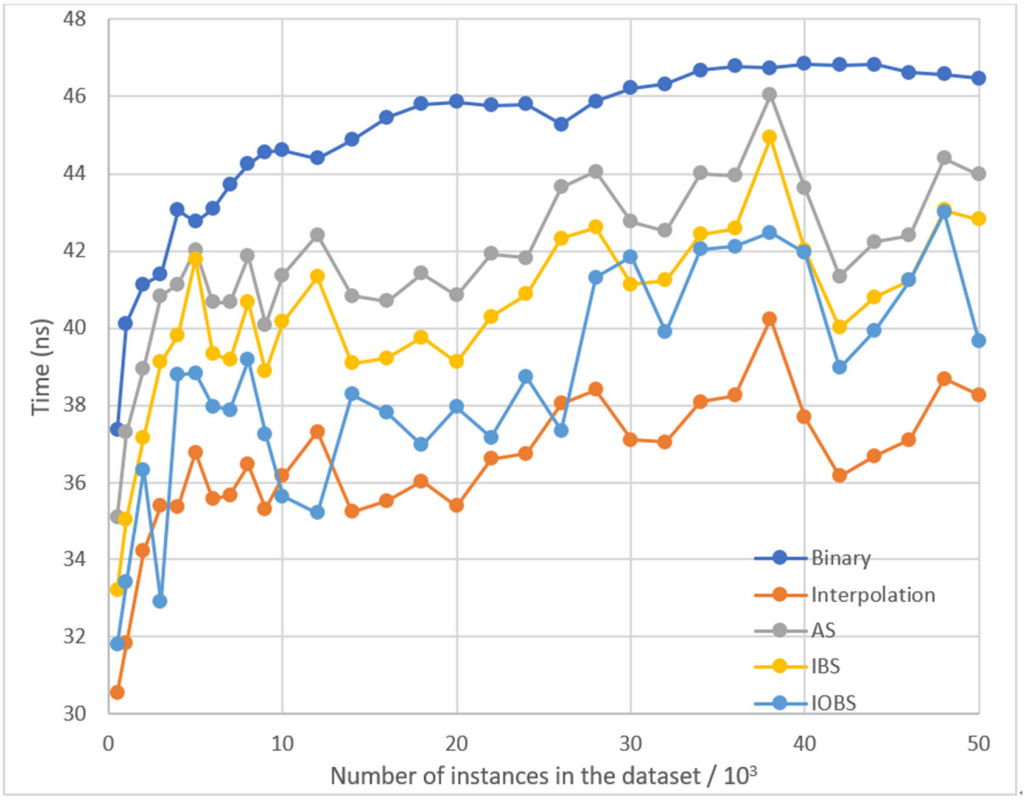

Figure 1. Average running time per search of Binary Search, Interpolation Search, AS, IBS and IOBS over small datasets with uniform distribution.

