Parameterized Complexity: A Statistical Approach Combining Factorial Experiments with Principal Component Analysis

Anchala Kumari[1] and Soubhik Chakraborty[2*]
[1]University Department of Statistics, Patna University, Patna-800005, India
[2]Department of Applied Mathematics, BIT Mesra, Ranchi-835215, India
*Email address of corresponding author: soubhikc@yahoo.co.in (S. Chakraborty)

Abstract

The new sort developed by Sundararajan and Chakarborty (2007), a modification of Quick sort that removes the interchanges, considers the first element of the array as pivot element. In this paper the pivot element taken is a randomly selected element of the array. The effect of binomial parameters are examined on the average sorting complexity using Principal Component Analysis approach. An attempt is also made to focus on a comparative study of the two algorithms: one called the random pivot element algorithm (RPA) and the other one with first element as the pivot element algorithm called first pivot element algorithm (FPA). The results reveal that the RPA can sort larger number of observations than the FPA.

Keywords

Parameterized Complexity; Principal Component Analysis; random pivot; factorial experiments

1. Introduction

Parameterized complexity focuses on classifying computational problems according to their inherent difficulty with respect to *multiple* parameters of the input. However, since time is both a cost and a weight, we have ventured to work directly on program run time.

Our work is motivated by the pioneering works on computer experiment (a series of runs of a code for various inputs) of Prof. Jerome Sacks and others (Sacks et. al. 1989). The recently published book by Chakraborty and Sourabh (2010) describes the design and analysis of computer experiments when the response is a complexity such as time.

A new sorting mechanism developed by Sundararajan and Chakraborty (2007) is based on the same logic as that of Quick sort but sorts the data without using interchanges and has the same average time complexity (O(nlogn)) and the same worst case complexity $O(n^2)$ as Quick sort has. In the new sort originally proposed, the pivot element taken was the first element of the array. In the present study, the pivot element is taken as a randomly selected array element.

In the recent past much of the work is done on parameterized complexity. For a helpful review, with special emphasis on sorting, Mahmoud (2000) may be consulted. Some of our contributions on parameterized complexity are the recent works of Anchala and Chakraborty (2007), Anchala and Chakraborty (2008), Prashant, Anchala and Chakraborty (2009). For the general problem of parameterized complexity theoretically, we refer to Flum and Grohe (2006). The first systematic work on this is credited to Downey and Fellows (1999).

In this paper we have examined the parameterized complexity of the new sorting technique when the pivot element is one of the randomly selected elements of the array generated randomly from binomial distribution. The logic behind having binomial inputs is that any population can be easily dichotomized by some criterion.

To investigate the effect of binomial parameters on the sorting efficiency, a 3 cube factorial experiment is employed. To investigate further the form of relationship between the sorting time and binomial parameters, the principal component analysis is used. (Neter et. al. 1996)

From now, we shall abbreviate the sorting algorithm with pivot element randomly selected element by RPA (Random Pivot element Algorithm) and the algorithm with pivot element the first element by FPA (First Pivot element Algorithm).

2. RPA Analysis

To examine the effect of binomial parameters on sorting efficiency, a 3-cube factorial experiment was employed with 3 factors each at three levels. The three factors included were the numbers to be sorted (n) and the two parameters of binomial distribution (m and p).The levels of the factors are given below in table 1

Table 1: Factors and their levels

Factor	level		
n	10000	15000	20000
m	100	150	200
p	0.2	0.5	0.8

Five replications of sorting time were made for each of the 27 combinations (different factors combined at different levels of each factor which generates 27 combinations) were obtained using Visual C++ code (see Appendix), and average run time was evaluated thereon.
To know the singular and interaction effects of different factors, the 3-cube factorial experiment was performed using MINITAB (13.0). The results are given in the table 2.1.

