International Journal of Mathematical Archive-2(8), 2011, Page: 1274-1278
Available online through www.ijma.info ISSN 2229 – 5046

AN EMPIRICAL STUDY ON K-SORT FOR BINOMIAL INPUTS

Kiran Kumar Sundararajan[1], Mita Pal[2], Soubhik Chakraborty[3*], Bijeeta Pal[4] & N. C. Mahanti[5]

[1]Barclays Bank PLC, United Arab Emirates, Dubai
[2,3,5]Department of Applied Mathematics, Birla Institute of Technology, Mesra, Ranchi-835215, India
[4]Department of Computer Engineering, Institute of Technology, Banaras Hindu University, Varanasi, India
**Corresponding author email: soubhikc@yahoo.co.in*

(Received on: 21-07-11; Accepted on: 03-08-11)

ABSTRACT

The present paper examines the behavior of a new version of Quick sort, which we call K-sort, when the sorting elements follow a Binomial distribution.

Key words: *K-sort; parameterized complexity; statistics; factorial experiments.*

---

1. 1 INTRODUCTION

The present paper examines the behavior of K-sort (a new version of Quick sort) for binomial distribution inputs and is in continuation of our earlier work on this new algorithm for uniform U[0, 1] inputs [5] with the acknowledgement that here the focus will be on how the parameters of Binomial distribution affect the average sorting time. In other words, this is a work in parameterized complexity [4]. Use is made of factorial experiments when the n observations to be sorted come independently from binomial distribution B (m, p). To investigate the individual effect of number of sorting elements (n), binomial distribution parameters (m and p which give the fixed number of trials and the fixed probability of success in a single trial) and also their interaction effects, a 3-cube factorial experiment is conducted with three levels of each of the three factors n, m and p. Further, we have obtained some interesting patterns showing how the Binomial parameters influence the average sorting time. We have attempted a justification for the same. The next section describes our K-sort.

1.2 K-sort

The steps of K-sort are given below:-

Step-1: Initialize the first element of the array as the key element and i as left, j as (right+1), k = p where p is (left+1).

Step-2: Repeat step-3 till the condition (j-i) ≥ 2 is satisfied.

Step-3: Compare a[p] and key element. If key ≤ a[p] then

Step-3.1: if (p is not equal to j and j is not equal to (right + 1))
then set a[j] = a[p]
else if (j equals (right + 1)) then
set temp = a[p] and flag = 1
decrease j by 1 and assign p = j
else (if the comparison of step-3 is not satisfied i.e. if key > a[p])

Step-3.2: assign a[i] = a[p] , increase i and k by 1 and set p = k

Step-4: set a[i] = key

if (flag = = 1) then
assign a[i+1] = temp

Step-5: if (left < i - 1) then

Split the array into sub array from start to i-th element and repeat steps 1-4 with the sub array

Step-6: if (left > i + 1) then

Split the array into sub array from i-th element to end element and repeat steps 1-4 with the sub array

2. EMPIRICAL RESULTS

Our first study examines the behavior of K sort for varying p with fixed n and m.

Table - 1 gives the average run time y (average taken over 30 readings) for different values of the argument p for fixed n=50000 and m=100.

Table – 1

Average sorting time

P	average sorting time in sec
0.1	0.3782
0.2	0.2862
0.3	0.25
0.4	0.2348
0.5	0.2298
0.6	0.233
0.7	0.2484
0.8	0.286
0.9	0.3812