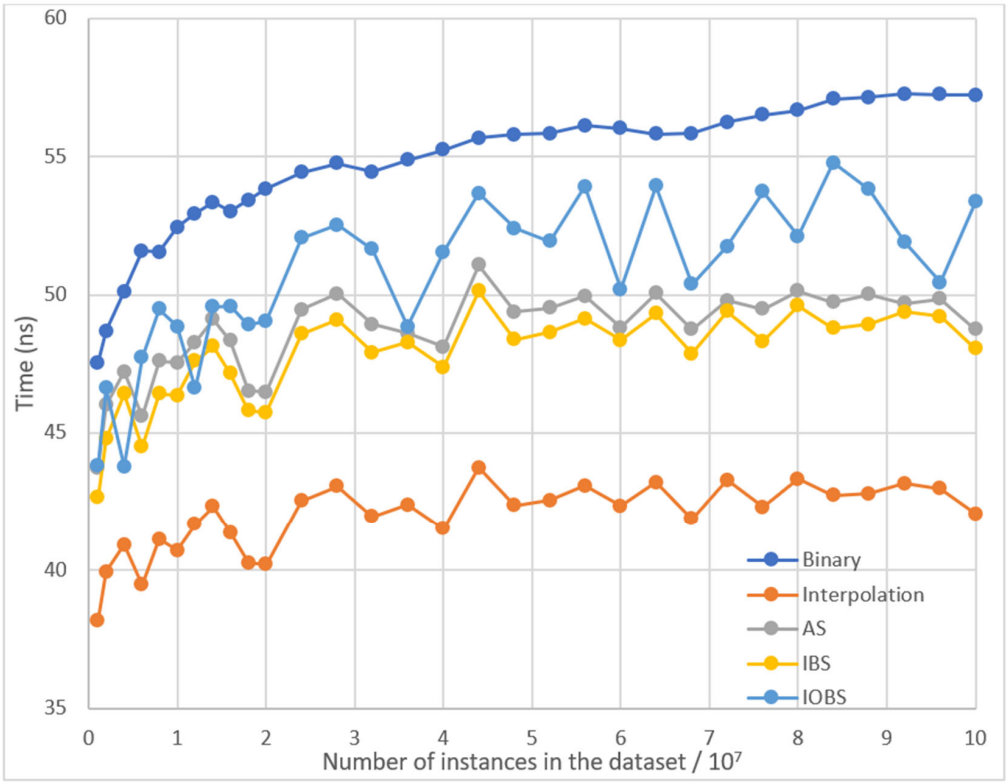

Figure 2. Average running time per search of Binary Search, Interpolation Search, AS, IBS and IOBS over large datasets with uniform distribution.

5.2. Test Results over Normally Distributed Datasets

Figures 3 and 4 show the experimental results for small and large datasets with normal distribution, where the result of Interpolation Search is excluded for its poor performance and clear demonstration of other methods' results. IOBS consistently outperforms IBS and AS and achieves performance only next to Binary Search.

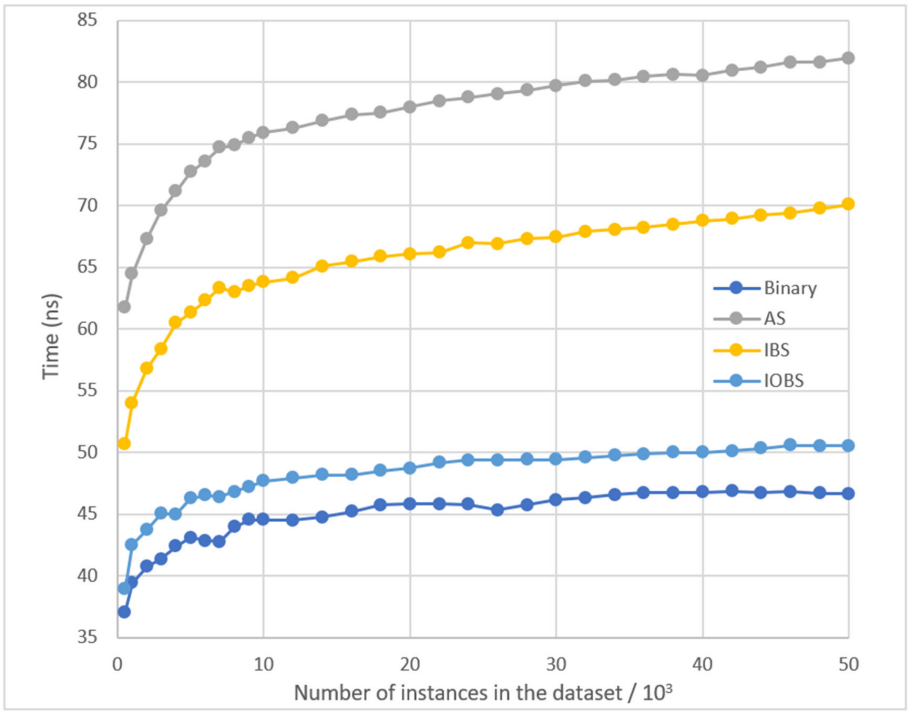

Figure 3. Average running time per search of Binary Search, AS, IBS and IOBS over small datasets with normal distribution.