Table 2.1: Results of 3-cube factorial experiment

Sources	d.f	s.s	m.s.s	F
Rep	4	.0003	.0001	0.36
n	2	38.6580	19.3290	8.6E+04
p	2	.0158	.0079	35.20
m	2	.1050	.0520	233.21
n*p	4	.3920	.0980	435.52
n*m	4	.0086	.0021	9.54
p*m	4	.2375	.0594	263.83
n*p*m	8	.1729	.0216	96.03
error	104	.0234	.0002	
Total	134			

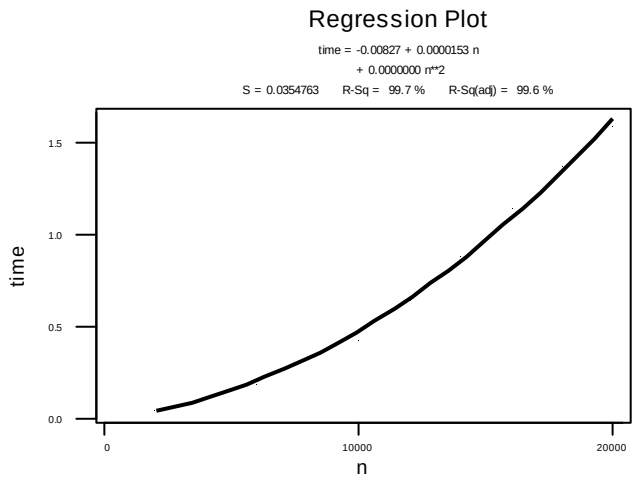
It is obvious that the sorting time has to increase as the number of observations to be sorted (n) is increased. Besides n, the parameters p and m are also important factors in explaining the complexity of the algorithm, as is clear from the above table, where m is highly significant and p is moderately significant. As regards the two factor interaction effects, np and pm are very highly significant and nm is nearly significant. It may be argued that the n numbers to be sorted and m (binomial parameter), when considered individually, have high precedence in explaining sorting complexity; at the same time these two also show marked effects at different probability points as is clear from very high values of F corresponding to np and pm interactions. Interestingly, the two parameters of the binomial distribution also are found to interact among themselves, but this effect is moderately significant. (F =9.54 only for pm interaction)

We have observed that all the three factors play an important role in describing the complexity of RPA algorithm . Now we try to examine the order of complexity explained by individual factor.

(i) **n-t plot**: Taking m=150 and p=.5 fixed, the values of time t were

plotted against different values of n using MINITAB statistical package . The graph obtained is given in fig. 2.1.

Fig :2.1 : n-t plot



The figure shows time as a second degree polynomial in n .As such the average time complexity of RPA sorting algorithm is empirically of the order $O_{emp}(n^2)$.

Remark: Empirical O here is the bound-estimate of a statistical bound. A statistical bound is a weight based bound which takes all the operations collectively. A mathematical complexity bound, in contrast, is always count based and operation specific. [Chakraborty and Sourabh (2010)].

(ii**) m-t plot** and **p-t plot**

The plot of time verses m keeping n=15000 and p=.5 fixed is given in figure 2.2 and the plot of time verses p, keeping n=15000 and and m=150 fixed , is given in fig 2.3

