Journal on Big Data
DOI:10.32604/jbd.2021.017017
Article

Application of Quicksort Algorithm in Information Retrieval

Jiajun Xie[1], Zuyan Li[1], Han Wu[1], Linhan Li[2], Bin Pan[1], Peng Guo[3] and Guang Sun[1,*]

[1]Hunan University of Finance and Economics, Changsha, 410205, China
[2]Changjun Meixihu Middle School, Changsha, 410205, China
[3]University Malaysia Sabah, Kota Kinabalu, 999004, Malaysia
[*]Corresponding Author: Guang Sun. Email: simon5115@163.com
Received: 06 May 2021; Accepted: 19 October 2021

Abstract: With the development and progress of today's network information technology, a variety of large-scale network databases have emerged with the situation, such as Baidu Library and Weipu Database, the number of documents in the inventory has reached nearly one million. So how do you quickly and effectively retrieve the information you want in such a huge database? This requires finding efficient algorithms to reduce the computational complexity of the computer during Information Retrieval, improve retrieval efficiency, and adapt to the rapid expansion of document data. The Quicksort Algorithm gives different weights to each position of the document, and multiplies the weight of each position with the number of matches of that position, and then adds all the multiplied sums to set a feature value for Quicksort, which can achieve the full accuracy of Information Retrieval. Therefore, the purpose of this paper is to use the quick sort algorithm to increase the speed of Information Retrieval, and to use the position weighting algorithm to improve the matching quality of Information Retrieval, so as to achieve the overall effect of improving the efficiency of Information Retrieval.

Keywords: Quicksort; Information Retrieval; information processing

1 Introduction

With the rapid development of Internet technology and the increasingly widespread use of the Internet, more and more information needs to be stored in the form of electronic data. How do you find the information you want in such a huge data storage warehouse? In response to this demand, information retrieval technology has emerged. Information Retrieval technology is one of them [1]. Compared with the original immature Information Retrieval technology, this type of technology has now been greatly improved and gradually matured. Literature retrieval is an important way for researchers to obtain resource information, and it has become a very important field in Information Retrieval. Scientific literature retrieval can help researchers learn from and summarize the research results of predecessors. It can not only promote the rapid development and utilization of literature resources, but also avoid repeated research and other phenomena [2].

The previous traditional Information Retrieval techniques generally have a single function. Either only considers word frequency and ignores the document value manifestation brought by the number of references between users and the number of document downloads, or considers the latter and ignores the former, and ultimately cannot Retrieve documents that are closest to user needs, which reduces user experience. On the basis of combining these loopholes, this paper further proposes a comprehensive idea, that is, to increase the function of users to independently select more detailed requirements, and finally meet the requirements of full accuracy of Information Retrieval [3]. Furthermore, this paper is committed to achieving comprehensiveness and accuracy of Information Retrieval, and at the same time, it uses a Quicksort Algorithm to sort and output the documents. The Quicksort Algorithm can achieve the fastest sorting when there are many and disorderly arranged data. The speed is very suitable for the huge amount of literature nowadays, and it can well meet the requirements of rapid Information Retrieval. This paper simulates the experiment under ubuntu with C++ environment installed, and finally proves that the research content of the paper is correct and can be implemented. The Quicksort Algorithm can improve the Information Retrieval rate very well without being affected by the hardware equipment, and has real application prospects.

2 Related Works

One of the core problems of Information Retrieval technology is to retrieve the results through a certain rule algorithm, and then use the Sorting algorithm to sort and output the retrieval results in a certain order [4]. There have been many research precedents for retrieval technology at home and abroad. The generalization can be divided into three generations: the first–generation Information Retrieval system based on word frequency, the second–generation Information Retrieval system based on links, and the third–generation Information Retrieval system based on intelligent sorting [5]. Take the three–generation Information Retrieval system as a clue to introduce the research status at home and abroad [6].

The first–generation Information Retrieval system based on word frequency is sorted according to the frequency and position of the retrieved keywords in the document [7]. Its operating principle is: the higher the number of search terms in a document and the more important the Position, the greater the correlation between the document and the search term, the TFIDF (Term Frequency–Inverse Document Frequency) algorithm can better handle the relationship between the frequency of the search term and the position where it appears, and the relevance score is calculated for ranking, which is considered to be this One of the most important inventions of the stage [8–9].

Next is the second–generation Information Retrieval system based on links. According to historical evidence, we know that although the PageRank algorithm improves the efficiency of Google's web search system, it only determines the importance of the document by considering the number of times the document has been cited, while ignoring the relevance of the content of the document itself and the user's search terms [10–11]. Although the recommended literature given to users is of high value and authority, it is not what users need most [12–13].

The third–generation Information Retrieval system is to solve the problem of the single retrieval result of the second–generation retrieval system. Intelligent sorting is dedicated to providing personalized services and realizing intelligent retrieval of documents [14–16]. What is intelligent retrieval? Even if the retrieval technology is more user–friendly. Intelligent retrieval technology can analyze the relevant keywords of the retrieved keywords on the current Internet, increase the semantic retrieval function and user feedback function, integrate these for personalized analysis, and finally select and arrange the most relevant to the user's search terms. Documents that can meet user needs. Therefore, the third–generation Information Retrieval system solves the problem of single and inaccurate Information Retrieval results.

