Bond-Sequential Search (BSS)

Omer H. Abu El Haija, and Azmi Alazzam

Abstract—**This paper presents a new search algorithm for searching a list or array for two or more keys at the same time. The new algorithm is named Bond Sequential Search (BSS), and the objective in this algorithm is to search for two keys in one array using sequential search with some enhancements.**

The classic sequential search is one of the easiest and cheapest search techniques, but it is very slow and requires comparing each element in the array until finding the desired element. This process is time consuming, and in the worst case when the element is at the end of the array or when it does not exist in the array, the number of comparisons required will be equal to the array size. Thus, the complexity of the sequential search will be equal to the length of the search array, and if two keys are searched this complexity will double.

There are other faster search methods used like the binary search and the parallel sequential search. However, each of the two methods has its own disadvantages; binary search requires sorting the data, and parallel search requires the use of multiple processors, and both methods will require additional cost compared to sequential search.

Bond Sequential search (BSS) introduced in this paper is based on sequential search. Simple logic gates were added to the classic sequential search in order to combine two or more searches on one cycle, which enhanced the performance of classic sequential search without any need for data sorting as in the binary search or using multiple processors as in the parallel search.

The BSS algorithm is discussed and explained in details in this paper. The BSS is also compared to the classic sequential search, and the results showed that our algorithm is two times faster than the classic sequential search.

Index Terms—**Sequential search, Bond Sequential search, Binary Search, Parallel Search.**

Manuscript received July 27, 2012, revised August 16, 2012.

Omer H. Abu El Haija is with the Jordan University of Science and Technology, Irbid, Jordan (Tel:+(962)799145454 email:omer.haijaa@gmail.com)

Azmi Alazzam is with the State University of New York at Binghamton, Binghamton, NY 13902, USA (Tel:(832) 206-351 email:aalazza1@binghamton.edu)

I. INTRODUCTION

THE huge amount of modern era data require choosing fast searching mechanisms. Different search mechanisms have been proposed and used to facilitate the process of searching for small list items or in unsoted large databases. The best search algorithm that can be used in any search scenario is the one that save time and power without high cost calculations or hardware modifications. Determination of that best fit mechanism depends on many factors; those factors are related to list properties, searching environment and hardware specifications. The factors that are used to determine the best search algorithm are related to previse knowledge of: list order (order list vs. disorder list), searching paradigm (external vs. internal searching), size of data (small vs., large list size), and list updating (static vs. dynamic updating lists).

Sequential search is a good technique in both cases when the search list size is small and when the list is unsorted. However sequential search doesn't have the ability to search for more than one key at the same time, unless costly parallel techniques are used, also binary search cannot be used unless costly sorting techniques were pre applied.

Many researchers have discussed the use of different searching algorithms, also discussed different techniques and modifications to enhance the searching process. Donald Knuth wrote "Searching is the most time –consuming part of many programs, and the substitution of a good search method for a bad one often leads to a substitution increase in speed" [1]. It is known that the best running time in non-parallel searching environment is when the list is in order because binary search has a complexity of $\Theta(\log n)$ time which is better time than sequential that has a complexity of $\Theta(n)$ [2] .

It was also suggested that sorting process is not desirable especially when the list or array to be sorted is large in size, since sorting time will be long which will lead to slowing the searching time, increasing the search complexity and degrading the performance of the search mechanism [3] and [4]. A searching mechanism using floating point was also proposed [5]. The idea of this algorithm was based on mantissa separation inside bins, where logic instruction and masked formations can maintain significant bit. A random algorithm for multiselection was recently introduced; the searching is used to find the i-th position key, and also multi selection and quick selection were used for the same reason [6]. Although binary search is preferred than sequential search since it has less complexity but it also has its own drawbacks. It was found that Binary search is not suitable for searching in m dimension list where m >1 [7].

Multithreading could be used to enhance sequential search and can beat binary search because binary search doesn't support frequent updates and requires time consuming sorting [8]. In [9] external search without sorting

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

was discussed and two scenarios were elaborated; whether in practice one can perform external selection faster than sorting, and if there is a deterministic algorithm that has the same performance of randomized algorithm.