Fig 2.2 m- t graph

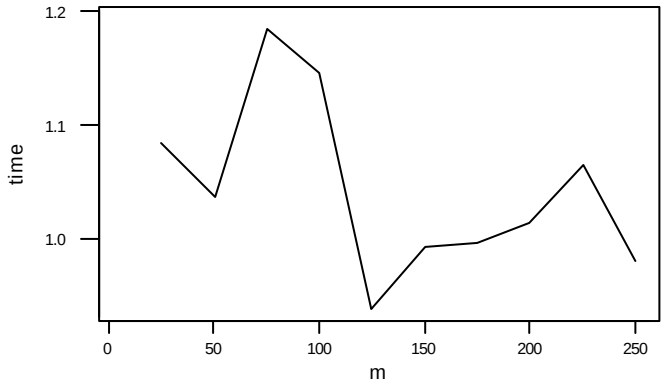
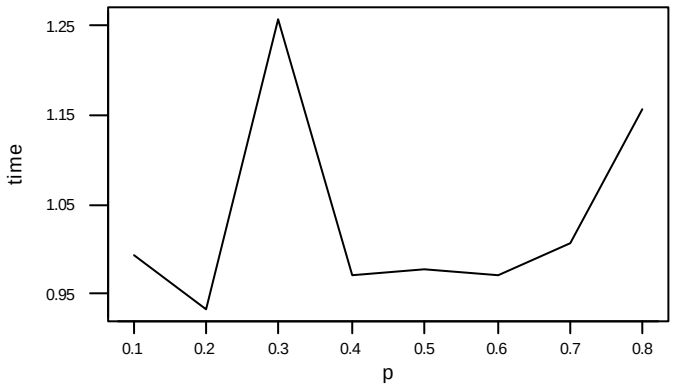
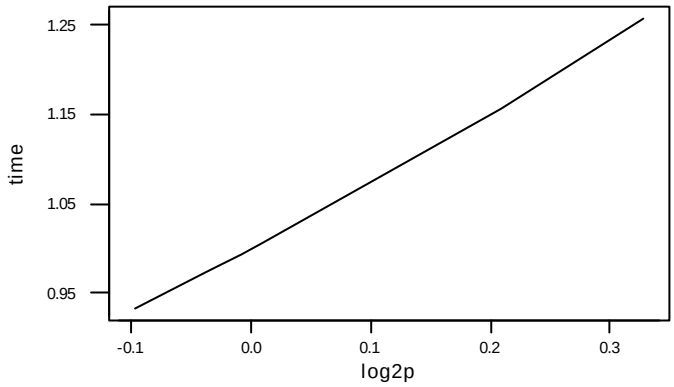


Fig 2.3 p-t plot



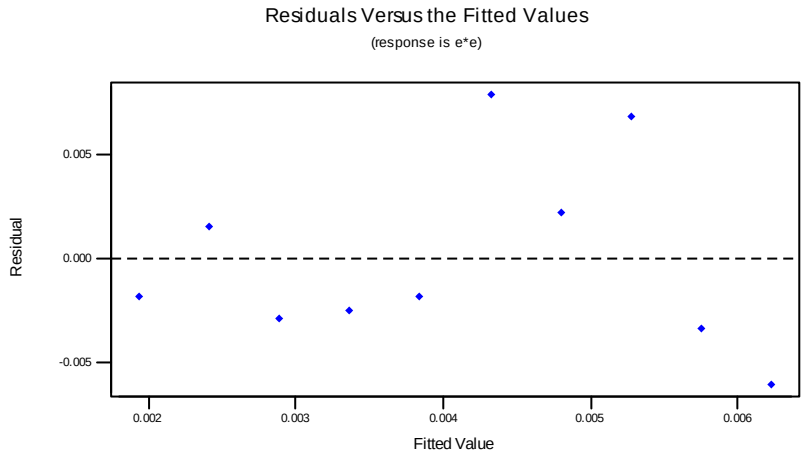
Time versus m and time versus p graphs do not show any systematic pattern. However, time plotted against the values of $\log_2 p$ gives a linear trend implying thereby that sorting efficiency of RPA is of $O_{emp}(\log_2 p)$. See fig. 2.4.

Fig. 2.4 time verses $\log_2 p$ plot



But as for as m-t plot stands, any transformation made on time or m or both to find a systematic pattern between time and m, remained unsuccessful surprisingly even though the time complexity significantly depends on the value of m. It may be due to the non independence of error terms which is clear from the following plots of residuals versus fitted value (fig. 2.5).

Fig 2.5 : Residuals verses fitted values



The sorting time when regressed on the three factors gives a high value of R^2=97.3, indicating thereby that all the three factors highly qualify for explaining the time complexity of the algorithm.

The regression equation is

time = - 0.788 +0.000131 n - 0.0380 p -0.000512 m

Interestingly, the distributional parameters produce negative effect on the complexity. With increase in p and m, average RPA complexity decreases. A high value of R^2 does not always mean a good fit. The high values of interactions may cause the presence of multicollinearity in the independent variables. As such to nullify the effect of multicollinearity and to find the exact relationship between time and the three factors, we apply the method of Principal Component analysis .

Table 2.2 : The results of the principal component analysis

Variable	PC1	PC2	PC3
n	1.000	0.000	0.000
p	0.000	0.000	-1.000
m	0.000	1.000	0.000

Where pc1, pc2 and pc3 are the three principal components.

