Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

There are various techniques for searching a data from a data base. One of them is interpolation search. It works on uniformly distributed and sorted numerical tables and considered to be one of the fastest methods. On an average this method takes *'lg lg n'*

Burton and Lewis [BL] shows the inefficiency of interpolation sarch for an alphabetic table whose distribution is not known or non-uniform. They introduce GAP variations of interpolation search to compare the inefficiency. However another approach to a non-uniform is to apply the cumulative distribution function F which transfer a non-uniform distribution to uniform one, for which interpolation search is the best.

In Arithmetic Coding a string of characters is mapped into the [0,1) interval according to the probabilities of its characters using arithmetic code. We found that this transformation, designed for data compression, is actually the cumulative distritution funciton F for alphabetic tables. However the tables needed for applying Arithmetic Coding require too much memory and only an approximated transformation using only few tables can be applied. This transformation gave a semi-uniform distribution and interpolation search gave higher results than *'lg lg n'*. Applying then the GAP variations improved the results where, the optimum close to *'lg lg n'* accesses was obtained for the accelerated GAP variation for GAP = 2 rather than $\sqrt{n}$ used in [BL]. An experimental analysis show GAP = 2 to be the best function for uniformly distributed files. We analyzed the regular GAP =2 theoretically to support the experimental result.

EXPERIMENTS WITH THE GAP VARIATION OF INTERPOLATION SEARCH FOR SEMI UNIFORM DISTRIBUTED ALPHABETIC FILES

by

Jatin M. Bhavsar

Submitted to

The Department of Computer and Information Science of the

New Jersey Institute of Technology

in partial fulfillment of the requirement for the degree of

Master of Science in Computer and Information Science.

May 1991.

Approval Sheet

Title of Thesis: Experiments with the Gap Variation of Interpolation Search for semi uniform distributed Alphabetic files

Name of candidate: Jatin M Bhavsar.

Thesis and abstract approved:

Dr. Y. Perl **Date**

Professor

Dept of Computer & Informaton Sc.

New Jersey Institute of Technology

VITA

Name	:	Jatin M. Bhavsar
Permanent Address	:	55 Manor Dr.,Apt. 9 - O
		Newark, NJ 07106
Degree and date to be conferred	:	MS in CIS, May 1991
Date of birth	:	
Place of birth	:	

Collegiate inst. attended	Dates	Degree	Date of Degree
N.J.I.T.	1989-1991	MS	May 1991
M.S.U., India	1983-1988	BS	Feb. 1988

Major : Computer Science

Contents

Chapter 1

INTRODUCTION

Tables with alphabetic keys play an important role in data bases. Two typical exapmles of such keys are names and dictionary entries. For such tables fast search procedures are required.

For a list which has a uniform random distribution, interpolation search gives the best result. Average number of key comparisons (accesses) for a list consisting of n keys drawn from a uniform distribution was shown by [PIA],[YY],[GRG] to be *'lg lg n'* using interpolation search.

Although interpolation search has excellent performance for uniformly distributed tables, it has quite a poor performance for tables for which the keys do not have a uniform distribution but another distribution e.g. normal distribution or Poison distribution or an unknown distirbution as is the case for alphabetic tabls as a name list or a language dictionary. This is not suprising since applying interpolation search for such distributions means working according to the worng assumptions. Burton and Lewis [BL] used interpolation search on a list of 39976 names of Michigan Technological University Alumni. The results were very disappointing. On an average, interpolation search required 134 accesses while the worst case required 596 accesses.

The reason for the inefficiency of interpolation search for alphabetic tables

is that prediction made according to a uniform distribution assumption does not fit the non-uniform probabilities of different characters and the interpolation search deteriorates (partially) to a very slow sequential search.

[BL] modified interpolation search in such a way that the gap between an interval endpoint and the probe location is always at least $\sqrt{n}$, where n is interval length. Even if we overshoot because of this, the desired key is trapped in an interval of length $\sqrt{n}$. [BL] introduced two variations of the GAP methods. One of them is *Accelerated Gap Method* and the other one is *Regular Gap Method.* In *Accelerated Gap Method*, when ever the key is found to lie in the larger of the two subintervals produced by a probe, the size of the gap will be doubled (from $\sqrt{n}$ to $2\sqrt{n}$ etc), except the gap size is not allowed to exceed half the interval size. The gap is reset whenever the key is found to lie in the samller of the two subintervals produced by the probe. In *Regular Gap Method* no matter what each time the same gap is applied, i.e. the gap is not doubled even if the key is found to lie in the larger of the two subintervals produced by a probe.

For both GAP variations we experimented with a large file and a small file for different function of n rather than $\sqrt{n}$ used in [BL]. Our experiment show that $\sqrt{n}$ or $2\sqrt{n}$ is the optimum function for GAP.

Another approach [PIA] to apply interpolation search for non-uniform distributed file is to apply for each key the F cumulative distribution function which transfter the file to uniform distribution and then apply interpolation search. But how can we do this for name files for which F is not known. The solution to this problem is to apply arithmetic coding to a string which transfters [PG] non-uniform distribution into uniform distribution and then apply interpolation search. As we know interpolation search gives best result for uniform distribution, we can inprove efficiency for alphabetic tables. The arithmetic coding is based on calculating prob-

abilities of characters in a given string. The formula to finding arithmetic code for any string is explained in section 5.

However due to shortage in space for tables the arithmetic coding is applied to the strings, only approximately yielding a semi-uniform distribution for which interpolation search requires more than $lg\ lg\ n$ accesses. To overcome this difficulty we tried to apply the accelerated Gap variation with different function. For GAP = 2 we received the best results which are close enough to $lg\ lg\ n$ accesses. To verify this we analyze both theoretically and experimentelly the performance of the accelerated Gap variations for uniform distirbutions. The function GAP = 2 gave best results for experimental analysis for accelerated GAP. We analyzed the regular GAP = 2 theoretically to support the experimental result.

Chapter 2

INTERPOLATION SEARCH

Interpolation search was introduced by Peterson[Pet] as a variation of binary search for a sorted table whose keys are uniformly distributed. Suppose for example that we search for a key x = 0.7 in a sorted table A of 1000 keys, uniformly distributed in the interval [0,1]. (From now on whenever we refer to a table we mean a table sorted in increasing order.) The binary search technique employes the divide and conquer approach as it divides the table into two equal parts by accessing the median key A(500) and continue the search similarly in the appropriate half of the table if x <> A(500).

The interpolation search takes another approach as it accesses in this case the A(700) key. As it is shown in [PIA] this is the expected position of the key x = 0.7 in the table A. Furthermore this choice reflects two extra properties which are due to the uniform distribution, which is not assumed for binary search. First, A(700) is the most probable entry to contain the x = 0.7 key. Second, if x <> A(700) then x has equal probability to be in each of the sides of A(700).

Formally, let $(X_1<..<X_N)$ be a sorted file of uniformly distributed keys between a and b. For technical reasons, the keys X_0=a and X_{N+1}=b are added as the first and the last keys of the file. Let P be the probability that a random key in the file is less than or equal to Y ,

$$P = \frac{(Y - X_0)}{(X_{N+1} - X_0)}$$

The expected and most probable location of the record is (N*P). Hence interpolation search combines both the greedy approach and the divide and conquer approach. The greedy approach is accessing the most probable key. The divide and conquer approach since although the table is divided into two parts not of equal size but by the equal probability of the required key to be in them. For another example of the combination of the greedy approach and the divide and conquer approach in a search technique see split trees [S][HW][Per]. Interpolation search continues to search similarly the appropriate part of the table if x <> A(N*P). The technique was shown to require an average of *lg lg n* accesses for a table of size n for which binary search requires *lg n* accesses in the worst case and (lg n - 1) accesses in the average case.

Several independent techniques were used to prove this result. In[YY] a complicated combinatorial analysis based on double induction is used. They also prove that interpolation search is an optimal search on the average for a uniformly distributed table. In [PIA] we used proof techniques from Martingale Theory. Experiments are reported to confirm the theoretical results. Another proof appears in [GRG]. All those proofs are quite complicated. In [PR] we provide a simple proof of a variation of interpolation search which requires an average of *2.4 lg lg n* accesses. In this variation interpolation search is interpreted as quadratic application of binary search yielding an intuitive understanding for the (lg lg n) performance. This variation was the basis for Reif's[R] parallel interpolation search algorithm.

Interpolation search doesn't helps in non-uniform distributed list and it deteriorates to a very slow sequential search. One approach to modify interpolation

search for non-uniform distribution is using interpolation and binary search interchangably. In [PR] the idea to bound the worst case behaviour by $2 \lg n$ is suggested for the price of doubling average case behaviour to $2 \lg \lg n$ for uniform distribution. In [SS] Santoro and Sidney suggested a variation to switch totally to binary search after a specified parameter of interchanging interpolation and binary search accesses did not retrieve the desired key.