3. Technical Foundation

3.1 The meaning of Quicksort Algorithm

In 1962, Tony Hoare developed a sorting algorithm that relied on recursion, called a Quick sort Algorithm. The Quicksort Algorithm adopts a divide-and-conquer method. In the average state, the time complexity of the Quicksort Algorithm is O(nlogn), that is, nlogn comparisons are required to quickly sort n data.

The algorithm rules of the Quicksort Algorithm can be stated as: Pick an element from the sequence to be sorted and use it as the "benchmark". Generally, the first number in the sequence is selected as the benchmark.

Advantages of the Quicksort Algorithm: History has proved through countless experiments that the Quicksorting Algorithm has a speed advantage over other algorithms when the larger and more disordered the sequence to be sorted is. Nowadays, the number of documents that can only be described as extremely large is suitable for Quicksorting Algorithms. Under this condition, the advantages of Quicksort are more obvious.

3.2 Application of Quicksort Algorithm

In the process of Information Retrieval, we multiply the number of times the search term appears in a certain position of the document and the weight of that position to obtain a sub-Eigen value, and then add the sub-Eigen values of all the positions of the document as the relative The Eigen value of this search term. The eigenvalues of all documents form an unordered number sequence. At this time, we use the Quicksort Algorithm to sort these eigenvalues, and output the documents with the largest eigenvalues first to meet the user's Information Retrieval requirements. In this process of Information Retrieval, the Quicksort Algorithm has played an important role. We know that, in general, the time complexity of the Quicksort Algorithm is O(nlogn), which is significantly better than the O(n2) time complexity of some traditional sorting algorithms such as Selection Sort, Swap Sort, and Insertion Sort. Today, the number of documents on the Internet is increasing and becoming more and more complex. In the case that the larger the sequence to be sorted, the more disorderly it is, the Quicksort Algorithm is also superior to some advanced sorting algorithms with O(nlogn) time complexity, such as Merge Sort. In this way, in today's rapid expansion of the number of documents, the Quicksort Algorithm has great advantages to be used in Information Retrieval, and has great application prospects.

3.3 Technical Basis of Information Retrieval Technology

When the user enters the word he wants to search in the search box, the search engine searches the document resource database according to the user search word, and when it finds a document that matches the user search, it uses a preset algorithm to calculate the document Compare the matching degree of search terms. Use the same method to retrieve the relevance of each relevant document in the literature resource database, and then return the corresponding documents to the user according to the order of relevance. To facilitate understanding, this paper uses word frequency and location weighting algorithms (that is, giving different weights to the title, subtitle, abstract, text, reference of the document, etc., and then multiplying the location weight with the matching degree of the location to obtain a sub Eigen value, add the Eigen values of all positions to get a final Eigen value) Calculate the Eigen value, use the Quicksort Algorithm to sort the Eigen values, and then sort and output the documents in the sorted sequence. In order to better meet the needs of users, we preset several priority selection buttons under the search interface. When users pay attention to the matching degree of search words in a certain position of the document, they can click the button and the background will check that position. The weight is weighted. Through this method, the document resource database can efficiently retrieve documents that match the user's needs.

4. Quicksort Algorithm Design

Assuming that the online literature resource library to be selected already exists, the order of the literature is random. Simulate the user's input of search terms, regard the search terms as a pattern string, and the documents in the resource library as the target string. Match the target string and the pattern string formed by each document (KMP Algorithm principle). If there is a segment equal to the pattern string in the target string, that is, the target substring, it means that a match is successful, and the document Eigen value is weighted once Processing, otherwise the matching is unsuccessful.

4.1 Design and Calculation of Document Matching

A document resource database of 15 documents has been simulated and established, simulating user needs to input search terms, the search terms are used as pattern strings, and the documents to be retrieved are used as target strings, and matching is performed according to the KMP (The Knuth-Morris-Pratt) algorithm.

Set the pattern string to the sliding window to start matching with the target string one by one. The matching process is shown in the following simulation:

```
First match:        Target string      X Y X Y Z X Y Z X Z Y X Y
                                       = = !=
Pattern string (search keywords)       X Y Z X Z
Second match:       Target string      X Y X Y Z X Y Z X Z Y X Y
                                           = = = = !=
Pattern string (search keywords)           X Y Z X Z
Third match:        Target string      X Y X Y Z X Y Z X Z Y X Y
                                                 = = = = =
Pattern string (search keywords)                 X Y Z X Z
```

In this simulation display, when the first match is performed, the third character is not equal. At this time, according to the principle of the KMP algorithm, the pattern string slides back two characters, and the third character is compared one by one again. When encountering a situation where the comparison characters are not equal again, slide and compare according to the same principle until the pattern string slides to the end of the target string.

