Sequential vs. Binary Batched Searching

Y. MANOLOPOULOS
Department of Electrical Engineering, University of Thessaloniki, Thessaloniki, Greece

J. G. KOLLIAS†
Department of Electrical Engineering, Division of Computer Science, National Technical University of Athens, 9 Heroon Polytechniov Avenue, 157 73 Zographov, Athens, Greece

M. HATZOPOULOS
Department of Mathematics, University of Athens, Greece

This study considers an ordered array of N keys and estimates the required number of key comparisons to locate M requested keys when a binary search is performed to find each key. The problem is analysed for the cases where the M requests (a) are performed individually on a First Come First Served basis, and (b) are treated as a batch. For the second case a break point is established which indicates whether it is preferable to apply binary or sequential search for a batch of M keys.

Received September 1984

1. INTRODUCTION

Binary search is one of the most commonly used methods of searching for the existence or not of **one** key value in an **ordered** array which is maintained entirely in the high speed storage.[10] The method takes advantage of the fact that the array is ordered to eliminate half of the remaining array elements with each probe in the array. Binary search may also be applied to ordered files residing entirely in Direct Access Storage Devices (e.g. disks). For the rest of this study we concentrate on ordered arrays, and we postpone until section 4 the discussion on how the reported results may be extended to cover secondary storage structures.

Consider an ordered array of N keys and assume (a) that M random keys are to be searched, and (b) that the search cost metric is the number of key comparisons required. The following four possible search strategies may be considered:

1. **Sequential Search on a First Come First Served basis** (FCFSSS). Under this strategy the M searches are performed individually. The expected cost for satisfying all the M requests is obviously:

$$M(N+1)/2. \tag{1}$$

2. **Sequential Search when keys are sorted and batch examined** (BSS). Several authors[9, 12] have observed that (1) can be reduced if the M requests are considered as a batch. In particular, if the M keys have been sorted on the same order with the array then the expected cost is:

$$(N+1)\,M/(M+1). \tag{2}$$

A proof of the above formula can be found in the first appendix. This searching strategy starts with the first request in the batch and tries to locate the first key in the array. The strategy proceeds with the second, ..., Mth request but the array is searched from the point where the previous request has stopped. We shall call the strategy Batched Sequential Searching (BSS). If we assume that the cost of sorting is negligible then it is easy to verify that BSS is in general better than FCFSSS.

At this point we note that batching and BSS in particular have been studied in the past in the context of large information systems. For example, Burton and Kollias,[2] and Kollias[11] estimate the seek time when batching is used for primary and secondary key retrievals. Schneiderman and Goodman[12] consider a file and derive formula for the average relative savings of batching M requests over M successive tree searches. Wong[14] proposes some rules for the minimization of the expected head movement in one-dimensional and two-dimensional mass storage systems when batching is used. The effect of batching in database performance is studied in Refs. 1 and 4. In Ref. 3 an access mechanism is proposed to facilitate batching queries in large information systems.

The present study focuses on the following batched binary searching strategies (section 2).

3. **Binary Search on a First Come First Served basis** (FCFSBS). With this strategy the M keys of the batch are searched for individually. Section 2.1 analyses the performance of the strategy.

4. **Binary Search when keys are sorted and batch examined** (BSS). The BSS strategy assumes again that the M keys in the batch are sorted and tries to locate each key value in sequence by performing a binary search on the array. It is noted that Schneiderman and Goodman suggested the study of the BBS problem because 'this strategy is so frequently used'.[12] (For example, the vector searching problem (i.e. given a k-vector A and a set B of n distinct k-vectors determine whether or not A is a member of set B) can be solved by applying BBS).[8] The strategy is analysed in section 2.2. Section 3 introduces a break point which indicates whether it is preferable to perform the BSS or BBS strategies. Section 4 discusses the conclusion.