Foster[Fo] suggested to make each step a decision between a binary search access and an interpolation search access based on a statistical test which uses the accessed key to measure the uniformity of the distribution in the search interval. Foster uses the alumni list of 24430 names(North Carolina at Charlotte) as the worst example of a non-uniform distributed table. The average numbers of accesses reported are 13.5 and 15 respectively depending on a parameter of the statistical test.

Chapter 3

THE GAP METHOD

[PR] found that it is possible to reduce the search interval from n to $\sqrt{n}$ in average constant time and complete the search on O(lg lg n) average time. [BL] modified interpolation search in such a way that the gap between an interval endpoint and the probe location is always at least $\sqrt{n}$, where n is interval length. Even if we overshoot because of this, the desired key is trapped in an interval of length $\sqrt{n}$. This alone is not enough as interpolation search can still degenerate into a sequential search with step size $\sqrt{n}$, so at least $O(\sqrt{n})$ time may be required to complete the search in the worst case. It is necessary to detect when the algorithm is bogged down in a cluster and to increase the gap size until escape is accomplished. Therefore, whenever the key is found to lie in the larger of the two subintervals produced by a probe, the size of the gap will be doubled (from $\sqrt{n}$ to $2\sqrt{n}$ etc.), except that the gap size is not allowed to exceed half the interval size. The gap formula is reset whenever the key is found to lie in the smaller of the two subintervals produced by a probe.

The search algorithm, which [BL] refers to as *fast search*, is given in a FORTRAN-like language.

```
LOGICAL FUNCTION FIND (KEY,POSITION,LIST,SIZE)
CHARACTER*30 KEY, LIST(SIZE)
INTEGER POSITION, SIZE
```

```
CHARACTER*30 MINKEY, MAXKEY
INTEGER LOWER,UPPER,PROBE,GAP
REAL LDIFF, UDIFF, DIFF
DATA MINKEY/30*'A',MAXKEY/30*'Z'
LOWER=0
UPPER=SIZE+1
GAP=SQRT(FLOAT(UPPER-LOWER))
LDIFF=SUBTRACT(KEY,MINKEY)
UDIFF=SUBTRACT(MAXKEY,KEY)
WHILE(LOWER. LT. UPPER-1) DO
    PROBE=LOWER+LDIFF/(LDIFF+UDIFF)*(UPPER-LOWER)+0.5
    PROBE=MAX(PROBE,LOWER+GAP)
    PROBE=MIN(PROBE,UPPER-GAP)
    GAP=GAP*2
    DIFF=SUBTRACT(KEY,LIST(PROBE))
    IF (DIFF.LT.0.0) THEN
        IF (PROBE.LE.(UPPER+LOWER)/2) THEN
          GAP=SQRT(FLOAT(PROBE-LOWER))
        UPPER=PROBE
        UDIFF=-DIFF
    ELSE IF (DIFF.GT.0.0) THEN
        IF (PROBE.GE.(UPPER+LOWER)/2)
          GAP=SQRT(FLOAT(UPPER-PROBE))
        LOWER=PROBE
        LDIFF=DIFF
    ELSE
```

```
            FIND=TRUE
            POSITION=PROBE
            RETURN
        END IF
        GAP=MIN(GAP,(UPPER-LOWER)/2)
    END WHILE
    FIND=FALSE
    RETURN
    END
```

To determine the scaled difference between two key values, [BL] describes a function called SUBTRACT. A 64 element character set is assumed. Key values are assumed to be character strings of length 30. The function is as follows in a FORTRAN-like language.

```
        REAL FUNCTION SUBTRACT(HIGH,LOW)
        CHARACTER*30 HIGH,LOW
        INTEGER I
        SUBTRACT=0
        DO 100 I=1,30
100     SUBTRACT=SUBTRACT*64+ICHAR(HIGH(I:I))-ICHAR(LOW(I:I))
        RETURN
        END
```

It is shown in [BL] that in the worst case this algorithm requires $O(\lg n)^2$ accesses. On the other hand if the search file is uniformly distributed then the average complexity is $O(\lg \lg n)$ as for the usual interpolation search. When applied to the non-uniform alphabetic tables this modification improves the performance. However it is still higher than the $\lg \lg n$ performance for uniform distribution.

When this modified interpolation algorithm, to which [BL] call fast search algorithm, is applied to the list of 39976 names which gave interpolation search so much trouble, it was found that an average of 12.5 data accesses were required to complete the search. In the worst case, 23 searches were required. On a list of 39976 uniformly distributed random values, fast search produced the desired result using an average of 6.7 accesses.

The reason for accpeting $\sqrt{n}$ as the gap function , as we have mentioned above, is that [PR] showed that the search interval can be reduced from n to $\sqrt{n}$ in average constant time. But there was no confimation on the selection of $\sqrt{n}$ as the gap that this gap function gives us the better results than other gap functions . So we started doing the experimental work of finding out whether $\sqrt{n}$ is the best choice.

Chapter 4

EXPERIMENTAL RESULTS

We used for our experiment two file. The first file is the Master file containing the alumni list of N.J.I.T. This list consists about 26,000, sorted alphabetically. We concatenate the first name following the last name. The second file contains 4096 names taken uniformly from the Master file so that the probabilities of the characters will be conserved.

We tried many functions for gap, e.g. $\sqrt{n}/\lg(n)$, $\lg(n)$, $\sqrt{n}/2$, $2\sqrt{n}$ and so on. We also tried constants like 2,3,...25 as the gap function. The purpose of doing so was the fear of overshooting the actual key location sometimes when using the gap method.

The results of the experiments for the small and large files are reported in the tables below for various gaps.

For comparisons we applied the same technique for same size of files of uniformly distributed sorted list.

The overall results of our experiments using various functions for GAP and using Accelerated as well as Regular gap mathods on both files are geiven below. We also used Interpolation Search on uniformly distributed list for comparison. Then we have given detailed distribution of some of the experiments which showed some interesting results.

EXPERIMENTAL RESULTS ON SMALL FILE OF 4096 NAMES

ACC GAP	AVG ACCESSES	WORST CASE ACCESSES
$\sqrt{n}$	9.226807	16
$\sqrt{n}$ / lg n	10.367432	21
lg(n)	9.789063	18
$\sqrt{n}/2$	9.551758	18
$2\sqrt{n}$	9.463379	15
$4\sqrt{(n)}$	9.868164	15
$3\sqrt{(n)}$	9.408691	15
$\sqrt{(n)}/4$	10.246338	19
$5\sqrt{(n)}$	10.152832	13
2	10.129639	20
3	9.929443	19
4	9.709229	18
5	9.706299	18
6	9.898682	17
7	9.766846	18
8	9.783447	17
9	9.555908	16
10	9.438232	16
11	9.355713	17
12	9.384277	15
13	9.437012	15
14	9.454834	16
15	9.542236	15
16	9.647217	16
17	9.695801	16
18	9.762939	16
19	9.743408	16
20	9.769287	15
21	9.707520	15
22	9.672119	15
23	9.672363	15
24	9.707520	16
25	9.632080	15

EXPERIMENTAL RESULTS ON LARGE FILE OF 25 K NAMES

ACC GAP	AVG ACCESSES	WORST CASE ACCESSES
$\sqrt{n}$	11.843008	21
$\sqrt{n}$ / lg n	13.370195	28
lg(n)	12.648125	24
$\sqrt{n}/2$	12.484571	24
$2\sqrt{n}$	11.663828	20
$4\sqrt{(n)}$	11.847422	17
$3\sqrt{(n)}$	11.665117	18
$\sqrt{(n)}/4$	13.019648	28
$5\sqrt{(n)}$	11.954531	18
2	13.555547	29
3	13.192500	26
4	12.992188	26
5	12.610703	25
6	12.587773	25
7	12.435860	24
8	12.541523	25
9	12.424453	23
10	12.387383	22
11	12.256758	22
12	12.228984	23
13	12.116875	23
14	12.085352	23
15	12.058984	23
16	12.123945	24
17	12.023594	21
18	11.975938	21
19	11.881953	21
20	11.931055	22
21	11.878125	20
22	11.914297	21
23	11.932813	20
24	12.059414	21
25	12.114961	20