4.2 Design and Calculation of Document Eigenvalues

How does the matching degree of the user's demand reflect? It can be reflected in this way. First, we assume that when the searched matching position is at the document title, a certain weight is added to the document, and the corresponding weight needs to be added for each match. Similarly, when the searched matching position is in the subtitle, in the text, or in the document, the specified weight is added, and the weight is added once for each match. In addition, in order to consider the value of the literature itself and the fluidity brought about by mutual references between the literature. We set that when a document is cited once, it also needs to be marked once, and the corresponding weight is added to increase its relevance. This requires that the documents in the database have established links. The more citations, the more authoritative and valuable the documents, and should be output first. Secondly, the number of downloads of a document can also reflect the needs of users. A document is marked once every time it is downloaded, weighted, and finally the corresponding Eigen value of each document can be obtained according to the formula.

The above fully demonstrates the method of using position weighting to calculate Eigen values to represent the relevance of documents in the conventional mode, and user needs are further considered here. If the user pays more attention to the matching degree of the terms in the title when searching documents, then we will weight the matching weight at the title to meet the needs of this user. Similarly, when users feel that the degree of matching in the text is more important, we give the weight of the text a proper weight. How to show this choice? We envisage adding a few more priority matching buttons on the Information Retrieval interface, giving priority to the corresponding positions, and users can choose by themselves.

We preset the weight settings as shown in Table 1.

According to the above rules, the Information Retrieval system is constructed. When the user enters the information to be retrieved in the search box, the program starts to analyze and calculate the Eigen value of each document in the resource library. The Eigen value calculation principle is R = ∑[(The weighted coefficient + (Priority weighting)) * Matching success times], the finally calculated Eigen value defaults to the retrieval relevance, importance, and user demand of the corresponding literature, but these Eigen values are still arranged in disorder. At this time, it is necessary to introduce a Quicksort Algorithm, and use the

Eigen value of the literature as the sorting element to sort and output the literature in the resource library, so that users can obtain better literature resources first. The use of Quicksort Algorithm is to improve the efficiency of the system, so that users can retrieve the desired results as quickly as possible.

Table 1: Principles for setting the weight of the Retrieval algorithm

Match success location	The weighted coefficient	Priority weighting
Title	5	2
Subtitle	4	3
Abstract	3	4
Keywords	5	2
Text	2	5
References	2	5
Number of downloads	5	2

5 Implementation and Analysis of the Quicksort Algorithm

The working principle of quick sort is Divide and Conquer, namely, a huge problem that needs to be dealt with is transformed into several small problems. These small problems are essentially the same as the original problem, but they are far less complex than the original problem. In this way, the decomposition layer by layer is approached successively, and finally the big problem is solved. In the sorting process, introducing the idea of quick sorting can effectively improve the efficiency of Information Retrieval.

Use the eigenvalue of the document as the key, and use the Quicksort Algorithm and several traditional sorting algorithms to sort the output, and compare their operating efficiency. After actual simulation, we will find that the Quicksort Algorithm is significantly better than the traditional sort algorithm O(n^2) in time complexity. Output the sorted documents, that can meet the user's retrieval needs. The following is a comparison simulation with a set of eigenvalues.

After a predetermined Weighted Rule, the feature value of each document is calculated, and finally the 15 documents in the simulated resource library have obtained their eigenvalue, and these eigenvalues are recorded on each document as a mark of the document. As shown in Table 2, at this time, these 15 eigenvalues are still out of order and cannot be provided to users. At this time, the Quicksort Algorithm needs to be executed.

Table 2: The simulated eigenvalues of the first 15 documents sorted

Document serial number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Eigenvalue	4	10	11	8	5	12	14	10	10	10	1	2	1	9	14

5.1 Simulation Implementation of the Quicksort Algorithm

Use the Quicksort Algorithm to sort documents with eigenvalue as keywords.

For example, a group of documents with feature values {4,10,11,8,5,12,14,10,10,10,1,2,1,9,14} are sorted by the Quicksort Algorithm:

a: {4,10,11,8,5,12,14,10,10,10,1,2,1,9,14}

For the first execution, use the document with the eigenvalue of 4 in the first position as the reference, and partition. The document with the eigenvalue greater than it is listed on the right, and the remaining columns are on the left;

b: {1,2,1,4,5,12,14,10,10,10,8,11,10,9,14}

In the second execution, the fifth-ranked document with the eigenvalue of 5 is used as the reference, and the partition is:

c: {1,2,1,4,5,12,14,10,10,10,8,11,10,9,14}

In the third execution, the first document with the eigenvalue of 1 is used as the reference, and the partition is:

d: {1,1,2,4,5,12,14,10,10,10,8,11,10,9,14}

In the fourth execution, the document with the eigenvalue of 12 in the sixth position is used as the reference, and the partition is:

e: {1,1,2,4,5,9,10,10,10,10,8,11,12,14,14}

In the fifth execution, the document with the eigenvalue of 9 in the sixth position is used as the reference, and the partition is:

f: {1,1,2,4,5,8,9,10,10,10,10,11,12,14,14}

After five operations, the simulation sorting is completed, and the sorted documents are output in order, which can meet the user's retrieval needs.