In [10] a hint of using integers sign bit for the Boolean values was proposed. In [11] the authors suggested forming data structure to speed up the searching processes. In [5] a searching technique inside compressed data is proposed, the technique compressed the key the same way the file was compressed, then bit by bit is used to search without the need of decompression. It was also found using bit manipulation can reduce sequential search time significantly [12]. Basic and intermediate ways to use logic gates introduced [13]. A recursive sequential search was also used to improve the search speed of patterns [5].

While sequential search takes one key at once, our proposed sequential based algorithm facilitate logic gates and mask formation to combine two or more keys performing better searching, by eliminating the excluded list items and taking equality candidates. Finally the algorithm shift to normal equality comparison for candidates process, and knows if the candidate item matches one of the two keys.

Our algorithm has the ability to combine two or more key to one composite key. The composite key is used to look for two or more keys on the same time. BSS takes half cycles compared to sequential search and have better performance if two or more keys are used; making BSS superior over sequential search in most cases.

In section II BSS is discussed in more details. In section III the numerical results are shown and BSS simulation results are compared to sequential search. We find that BSS has better performance and is 100% faster than classic sequential search when we aim to search two or more keys. In section IV the conclusion is discussed, also a comparison of time complexity between sequential and the proposed BSS is shown.

II. Materials and Methods

A. *Logic formation and nondeterministic comparison for one key*

Comparison for identifiers equality has classic syntax:

Left identifier == right identifier

For nondeterministic equality used by BSS both and gate, or gate, and classic equality are used.

The nondeterministic equality for single non composite key has two phases

Phase 1: The choice of candidate is done by applying and gate, between key and test item, let's have key k and item b to compare, our comparison equation is:

K and b

Phase 2: next is testing equality between phase 1 result and candidate so when the comparison of (phase1 results == b) returns true then b are candidate and it has a solution where

A = {b==k, b==0} and b ϵ A

b==k solution take place when the candidate is equal to the key

b==0 solution happen when the candidate fail, and this solution appears in rare cases .

B. *Logic formation and nondeterministic comparison with composite key*

The steps for comparing more than one key on the same process are:

1. We use or gate to combine source keys ka, kb to one composite key kc, where kc = ka or kb
2. We test element b to find candidate q, If b and kc = b, then we have candidate q, where q = kc and b with solution set A={q==ka, q==ka, q==kb, q==0} and q ϵ A.
3. We test candidate q for classic equality within solution set {q=ka, q=kb}, if q matches one of the deserved keys, else if no one of the deserved keys are identical then we exclude this candidate.

C. *BSS algorithm assumption*

To understand the algorithm in details let us have some assumptions:

ka,kb two different keys that we want to look for
D is a list of items that we want to look in
n is the index of last item of D
i is the index of i th item

For classical sequential search ka must be compared with all D items figure 1, then the same procedure is done for the kb is done figure 2 consuming 2n cycles.

For BSS we combine both keys in a composite key kc = ka or kb

Then we test for candidate for all D items using

If (kc and D[i]) ==D[i]

For every candidate we find we try to test if it is identical with one of the compound keys, if it is identical then we reach our goal to find first item, else if it is not identical we exclude this candidate.

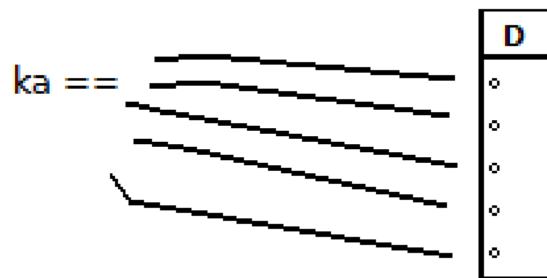

Fig. 1. Sequential search key ka comparison with all list items.

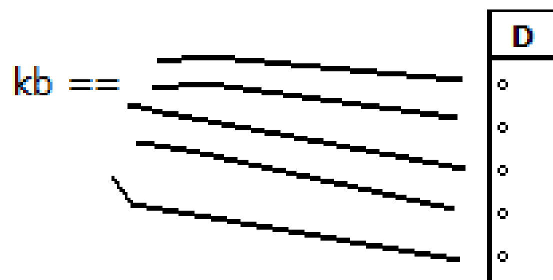

Fig. 2. Sequential search key kb comparison with all list items.