EXPERIMENTAL RESULTS ON SMALL FILE OF 4096 NAMES

REG GAP	AVG ACCESSES	WORST CASE ACCESSES
$\sqrt{n}$	9.956055	22
$\sqrt{n}$ / lg n	13.481445	49
lg(n)	11.581543	31
$\sqrt{n}/2$	11.226562	38
$2\sqrt{n}$	9.643066	18
$4\sqrt{(n)}$	10.395508	17
$3\sqrt{(n)}$	9.894043	18
$\sqrt{(n)}/4$	12.220459	49
$5\sqrt{(n)}$	10.534668	16
2	15.603516	49
3	14.239014	47
4	13.127197	42
5	12.237793	34
6	11.869141	32
7	11.554932	30
8	11.393311	30
9	11.096436	28
10	10.923828	26
11	10.756836	25
12	10.790039	25
13	10.638916	24
14	10.545898	24
15	10.510742	23
16	10.558105	22
17	10.423096	22
18	10.334473	21
19	10.261230	21
20	10.265137	21
21	10.185059	21
22	10.118896	20
23	10.101074	21
24	10.185059	20
25	10.066162	21

EXPERIMENTAL RESULTS ON LARGE FILE OF 25 K NAMES

REG GAP	AVG ACCESSES	WORST CASE ACCESSES
$\sqrt{n}$	12.510430	31
$\sqrt{n}$ / lg n	19.250703	49
lg(n)	18.288945	49
$\sqrt{n}/2$	14.653437	48
$2\sqrt{n}$	11.988750	24
$4\sqrt{(n)}$	12.391719	22
$3\sqrt{(n)}$	12.118281	22
$\sqrt{(n)}/4$	17.492461	49
$5\sqrt{(n)}$	12.548477	22
2	17.540156	49
3	18.104297	49
4	18.699297	49
5	18.803789	49
6	18.747930	49
7	18.693867	49
8	18.392734	49
9	17.934883	49
10	17.485898	49
11	17.114961	49
12	16.941406	49
13	16.514219	49
14	16.229961	49
15	15.959492	49
16	15.850195	49
17	15.630664	49
18	15.371094	49
19	15.182344	49
20	15.024414	47
21	14.807031	46
22	14.637773	44
23	14.507539	43
24	14.472656	41
25	14.290352	40

EXPERIMENTAL RESULTS ON LARGE FILE OF 4096 NUMBERS

ACC GAP	AVG ACCESSES	WORST CASE ACCESSES
$\sqrt{n}$	5.088135	9
$\sqrt{n}$ / lg n	3.936279	8
lg(n)	4.286377	8
$\sqrt{n}/2$	4.455078	8
$2\sqrt{n}$	6.068604	9
$4\sqrt{(n)}$	7.310547	10
$3\sqrt{(n)}$	6.704346	9
$\sqrt{(n)}/4$	4.427490	8
$5\sqrt{(n)}$	7.818359	10
2	3.911377	8
3	4.091797	8
4	4.366943	8
5	4.397949	8
6	4.640381	8
7	4.678467	7
8	4.958008	7
9	4.933594	8
10	5.029785	8
11	5.008057	8
12	5.245605	8
13	5.231689	8
14	5.334961	8
15	5.313721	8
16	5.559814	8
17	5.530762	8
18	5.497559	8
19	5.471924	8
20	5.559326	9
21	5.533203	9
22	5.530518	9
23	5.576416	9
24	5.808838	9
25	5.793945	9

EXPERIMENTAL RESULTS ON LARGE FILE OF 25 K NUMBERS

ACC GAP	AVG ACCESSES	WORST CASE ACCESSES
$\sqrt{n}$	5.666680	10
lg(n)	4.630977	10
$\sqrt{n}$/lg(n)	4.387891	10
$\sqrt{n}/2$	5.118008	10
$2\sqrt{n}$	6.744531	11
$4\sqrt{n}$	8.038672	11
$3\sqrt{n}$	7.527070	10
$\sqrt{n}/4$	4.867383	10
$5\sqrt{n}$	8.588672	11
2	4.203437	9
3	4.408242	9
4	4.666836	9
5	4.720039	9
6	4.955664	9
7	5.001484	9
8	5.231797	9
9	5.220352	9
10	5.292148	9
11	5.304258	9
12	5.547305	9
13	5.528398	9
14	5.616211	10
15	5.622383	9
16	5.870664	10
17	5.860898	10
18	5.863594	9
19	5.862188	10
20	5.958945	9
21	5.940938	9
22	5.971602	9
23	5.993867	9
24	6.243711	9
25	6.225000	9

EXPERIMENTAL RESULTS ON LARGE FILE OF 4096 NUMBERS

REG GAP	AVG ACCESSES	WORST CASE ACCESSES
$\sqrt{n}$	4.813477	8
lg(n)	3.961426	7
$\sqrt{n}$/lg(n)	3.680420	8
$\sqrt{n}/2$	4.143799	8
$2\sqrt{n}$	5.923340	8
$4\sqrt{n}$	7.303955	9
$3\sqrt{n}$	6.641846	9
$\sqrt{n}/4$	4.107666	8
$5\sqrt{n}$	7.698730	10
2	3.547607	7
3	3.686035	7
4	3.958740	7
5	3.971268	7
6	4.218506	7
7	4.255615	7
8	4.517822	7
9	4.515625	7
10	4.589844	7
11	4.582031	7
12	4.804199	7
13	4.802979	7
14	4.872803	7
15	4.866455	7
16	5.118408	7
17	5.086182	7
18	5.060059	7
19	5.052979	7
20	5.158691	8
21	5.143555	8
22	5.153076	8
23	5.209961	8
24	5.450195	8
25	5.439697	8

EXPERIMENTAL RESULTS ON LARGE FILE OF 25 K NUMBERS

REG GAP	AVG ACCESSES	WORST CASE ACCESSES
$\sqrt{n}$	5.355234	9
lg(n)	4.222344	8
$\sqrt{n}/\lg(n)$	4.120781	9
$\sqrt{n}/2$	4.675508	9
$2\sqrt{n}$	6.422734	10
$4\sqrt{n}$	7.858320	10
$3\sqrt{n}$	7.283437	10
$\sqrt{n}/4$	4.452344	10
$5\sqrt{n}$	8.349531	11
2	3.795000	9
3	3.934492	9
4	4.164453	8
5	4.193867	8
6	4.427227	8
7	4.472422	8
8	4.719922	8
9	4.715547	7
10	4.799961	8
11	4.815391	8
12	5.053477	8
13	5.042617	8
14	5.121641	8
15	5.135000	8
16	5.378516	8
17	5.362578	8
18	5.386289	8
19	5.372070	8
20	5.481055	9
21	5.478008	8
22	5.508711	8
23	5.519648	8
24	5.741328	8
25	5.731211	8

EXPERIMENTAL RESULTS ON NUMBERS

INTERPOLATION METHOD

DATA	AVG ACCESSES	WORST CASE ACCESSES
4096	3.438965	8
25600	3.738633	10

DATA : ALPHABETIC LIST

METHOD : ACCELERATED GAP METHOD

ACC. GAP = $\sqrt{n}$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	4	1	3
2	16	2	9
3	27	3	47
4	68	4	108
5	144	5	267
6	264	6	530
7	395	7	946
8	567	8	1504
9	661	9	2093
10	702	10	2701
11	556	11	3148
12	393	12	3356
13	203	13	3218
14	67	14	2931
15	27	15	2154
16	1	16	1298
		17	680
		18	407
		19	145
		20	50
		21	4

ACC. GAP = lg(n) DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	5	1	3
2	18	2	20
3	39	3	56
4	99	4	148
5	153	5	330
6	267	6	560
7	303	7	925
8	460	8	1386
9	515	9	1872
10	495	10	2203
11	574	11	2464
12	436	12	2606
13	304	13	2495
14	235	14	2392
15	107	15	2187
16	53	16	1856
17	28	17	1562
18	4	18	1079
		19	740
		20	408
		21	222
		22	76
		23	8
		24	1

ACC. GAP = $\sqrt{n}/2$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	4	1	3
2	15	2	12
3	39	3	52
4	76	4	104
5	181	5	270
6	285	6	528
7	390	7	871
8	485	8	1308
9	514	9	1848
10	535	10	2198
11	518	11	2622
12	400	12	2830
13	316	13	2873
14	238	14	2647
15	62	15	2374
16	30	16	2078
17	6	17	1471
18	1	18	831
		19	428
		20	160
		21	61
		22	21
		23	8
		24	1

ACC. GAP = $\sqrt{n}/\lg(n)$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	5	1	3
2	25	2	17
3	41	3	52
4	89	4	150
5	162	5	308
6	226	6	523
7	302	7	857
8	355	8	1228
9	436	9	1666
10	454	10	1875
11	440	11	2099
12	422	12	2227
13	358	13	2276
14	317	14	2283
15	246	15	2088
16	111	16	1994
17	55	17	1675
18	29	18	1335
19	14	19	1041
20	6	20	740

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
21	2	21	514
		22	355
		23	160
		24	79
		25	27
		26	17
		27	9
		28	1