The regression equation is given by

time = - 0.788 + 0.0380 pc1 -0.000512 pc2 +0.000131 pc3

with a value of R^2 = 97.4%

3. FPA Analysis

In this section we consider the usual New sorting technique with pivot element as the first element of the array.

The n array elements are randomly generated from B (m,p) and the three levels of input parameters are as given below

n: 5000 10000 15000

m: 50 100 150

p: 0.3 0.5 0.8

the results 0f 3-cube factorial experiments are presented in the table 3.1.

Table 3.1 : Results of 3-cube Factorial Experiment

Sources	d.f	s.s	m.s.s	F
Rep	4	.00052	.00013	0.58
n	2	14.01733	7.00867	3.2E+04
m	2	.08025	.04013	180.35
p	2	.05721	.02860	128.57
n*m	4	.03740	.00935	42.03
n*p	4	.01230	. 00308	13.82
m*p	4	.03043	.00761	34.19
n*p*m	8	.07930	.00991	44.56
error	104	.02314	.00022	
Total	134	14.33788		

Here all the three factors considered individually, have very high impact on sorting efficiency of FPA. As far as their interactive effects are considered , all the two factor interactions are moderately significant.

Next we study n-time, p-time , m- time plots separately.

(i) **n-time graph**

Fig-3.1

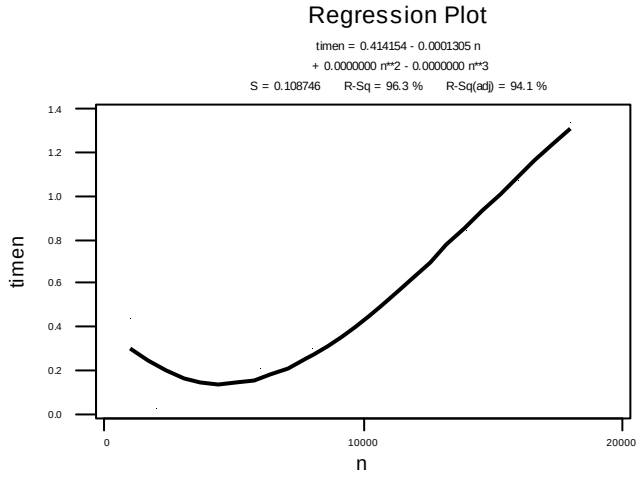
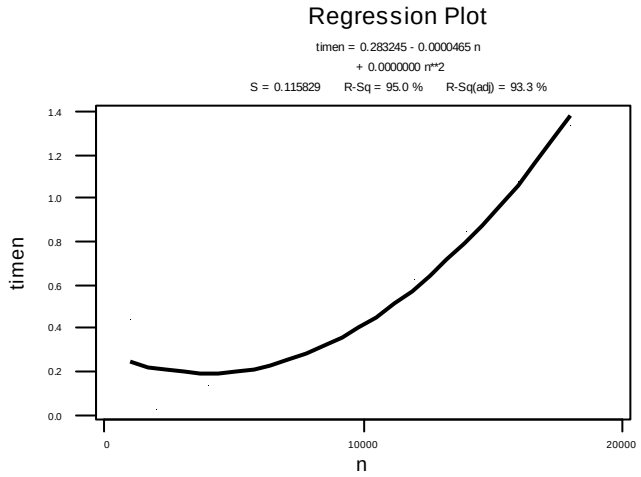
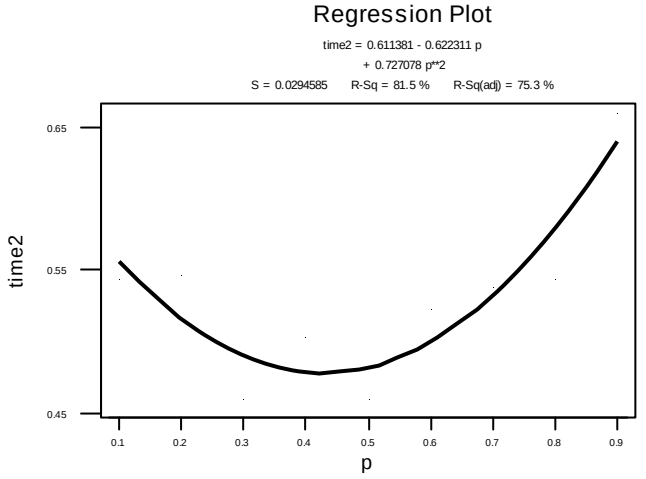