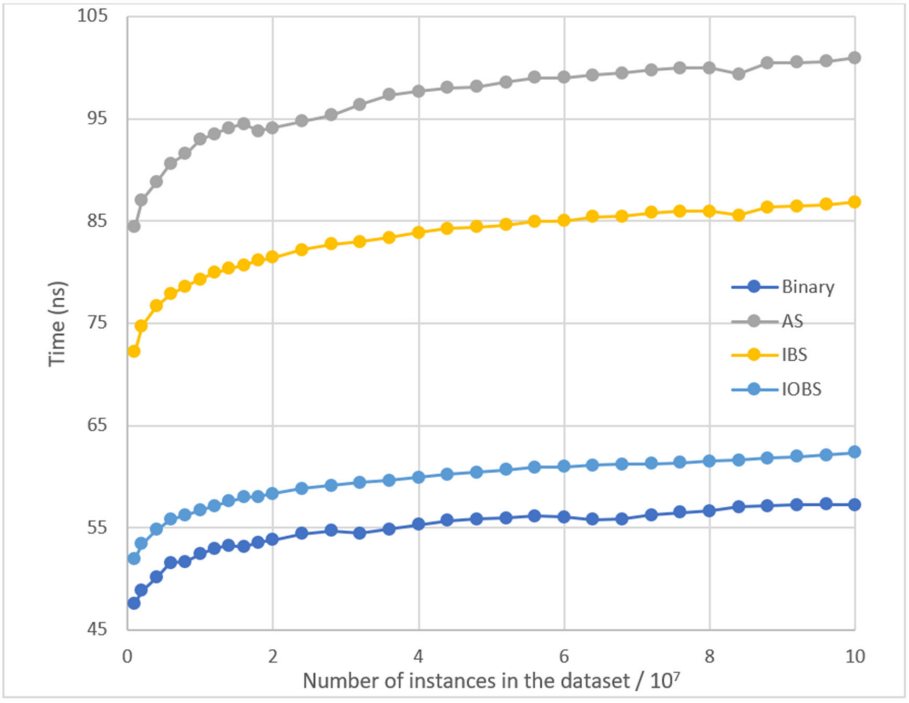

Figure 4. Average running time per search of Binary Search, AS, IBS and IOBS over large datasets with normal distribution.

The experimental results demonstrate that interpolation is ineffective at determining the pivot element's index. First, interpolation often fails to narrow the search range for normally distributed datasets effectively. Second, the computational cost of calculating the index for an Interpolation Search probe exceeds that of simply using the middle element's index. IOBS outperforms IBS and AS when dealing with normally distributed datasets because it applies interpolation less frequently.

5.3. Test Results over Exponentially Distributed Datasets

Figures 5 and 6 show the experimental results for small and large datasets with exponential distribution, where the result of Interpolation Search is excluded because of its poor performance and clear demonstration of other methods' results. Similar to the results for datasets with normal distribution, IOBS consistently outperforms IBS and AS, and achieves performance only next to Binary Search.