2. BATCHED BINARY SEARCHING STRATEGIES

In the following analysis we assume that retrieval requests are made randomly for the keys in the sorted array of N keys, so that each key is equally likely to be requested. We also consider that N is expressed as:

$$N = 2^k - 1 + A \quad \text{where} \quad 0 \leqslant A < 2^k.$$

If **one** binary search for finding one key is performed then the key may be located at best with just one comparison, and this happens with probability $1/N$. In

† This work is part of KRINO, a project aiming at improving software technology in Greece.

the **worst** case k or $k+1$ comparisons are made depending on whether $A = 0$ (with probability $2^{k-1}/N$) or $A \neq 0$ (with probability A/N). The **expected** number of comparisons is given by:

$$\sum_{i=1}^{k} i2^{i-1} + A(k+1) = (k+1) + (k+2-2^{k+1})/N. \quad (4)$$

(**Note**: a similar formula has been derived in Ref. 7.)

In the next two subsections we consider that binary search is applied to locate M distinct keys, where M is expressed as

$$M = 2^m - 1 + B \quad \text{where} \quad 0 \leqslant B < 2^m. \quad (5)$$

Subsection 2.1 (2.2) assumes that the keys are searched by applying the FCFSBS (BBS) strategy. In both analyses we assume that all the M keys exist in the table.

2.1. FCFSBS strategy

With this strategy one key is processed at a time. Below we analyse the performance of this strategy for the best, worst and average cases. The **optimal** number of comparisons is:

$$\sum_{n=1}^{\lfloor \log M \rfloor} n2^{n-1} + B\lceil \log M \rceil,$$

where $\lceil x \rceil$ ($\lfloor x \rfloor$) is the least (greatest) integer greater (less) than or equal to the real number x, and $\log M$ is the base 2 logarithm of M. This number is derived as follows. Suppose that binary searching corresponds to traversing a perfectly balanced tree of N nodes. Assuming that the M keys match the keys at the top of the tree we have the above number. This may occur with probability equal to:

$$\frac{1}{\binom{N}{M-B}} \frac{\binom{2^m}{B}}{\binom{N-M+B}{B}}.$$

If $B = 0$, then the probability reduces to:

$$\frac{1}{\binom{N}{M}}.$$

If $M \leqslant A$ or ($A = 0$ and $M \leqslant 2^{k-1}$) then the **worst** case of comparisons is $M\lceil \log N \rceil$ and it may occur with probabilities:

$$\frac{\binom{A}{M}}{\binom{N}{M}} \quad \text{or} \quad \frac{\binom{2^{k-1}}{M}}{\binom{N}{M}} \quad \text{respectively.}$$

If we have $0 \leqslant M - A < 2^{k-1}$ then the worst case is $A\lceil \log N \rceil + (M-A)\lfloor \log N \rfloor$ and it may occur with probability:

$$\frac{\binom{2^{k+1}+A}{M}}{\binom{N}{M}}.$$

For other conditions of the variables M, A and k similar results may be obtained by considering that the M keys match the keys at the lowest level(s) of the tree formed by the N keys.

From (4) it is clear that the **expected** number of comparisons for finding M keys in FCFSBS strategy is:

$$M((k+1) + (k+2-2^{k+1})/N) \quad (6)$$

2.2. BBS strategy

With this strategy the first, second, ..., Mth keys are again binary searched in the ordered table. However, the location of the first key in the array enables us to neglect part of the array in the searching for the second key. For example, if the first key is located at position i (where $1 \leqslant i < N$) then the searching for the next key may be reduced to the array elements $i+1$ to N instead of 1 to N which occurred in the FCFS strategy. The following algorithm illustrates further the BBS strategy. In this algorithm it is assumed that key_i keys are to be searched ($i = 1, 2, \ldots, M$) against the ordered array A of N keys. The variables low, high and mid are all indices to the elements of A.

```
found := false;
low := 1;
high := n;
mid := (low + high)/2;
for i := 1 to M do
  begin
    while low ⩽ high and not found do
      begin
        if key_i < A[mid]
          then
            begin
              high := mid − 1;
              mid := (low + high)/2;
            end
        else if key_i > A[mid]
          then
            begin
              low := mid + 1;
              mid := (low + high)/2;
            end
        else
          begin
            print (key_i, is in the table);
            low := mid + 1;
            found := true;
          end
      end
    high := n;
    mid := (low + high)/2;
    found := false;
  end
```

Figure 1. The BBS strategy

We now proceed to analyse the strategy for the best, worst and average cases. Before doing so we note that the array of dimension N is divided by the M keys in $M+1$ subintervals. We shall denote these subintervals by K_i, where $1 \leqslant i \leqslant M+1$. If the keys are randomly drawn from an underlying uniform distribution it can be proved that:

$$E[K_i] = (N-M)/(M+1).$$

The proof of the above equation can be found in the second appendix. Therefore, the number of keys remaining when the ith subinterval plus the already found key are exempted is:

$$N_i = N - i(E[K_i]+1) = N - i\frac{N+1}{M+1},$$

where $0 \leqslant i \leqslant M-1$.