ACC. GAP $= 2\sqrt{n}$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	2	1	2
2	12	2	11
3	21	3	30
4	42	4	77
5	103	5	165
6	187	6	381
7	369	7	717
8	561	8	1350
9	639	9	2117
10	757	10	3058
11	629	11	3779
12	526	12	4012
13	214	13	3835
14	32	14	2919
15	1	15	1716
		16	948
		17	421
		18	50
		19	10
		20	1

ACC. GAP = 2 DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	5	1	3
2	22	2	21
3	54	3	55
4	92	4	153
5	173	5	287
6	246	6	488
7	317	7	777
8	415	8	1142
9	444	9	1566
10	456	10	1799
11	431	11	2107
12	420	12	2326
13	331	13	2223
14	263	14	2239
15	219	15	2169
16	135	16	1932
17	47	17	1687
18	14	18	1338
19	6	19	1025

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
20	5	20	888
		21	579
		22	365
		23	221
		24	123
		25	68
		26	13
		27	4
		28	1

ACC. GAP = 3 DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	5	1	3
2	20	2	21
3	52	3	53
4	102	4	150
5	181	5	300
6	249	6	530
7	346	7	868
8	425	8	1230
9	477	9	1632
10	460	10	1978
11	478	11	2210
12	426	12	2445
13	307	13	2435
14	208	14	2190
15	172	15	2118
16	115	16	1827
17	47	17	1641
18	19	18	1418
19	6	19	1030
		20	672
		21	464
		22	225
		23	81
		24	48
		25	27
		26	3

DATA : ALPHABETIC LIST

METHOD :REGULAR GAP METHOD

REG. GAP = $\sqrt{n}$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	4	1	3
2	15	2	9
3	36	3	53
4	98	4	130
5	154	5	293
6	315	6	642
7	484	7	1126
8	488	8	1796
9	521	9	2488
10	456	10	2818
11	340	11	2763
12	271	12	2456
13	209	13	1995
14	182	14	1664
15	126	15	1441
16	126	16	1259
17	103	17	1141
18	80	18	959
19	56	19	744
20	25	20	554
21	5	21	412
22	1	22	314
		23	206
		24	134
		25	96
		26	54
		27	26
		28	13
		29	4
		30	5
		31	1

REG. GAP = lg(n) DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	5	1	4
2	20	2	19
3	51	3	65
4	114	4	165
5	201	5	351
6	302	6	581
7	380	7	901
8	385	8	1161
9	415	9	1258
10	376	10	1250
11	308	11	1198
12	248	12	1120
13	183	13	1073
14	142	14	1034
15	100	15	1008
16	75	16	974
17	71	17	910
18	68	18	835
19	77	19	803
20	79	20	790
21	97	21	765
22	81	22	697
23	87	23	696
24	64	24	663
25	42	25	629
26	37	26	584
27	31	27	547
28	23	28	531
29	17	29	461
30	11	30	453
31	5	31	418

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	330
		33	283
		34	237
		35	235
		36	193
		37	201
		38	183
		39	190
		40	190
		41	184
		42	168
		43	142
		44	111
		45	85
		46	70
		47	56
		48	44
		49	33

REG. GAP = $\sqrt{n}/2$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	4	1	3
2	15	2	13
3	43	3	53
4	98	4	140
5	196	5	346
6	284	6	645
7	362	7	1054
8	430	8	1498
9	406	9	1733
10	388	10	1921
11	308	11	2015
12	251	12	1880
13	193	13	1771
14	157	14	1504
15	144	15	1392
16	115	16	1207
17	128	17	1139
18	111	18	1023
19	103	19	971
20	102	20	856
21	78	21	754
22	57	22	667
23	38	23	549
24	30	24	438
25	23	25	374
26	10	26	317
27	6	27	254
28	4	28	210
29	2	29	178
30	1	30	144
31	1	31	136

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
32	1	32	92
33	1	33	61
34	1	34	57
35	1	35	54
36	1	36	37
37	1	37	24
38	1	38	19
		39	18
		40	14
		41	19
		42	7
		43	4
		44	3
		45	2
		46	2
		48	1

REG. GAP = $\sqrt{n}/\lg(n)$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	5	1	5
2	19	2	18
3	57	3	66
4	107	4	166
5	187	5	319
6	217	6	525
7	270	7	733
8	316	8	869
9	323	9	966
10	305	10	980
11	279	11	1025
12	258	12	973
13	231	13	980
14	177	14	937
15	163	15	922
16	110	16	907
17	101	17	836
18	82	18	817
19	78	19	839
20	57	20	801
21	68	21	794
22	70	22	798
23	75	23	761
24	65	24	728
25	63	25	665
26	64	26	610
27	53	27	551
28	47	28	519
29	46	29	516
30	31	30	463
31	24	31	462

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
32	13	32	426
33	14	33	421
34	17	34	388
35	18	35	307
36	15	36	293
37	13	37	255
38	6	38	218
39	5	39	197
40	5	40	168
41	5	41	168
42	2	42	163
43	8	43	152
44	4	44	135
45	4	45	126
46	1	46	113
47	2	47	104
48	2	48	84
49	2	49	85

REG. GAP = $2\sqrt{n}$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	2	1	2
2	13	2	10
3	21	3	35
4	55	4	93
5	111	5	231
6	199	6	447
7	426	7	888
8	586	8	1536
9	647	9	2367
10	618	10	3217
11	496	11	3456
12	330	12	3237
13	253	13	2594
14	143	14	1962
15	102	15	1638
16	62	16	1269
17	22	17	1024
18	9	18	680
		19	465
		20	272
		21	131
		22	37
		23	5
		24	3

REG. GAP = 2 DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	5	1	6
2	21	2	22
3	65	3	53
4	93	4	150
5	150	5	270
6	200	6	411
7	249	7	558
8	256	8	662
9	252	9	683
10	227	10	670
11	211	11	644
12	170	12	628
13	153	13	636
14	139	14	604
15	130	15	598
16	127	16	587
17	121	17	533
18	123	18	555
19	114	19	557
20	102	20	575
21	91	21	565
22	94	22	526
23	84	23	512
24	95	24	479
25	79	25	462
26	68	26	433
27	62	27	424
28	58	28	407
29	52	29	423
30	53	30	409
31	46	31	424

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
32	36	32	387
33	40	33	381
34	38	34	370
35	41	35	339
36	33	36	327
37	24	37	314
38	20	38	313
39	11	39	304
40	13	40	276
41	14	41	278
42	15	42	254
43	6	43	245
44	6	44	247
45	6	45	228
46	6	46	227
47	7	47	221
48	8	48	198
49	11	49	200

REG. GAP = 3 DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	5	1	5
2	21	2	22
3	59	3	56
4	109	4	154
5	170	5	289
6	228	6	486
7	309	7	669
8	298	8	777
9	298	9	867
10	247	10	833
11	231	11	784
12	213	12	774
13	171	13	754
14	160	14	708
15	152	15	679
16	135	16	665
17	113	17	642
18	110	18	653
19	100	19	643
20	83	20	629
21	88	21	639
22	85	22	572
23	90	23	516
24	78	24	505
25	71	25	465
26	83	26	462
27	75	27	432
28	45	28	451
29	31	29	421
30	23	30	425
31	21	31	410

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
32	27	32	390
33	19	33	390
34	16	34	367
35	18	35	360
36	19	36	351
37	14	37	306
38	13	38	278
39	14	39	289
40	10	40	269
41	10	41	268
42	5	42	255
43	10	43	210
44	7	44	178
45	5	45	184
46	5	46	193
47	1	47	198
		48	187
		49	178

DATA : UNIFORMLY DISTRIBUTED NUMBERS

METHOD : ACCELERATED GAP METHOD

ACC. GAP = $\sqrt{n}$ DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	138	1	366
2	53	2	62
3	488	3	2230
4	518	4	2740
5	1131	5	5451
6	1067	6	6807
7	610	7	5440
8	76	8	1826
9	14	9	628
		10	49

ACC. GAP = lg(n) DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	153	1	383
2	347	2	1822
3	653	3	3571
4	1067	4	5828
5	1013	5	6598
6	623	6	4892
7	232	7	1935
8	7	8	517
		9	52
		10	1

ACC. GAP = $\sqrt{n}/2$ DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	142	1	367
2	184	2	167
3	707	3	3217
4	967	4	4254
5	1168	5	7348
6	603	6	5939
7	301	7	3322
8	23	8	841
		9	143
		10	1

ACC. GAP = $\sqrt{n}/\lg(n)$ DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	154	1	383
2	433	2	1572
3	957	3	4578
4	1133	4	7668
5	889	5	5944
6	453	6	3821
7	71	7	1340
8	5	8	269
		9	21
		10	3