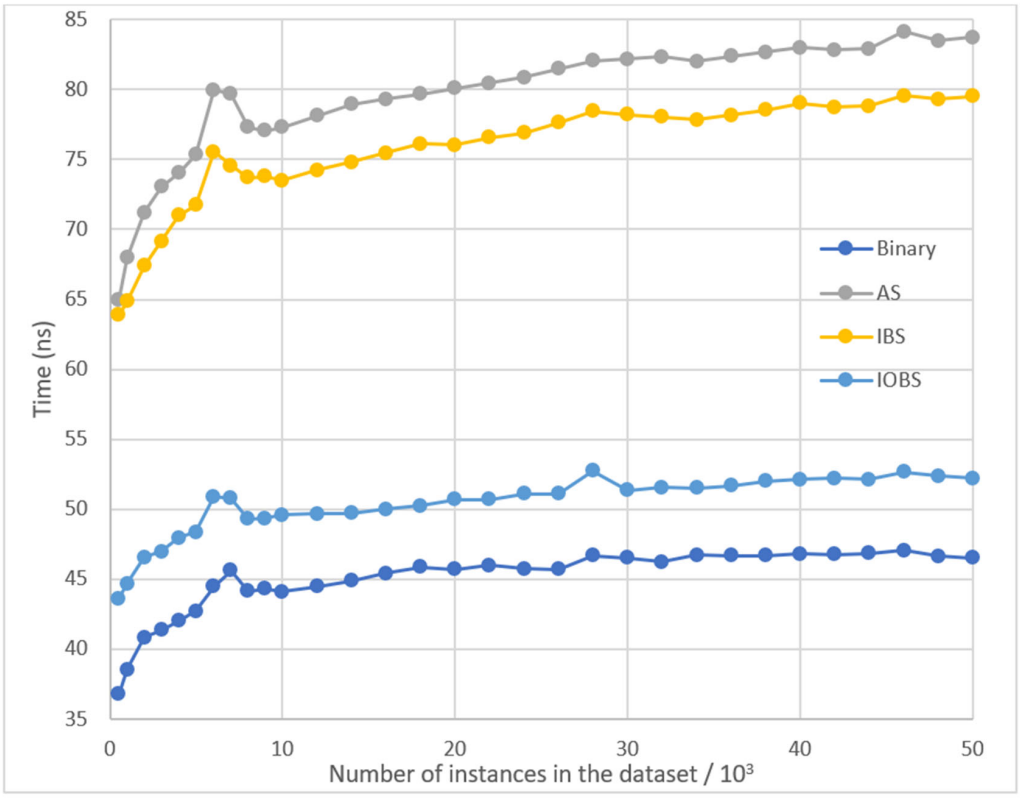

Figure 5. Average running time per search of Binary Search, AS, IBS and IOBS over small datasets with exponential distribution.

Comparing the results of IOBS in Figures 4 and 6 reveals that IOBS is almost unaffected by data distribution. In contrast, IBS and AS exhibit poorer results with exponentially distributed datasets than with normally distributed ones. Intuitively, the exponential distribution is more uneven than the normal distribution, and consequently, Interpolation Search probes are less effective with exponentially distributed datasets than with normally distributed datasets.

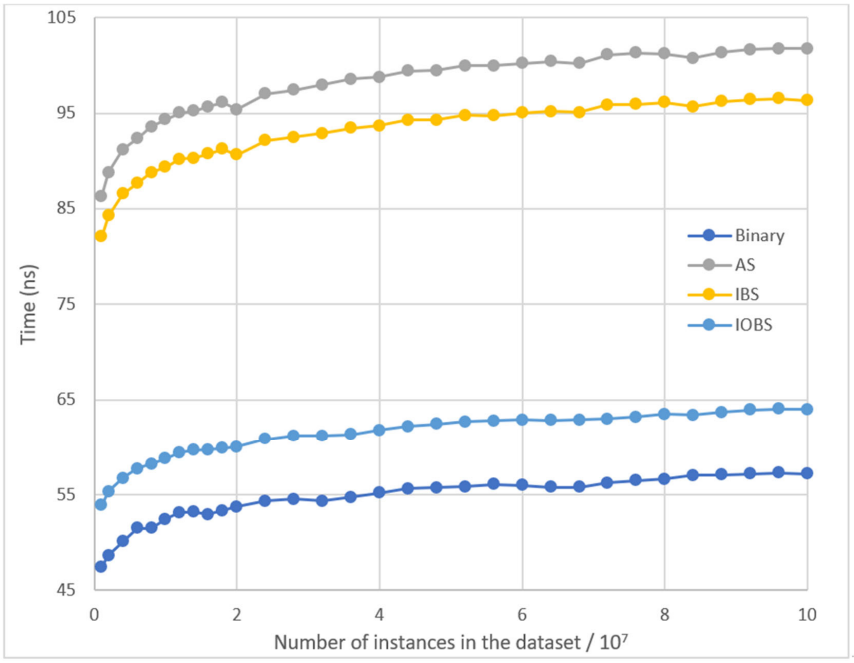

Figure 6. Average running time per search of Binary Search, AS, IBS and IOBS over large datasets with exponential distribution.