The **best** case arises if the search of the M keys terminates with exactly M comparisons. This may occur with probability:

$$\prod_{i=0}^{M-1} \frac{1}{N_i}.$$

The **worst** case arises when every search reaches the lowest level of the tree and may occur with probability:

$$\prod_{i=0}^{M-1} \left(p\frac{2^{\log N_i - 1}}{N_i} + \frac{A_i}{N_i}\right),$$

where $p = 0$ if $A_i \neq 0$ and $p = 1$ if $A_i = 0$. The number of comparisons required in the worst case is:

$$\sum_{i=0}^{M-1} \lceil \log N_i \rceil.$$

Using a similar reasoning as with the derivation of (4) and (6) we observe that the mean value of comparison is:

$$\sum_{i=0}^{M-1} \{\lfloor \log N_i \rfloor + 1 + (\lfloor \log N_i \rfloor + 2 - 2^{\lfloor \log N_i \rfloor + 1})/N_i\}. \quad (7)$$

Table 1 presents the average performance of the FCFSBS and BBS strategies (as expressed by the formulae (6) and (7)) along with the percentage of gain achieved by using the BBS strategy as a function of M and N. It is easy to verify that (7) produces a smaller value than (6), i.e. the BBS outperforms the FCFSBS strategy.

Table 1. The average performance of FCFSBS and BBS strategies

M, N	100			1000			10000		
2	5.8	5.49	5.33%	8.99	8.73	2.83%	12.36	12.07	2.39%
5		5.13	11.57%		8.35	7.12%		11.66	5.67%
10		4.91	15.34%		8.11	9.67%		11.41	7.70%
20		4.75	18.11%		7.93	11.76%		11.23	9.19%
50		4.61	20.44%		7.78	13.44%		11.07	8.52%

3. BINARY vs. SEQUENTIAL BATCHING

In the previous sections we showed that the BSS and BBS strategies are always better than the FCFSSS and FCFSBS strategies respectively. We also showed that the average performances of the BSS and BBS strategies are expressed by the formulae (2) and (7) respectively. In this section we try to establish a threshold value of M, for a given N, beyond which BSS is preferable to BBS. We first prove the following lemma.

Lemma

Expression (7) can be approximated by the expression:

$$\frac{1}{\ln 2}\left(M\ln\frac{N}{2} - \frac{M-1}{M+1}\left(\frac{M}{2} + \frac{M(2M+1)}{12(M+1)} + \frac{M-1}{12}\right)\right). \quad (8)$$

Proof

Following [7] we have that

$$\sum_{i=0}^{M-1} (\lfloor \log N_i \rfloor + 1 + (\lfloor \log N_i \rfloor + 1 - 2^{\lfloor \log N_i \rfloor + 1})/N_i)$$
$$\approx \sum_{i=0}^{M-1} (\lfloor \log N_i \rfloor + 1 - 2^{\lfloor \log N_i \rfloor + 1}/N_i)$$

By assuming that $[\log N_i] + 1 \approx \log N_i$ we have:

$$\sum_{i=0}^{M-1} (\log N_i - 2^{\log N_i}/N_i)$$
$$= \sum_{i=0}^{M-1} \log N_i - M$$
$$= \sum_{i=0}^{M-1} \log\left(N - i\frac{N+1}{M+1}\right) - M$$
$$\approx \sum_{i=0}^{M-1} \log N\left(1 - \frac{i}{M+1}\right) - M$$
$$= M\log N + \sum_{i=0}^{M-1} \log\left(1 - \frac{i}{M+1}\right) - M$$
$$= M\left(\frac{\ln N}{\ln 2} - 1\right) + \frac{1}{\ln 2}\sum_{i=0}^{M-1} \ln\left(1 - \frac{i}{M+1}\right)$$
$$= \frac{M}{\ln 2}\ln\frac{N}{2} - \frac{1}{\ln 2}\sum_{i=0}^{M-1}\left(-\frac{i}{M+1} - \frac{1}{2}\left(\frac{i}{M+1}\right)^2 - \frac{1}{3}\left(\frac{i}{M+1}\right)^3 + \dots\right)$$
$$= \frac{1}{\ln 2}\left(M\ln\frac{N}{2} - \left(\frac{M(M-1)}{2(M+1)} + \frac{(M-1)M(2M+1)}{12(M+1)^2} + \frac{(M-1)^2}{12(M+1)} + \dots\right)\right)$$
$$= \frac{1}{\ln 2}\left(M\ln\frac{N}{2} - \frac{M-1}{M+1}\left(\frac{M}{2} + \frac{M(2M+1)}{12(M+1)} + \frac{(M-1)}{12} + \dots\right)\right)$$
$$\approx \frac{1}{\ln 2}\left(M\ln\frac{N}{2} - \frac{M-1}{M+1}\left(\frac{M}{2} + \frac{M(M+1)}{12(M+1)} + \frac{M-1}{12}\right)\right) \quad \text{Q.E.D.}$$

Theorem

The threshold value M beyond which BSS is preferable to BBS is estimated by solving the equation:

$$M^2(\ln N - 1.44) + M(2\ln N - 0.69N - 1.22) + (\ln N - 0.69N) = 0.$$

Proof

By combining (8) and (2) we get

$$\frac{1}{\ln 2}\left(M\ln\frac{N}{2} - \frac{M-1}{M+1}\left(\frac{M}{2} + \frac{M(2M+1)}{12(M+1)} + \frac{M-1}{12}\right)\right) = (N+1)\frac{M}{M+1}.$$

After some algebra we get:

$$M^3\left(12\ln\frac{N}{2} - 9\right) + M^2\left(24\ln\frac{N}{2} + 2 - 12N\ln 2\right) + M\left(12\ln\frac{N}{2} + 8 - 12(N+1)\ln 2\right) - 1 = 0$$

$$M^2\left(12\ln\frac{N}{2} - 9\right) + M\left(24\ln\frac{N}{2} + 2 - 12N\ln 2\right) + \left(12\ln\frac{N}{2} + 8 - 12(N+1)\ln 2\right) = 0$$

$$M^2(\ln N-1.44)+M(2\ln N-0.69N-1.22)+(\ln N-0.69N)=0 \quad \text{Q.E.D.}$$

We finish by making two observations. First, equation (9) always gives real roots, one of which is always positive for $N \geqslant 5$. The second observation relates to the fact that in many practical situations it is advantageous to know a simple – but reasonably accurate – expression for the positive root of (9). The following corollary derives such an expression, which may be used by software designers for estimating purposes.

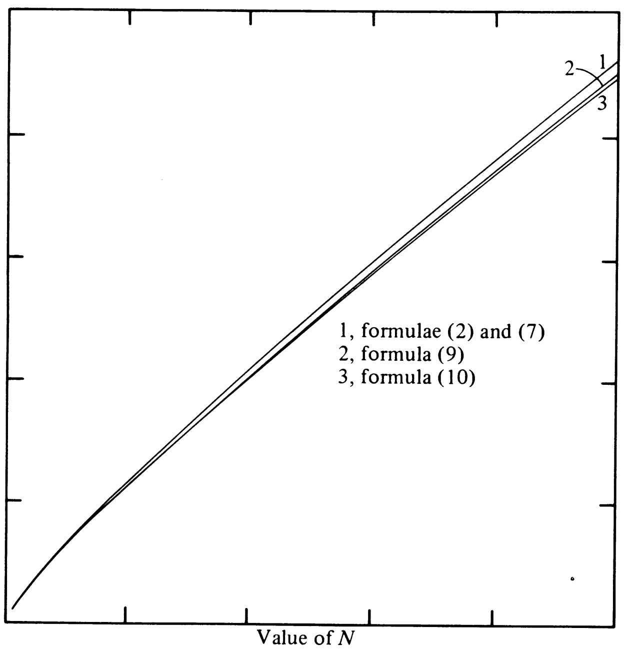

Figure 2. The threshold M beyond which BSS is preferable to BBS.