Fig:3.2



The first figure (fig. 3.1) expresses time as a polynomial of third degree in n and the second figure (fig. 3.2) shows it as a polynomial of degree 2 in n. for both the plots R^2 >=95% Thus we may conclude that average time complexity of APF may be taken as $O(n^2)$ or $O(n^3)$ experimentally. We shall accept the former as it is a stronger statement.

(ii) Fig. 3.3 p-time graph



The above fig (fig. 3.3) shows that average time complexity can nearly be expressed as a quadratic function in p with a value of R^2=81.5%

(iii) m-time graph

Fig : 3.4

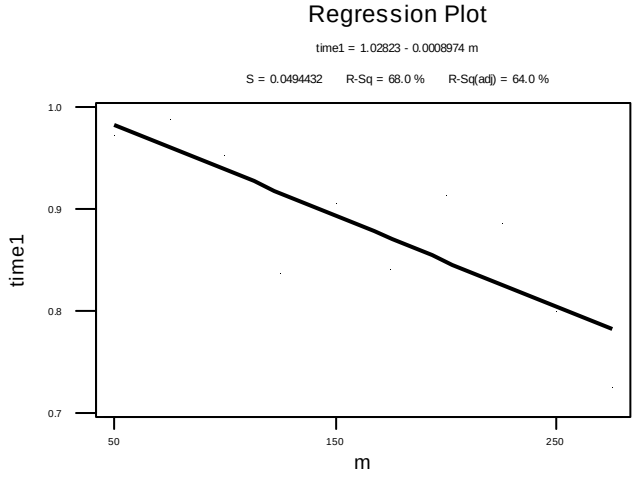
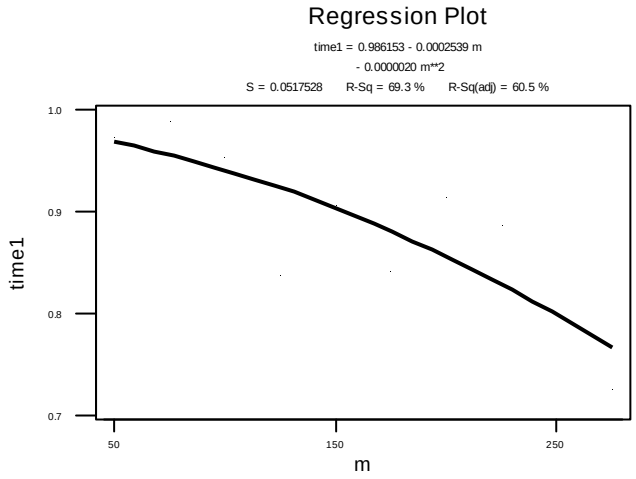


Fig: 3.5



The sorting time as a function of firsst degree or 2nd degree in m does not give a moderately good fit as the value of R^2 is only 68% in first case and 69% in 2nd case.

Let us have a look on residuals versus fitted values and normal probability plots.