5.2 The Advantages of the Quicksort Algorithm over Merge Sort Algorithm

The time complexity of Merge Sort is also O(nlogn), which is also better than traditional sorting algorithms. Comparing it with Quicksort Algorithm, it can intuitively reflect the advantages of Quicksort Algorithm over Merge Sorting algorithms and other sorting algorithms.

1: Now compare the Quicksort Algorithm and Merge Sort by simulation experiment:

Experimental environment: ubuntu operating system with configured C language and C++ environment.

(1) Merge Sort

The operation result is shown in Fig. 1:

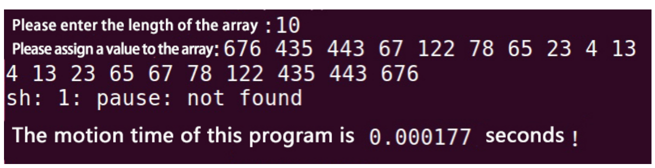


Figure 1: The experimental results of the Merge Sort algorithm when arranging 10 numbers

(2) The Quicksort Algorithm:

The operation result is shown in Fig. 2:

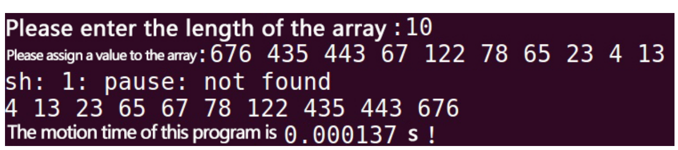


Figure 2: The experimental results of the Quicksort Algorithm when arranging 10 numbers

The comparison shows that when the data to be sorted is small, the Quicksort Algorithm may be faster than the Merge Sort algorithm (because only one experimental result is simulated, so no conclusion can be drawn!), we will further increase the length and the degree of randomness fully proves that when the data is large enough, the Quicksort Algorithm has an absolute advantage over the Merge Sort.

2: Now use the arrangement of ten arrays with lengths of 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000 to compare the time complexity of these two algorithms.

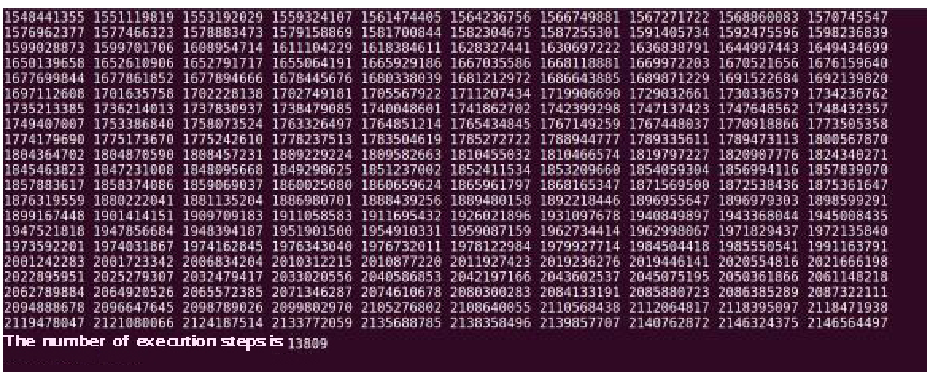
Experimental environment: Ubuntu operating system with configured C language and C++ environment.

Here we need to slightly modify the previous algorithm, add a Random function to generate random numbers, and sort them. When the array is greater than 700, the execution steps can be used to replace the running time:

(1) The Merge Sort algorithm randomly calls part of the function code:

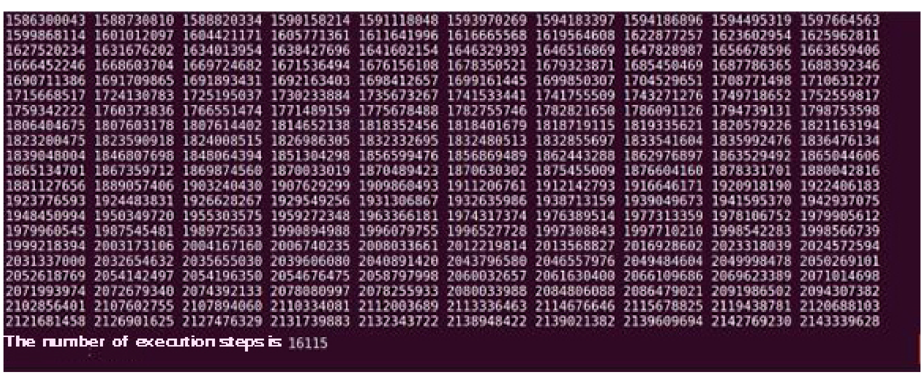
```
int const n(700);
  int a[n];
  srand((int)time(NULL));
  for(int i=0;i<n;i++)
     a[i]=rand();
  mergeSort(a,0,n-1);
  for(int i=0;i<n;++i){
     cout<<a[i]<<" ";
     if((i+1)%10==0) cout<<endl;}
  cout<<endl;
  cout<<"The number of execution steps is:"<<count<<endl; //count set as a global variable
  return 0;
```