Corollary

If N is large then the positive root of (9) can be approximated by the formula:

$$0.69N/(\ln N-1.44) \qquad (10)$$

Proof

From (9) we have that:

$$M=\frac{(0.69N+1.22-2\ln N)+\sqrt{[2\ln N-0.69N-1.22)^2-4(\ln N-1.44)(\ln N-0.69N)}}{2(\ln N-1.44)}.$$

After some algebra it is easy to verify that the expression involving the operator square root approximates to $0.69N$. Therefore the positive root of (9) can be approximated by:

$$0.69N/(\ln N-1.44) \quad \text{Q.E.D.}$$

Fig. 2 provides a graph for the values of the threshold values M. The upper line has been derived by considering formulae (2) and (7), while the middle and the lower lines depict formulae (9) and (10) respectively. The fact that the deviation between the upper and middle (lower) line is less than 2.5% (3%) demonstrates the validity of the lemma (corollary).

4. CONCLUDING REMARKS

This study has analysed the Batched Binary Searching (BBS) strategy and established a break point between BBS and Batched Sequential Searching (BSS) strategy. We anticipate that the results of this paper may be used by software designers to automate the selection of an appropriate search strategy for their applications. Note here that one comparison in sequential search is cheaper than one comparison in binary search. It is evident that the value of the threshold M rises if this fact is taken into consideration. However, the evaluation of a corrective coefficient is not easy as it is machine-dependent.

Some of the results of this study may also be applied for secondary storage structures, where the search cost metric is the number of blocks accessed for satisfying the M requests. Consider for example an ordered file consisting of N blocks which is sequentially allocated on a disk without considering the effect of data management problems for handling eventual overflow records. If we make the added assumption that the record size approaches the block size then (1), (2), (6) and (7) are the search costs for the four possible strategies presented in Section 1.

The study can be extended and applied in many contexts. We just mention three possible extensions. First, to investigate further the four strategies for secondary storage.[4] (In fact one possible approach to the problem is shown in pp. 228–231 of Ref. 13). Secondly, to derive break points for other strategies as well, e.g. to establish a break point among FCFSSS and FCFSBS. Another extension in the same direction is to consider alternative structures of the ordered table, e.g. batched tree searching. Thirdly, to study the BBS strategy for the case of searching secondary indices, which are stored as lists of values, to find the pointers which satisfy a batch of queries based on secondary key values. Ref. 2 studies the same problem for the BSS strategy.

APPENDIX A

Proposition

The expected cost for satisfying all the M requests when the array and the M keys are sorted on the same order and the requests are batch-examined is: $(N+1)M/(M+1)$.

Proof

It is known from Ref. 4 that the probability that exactly k elements have to be examined sequentially in order to find all the M keys is:

$$P(k)=\frac{\binom{k}{M}-\binom{k-1}{M}}{\binom{N}{M}}.$$

Thus the expected cost is $\sum_{k=M}^{N} kP(k)$ and the following equation has to be proved:

$$\sum_{k=M}^{N} k\frac{\binom{k}{M}-\binom{k-1}{M}}{\binom{N}{M}}=(N+1)M/(M+1);$$

24-2

or equivalently by using the properties of combinations (see Ref. 6) we have that:

$$\sum_{k=M}^{N} k\binom{k-1}{M-1} = M\binom{N+1}{M+1}.$$

The proof follows by induction.