Fig- 3.6: Residuals verses fitted value plot

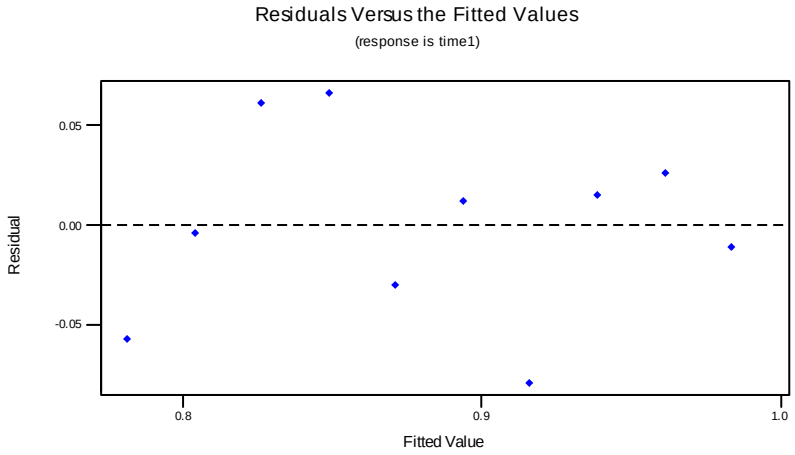
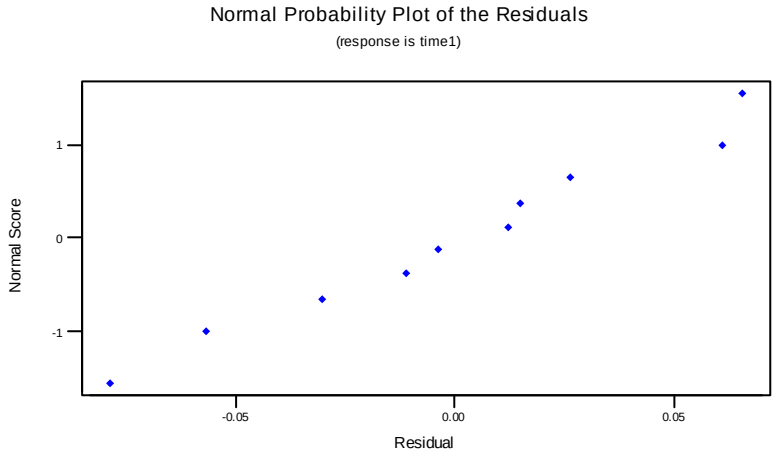


Fig-3.7 : Normal probability plot



Normal probability plot is a clear indication of the fact that there is almost a linear trend between sorting time and m but on the other hand the residual verses fitted plot gives an indication that the errors terms are not independent.

It may be argued therefore that the average time complexity of FPA in fact should be of O(m), but due to the fact that the error terms are not independent , it depicts an approximate linear relation between time and the binomial parameter m.

4. Comparative study of RPA and FPA

Average sorting time for RPA

n	p=0.2			p=0.5			p=0.8		
	m			m			m		
	50	100	150	50	100	150	50	100	150
10000	.5094	.4970	.4248	.4904	.5186	.4236	.4716	.4750	.4524
15000	.9516	1.056	.9188	1.408	1.1658	.9812	1.1374	1.066	1.159
20000	1.872	1.866	1.873	1.903	1.878	1.606	1.611	1.656	1.753

Average sorting time for FPA

n	p=0.3			p=0.5			p=0.8		
	m			m			m		
	50	100	150	50	100	150	50	100	150
5000	.1750	.1628	.1596	.1842	.1658	.1586	.1906	.1816	.1720
10000	.4752	.4560	.3972	.4816	.4376	.4438	.5218	.5252	.4748
15000	.9876	.9592	.8278	.9718	.9404	.9170	1.0874	.8778	1.006

The results of two algorithms reveal the following facts:

1. As far as the two algorithms are considered, the RPA is more efficient than the FPA in the sense that we can sort an array of large size in the first case than in the second. Moreover for small probability points the second algorithm is not valid.

2. As for as the sorting time is considered for same number of observations to be sorted, except for some incidences, the sorting time is usually less for FPA than for RPA. The reason is that every time the sorting takes place

new _sort function is called and pivot element is selected every time, That causes increase in the sorting time. See the code given in the appendix.

3. In both the algorithms for fixed m and p, the sorting time increases with increase in the sample size which is otherwise also obvious. But for fixed n, the sorting time responses in a very erratic way when we change the value of p keeping m fixed or m is changed while keeping p fixed in RPA. On the other hand in FPA for fixed n and fixed m, the sorting time shows almost decreasing trend while the probability points are increased. However for fixed n and p, no systematic pattern is found in the time value corresponding to the change in the value of m.

5. Conclusion and Future Work

It may be concluded here that if we want to sort an array with larger number of observations then RPA (Random Pivot Algorithm) should be preferred.