5.4. Test Results over Datasets with Mixed Distributions

To evaluate the performance of these search algorithms over datasets that do not follow a specific distribution, we conduct the same experiment over the mixtures of the large datasets from Section 5.1, Section 5.2, and Section 5.3. Specifically, each dataset is a mixture of three equal-sized sub-datasets: one with uniform distribution, one with normal distribution, and one with exponential distribution. Figure 7 shows the experimental results, where Interpolation Search is excluded because of its poor performance and clear demonstration of other methods' results. IOBS's performance is almost unaffected by the data distribution and superior to that of AS and IBS.

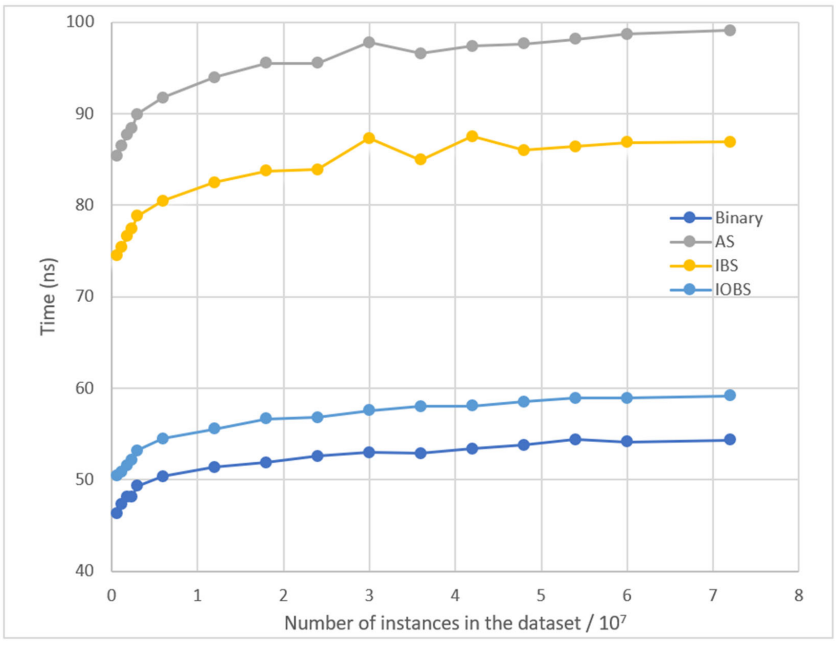

Figure 7. Average running time per search of Binary Search, AS, IBS and IOBS over large datasets with mixed distributions.

6. Discussion

For ease of exposition, given a dataset, a search key, and a search method, let m denote the total number of times needed to determine the pivot element's index, and m_i denote the number of times interpolation is used to determine the pivot element's index. The interpolation ratio is defined as m_i/m. The values of interpolation ratio for Interpolation Search, IOBS and Binary Search are 1, $1/m$, and 0, respectively. IBS alternates between using Interpolation Search probes and Binary Search probes, and thus yields an interpolation ratio of 0.5. AS consistently uses Interpolation Search probes, except when the current Interpolation Search probe is ineffective. In such cases, a Binary Search probe is used as the next pivot element, and thus the interpolation ratio of AS is between 0.5 and 1. Notably, if all the Interpolation Search probes in AS are effective, then no Binary Search probe is needed, and consequently, AS uses the same probes as Interpolation Search does. Conversely, if all the Interpolation Search probes in AS are ineffective, then they will be followed by a Binary Search probe, and consequently, AS uses the same probes as IBS does. For the same value of m, the ordering of interpolation ratio is Binary Search < IOBS $\leq$ IBS $\leq$ AS $\leq$ Interpolation Search. Notably, IOBS = IBS only when $m \leq 2$; IBS = AS only when all Interpolation Search probes of AS are ineffective; AS = Interpolation Search only when all Interpolation Search probes of AS are effective.