ACC. GAP = $2\sqrt{n}$ DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	138	1	361
2	33	2	37
3	222	3	591
4	285	4	1910
5	345	5	1569
6	1013	6	4530
7	1486	7	7738
8	515	8	6207
9	58	9	2596
		10	59
		11	1

ACC. GAP = 2 DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	158	1	388
2	571	2	3051
3	775	3	4483
4	1118	4	6736
5	983	5	6011
6	430	6	4077
7	51	7	626
8	9	8	217
		9	10

ACC. GAP = 3 DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	158	1	388
2	508	2	2833
3	581	3	3268
4	1102	4	6595
5	1151	5	6212
6	490	6	4635
7	100	7	1610
8	5	8	49
		9	9

DATA : UNIFORMLY DISTRIBUTED NUMBERS

METHOD : REGULAR GAP METHOD

REG. GAP = $\sqrt{n}$ DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	138	1	366
2	53	2	62
3	521	3	2431
4	619	4	2843
5	1409	5	7162
6	1106	6	7688
7	240	7	4380
8	9	8	662
		9	5

REG. GAP = lg(n) DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	153	1	383
2	427	2	2322
3	695	3	3781
4	1383	4	8068
5	1042	5	7260
6	373	6	3367
7	22	7	396
		8	22

REG. GAP = $\sqrt{n}/2$ DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	142	1	367
2	184	2	179
3	802	3	3880
4	1361	4	6429
5	1099	5	8392
6	434	6	4967
7	70	7	1238
8	3	8	139
		9	8

REG. GAP = $\sqrt{n}/\lg(n)$ DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	154	1	383
2	507	2	2166
3	1051	3	4882
4	1402	4	8742
5	768	5	6301
6	189	6	2380
7	20	7	604
8	4	8	119
		9	22

REG. GAP $= 2\sqrt{n}$ DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	138	1	361
2	33	2	37
3	226	3	1104
4	280	4	1629
5	427	5	2150
6	1078	6	5212
7	1647	7	9197
8	266	8	5430
		9	472
		10	7

REG. GAP = $3\sqrt{n}$ DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	137	1	357
2	14	2	46
3	21	3	224
4	181	4	1214
5	322	5	1139
6	657	6	2498
7	1405	7	5715
8	1356	8	10027
9	2	9	4215
		10	164

REG. GAP = 2 DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	158	1	388
2	641	2	3218
3	1144	3	6909
4	1289	4	7845
5	688	5	5472
6	162	6	1532
7	13	7	212
		8	22
		9	1

REG. GAP = 3 DATA = UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	158	1	388
2	565	2	3005
3	926	3	5792
4	1387	4	7739
5	879	5	6343
6	171	6	2061
7	9	7	258
		8	11
		9	2

Chapter 5

ARITHMETIC INTERPOLATION SEARCH

As discussed in [PIA], if the distribution function is known then we can use the property from Probability Theory that applying the cumulative distribution function F to the keys of the table according to this distribution transfers the table into a uniformly distributed table. Thus if F is for example the cumulative distribution function of the normal distribution then applying interpolation search to the value F(x) of a key x in a normally distributed table will result the usual $lg\ lg\ n$ average performance of interpolation search. Thus the extra cost required for applying interpolation search for non-uniformly distributed tables, whose cumulative probability function F is known, is computing the function F for the $lg\ lg\ n$ accessed keys. If the table is stored in the main memory then depending upon the function F it might be that these extra computations require more time than the lg n accesses required for the binary search. However if the table is stored on a disk as is the usual situation for a large table then the computation of F is much faster than the time required to access a key on the disk. Hence, for a non-uniform distributed large table stored on the disk for which the distribution function is known the modified interpolation search requires an average of $lg\ lg\ n$ accesses to the disk.

The arithmetic coding when applied properly serves as a cumulative distribution function F which transfers alphabetic entries in the given table into a uniformly distributed table as was shown in [PG].

Arithmetic coding ([WNC][L]) is a technique for data compression which maps a string of characters to an interval [0,1). This mapping is based on the probabilities of the different characters in the coded text either known in advance or accumulated during encoding. Let us demonstrate this mapping by an example of a ternary alphabet set {A, B, C} for which we assume the probabilities P(A) = 1/2, P(B) = 1/8 and P(C) = 3/8. The one character strings A, B and C are mapped to the intervals [0,0.5), [0.5,0.625) and [0.625,1) respectively. Now, every string starting with the letter A is mapped into an interval contained in [0,0.5), the interval being further reduced by the following letters. For example consider strings AA, AB and AC. These strings starts with A so it can mapped into an interval [0,0.5). Individual interval will depends on following letters. If following letter is A then interval for AA will be first half of [0,0.5) i.e. [0,0.25). Similarly for AB [0.25,0.3125) and for AC [0.3125,0.5). Similarly the intervals, corresponding to the strings ACA, ACB and ACC are [0.3125,0.40625), [0.40625,0.4295875) and [0.4295875,0.5) respectively. The string ACAA can be represent by [0.4295875,0.447265625). In arithmetic coding, the expansion of the fraction, which is the end point of the interval corresponding to a string, is sent as the code for these strings.

The theorem for Arithmetic Interpolation Search was developed by [PG]. They defined some probability terms before presenting the theorem.

Let *s(i)* denote the *i th* character in the string *s*, and let *s(i,j)* denote the substring of *s* containing the characters in the position of *i, i+1, ..., j* of *s*. Thus *s(1,j)* contain the prefix of *j* characters of *s*.

Let P1(a) denote the probability of the first character in a string to be 'a'.Let

Q1(a) denote the probability of the first character in a string to preceed 'a' in the alphabetic order. That is

$$Q1(a) = \sum_{x<a} P1(x)$$

Let P(a,i,t) denote the conditional probability of a character 'a' in a position i of a string which has a prefix t. Let Q(a,i,t) be the conditional probability of a character preceeding 'a' (in the alphabetic order) in a position i of a string which has prefix t. That is

$$Q(a,i,t) = \sum_{x<a} P(x,i,t)$$

Let R(t) denote the probability of a string in the file to have a prefix t of j characters. Then

$$R(t) = P1(t(1)) * \prod_{i=2}^{j} P(t(i), i, t(1, i-1))$$

Now given the appropriate probabilities, the computation of the arithmetic code of a string s of k characters is

$$A(s) = Q1(s(1)) + \sum_{i=2}^{k} R(s(1, i-1)) * Q(s(i), i, s(1, i-1))$$

We calculate separately the probabilities of the first character in each string. For the second position in the string we calculate the probability of the character dependent on the first character in the string. Thus for the probabilities for the second position we need a table of k^2 entries where k is the number of characters in our alphabet. Similarly for the i-th position we shall need a table of k^i entries. This is beyond the available space resources.

Thus we decided to treat all the characters from position three and on in the same way as follows. For each character disregarding its position we collect

the probabilities depending only on the previous character in the string, rather tha depending on the whole prefix preceeding the character. This way the extra table of dependent probabilities will require only k^2 entries.

The R conditional probabilities will be also computed in a similar way by using the same probabilities depending only on the previous character in the string rather than the whole prefix preceeding the characters and disregarding the position of the characters.

This way we only approximate the conditional probabilities which appear in the formula for arithmetic coding. Nevertheless we believe this is a resonable approximation since the distribution of the characters in the first and second position are different from those for the rest of the positions and are also the most critical part in the computation of the arithmetic coding. The differences between the other positions are less significant and putting them together will not be that harmfull. We also realize that the depedency of the characters is mainly on the previous character and less on the whole precceding prefix. Hence these assumptions seem reasonable under the space limitation encountered. However the distribution of the keys after applying the arithmetic coding with the approximated tables is only approximately uniform distribution. We call such a distribution a semi-uniform distribution.

Chapter 6

EXPERIMENTAL RESULTS USING ARITHMETIC CODING

After applying arithmetic coding to a string, we experimented with different functions for the gap and accelerated gap for semi-uniform distributed data. Compared to the previous results (i.e. without using arithmetic coding) we got better results. We tried many functions for gap, e.g. $\sqrt{n}$, $\sqrt{n}/\lg(n)$, $\lg(n)$, $\sqrt{n}/2$ and so on. We also tried constants like 2, 3, .., 8 as the gap function.

For comparisons we used the same files to all experiments. The overall results of our experiments using various functions for GAP and using Accelerated Gap as well as Regular Gap methods on both files are given below. Then we have given detailed distribution of some of the experiments which showed some interesting results.