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

D. BSS algorithm pseudo code

```
seqsearch(D, first, last, key)
while key=D[first] and first< last
   first <- first + 1
if  key=D[first]
   return first
Bss(D, first, last, keya, keyb, keyapos, keybpos)
Keyc <- keya or keyb
Done <- 1
i <- first
while done>0 and first < last
   if keyc and D[first] = D[first]
                        if  keya = D[first]
                           done <- done -1
                                  keyapos <- first
                                  keybpos <- seqsearch(D, first,
                        last, keyb)
                        else
                           if  keya = D[first]
                              done <- done -1
                                         keybpos <- first
                                         keyapos <- seqsearch(D,
                                  first, last, keya)

   first <- first +1
```

The flow chart for the proposed BSS algorithm is shown in figure 3.

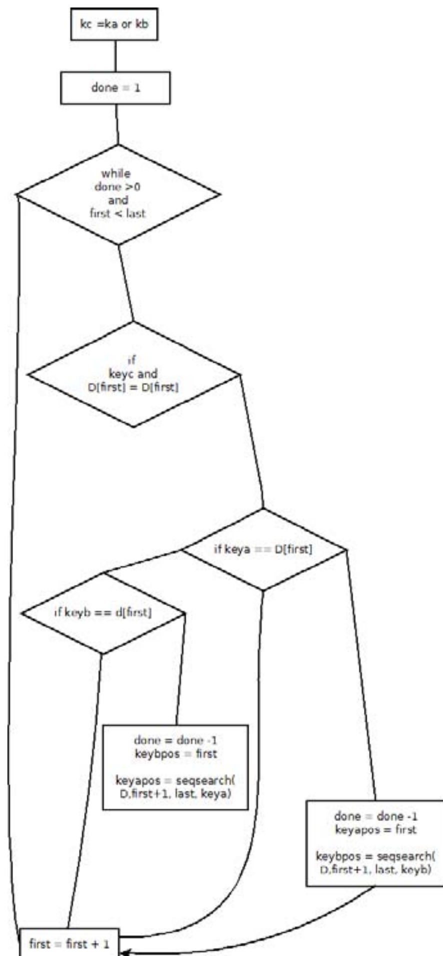

Fig. 3. BSS flow chart diagram.

E. Complexity analysis

Table 1 shows a comparison between three known searching algorithms: sequential search, BSS, and Binary search. For non sorted, dynamically changed arrays BSS beats binary search because binary search needs costly sorting, on the other hand BSS beats sequential search in most cases because it needs half of sequential search in most cases(see table 1.)

Table 1: running time comparison complexity for an array of size n with two keys searching

	sequential	BSS	Binary search
Comparison	2N	N	2 log n
Sorting	-	-	n log n .. n2
searching in the worst case	2n	N	2 log n
total running time	2n	N	2log n + n log n .. 2log n + n^2

Skipping non candidates and the ability to combine more than one key is what gives BSS the speed over classic sequential search.

III. RESULTS

To study the effectiveness of BSS with respect to sequential search algorithm, we chose to build a simulator using educational version of Microsoft C sharp, under Microsoft windows 7 multi-threading environment.

Many other authors tried to compare their results due to time, but it looks more applicable for multi core and multi-threading environment to compare due count of comparisons, so we counted comparisons for both BSS and sequential search, those comparisons are done to many different data sizes and for both cases; when key exist and when key does not exist.

Microsoft C sharp has integrated development environment under win32, it looks similar to C++ language. We ran our simulator on Intel T2300 running at 1.66 GHZ with 2GB RAM. Our simulation results showed that the performance of BSS is twice the performance of sequential search for the term of comparisons count as illustrated in table 2, table 3, table 4, table 5, figure 4, figure 5, and figure 6.

Bss is superior compared to sequential search, and it looks clear in figure 4, and figure 5.

Figure 4 is visual representation for comparison between sequential search and BSS on many data sizes and key sizes, when the key exists. The data that is shown here is same when key found on tables 2,3,4,5

Figure 5 is a visual representation for comparison between sequential search and BSS on many data sizes and key sizes, when the key doesn't exists. The data that is shown here is similar to that when the key is not found presented on tables 2,3,4,5.