It is easily derived that for $N = M$ and $N = M+1$ the two sides of the relation are equal to M and $M(M+2)$ respectively. For $N = M+L$ we accept that the following equation holds:

$$\sum_{k=M}^{N+L} k\binom{k-1}{M-1} = M\binom{M+L+1}{M+1}.$$

We will prove that for $N = M+L+1$ the following equation holds:

$$\sum_{k=M}^{N+L+1} k\binom{k-1}{M-1} = M\binom{M+L+2}{M+1}.$$

$$\sum_{k=M}^{N+L+1} k\binom{k-1}{M-1}$$

$$= M\binom{M+L+1}{M+1} + (M+L+1)\binom{M+L}{M-1}$$

$$= M\left(\binom{M+L+1}{M+1} + \binom{M+L+1}{M}\right)$$

$$= M\left(\binom{M+L+1}{M+1} + \binom{M+L+1}{M}\right)$$

$$= M\binom{M+L+2}{M+1} \quad \text{Q.E.D.}$$

APPENDIX B

Proposition

The expected value of the length of the subintervals (*a*) between any two successive M elements or (*b*) between the beginning of the array and the first element of M or (*c*) between the last element of M and the end of the array is $(N-M)/(M+1)$.

Proof

From Ref. (5) it is known that the probability distribution of the three above cases is:

$$P(K) = \frac{\binom{N-K-1}{M-1}}{\binom{N}{M}},$$

where K is the size of the subinterval. Thus it is easily understood that the following equation has to be proved:

$$\sum_{K=0}^{N-M} K\frac{\binom{N-K-1}{M-1}}{\binom{N}{M}} = \frac{N-M}{M+1}$$

or equivalently:

$$\sum_{K=1}^{N-M} K\binom{N-K-1}{M-1} = \binom{N}{M+1}.$$

The left-hand side of the equation equals to:

$$\sum_{K=1}^{N-M}\binom{N-K-1}{M-1} + \sum_{K=2}^{N-M}\binom{N-K-1}{M-1} + \sum_{K=3}^{N-M}\binom{N-K-1}{M-1} + \dots + \sum_{K=N-M}^{N-M}\binom{N-K-1}{M-1}.$$

According to Ref. (6), formula 12.8, p. 64, the above relation becomes:

$$\binom{N-1}{M} + \binom{N-2}{M} + \binom{N-3}{M} + \dots + \binom{M}{M}$$

$$= \sum_{K=1}^{N-M}\binom{N-K}{M}$$

$$= \binom{N}{M+1} \quad \text{Q.E.D.}$$

Acknowledgement

The authors wish to thank the referee for his constructive criticism.

REFERENCES

1. D. S. Batory and C. C. Gotlieb, A unifying model of physical databases. *ACM Transactions on Database Systems*, **7**(4), 509–539 (1982).
2. F. W. Burton and J. Kollias, Optimising disc head movements in secondary key retrievals. *The Computer Journal* **22**(3), 206–208 (1979).
3. S. Christodoulakis, Access files for batching in large information systems. *Proc. ICOD-2 Conference, Glasgow 1983*, pp. 127–150 (1983).
4. S. Christodoulakis, Estimating block transfers and join sizes. *Proc. SIGMOD 83, San José*, pp. 40–54 (1983).
5. S. Christodoulakis, Analysis of performance of an archiver using optical disk technology (submitted for publication, 1985).
6. W. Feller, *An Introduction to Probability Theory and its Applications*. Wiley, Chichester 3rd edition (1968).
7. I. Flores and G. Madpis, Average binary search lengths for dense ordered lists, *Communications of ACM* **14**(9), 602–603 (1971).
8. D. S. Hirschberg, On the complexity of searching a set of vectors. *SIAM Journal on Computing* **9**(1), 126–129 (1980).
9. D. E. Knuth, *The Art of Computer Programming*, Vol. 1, *Fundamental Algorithms*. Addison-Wesley, Reading, Massachusetts (1968).
10. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, *Sorting and Searching*, pp. 406–419. Addison-Wesley, Reading, Massachusetts (1973).
11. J. Kollias, An estimate of seek time for batched searching of random or index sequential structured files. *The Computer Journal* **21**(2), 132–133 (1978).
12. B. Schneiderman and V. Goodman, Batched searching of sequential and tree structured files. *ACM Transactions on Database Systems* **1**(3), 268–275 (1976).
13. T. J. Teorey and J. P. Fry, *Design of Database Structures*. Prentice-Hall, Englewood Cliffs, N.J. (1982).
14. C. K. Wong, Minimizing expected head movements in one-dimensional and two-dimensional mass storage systems. *Computing Surveys of ACM* **12**(2), 167–178 (1980).