EXPERIMENTAL RESULTS ON SMALL FILE OF 4096 NAMES

REG GAP	AVG ACCESSES	WORST CASE ACCESSES
$\sqrt{n}$	5.661865	15
$\sqrt{n}$ / lg n	5.541748	27
lg(n)	5.194092	14
$\sqrt{n}/2$	5.500488	24
$2\sqrt{n}$	6.737549	11
2	5.763916	28
3	5.456055	21
4	5.423340	18
5	5.293457	16
6	5.369141	15
7	5.348389	14
8	5.509033	13
9	5.408936	12
10	5.433350	12
25	5.956299	13

EXPERIMENTAL RESULTS ON LARGE FILE OF 25 K NAMES

REG GAP	AVG ACCESSES	WORST CASE ACCESSES
$\sqrt{n}$	7.427617	49
$\sqrt{n}$ / lg n	8.808438	49
lg(n)	8.016836	49
$\sqrt{n}/2$	8.007969	49
$2\sqrt{n}$	7.851953	47
2	10.237930	49
3	9.370781	49
4	8.944922	49
5	8.579727	49
6	8.426602	49
7	8.258867	49
8	8.157930	49
9	7.987969	49
10	7.911523	49
25	7.512930	49

EXPERIMENTAL RESULTS ON SMALL FILE OF 4096 NAMES

ACC GAP	AVG ACCESSES	WORST CASE ACCESSES
$\sqrt{n}$	6.167725	13
$\sqrt{n}$ / lg n	5.581055	17
lg(n)	5.560303	15
$\sqrt{n}/2$	5.834961	15
$2\sqrt{n}$	6.967285	12
2	5.460693	14
3	5.521973	13
4	5.580811	15
5	5.558838	15
6	5.703613	12
7	5.743164	14
8	5.904785	12
9	5.834961	11
10	5.849609	11
25	6.383545	13

EXPERIMENTAL RESULTS ON LARGE FILE OF 25 K NAMES

ACC GAP	AVG ACCESSES	WORST CASE ACCESSES
$\sqrt{n}$	7.738359	44
$\sqrt{n}$ / lg n	7.878984	49
lg(n)	7.399414	49
$\sqrt{n}/2$	7.754961	41
$2\sqrt{n}$	8.221211	45
2	7.727188	49
3	7.647109	42
4	7.559141	37
5	7.488320	49
6	7.530625	49
7	7.492773	49
8	7.520156	45
9	7.425820	41
10	7.444102	37
25	7.720937	49

EXPERIMENTAL RESULTS ON NUMBERS

INTERPOLATION METHOD

DATA	AVG ACCESSES	WORST CASE ACCESSES
4096	3.438965	8
25600	3.738633	10

DATA : ALPHABETIC LIST

METHOD :REGULAR GAP METHOD

REG. GAP = $\sqrt{n}$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	38	1	116
2	58	2	192
3	253	3	592
4	528	4	1555
5	951	5	3154
6	1142	6	4818
7	717	7	5001
8	253	8	3769
9	82	9	2410
10	46	10	1411
11	16	11	882
12	5	12	515
13	3	13	325
14	2	14	223
15	1	15	144
		16	107
		17	75
		18	50
		19	39
		20	28
		21	22
		22	19
		23	19
		24	15
		25	14
		26	11
		27	11
		28	10
		29	10
		30	9
		31	7

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	5
		33	4
		34	4
		35	4
		36	4
		37	3
		38	2
		39	2
		40	2
		41	2
		42	2
		43	2
		44	2
		45	2
		46	2
		47	2
		48	1
		49	1

REG. GAP = lg(n) DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	39	1	119
2	201	2	421
3	359	3	1152
4	827	4	2434
5	1072	5	3563
6	832	6	3901
7	408	7	3509
8	166	8	2690
9	86	9	1888
10	48	10	1360
11	33	11	956
12	18	12	697
13	4	13	463
14	2	14	372
		15	306
		16	249
		17	225
		18	195
		19	149
		20	122
		21	107
		22	90
		23	83
		24	74
		25	68
		26	61
		27	56
		28	39
		29	33
		30	35
		31	41

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	30
		33	20
		34	15
		35	13
		36	7
		37	7
		38	6
		39	4
		40	3
		41	3
		42	3
		43	3
		44	3
		45	3
		46	3
		47	3
		48	3
		49	3

REG. GAP = $\sqrt{n}/\lg(n)$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	39	1	119
2	204	2	422
3	470	3	1142
4	869	4	2274
5	865	5	3035
6	614	6	3254
7	388	7	3003
8	231	8	2432
9	142	9	1926
10	89	10	1527
11	51	11	1183
12	36	12	928
13	24	13	722
14	15	14	559
15	12	15	450
16	8	16	373
17	6	17	348
18	4	18	275
19	4	19	221
20	3	20	187
21	4	21	171
22	3	22	154
23	3	23	140
24	5	24	122
25	4	25	110
26	1	26	93
27	1	27	76
		28	49
		29	39
		30	33
		31	25

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	24
		33	19
		34	21
		35	21
		36	20
		37	14
		38	13
		39	11
		40	9
		41	7
		42	4
		43	4
		44	3
		45	3
		46	3
		47	3
		48	2
		49	2

REG. GAP = 2 DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	40	1	117
2	224	2	502
3	506	3	1266
4	802	4	2138
5	817	5	2817
6	631	6	2857
7	382	7	2502
8	210	8	2018
9	117	9	1533
10	72	10	1251
11	64	11	1011
12	39	12	871
13	36	13	735
14	29	14	644
15	27	15	550
16	18	16	462
17	18	17	378
18	18	18	336
19	8	19	305
20	6	20	268
21	4	21	222
22	4	22	205
23	4	23	173
24	4	24	156
25	4	25	156
26	4	26	127
27	4	27	130
28	3	28	117
		29	130
		30	125
		31	98

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	102
		33	70
		34	74
		35	70
		36	74
		37	58
		38	62
		39	55
		40	67
		41	57
		42	44
		43	43
		44	42
		45	38
		46	28
		47	25
		48	23
		49	18

REG. GAP = 3 DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	39	1	119
2	225	2	509
3	454	3	1262
4	830	4	2236
5	913	5	2999
6	701	6	3210
7	376	7	2855
8	188	8	2221
9	115	9	1654
10	68	10	1259
11	38	11	1082
12	34	12	898
13	29	13	672
14	30	14	548
15	17	15	451
16	11	16	357
17	8	17	332
18	6	18	291
19	5	19	236
20	6	20	207
21	2	21	166
		22	150
		23	141
		24	122
		25	129
		26	130
		27	110
		28	108
		29	96
		30	98
		31	98

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	83
		33	66
		34	56
		35	56
		36	46
		37	39
		38	39
		39	28
		40	22
		41	17
		42	19
		43	16
		44	13
		45	15
		46	13
		47	12
		48	12
		49	15

REG. GAP = 4 DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	39	1	118
2	222	2	503
3	381	3	1148
4	714	4	2134
5	991	5	3138
6	819	6	3502
7	456	7	3189
8	177	8	2446
9	95	9	1746
10	61	10	1321
11	48	11	1055
12	25	12	833
13	25	13	623
14	17	14	493
15	11	15	402
16	6	16	318
17	6	17	268
18	2	18	227
		19	207
		20	173
		21	159
		22	150
		23	142
		24	136
		25	121
		26	105
		27	100
		28	89
		29	64
		30	61
		31	68

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	54
		33	45
		34	35
		35	25
		36	21
		37	19
		38	22
		39	15
		40	19
		41	15
		42	19
		43	20
		44	15
		45	15
		46	13
		47	13
		48	11
		49	9

REG. GAP = $\sqrt{n}/2$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	39	1	119
2	176	2	271
3	354	3	849
4	830	4	2077
5	973	5	3412
6	724	6	4022
7	441	7	3657
8	216	8	2775
9	127	9	2137
10	77	10	1540
11	42	11	1059
12	30	12	822
13	26	13	620
14	12	14	509
15	7	15	363
16	7	16	268
17	3	17	215
18	3	18	183
19	2	19	138
20	1	20	107
21	1	21	77
22	2	22	67
23	1	23	58
24	1	24	43
		25	36
		26	27
		27	24
		28	18
		29	14
		30	14
		31	15

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	9
		33	8
		34	7
		35	6
		36	6
		37	3
		38	3
		39	2
		40	2
		41	2
		42	2
		43	2
		44	2
		45	2
		46	1
		47	1
		48	1
		49	1

REG. GAP = $2\sqrt{n}$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	38	1	108
2	11	2	135
3	148	3	355
4	209	4	915
5	363	5	1726
6	845	6	3081
7	1050	7	4871
8	932	8	5633
9	433	9	4461
10	57	10	2359
11	9	11	966
		12	375
		13	210
		14	91
		15	49
		16	41
		17	30
		18	24
		19	19
		20	17
		21	12
		22	12
		23	11
		24	10
		25	10
		26	10
		27	9
		28	9
		29	7
		30	7
		31	5