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Table 2: BSS and sequential search small data size results

2^ data size	4	5	6	7
data size	16	32	64	128
2^ keys	1	2	3	4
Keys	2	4	8	16
seq found/compares	6	33	196	1,119
bss found/compares	5	33	136	777
seq ! found/compares	30	124	504	2,032
bss ! found/compares	16	64	255	1,020
FOUND TIME	0.83	1.00	0.69	0.69
!FOUND TIME	0.53	0.52	0.51	0.50
found performance	1.2	1.0	1.4	1.4
!found performance	1.9	1.9	2.0	2.0

Table 3: BSS and sequential search medium data size results

2^ data size	8	9	10	11
data size	256	512	1024	2048
2^ keys	5	6	7	8
Keys	32	64	128	256
seq found/compares	4,358	18,157	66,735	267,166
bss found/compares	2,762	10,695	37,339	143,265
seq ! found/compares	8,160	32,704	130,944	524,032
bss ! found/compares	4,083	16,359	65,484	262,027
FOUND TIME	0.63	0.59	0.56	0.54
!FOUND TIME	0.50	0.50	0.50	0.50
found performance	1.6	1.7	1.8	1.9
!found performance	2.0	2.0	2.0	2.0

Table 4: BSS and sequential search above medium data size results

2^ data size	12	13	14
data size	4096	8192	16384
2^ keys	9	10	11
keys	512	1024	2048
seq found/compares	1,032,530	4,261,386	17,252,084
bss found/compares	565,598	2,262,795	8,775,585
seq ! found/compares	2,096,640	8,387,584	33,552,384
bss ! found/compares	1,048,328	4,193,799	16,776,200
FOUND TIME	0.55	0.53	0.51
!FOUND TIME	0.50	0.50	0.50
found performance	1.8	1.9	2.0
!found performance	2.0	2.0	2.0

Table 5: BSS and sequential search large data size results

2^ data size	15	16	17
data size	32768	65536	131072
2^ keys	12	13	14
Keys	4096	8192	16384
seq found/compares	67,368,857	267,854,212	1,067,384,476
bss found/compares	34,189,460	135,939,450	538,662,587
seq ! found/compares	134,213,632	536,862,720	2,147,467,264
bss ! found/compares	67,106,824	268,431,363	1,073,733,633
FOUND TIME	0.51	0.51	0.50
!FOUND TIME	0.50	0.50	0.50
found performance	2.0	2.0	2.0
!found performance	2.0	2.0	2.0

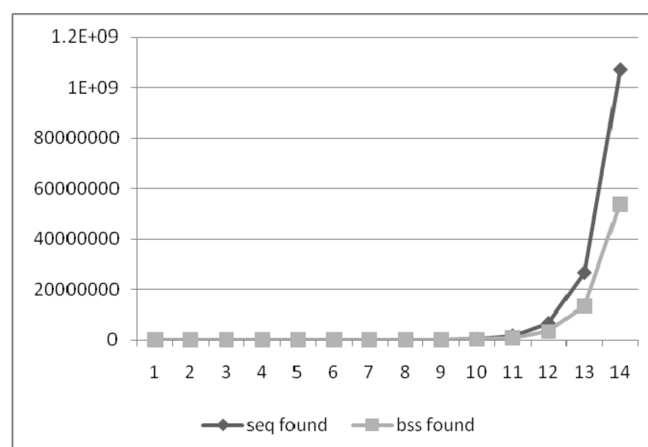

Fig. 4. Comparisons count for both BSS and sequential search when key exist.

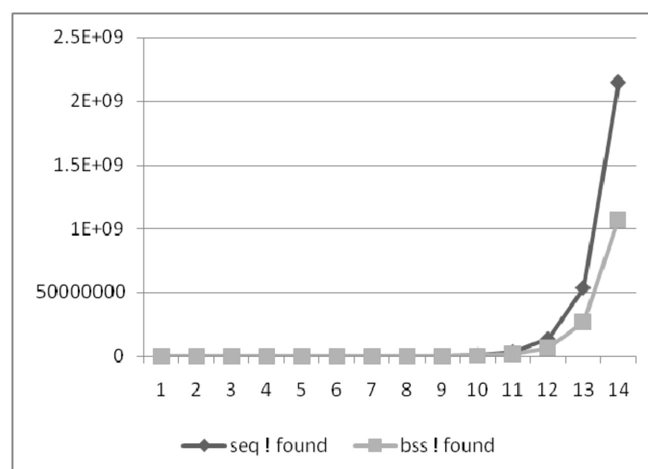

Fig 5. Comparisons count for both BSS and sequential search when key doesn’t exist.

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

The performance in both cases, when key exists and when it doesn't exist is shown in figure 6.

Also in figure 6 BSS shows that the proposed technique is twice the speed of sequential search in most cases.