Based on the experimental results with uniformly distributed datasets in Section 5.1, using interpolation to determine the pivot element's index is more effective at narrowing the search range than using the index of the middle element, as done in Binary Search. Furthermore, despite its higher computational cost, interpolation remains highly effective for uniformly distributed datasets. A higher interpolation ratio often yields better performance for uniformly distributed datasets. Consequently, Interpolation Search, Binary Search, and IOBS exhibit the best, worst, and second-worst performances for large datasets with uniform distribution, as shown in Figure 2.

Even though the interpolation ratio of AS is always greater than or equal to that of IBS, this does not necessarily make AS quicker than IBS. Despite having the same average time complexity of $O(\log \log n)$ for uniformly distributed datasets and the same worst-case time complexity of $O(\log n)$ for non-uniformly distributed datasets, IBS consistently outperforms AS. When analyzing the time complexity of a search algorithm, the focus is on the number of probes needed relative to the dataset size. Typically, the cost of choosing a probe is treated as a constant. However, AS takes more time to choose a probe than IBS does because AS needs to evaluate the effectiveness of each Interpolation Search probe. This observation also addresses our second research question posed in Section 1: Why does the seemly more thoughtful strategy in AS yields worse performance compared to the straightforward turn-taking strategy in IBS? Notably, there is no evidence to support the notion that an ineffective Interpolation Search probe should not follow another Interpolation Search probe. As a result, the seemly thoughtful strategy in AS becomes a computational time waste, rendering AS slower than IBS. Similarly, IOBS also avoids complex logic to switch between Interpolation Search probes and Binary Search probes to improve performance.

Based on the experimental results with nonuniformly distributed datasets in Sections 5.2 and 5.3, using Interpolation Search probes is less effective than using Binary Search probes. This observation also addresses our first research question posed in Section 1: why does Interpolation Search perform poorly over nonuniformly distributed data? As discussed in Section 2, calculating the Interpolation Search probe using Equation (2) is costlier than calculating the Binary Search probe using Equation (1). Since the effectiveness of an Interpolation Search probe for nonuniformly distributed datasets does not outweigh its computational cost, a smaller interpolation ratio often yields better performance. IOBS outperforms AS, IBS and Interpolation Search when searching nonuniformly distributed datasets because IOBS uses Interpolation Search probe only once to mitigate the computational cost.

7. Conclusions

Searching over a sorted list is a common operation in many applications, and thus improving this operation can greatly enhance the overall performance. The main contributions of this work include the following:

(1) a new algorithm, IOBS, that outperforms other hybrids of Interpolation Search and Binary Search over nonuniformly distributed datasets,
(2) exploring the tradeoff between the cost of calculating an Interpolation Search probe and its effectiveness on reducing the search range, and
(3) showing that a seemly reasonable but unfounded heuristic (e.g., AS chooses between an Interpolation Search probe and a Binary Search probe, based on the effectiveness of the previous Interpolation Search probe) can be harmful to the performance.

Notably, the latter two can be checked when developing new search algorithms. The experimental results in Section 5 show that Interpolation Search yields the worst results for datasets with normal or exponential distributions. AS and IBS improve Interpolation Search by integrating Binary Search and Interpolation Search to mitigate the interpolation cost. Furthermore, IBS outperforms AS due to its simplicity. These results suggest that hybrids of Binary Search and Interpolation Search benefit from reducing the interpolation cost and avoiding complex rules for switching between Binary Search and Interpolation Search. Accordingly, this study proposes IOBS, which incurs less interpolation cost and is more straightforward than AS and IBS. Performance results show that IOBS outperforms AS and IBS over datasets with normal or exponential distribution and that IOBS outperforms Binary Search for uniformly distributed datasets. Thus, IOBS is a better alternative to other hybrids of Binary Search and Interpolation Search, such as AS and IBS, unless the data distribution is known to be uniform.