As regards the future work we may examine the algorithm with pivot element as the average of all array elements.

References

[1] Anchala Kumari and Soubhik Chakraborty, Software Complexity: A statistical Case Study Through Insertion Sort, Applied Mathematics and Computation, Vol. 190 (2007), p. 40-503.

[2] Anchala kumari and S.Chakarborty, A Simulation study on Quick Sort Parameterized complexity using Response Surface Design, International Journal of Mathematical Modeling, Simulation and Applications,Vol .1 No.4, 2008, 448-458

[3] Hosam Mahmoud, Sorting: A Distribution Theory, John Wiley, 2000

[4] Kiran Kumar Sundararajan and Soubhik Chakarborty,A New sorting Algorithm, Applied Mathematics and Computation, Vol. 188(1), 2007, p1037-1041

[5] Jerome Sacks, William J. Welch, Toby J. Mitchell and Henry P. Wynn, Design and Analysis of Computer Experiments, Statistical Science, Vol. 4, No. 4, 1989, p409-423

[6] John Neter, Michael Kutner, William Wasserman and Christopher Nachtsheim, Applied Linear Statistical Models Mc Graw-Hill/Irwin, 4th ed., 1996
[7] Jorg Flum and Martin Grohe, Parameterized Complexity Theory, Springer, 2006
[8] Prashant Kumar, Anchala Kumari and Soubhik Chakarborty, Parameterized Complexity on a new sorting algorithm : A study in Simulation, Annals. Computer Science Series, Vol. VII, Fasc.2, 2009, 9-22
[9] Rod Downey and Michael Fellows, Parameterized Complexity, Springer, 1999
[10] Soubhik Chakraborty and Suman Kumar Sourabh, A Computer Experiment Oriented Approach to Algorithmic Complexity, Lambert Academic Publishing, 2010

Appendix

FPA : First element as the Pivot element algorithm Visual C++ code.

```
#include<iostream.h>
#include<stdlib.h>
#include<sys/timeb.h>
#include<time.h>
#include<iomanip.h>
#include<conio.h>
void new_sort(int a[], int b[], int l, int r)
{
 int pivot=a[l],low=l;
int up=r;
for(int i=l+1;i<=r;i++)
{
if(a[i]<=pivot)
 {
b[low]=a[i];
low++;
}
else
{
b[up]=a[i];
up--;
}
}
b[low]=pivot;
for( i=1;i<=r;i++)
a[i]=b[i];
if(l<up-1)
new_sort(a,b,l,up-1);
if(up+1<r)
new_sort(a,b,up+1,r);
}
void mysort(int a[],long array_size)
{
clock_t start,finish;
int *b;
b=new int[array_size];
 double duration;
start=clock();
new_sort(a,b,0,array_size-1);
finish=clock();
```

```
duration=(double)(finish-start)/CLOCKS_PER_SEC;
cout.precision(4);
cout.setf(ios::showpoint);
cout<<"Elapsed time"<<  duration<<endl;
}
void main()
{
        int m, s;
  float p=.5, r;
  int *a;
int n;
rand();
cin>>n>>m;
a=new int[n];
for(int i=0;i<=n;i++)
{
s=0;
for(int j=0;j<m;j++)
{
r=(float)rand()/RAND_MAX;
if(r<p)
++s;
}
*(a+i)=s;
}
mysort(a,n);
getch();
}
```

RPA: Randomly selected element as the pivot element algorithm

The code is same as the FPA code except for the void new_sort function which is given below.

```
void new_sort(int a[], int b[], int l, int r)
{
        int i;
        int y=rand()%(r-l);
int pivot= a[y],low=l,up=r;
for( i=l+1;i<=r;i++)
{
if(a[i]<=pivot)
 {
b[low]=a[i];
low++;
}
else
{
b[up]=a[i];
up--;
}
}
```

```
b[low]=pivot;
for( i=1;i<=r;i++)
a[i]=b[i];
if(l<up-1)
new_sort(a,b,l,up-1);
if(up+1<r)
new_sort(a,b,up+1,r);
}
```