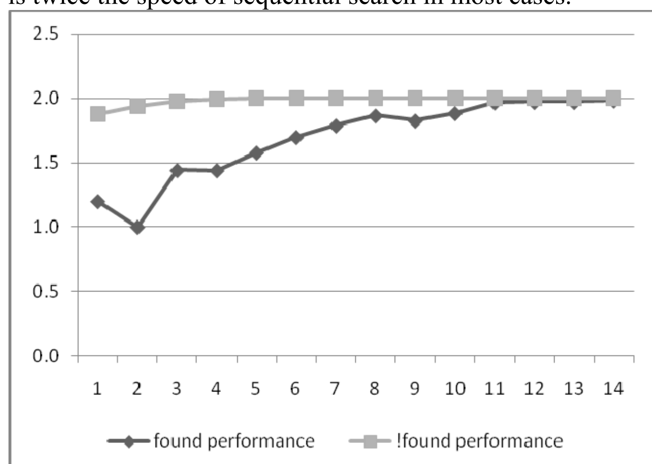

Fig. 6. Speed performance of BSS over sequential search in both cases when key exist and when key doesn't exist.

IV. Conclusion

In this study we proposed a novel searching algorithm that is based on logic instructions, masked formation, and sequential search. Comparison for equality of two identifiers has classic syntax:
Key Identifier1 == test identifier
Key Identifier2 == test identifier
For mask formation and logic gates we combine two keys:
Combined identifier = identifier 1 or identifier 2
And then one comparison is done between combined identifier and searching identfier3.
Masked formation and logic gates comparison way is not used mostly, because of its nondeterministic equality, but it is a good way to skip the non-candidates which is the reason of using it in this research.

Although BSS does two comparisons after finding candidates in some cases while sequential search has no candidates comparisons and find equality directly with classic comparison, but BSS is faster, because it can ignore all cases that doesn't belong to candidate set, by one comparison for finding candidate of two keys at the same time, while classic sequential search do two comparisons. BSS is more applicable than sequential search and if we only compare running time for two keys; BSS is ∑sipi where sequential search is 2∑sipi, where si is the size of data and pi is the size of keys we want to find. In the future work we will try to discuss multi-key version of BSS, also a paralleled version will be a good idea to see and compare the performance of BSS on parallel.

Acknowledgment

Thanks to my dad's soul for the knowledge he gave to me, my mom for giving me much care, my family for love, and all references for their valuable works that helped me to do this paper**.**

References

[1] D. E. Knuth. The Art of Computer Programming: Sorting and Searching, volume 3. Addison-Wesley,Reading, Mass., 3rd edition, 1998.
[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms. McGraw-Hill, 2nd edition, 2001.
[3] Andrew C. Yao. Should tables be sorted? J. Assoc. Comput. Mach., 31:245–281, 1984.13
[4] Lila Kaghazian, Dennis McLeod, Reza Sadri, Scalable complex patern search in sequential data,Acm, 2008
[5] Edleno Silva de Moura , Gonzalo Navarro , Nivio Ziviani , Ricardo Baeza-Yates, Fast and flexible word searching on compressed text, ACM Transactions on Information Systems (TOIS), v.18 n.2, p.113-139, April 2000
[6] M. H. Alsuwaiyel. A random algorithm for multiselection. Discrete Mathematics and Applications,16(2):175–180, 2006.
[7] G. Franceschini and R. Grossi. No sorting? better searching! In Proceedings of the IEEE Symposium on Foundations of Computer Science, 2004.
[8] Haboush, A. and S. Qawasmeh, 2011. Parallel sequential searching algorithm for unsorted array. Res. J. Applied Sci., 6: 70-75. 2011
[9] J. F. Sibeyn. External selection. Journal of Algorithms, 58:104–117, 2006.
[10] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. SIAM, Journal on Computing, 22(5):935–948, October 1993.
[11] J. Ian Munro and Hendra Suwanda. Implicit data structures for fast search and update. Journal of Computer and System Sciences, 21(2):236–250, 1980.
[12] John H. Reynolds, Using bit manipulation to reduce sequential search times, Journal of Computing Sciences in Colleges, v.17 n.2, p.263-270, December 2001
[13] M.Morris Mano , computer system architecture (3rd ed.) , Prentice – Hall , Inc . ,Upper Saddle River , NJ , 2002 .