The results of counting the number of steps performed when the array length is 700, 800, 900, and 1000 are shown in Fig. 3 to Fig. 6:



```
1548441355 1551119819 1553192029 1559324107 1561474405 1564236756 1566749881 1567271722 1568860083 1570745547
1576962377 1577466323 1578883473 1579158869 1581700844 1582304675 1587255301 1591405734 1592475596 1598236839
1599028873 1599701706 1608954714 1611104229 1618384611 1628327441 1630697222 1636838791 1644997443 1649434699
1650139658 1652610906 1652791717 1655064191 1665929186 1667035586 1668118881 1669972203 1670521656 1676159640
1677699844 1677861852 1677894666 1678445676 1680338039 1681212972 1686643885 1689871229 1691522684 1692139820
1697112608 1701635758 1702228138 1702749181 1705567922 1711207434 1719906690 1729032661 1730336579 1734236762
1735213385 1736214013 1737830937 1738479085 1740048601 1741862702 1742399298 1747137423 1747648562 1748432357
1749407007 1753386840 1758073524 1763326497 1764851214 1765434845 1767149259 1767448037 1770918866 1773505358
1774179690 1775173670 1775242610 1778237513 1783504619 1785272722 1788944777 1789335611 1789473113 1800567870
1804364702 1804870590 1808457231 1809229224 1809582663 1810455032 1810466574 1819797227 1820907776 1824340271
1845463823 1847231008 1848095668 1849298625 1851237002 1852411534 1853209660 1854059304 1856994116 1857839070
1857883617 1858374086 1859069037 1860025080 1860659624 1865961797 1868165347 1871569500 1872538436 1875361647
1876319559 1880222041 1881135204 1886980701 1888439256 1889480158 1892218446 1896955647 1896979303 1898599291
1899167448 1901414151 1909709183 1911058583 1911695432 1926021896 1931097678 1940849897 1943368044 1945008435
1947521818 1947856684 1948394187 1951901500 1954910331 1959087159 1962734414 1962998067 1971829437 1972135840
1973592201 1974031867 1974162845 1976343040 1976732011 1978122984 1979927714 1984504418 1985550541 1991163791
2001242283 2001723342 2006834204 2010312215 2010877220 2011927423 2019236276 2019446141 2020554816 2021666198
2022895951 2025279307 2032479417 2033020556 2040586853 2042197166 2043602537 2045075195 2050361866 2061148218
2062789884 2064920526 2065572385 2071346287 2074610678 2080300283 2084133191 2085880723 2086385289 2087322111
2094888678 2096647645 2098789026 2099802970 2105276802 2108640055 2110568438 2112064817 2118395097 2118471938
2119478047 2121080066 2124187514 2133772059 2135688785 2138358496 2139857707 2140762872 2146324375 2146564497
The number of execution steps is 13809
```

Figure 3: The experimental results of the Merge Sort Algorithm when arranging 700 numbers



```
1586300043 1588730810 1588820334 1590158214 1591118048 1593970269 1594183397 1594186896 1594495319 1597664563
1599868114 1601012097 1604421171 1605771361 1611641996 1616665568 1619564608 1622877257 1623602954 1625962811
1627520234 1631676202 1634013954 1638427696 1641602154 1646329393 1646516869 1647828987 1656678596 1663659406
1666452246 1668603704 1669724682 1671536494 1676156108 1678350521 1679323871 1685450469 1687786365 1688392346
1690711386 1691709865 1691893431 1692163403 1698412657 1699161445 1699850307 1704529651 1708771498 1710631277
1715668517 1724130783 1725195037 1730233884 1735673267 1741533441 1741755509 1743271276 1749718652 1752559817
1759342222 1760373836 1766551474 1771489159 1775678488 1782755746 1782821650 1786091126 1794739131 1798753598
1806404675 1807603178 1807614402 1814652138 1818352456 1818401679 1818719115 1819335621 1820579226 1821163194
1823200475 1823590918 1824008515 1826986305 1832332695 1832480513 1832855697 1833541604 1835992476 1836476134
1839048004 1846807698 1848064394 1851304298 1856599476 1856869489 1862443288 1862976897 1863529492 1865044606
1865134701 1867359712 1869874560 1870033019 1870489423 1870630302 1875455009 1876604160 1878331701 1880042816
1881127656 1889057406 1903240430 1907629299 1909860493 1911206761 1912142793 1916646171 1920918190 1922406183
1923776593 1924483831 1926628267 1929549256 1931306867 1932635986 1938713159 1939049673 1941595370 1942937075
1948450994 1950349720 1955303575 1959272348 1963366181 1974317374 1976389514 1977313359 1978106752 1979905612
1979960545 1987545481 1989725633 1990894988 1996079755 1996527728 1997308843 1997710210 1998542283 1998566739
1999218394 2003173106 2004167160 2006740235 2008033661 2012219814 2013568827 2016928602 2023318039 2024572594
2031337000 2032654632 2035655030 2039606080 2040891420 2043796580 2046557976 2049484604 2049998478 2050269101
2052618769 2054142497 2054196350 2054676475 2058797998 2060032657 2061630400 2066109686 2069623389 2071014698
2071993974 2072679340 2074392133 2078080997 2078255933 2080033988 2084806088 2086479021 2091986502 2094307382
2102856401 2107602755 2107894060 2110334081 2112003689 2113336463 2114676646 2115678825 2119438781 2120688103
2121681458 2126901625 2127476329 2131739883 2132343722 2138948422 2139021382 2139609694 2142769230 2143339628
The number of execution steps is 16115
```