Experimental results also show that Interpolation Search performs best for datasets with uniform distribution. Repeatedly using interpolation to determine the pivot elements for uniformly distributed datasets improves performance. However, it is known that the best-case time complexity of Interpolation Search is O(1) when the first pivot element derived from interpolation is the search key. For uniformly distributed datasets, the first pivot element by Interpolation Search is near the search key, and thus the importance of determining the subsequent pivot elements by interpolation is reduced. How to balance between repeated interpolation and reducing interpolation cost deserves further investigation. For example, although IOBS does not require any parameter, a more versatile version of IOBS could incorporate a parameter to fine-tune this balance, thereby enhancing the algorithm's adaptability to diverse datasets.

Funding: This research received no external funding.

Data Availability Statement: Source codes for performance study are available upon request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Knuth, D.E. *The Art of Computer Programming, Volume 3, Sorting and Searching*, 2nd ed.; Addison-Wesley: Boston, MA, USA, 1998.
2. Bentley, J.L.; Yao, A.C.-C. An almost optimal algorithm for unbounded searching. *Inf. Process. Lett.* **1976**, *5*, 82–87. [CrossRef]
3. Overholt, K.J. Efficiency of the Fibonacci search method. *BIT Numer. Math.* **1973**, *13*, 92–96. [CrossRef]
4. Peterson, W.W. Addressing for Random-Access Storage. *IBM J. Res. Dev.* **1957**, *1*, 130–146. [CrossRef]
5. Perl, Y.; Itai, A.; Avni, H. Interpolation search—A log logN search. *Commun. ACM* **1978**, *21*, 550–553. [CrossRef]
6. Yao, A.C.; Yao, F.F. The complexity of searching an ordered random table. In Proceedings of the 17th Annual Symposium on Foundations of Computer Science (sfcs 1976), Houston, TX, USA, 25–27 October 1976; pp. 173–177. [CrossRef]
7. Perl, Y.; Reingold, E.M. Understanding the Complexity of Interpolation Search. *Inf. Process. Lett.* **1977**, *6*, 219–222. [CrossRef]
8. Gonnet, G.H.; Rogers, L.D. The interpolation-sequential search algorithm. *Inf. Process. Lett.* **1977**, *6*, 136–139. [CrossRef]
9. Santoro, N.; Sidney, J.B. Interpolation-binary search. *Inf. Process. Lett.* **1985**, *20*, 179–181. [CrossRef]
10. Burton, F.W.; Lewis, G.N. A robust variation of interpolation search. *Inf. Process. Lett.* **1980**, *10*, 198–201. [CrossRef]
11. Bonasera, B.; Ferrara, E.; Fiumara, G.; Pagano, F.; Provetti, A. Adaptive search over sorted sets. *J. Discret. Algorithms* **2015**, *30*, 128–133. [CrossRef]

12. Mohammed, A.S.; Amrahov, S.E.; Çelebi, F.V. Interpolated binary search: An efficient hybrid search algorithm on ordered datasets. *Eng. Sci. Technol. Int. J.* **2021**, *24*, 1072–1079. [CrossRef]
13. Kabir, M.N.; Alginahi, Y.M.; Ali, J.; Abdel-Raheem, E. Optimal search algorithm in a big database using interpolation–extrapolation method. *Electron. Lett.* **2019**, *55*, 1130–1133. [CrossRef]
14. Mehlhorn, K.; Tsakalidis, A. Dynamic interpolation search. *J. ACM* **1993**, *40*, 621–634. [CrossRef]
15. Kaporis, A.; Makris, C.; Sioutas, S.; Tsakalidis, A.; Tsichlas, K.; Zaroliagis, C. Dynamic Interpolation Search revisited. *Inf. Comput.* **2020**, *270*, 104465. [CrossRef]
16. Chaitanya, P. Binary Search Sort Algorithm-Yet Another Sorting Algorithm with Binary Search having O(nlogn) and O(n) time complexity. In Proceedings of the 2023 1st International Conference on Optimization Techniques for Learning (ICOTL), Bengaluru, India, 7–8 December 2023; pp. 1–6. [CrossRef]
17. Bonasera, B. AdaptiveSearch (Java Source Code). Available online: https://bitbucket.org/ale66/adaptivesearch/src/master/ (accessed on 1 May 2024).