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	4
		33	4
		34	4
		35	3
		36	3
		37	3
		38	1
		39	1
		40	1
		41	1
		42	1
		43	1
		44	1
		45	1
		46	1
		47	1

DATA : ALPHABETIC LIST

METHOD : ACCELERATED GAP METHOD

ACC. GAP = $\sqrt{n}$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	39	1	116
2	48	2	189
3	230	3	527
4	350	4	1266
5	767	5	2398
6	905	6	3736
7	846	7	4454
8	521	8	4411
9	250	9	3394
10	107	10	2275
11	22	11	1239
12	9	12	723
13	1	13	283
		14	181
		15	80
		16	53
		17	38
		18	35
		19	30
		20	24
		21	22
		22	17
		23	14
		24	12
		25	11
		26	9
		27	9
		28	9
		29	9
		30	8
		31	6

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	3
		33	3
		34	2
		35	2
		36	2
		37	2
		38	1
		39	1
		40	1
		41	1
		42	1
		43	1
		44	1

ACC. GAP = lg(n) DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	39	1	120
2	183	2	370
3	299	3	992
4	652	4	2030
5	810	5	3130
6	899	6	3997
7	634	7	4001
8	362	8	3427
9	136	9	2683
10	48	10	1812
11	20	11	1170
12	9	12	712
13	2	13	391
14	1	14	232
15	1	15	149
		16	91
		17	59
		18	36
		19	24
		20	21
		21	19
		22	15
		23	14
		24	12
		25	11
		26	11
		27	10
		28	9
		29	8
		30	8
		31	5

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	4
		33	4
		34	3
		35	3
		36	2
		37	2
		38	1
		39	1
		40	1
		41	1
		42	1
		43	1
		44	1
		45	1
		46	1
		47	1
		48	1
		49	1

ACC. GAP = $\sqrt{n}/\lg(n)$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	39	1	119
2	189	2	364
3	376	3	1039
4	710	4	1975
5	830	5	2939
6	756	6	3429
7	528	7	3449
8	302	8	3144
9	179	9	2546
10	89	10	2000
11	42	11	1373
12	29	12	1059
13	13	13	674
14	4	14	422
15	5	15	292
16	2	16	196
17	2	17	127
		18	96
		19	85
		20	63
		21	38
		22	26
		23	18
		24	14
		25	11
		26	12
		27	13
		28	11
		29	9
		30	9
		31	5

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	4
		33	4
		34	4
		35	4
		36	4
		37	4
		38	2
		39	2
		40	2
		41	2
		42	2
		43	1
		44	1
		45	1
		46	2
		48	1
		49	1

ACC. GAP = 2 DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	39	1	119
2	226	2	504
3	378	3	1132
4	654	4	2043
5	866	5	2898
6	805	6	3486
7	544	7	3519
8	309	8	3140
9	157	9	2509
10	55	10	1918
11	35	11	1357
12	16	12	916
13	8	13	634
14	3	14	422
		15	308
		16	206
		17	146
		18	72
		19	51
		20	44
		21	28
		22	20
		23	16
		24	13
		25	11
		26	11
		27	9
		28	9
		29	8
		30	8
		31	6

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		32	5
		33	4
		34	3
		35	3
		36	2
		37	2
		38	1
		39	1
		40	1
		41	1
		42	1
		43	1
		44	1
		45	1
		46	1
		47	1
		48	1
		49	1

ACC. GAP = 3 DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	39	1	119
2	217	2	496
3	321	3	1085
4	619	4	2045
5	884	5	2973
6	857	6	3574
7	563	7	3764
8	333	8	3254
9	156	9	2579
10	54	10	1910
11	31	11	1259
12	19	12	756
13	2	13	481
		14	337
		15	229
		16	173
		17	136
		18	93
		19	66
		20	48
		21	31
		22	26
		23	23
		24	17
		25	17

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		26	14
		27	13
		28	12
		29	11
		30	10
		31	8
		32	6
		33	5
		34	5
		35	5
		36	5
		37	5
		38	3
		39	2
		40	2
		41	1
		42	1

ACC. GAP = 4 DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	39	1	119
2	215	2	482
3	273	3	992
4	533	4	1890
5	827	5	2921
6	984	6	3725
7	701	7	3892
8	310	8	3414
9	152	9	2705
10	41	10	1881
11	11	11	1258
12	6	12	840
13	1	13	485
14	1	14	314
15	1	15	178
		16	120
		17	94
		18	68
		19	40

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		20	32
		21	25
		22	19
		23	15
		24	13
		25	12
		26	12
		27	10
		28	10
		29	8
		30	8
		31	4
		32	3
		33	3
		34	3
		35	2
		36	1
		37	1

ACC. GAP = $\sqrt{n}/2$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	39	1	119
2	92	2	213
3	293	3	718
4	571	4	1675
5	863	5	2800
6	786	6	3711
7	729	7	4046
8	366	8	3766
9	194	9	2831
10	105	10	2056
11	38	11	1373
12	13	12	831
13	3	13	512
14	2	14	281
15	1	15	201
		16	111
		17	76
		18	48
		19	36
		20	27
		21	23
		22	16
		23	14
		24	12
		25	12

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		26	11
		27	11
		28	11
		29	10
		30	9
		31	7
		32	6
		33	6
		34	4
		35	3
		36	3
		37	3
		38	2
		39	3
		40	1
		41	1

ACC. GAP = $2^*\sqrt{n}$ DATA = NON-UNI. DISTRI. LIST

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
1	38	1	108
2	11	2	132
3	137	3	328
4	182	4	757
5	316	5	1411
6	728	6	2466
7	976	7	4347
8	1040	8	5293
9	492	9	4548
10	118	10	3092
11	53	11	1594
12	4	12	662
		13	315
		14	157
		15	90
		16	57
		17	40
		18	28
		19	21
		20	20
		21	16
		22	14
		23	13
		24	11
		25	10

Continued on the next page

SMALL FILE		LARGE FILE	
ACCESSES	SEARCHES	ACCESSES	SEARCHES
		26	10
		27	9
		28	8
		29	8
		30	9
		31	5
		32	2
		33	2
		34	2
		35	2
		36	2
		37	2
		38	1
		39	1
		40	1
		41	1
		42	1
		43	1
		44	1
		45	1

Chapter 7

ANALYSIS FOR REGULAR INTERPOLATION

It is obvious from the results of Interpolation Search on numbers that GAP =2 gives the best result among all other gap functions. To look for justification we analyze the GAP with 2 and 3 for uniform distribution. Analysis with GAP = 2 yields result which are very close to the result obtained using regular interpolation.

As we know

$$\left(1+\frac{1}{n}\right)^n \rightarrow e$$

$$\left(1-\frac{1}{n}\right)^n \rightarrow \frac{1}{e}$$

Let P be the probability for any position, n denotes total number of data, then access time for ith position can given by

$$\binom{n}{i} P^i(1-P)^{n-i}$$

The expected location for the required key is $\overline{E} = n \cdot p$. Interpolation search accesses this position. The case of GAP = 2 means that if the interpolation search has to access the key in the position 1, we will access the key in the position 2. Thus, the case is that, the expected location is $n \cdot P = 1$ or P = 1/n. Thus, in the case where interpolation search accesses location 1 the probability for the

required key to be in position i = 1

$$\binom{n}{1} P^1(1-P)^{n-1} = n \cdot P \cdot (1-P)^{n-1} = 1 \cdot \left(1-\frac{1}{n}\right)^{n-1} \to \frac{1}{e}$$

We have substituted $n \cdot P = 1$ and P = 1/n.

for position i = 2

$$\binom{n}{2} P^2(1-P)^{n-2} = \frac{n}{2} \cdot (n-1) \cdot P^2 \cdot (1-P)^{n-2} = \frac{1}{2} \cdot n \cdot P \cdot (nP-P) \cdot \left(1-\frac{1}{n}\right)^{n-2} =$$

$$\frac{1}{2} \cdot 1 \cdot \left(1-\frac{1}{n}\right) \cdot \left(1-\frac{1}{n}\right)^{n-2} = \frac{1}{2} \cdot \left(1-\frac{1}{n}\right)^{n-1} \to \frac{1}{2e}$$

for position i = 3

$$\binom{n}{3} P^3(1-P)^{n-3} = \frac{n}{6} \cdot (n-1) \cdot (n-2) \cdot P^3 \cdot (1-P)^{n-3} =$$

$$\frac{1}{6} \cdot nP \cdot (nP-P) \cdot (nP-2P) \cdot \left(1-\frac{1}{n}\right)^{n-3} =$$

$$\frac{1}{6} \cdot 1 \cdot \left(1-\frac{1}{n}\right) \cdot \left(1-\frac{2}{n}\right) \cdot \left(1-\frac{1}{n}\right)^{n-3} \simeq \frac{1}{6} \cdot \left(1-\frac{1}{n}\right)^{n-1} \to \frac{1}{6e}$$

Here we have assumed $\left(1-\frac{2}{n}\right) \simeq \left(1-\frac{1}{n}\right)$.

similarly we can find probability for a position

i = 4 converging to $\frac{1}{24e}$

i = 5 converging to $\frac{1}{120e}$

i = 6 converging to $\frac{1}{720e}$

Now consider regular interpolation

For a key at position i, the number of search iteration required is i average time for i = 1, 2, 3

$$P(i=1) \cdot 1 + P(i=2) \cdot 2 + P(i=3) \cdot 3 = \frac{1}{e} \cdot 1 + \frac{1}{2e} \cdot 2 + \frac{1}{6e} \cdot 3 = \frac{1}{e}\left[1+1+\frac{1}{2}\right] = \frac{2.5}{e}$$

For GAP method, using gap = 2

If we look for first element, first it will look for second element and then checks the first element. Therefore number of search for key at position i = 2 will be 1, but for a position i = 1 it will be 2.