Figure 4: The experimental results of the Merge Sort Algorithm when arranging 800 numbers

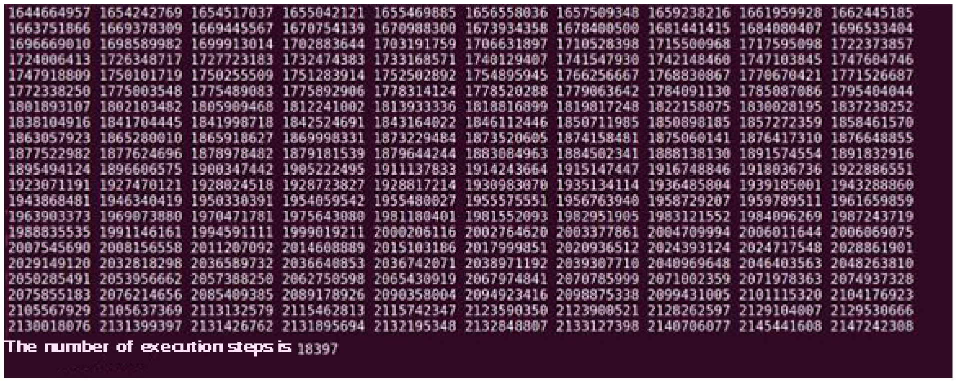


Figure 5: Experimental results of Merge Sort Algorithm when arranging 900 numbers

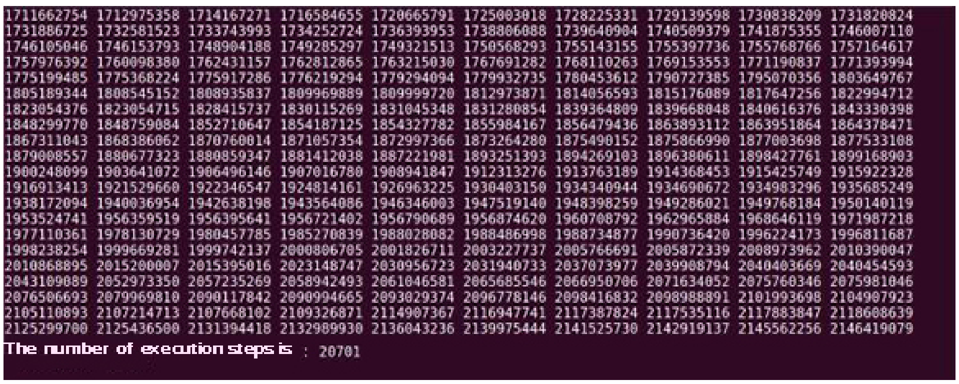


Figure 6: The experimental results of the Merge Sort Algorithm when arranging 1000 numbers

(2) The Quicksort Algorithm randomly generates part of the function code:

```
int const n(1000);
   int a[n];
   srand((int)time(NULL));
   for(int i=0;i<n;i++)
      a[i]=rand();
   Qsort(a,0,n-1);
   for(int i=0;i<n;i++){
      cout<<a[i]<<" ";
      if((i+1)%10==0)
         cout<<endl;
   }
   cout<<"The number of execution steps is"<<count<<endl;// Set global variables count
   return 0;
```

The results of counting the number of steps performed when the array length is 700, 800, 900, and 1000 are shown in Fig. 7 to Fig. 10:

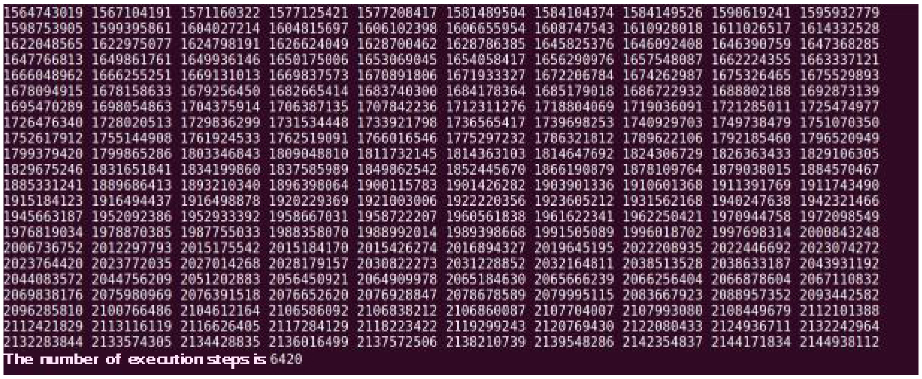


Figure 7: Experimental results of the Quicksort Algorithm when arranging 700 numbers