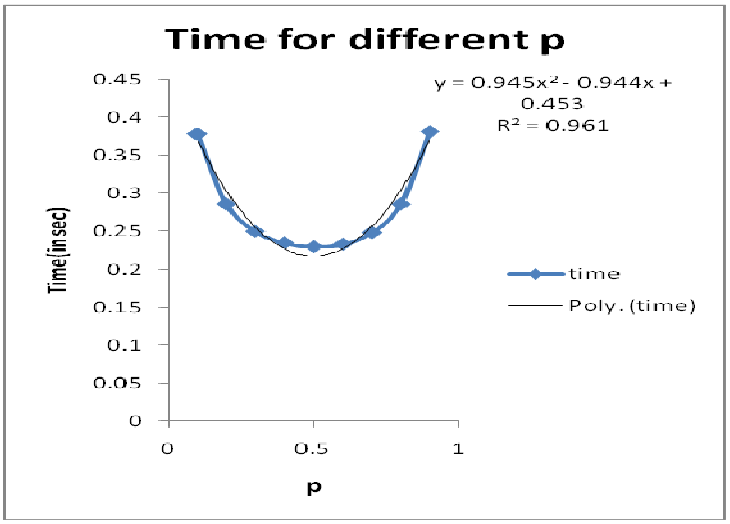

Fig-1 Average sorting time versus p: quadratic fit

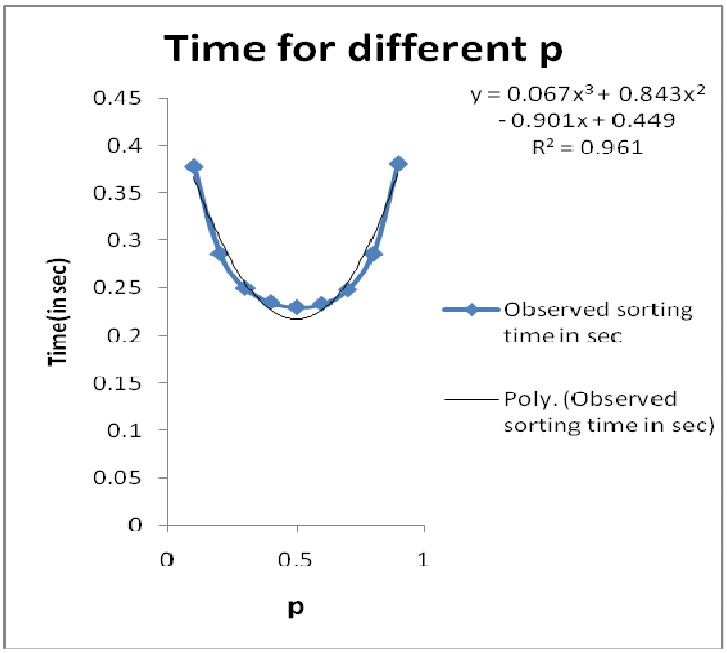

Fig 2 Average sorting time versus p: cubic fit

Comparing Figures 1 and 2, based on the experimental results given in table1, we find that a second degree polynomial is the adequate fit to represent the average sorting time in terms of p for binomial distribution input for fixed no. of trials m and array size n. The cubic and quadratic fits are equally good meaning thereby that cubic term is not contributing anything to the model.

In our second study, we keep p and n fixed and observe average sorting time for varying m.

Table - 2 gives the average run time y (average taken over 30 readings) for different values of the argument m for fixed n=150000 and p=0.5

Table 2: Avg sorting time vs m

For fixed p= 0. 5

m	Observed sorting time in sec
100	2.0516
500	0.9249
1000	0.6563
1500	0.5422
2000	0.4674

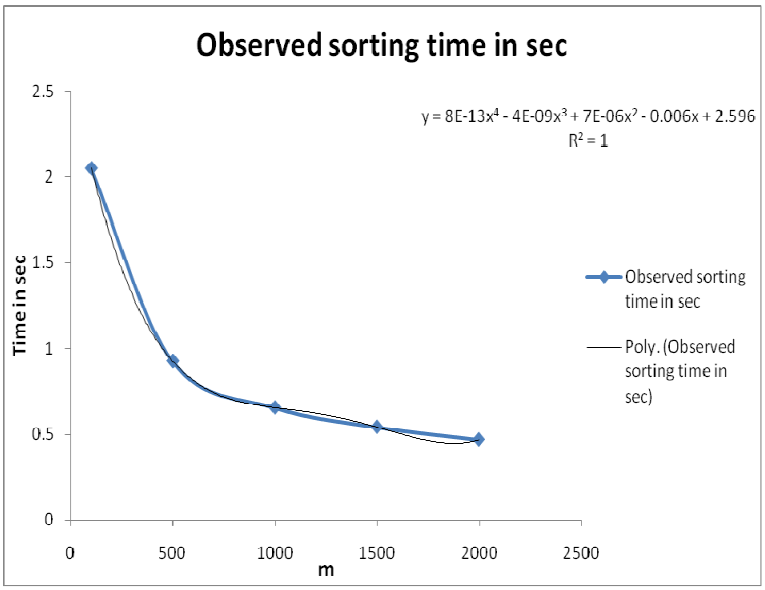

Fig-3: Avg sorting time vs m: fit of polynomial of deg four

Figure 3, based on the experimental results given in table 2, suggests a forth degree polynomial fit for binomial distribution input to predict the average sorting time in terms of m for fixed p and n. It is clear from table 2 that average sorting time decreases as m increases. This is because as m increases the number of ties decreases. Why this happens is discussed later (sec. 3).

In our last study, Table 3 gives the data for factorial experiments to accomplish our study on parameterized complexity.

Table 3: Data for 3^3 factorial experiment for K- sort

K- sort times in second Binomial (m , p) distribution input for various n (50000, 100000, 150000) , m (100 , 1000, 1500) and p (0.2, 0.5, 0.8).

n = 50000

m	p=0.2	p=0.5	p=0.8
100	0.2862	0.2298	0.286
1000	0.094	0.0748	0.0937
1500	0.0751	0.0623	0.078

n=100000

m	p=0.2	p=0.5	p=0.8
100	1.1422	0.9109	1.1407
1000	0.3671	0.2936	0.3655
1500	0.3014	0.2421	0.3014

n = 150000

m	p=0.2	p=0.5	p=0.8
100	2.5658	2.0516	2.5673
1000	0.8205	0.6563	0.8158
1500	0.672	0.5422	0.6704

Table 4 gives the results using MINITAB statistical package version 15.