For a position i = 3 it will be 3 because first it will look for second element then fourth element and finally it will check for third element.

Average time for i = 1, 2, 3

$$P(i=1)\cdot 2 + P(i=2)\cdot 1 + P(i=3)\cdot 3 = \frac{1}{e}\cdot 2 + \frac{1}{2e}\cdot 1 + \frac{1}{6e}\cdot 3 = \frac{1}{e}\left[2 + \frac{1}{2} + \frac{1}{2}\right] = \frac{3}{e}$$

Effect of adding next element

Adding 4 th element

For regular interpolation search

$$\frac{1}{e}\left[2.5 + \frac{1}{24}\cdot 4\right] = \frac{1}{e}\cdot 2\frac{4}{6}$$

For GAP method with gap = 2

$$\frac{1}{e}\left[3 + \frac{1}{24}\cdot 2\right] = \frac{1}{e}\left(3\frac{1}{12}\right)$$

Different between gap method and interpolation is equal to $\frac{5}{12}$ which is negligible.

Adding 5 and 6

For regular interpolation search

$$\frac{1}{e}\left[2\frac{4}{6} + \frac{1}{120e}\cdot 5 + \frac{1}{720e}\cdot 3\right]$$

For GAP method with gap = 2

$$\frac{1}{e}\left[3\frac{1}{12} + \frac{1}{120e}\cdot 4 + \frac{1}{720e}\cdot 3\right]$$

The difference is again very low. This indicates that performance of constant gap (i.e. 2) is as good as regular interpolation search. Since for every index except the first and last there is no difference and for these two the difference is very small.

Now analyzing regular interpolation search with gap = 3. Here finding out number of search iteration is quite complicated. First it will look for the third element and then depending upon required element it looks forward or backward. If key is less than the third element then it will either go for first element or second element. To find this, it follows regular interpolation instead of GAP method. Here we assume that probability of looking for first element is more than second element. Therefore number of search iteration to look for element at position one is 2 and for position two is 3. Similarly for postion 4, 5 and 6 it is 3, 4 and 2 respectively. for i = 1, 2, 3, 4 average access time

$$P(i=1)\cdot 2 + P(i=2)\cdot 3 + P(i=3)\cdot 1 + P(i=4)\cdot 3 =$$

$$\frac{2}{e} + \frac{3}{2e} + \frac{1}{6e} + \frac{3}{24e} =$$

$$\frac{1}{e}\left[2 + \frac{3}{2} + \frac{1}{6} + \frac{1}{8}\right] =$$

$$\frac{1}{e} \cdot \frac{91}{24}$$

This result indicates greater disparity with respect to that of regular interpolation, thus proving that gap = 3 is not a better method. Adding 5 and 6 give worse results.

Adding 5 and 6 average time = $\frac{1}{e}\ [\frac{1387}{360}]$.

The above analysis explains why the constant GAP function works better for a semi-uniform distribution. The difference between average time for regular interpolation and GAP method is very small. In GAP = 2, the average time is

approximately 1 which is quite resonable. In case of GAP = 3 it becomes worse than regular interpolation.

Chapter 8

CONCLUSIONS

For the comparison purposes we carried out the experiments on non-uniformly distributed as well as uniformly distributed lists.

Observations about experiments on uniformly distributed lists :

- Regular gap method gives better performance than Accelerated gap method on uniformly distributed list.
- As we went on dividing $\sqrt{n}$ by a constant e.g. 2, the performance kept on improving. Ultimately we terminated this sequence of experiments by using $\sqrt{n}/\lg(n)$ as gap function which gave us better average performance than $\sqrt{n}$.
- In using $\sqrt{n}$ as a gap there was one fear that we might overshoot the key location although the search interval was getting reduced everytime. So with this in mind we used the constant numbers as the gap function, e.g. constant 2,3,..7. Gap 2 gave us even better performance than $\sqrt{n}/\lg(n)$.
- We tried several other functions and constants as well as for the gap function . We also used interpolation search method on numbers and, as is already proved, we got the best results of all the experiments for uniformly distributed list.

Observations about experiments on non-uniformly distributed lists Using Arithmetic Code :

- For a small file of 4096 names, using Accelerated gap = 2 gave us the best results.
- For a samll file of 4096 names, using Regular gap = lg(n) gave us the best results.
- For a large file of 25600 names, using Accelerated gap = lg(n) gave us the best results.
- For a large file of 25600 names, Gap = $\sqrt{n}$ gave us better results. But the improvement was not significant and this result was not consistent with small file.
- In case of a small file using regular gap method for gap = constant, the results went on improving until gap = 5 and then onwards the performance started deteriorating.

General comments :

- For the uniformly distributed lists the interpolation search method gives the best results. On an average it requires *lg lg n* accesses and in the worst case *n* accesses. The average accesses of *lg lg n* is proven to be the optimal performance we can get for any search technique.
- For the list of non-uniformly distributed data the Regular gap method did give us better results when we were using gap other than $\sqrt{n}$ but the improvement was insignificant and the results were not consistent with those of a small file. Results were changing with the size and distribution of the files. So

everytime we have to find the best choice for a gap function depending upon the particular size and distribution of the file. Therefore the generalization of results was not possible and results were not comparable to lg lg n.

- The use of arithmetic code improves the result. Compared to the results obtained for a non-uniform without using arithmetic coding, this results were better. We found that it decreases the average access time and also the worst case accesses. We got an advantage on average access and worst case accesses using arithmetic code.

BIBLIOGRAPHY

[BL] Burton F.W. and Lewis G.N., A robust variation of interpolation search, Information processing Letter 10, 1980, pp. 198-201.

[Fe] Feller, W., An introduction to probability Theory and Its Applications, Vol. 1. Wiley, New York, Third ed., 1968.

[Fo] Foster K.E., a statistically-based interpolation binary search, T.R. of Winthrop College, SC.

[GRG] Gonnet G.H., Rogers L.D. and George J.A., An Algorithmic and Complexity Analysis of Interpolation Search, Acta Information 13, 1980, pp.39-52.

[L] Langdon G.G. An introduction to arithmetic coding, IBM J Res. Dev. 28,2, 1984, pp. 135-149.

[Per] Perl Y., Optimum split trees, J. Algorithms 5, 1984, pp. 364-374.

[Pet] Peterson, W.W. Addressing for random-access storage. IBM J. Res. Dev. I 1957, 131-132.

[PG] Y. perl, L. Gabriel, Arithmetic Interpolation Search for Alphabtic Tables, to appear in IEEE Trans. on Computer

[PIA] Perl, Y., Itai, A., and Avni, H., Interpolation Search -A Lg Lg N Search, CACM, 21, 1978, 550-553.

[PR] Perl, Y., Reingold, E.M., Undestanding the Complexity of Interpolation search, Information Processing Letters, 6, 1977, pp. 219-222.

[R] Reif, John H., Parallel Interpolation Search, TR-07-82
HARVARD UNIVERSITY.

[S] Shell B.A., Medium Split Trees : A fast lookup technique for frequently occuring keys, CACM 21, 1978, pp.947-958.

[SS] Santoro N. and Sidney J.B., Interpolation-binary search, Information Processing Letters 20, 1985, pp. 179-181.

[W] Willard D.E., Searching unindexed and nonuniformly generated files in lg lg n time, SIAM J. Computing 14, 1985, pp. 1013-1029.

[WNC] Witten I.H., Neal R.M. and Cleary J.G., Arithmetic coding for data compression, CACM, 30, 1987, 520-540.

[YY] Yao, A.C., and Yao F.F., The complexity of searching an ordered random table. Proc. Seventeenth Annual Symp. Foundations of Comptr. Sci., 1976, pp. 173-177.