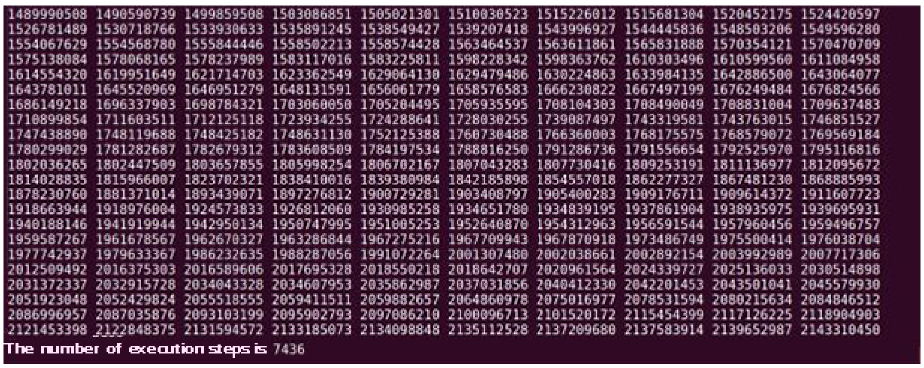


Figure 8: Experimental results of the Quicksort Algorithm when arranging 800 numbers

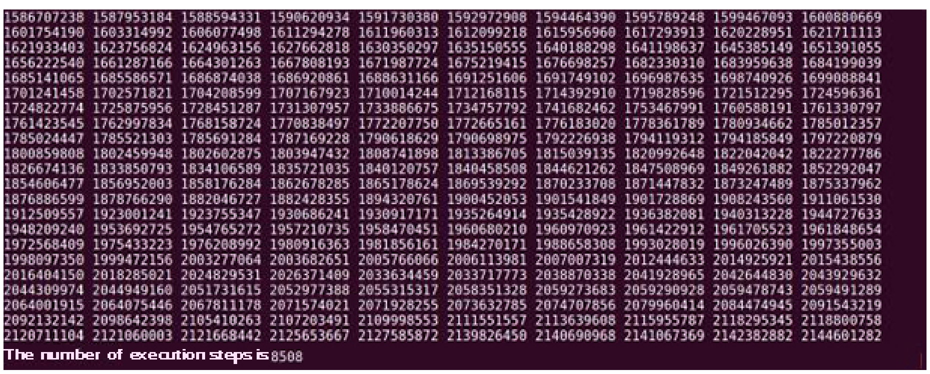


Figure 9: Experimental results of the Quicksort Algorithm when arranging 900 numbers

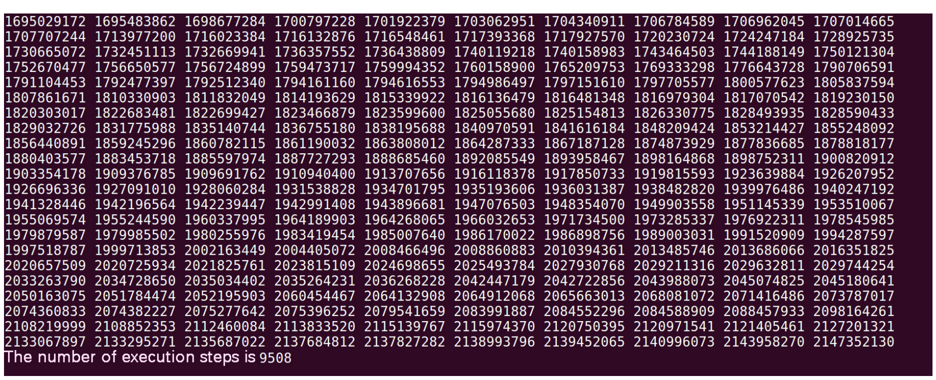


Figure 10: The experimental results of the sorting algorithm when arranging 1000 numbers

(3) Draw a broken line as shown in Fig. 11 for the result obtained:

n	100	200	300	400	500	600	700	800	900	1000
Merge							13809	16115	18397	20701
Quicksort							6420	7436	8508	9508
nlogn	664.3856	1528.771	2468.646	3457.542	4482.892	5537.291	6615.848	7715.085	8832.403	9965.784

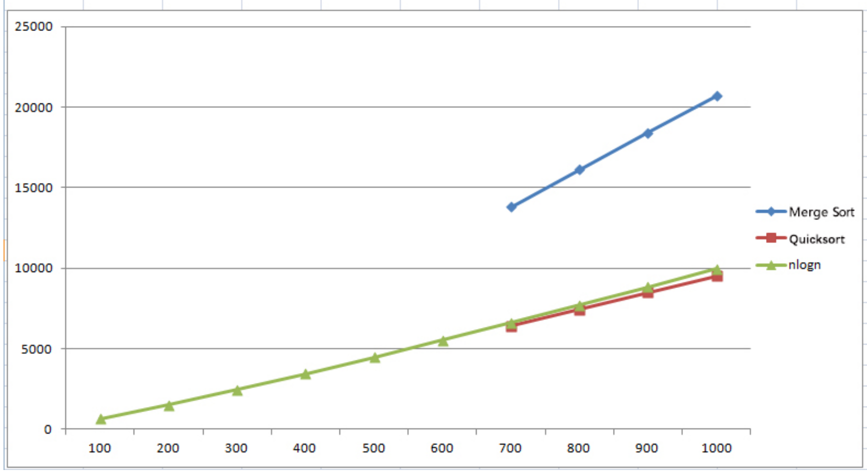


Figure 11: Comparison of the running speed of the Merge Sort and the Quicksort Algorithm when n = 700 to n = 1000