Table-4: Results of 3^3 factorial experiment on K-sort

General Linear Model: y versus n, m, p

```
Factor   Type   Levels Values
n       fixed       3  1, 2, 3
m       fixed       3  1, 2, 3
p       fixed       3  1, 2, 3

Analysis of Variance for y, using Adjusted SS for Tests

Source  DF   Seq SS   Adj SS  Adj MS          F      P
n        2  17.3422  17.3422  8.6711  879434.48  0.000
m        2  14.0687  14.0687  7.0343  713429.94  0.000
p        2   0.3512   0.3512  0.1756   17807.22  0.000
n*m      4   7.0538   7.0538  1.7635  178851.85  0.000
m*p      4   0.1445   0.1445  0.0361    3663.67  0.000
n*p      4   0.1721   0.1721  0.0430    4363.81  0.000
n*m*p    8   0.0716   0.0716  0.0090     908.08  0.000
Error   54   0.0005   0.0005  0.0000
Total   80  39.2047

S = 0.00314005   R-Sq = 100.00%   R-Sq(adj) = 100.00%
```


3. DISCUSSION

K- sort is highly affected by the main effects n, m and p. When we consider the interaction effects, interestingly we find that all interactions are significant in K- sort. Strikingly, even the three factor interaction n*m*p cannot be neglected. It is observed that y decreases for increasing p up to 0.5 in fig. 1 and then increases for increasing p. The only justification that can be readily given is that as p gets away from 0.5, either the lower values of the variate ($p<0.5$) or the higher values of the variate ($p>0.5$) are more likely resulting in greater number of ties. If interchanges were involved, greater number of ties would result in fewer interchanges leading to lesser sorting time. The question of interest is: when interchanges are not involved, why does an increase in the ties lead to an increase in the sorting time (in this case at least)? The answer is that the construction of the algorithm is such that more computations are required for "if (key<=a[p])" (step 3) than for "if (key>a[p])". *The crux of the debate lies in that the case of equality resulting in ties is attached with the less than type (<) operator*. If it were attached with the greater than type (>) operator, the story would be the other way round.

4. CONCLUSION AND SUGGESTIONS FOR FUTURE WORK

Three-cube factorial experiments conducted on K-sort reveal that for certain algorithms such as sorting, the parameters of the input distribution singularly as well as interactively are important factors, besides the size of input, for evaluating time complexity more precisely. While, our results will definitely pose an intellectual challenge for the theoretical analysts, we do emphasize here that cheap and efficient prediction [3] is the objective in computer experiments such as the ones conducted here. A computer experiment is a series of runs of a code for various inputs. Our study in parameterized complexity in algorithms also emphasizes the important role that ties play. Ties have been explored also in [6].

K-sort, however, has one drawback. The algorithm fails to take advantage of an already sorted array or sub array. Modern programmers keep bubble sort, which has this facility, as a subroutine to Quick sort and its different variations. We too propose the same in the case of K-sort.

Remark: It may be of interest to know that the first version of K-sort which used an auxiliary array was proposed by Sundararajan and Chakraborty [1] and called a new sorting algorithm. Khreisat [2] made a comparative study of several versions of Quicksort in terms of speed. The new sorting algorithm was also tested and found to be competing well with SedgewickFast, Singleton sort and Bsort, three of the fast versions of Quicksort, for number of sorting elements ranging from 3000 to 2, 00,000.

REFERENCES

[1] K. K. Sundararajan , S. Chakraborty , *A New Sorting Algorithm*, Applied Math. and Compu., vol. 188(1), 2007, p. 1037-1041

[2] L. Khreisat, *QuickSort A Historical Perspective and Empirical Study*, International Journal of Computer Science and Network Security, VOL.7 No.12, December 2007, p. 54-65

[3] J. Sacks, W. Weltch, T.Mitchel, H. Wynn, *Design and Analysis of Computer Experiments*, Statistical Science 4 (4), 1989

[4] M. Pal, S. Chakraborty and N. C. Mahanti, How does the Shift-insertion Sort behave when the sorting elements follow a Normal distribution?, Annals Computer Science Series, Vol. VIII, Fasc. 2, 2010, 93-98.

[5] K. K. Sundararajan, M. Pal, S. Chakraborty and N. C. Mahanti, K-sort: A new sorting algorithm that beats Heap sort for $n \leq 70$ lakhs! arXiv: 1107.3622v1 [cs.DS]

[6] S. Chakraborty, S. K. Sourabh, M. Bose and K. Sushant, *Replacement sort revisited: The "gold standard" unearthed!* , Applied Math. and Compu. ,vol. 189(2), 2007, p. 384-394.

****