It can be seen from Fig. 11 that when the data becomes larger and larger, the time complexity of the Quicksort Algorithm is almost O(nlogn), and the time complexity of the Merge Sort has far exceeded O(nlogn). Since the data is randomly generated, it can basically represent generality. Therefore, it can be proved that the Quicksort Algorithm is better than the Merge Sort when the number of permutations increases. the Quicksort Algorithm is more adaptable to the increasing number of documents, and can better improve the efficiency of Information Retrieval!

6 Conclusion

In recent years, with the increasing number and variety of documents on the Information Retrieval platform and the ever-expanding demand of users, the society urges us to put forward higher requirements for the technology and efficiency of the Information Retrieval. In the design ideas of the Information Retrieval system in this paper, we fully refer to the more common design ideas of position weighting and user behavior feedback in current Information Retrieval engines, and combine the characteristics of Information Retrieval to increase the function of independent selection by users. It further improves the comprehensive indexes such as the matching degree, value, importance, and user needs of the retrieved documents. While improving the retrieval accuracy, the quick sorting algorithm is introduced to improve the sorting rate of document eigenvalue, optimize the performance of the Information Retrieval system, and finally achieve the effect of searching documents that meet the needs at the fastest speed, which has great applications prospect.

Funding Statement: This work was supported in part by the National Natural Science Foundation of China, Grant No. 72073041; Open Foundation for the University Innovation Platform in the Hunan Province, Grant No. 18K103.2011; Collaborative Innovation Center for Development and Utilization of Finance and Economics Big Data Property. Hunan Provincial Key Laboratory of Finance & Economics Big Data Science and Technology; 2020 Hunan Provincial Higher Education Teaching Reform Research Project under Grant HNJG-2020-1130, HNJG-2020-1124; 2020 General Project of Hunan Social Science Fund under Grant 20B16.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] W. M. Yan and W. M. Wu, *Data Structure*. Beijing: Tsinghua University Press, 2016.

[2] F. Cheng, "Research on Information Retrieval model based on ranking learning," Doctoral dissertation, University of Science and Technology of China, 2012.

[3] J. Yao, "Information Retrieval technology based on the quicksort algorithm," Anhui Vocational and Technical College of Posts and Telecommunications, Hefei, 2014.

[4] H. X. Zuo, C. Zhang, L. Peng and Y. M. Niu. "CINAHL database retrieval system and related retrieval methods," *Chinese Journal of Evidence-Based Cardiovascular Medicine*, 2017.

[5] J. J. Tian, "Overview of two sorting algorithms based on divide and conquer strategy," School of Computer and Information Engineering, Tianjin Normal University, Tianjin, vol. 10, 2015.

[6] Q. Guo, "Expansion research based on the Quicksort method," Department of Information Management, Liaoning University of International Business and Economics, Dalian, vol. 5, 2017.

[7] Q. F. Wang and Y. Y. Wei, "Research on literature ranking method based on citation context analysis," School of Information Management, Central China Normal University, Wuhan, vol. 16, no. 5, pp. 146–169, 2017.

[8] S. L. Lou and Y. Xu, "Analysis of the impact of keyword expansion on search results," Hubei Center for Patent Examination Cooperation of the State Intellectual Property Office, Wuhan, vol. 27, no. 1, pp. 298, 2017.

[9] L. Y. Li, "Research on Semantic-based Technology Information Retrieval," Ph.D. dissertation, Master thesis, Guizhou University, Guizhou, 2011.

[10] Z. B. Huang, Q. Zhao and S. Y. Sun, "Design and optimization of multithreaded quick sort based on Java," North China Institute of Computer System Engineering, vol. 35, no. 16, 2016.

[11] Z. W. Wang, "Research and improvement of PageRank algorithm in Information Retrieval ranking," Ph.D. dissertation, Master's thesis, Nanchang University, 2016.

[12] R. Wang, S. Chen Shu and B. Zeng. "Research on Information Retrieval system based on metadata," Ph.D. dissertation, Naval Engineering University of Chinese People's Liberation Army, Wuhan, 2017.

[13] Y. Jiang, "Research and implementation of Information Retrieval engine technology," Ph.D. dissertation, Master's thesis, Guizhou University, Guizhou, 2011.

[14] M. Thelwall, "Exttracting macroscopic information from Web links," *Journal of the American Society for Information Science & Technology*, vol. 51, no. 13, pp. 1157–1168, 2012.

[15] M. J. K. E. Han, "Stablishing sustainable and scalable workflows for cataloging and metadata services," *Library Management*, vol. 2016, no. 6, pp. 308–316, 2016.

[16] C. F. Huang, "A hybrid stock selection model using genetic algorithms and support vector regression original research article," *Applied Soft Computing*, vol. 12, no. 2, pp. 807–818, 2012.