1998 International Computer Symposium
Workshop on Algorithms
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

Lottery Drawing

Jenshiuh Liu and Yeh-Ching Chung

Dept. of Information Eng. and Computer Science
Feng Chia University
Taichung, Taiwan 407
ROC

Abstract:

We consider the problem of an unbiased choice of k items at random from a total of n candidates, where $k \leq n$. This problem has many applications, for example, in quality control or lottery drawing. We present three algorithms to tackle the problem. All three algorithms have their own merit and are based on using random numbers. Due to the nondeterministic characteristics, the execution time for each algorithm is primarily determined by the number of random numbers used in its process. We formally analyze each algorithm by computing its expected number of random numbers required and the variance associated with it. Computer simulations are used to gain further insights of these algorithms. Finally, we present a general guideline to direct users in selecting an appropriate algorithm based on the values of k and n.

1 Introduction

We consider the problem of an unbiased choice of k items at random from a total of n candidates, where $k \leq n$. This problem has many applications, for example, in quality control or lottery drawing. In quality control, we usually need an unbiased sample of size k out of the total n items to examine if the chosen ones are with certain property or not. In lottery drawing, the k lucky numbers must be drawn unbiasly from the n balls. Many data processing applications can be modelled as a lottery drawing, for example, to select k out of the n students who want to enrol in a certain course, or to select k out of the n persons who are interested in purchasing a certain stock when it goes public.

In this work, we study the problem of efficient lottery drawing using computer. Although some lottery players let computers choose their lucky numbers, few people trust that the computer will draw lottery fairly. People used to watch on TVs and expect that balls labeled with their lucky numbers drop from the pot on the lottery drawing nights. It is hard to image how many people still want to play lottery if the drawing process is replaced by a computer to pick the lucky numbers. In some applications, it is almost impossible to use balls or other similar substitute for lottery drawing, especially when n gets very large. Thus, efficient and unbiased algorithms for lottery drawing are necessary.

We present three algorithms to tackle the problem. All of these are based on using random numbers. The execution time for each algorithm is primarily determined by the number of random numbers used in its process. Each algorithm has its strength for certain values of k and n. Due to the nondeterministic characteristic of random numbers, the execution time for each algorithm will vary for different drawings. To further investigate these algorithms, we choose the expected number of random numbers required in each algorithm as our performance matric. We formally analyze each algorithm by computing its expected number of random numbers and the variance associated with it. Although the expected number of trials needed is a good measurement for the performance of the three algorithms, there are other factors involved. Computer simulations were used to gain further insights of these algorithms. The rest of this paper is structured as follow. Section 2 presents the

three algorithms. Their formal analysis is given in Section 3. Section 4 shows computer simulation results. Finally, Section 5 contains some concluding remarks.

2 Unbiased Lottery Drawing

The lottery drawing problem has many applications in different fields. In some cases, for example real lottery, an ID number to represent a candidate is good enough. One the other hand, in some data processing, each candidate usually consists of a collection of data called a *record*. The drawing should *not* depend on the content of each record. To simplify our presentation, we label each candidate with an ID number i ($1 \leq i \leq n$), and view each candidate as a number. Therefore, we use an integer array to mark or store the selected items in all three algorithms. If the data been processed are in a form of records, an extra array of pointers may needed in order to access the data records after the selection.

2.1 Random Drawing without Replacement

An obvious approach is to select each item with probability k/n. It seems appropriate, but it gives only an average of k items in the sample. It is quite possible that the number of selected items will either too large ($> k$) or too small ($< k$) to give the necessary results. The following modification (see *e.g.*, [3]) gives what we want: We examine items one by one; the $i+1^{st}$ ($0 \leq i < n$) item should be selected with probability $(k-s)/(n-i)$, where s is the number of items have already been selected upto the i^{th} item. To see why the probability is appropriate, we focus on the selection of the $i+1^{st}$ item. Consider all the possible ways to choose k items form n under the condition that s items have been selected in the first i items: There are $\binom{n-i}{k-s}$ possibly ways to select $k-s$ items from the remaining $n-i$. The selection of the $i+1^{st}$ item implies that $k-s-1$ items must be selected from the remaining $n-i-1$. Therefore, exactly

$$\binom{n-i-1}{k-s-1} \Big/ \binom{n-i}{k-s} = \frac{k-s}{n-i}$$

of these will select the $i+1^{st}$ item.

The previous idea leads us to algorithm *Alg_o*.

Alg_o

```
1.  For i = 1 to n do
      Mark[i] = ' '.
2.  Set i = 0,  s = 0.
3.  While (s < k)
3.1   Generate a random number R,
      uniformly distributed in [0,1).
3.2   If R < (k-s)/(n-i) then
        Mark[i+1] = '*', s = s+1.
3.3   i = i+1.
4.  For i = 1 to n do
4.1   If Mark[i] = '*'  then select X[i].
```

Alg_o seems to be unreliable at the first glance, expecially one may ask how can we guarrante that step 3 terminates? In fact, step 3 will repeat no more than n times. To see this, we examine the following extreme case: Suppose that none of the first $n-k$ itmes are selected (*i.e.*, we are in a situation of $s = 0$ and $i = n-k$,), then the probability to select the $n-k+1^{th}$ item and what follows will all be, by *Alg_o*, exactly 1. In other words, all the remaining k itmes will be selected with probability 1. Hence, we have shown that step 3 repeats at most n times.

Another question one may ask is that whether *Alg_o* gives an unbiased result. It is not difficult to see that the probability of the first item been selected is k/n. Although, we do not select the $i+1^{st}$ item with probability k/n, the following argument shows that the probability of any item been selected is exactly k/n. We have seen that for $i = 0$, the probability to select the first item is k/n. For $i = 1$, we distinguish between the following two cases: (A) the first item was *not* selected (*i.e.*, $s = 0$), and (B) the first item was *selected* (*i.e.*, $s = 1$). The probabilities for cases A and B to occur are $(1-k/n)$ and k/n, respectively; The probabilities to select the second item are, by *Alg_o*, $\frac{k}{n-1}$ and $\frac{k-1}{n-1}$ for cases A and B, respectively. Therefore, the probability to select the second item is

$$\frac{n-k}{n}\frac{k}{n-1} + \frac{k}{n}\frac{k-1}{n-1} = \frac{k}{n}$$

In general, the probability to select the $i+1^{th}$ item can be expressed as

$$\sum_{s=0}^{i} \frac{k-s}{n-i} \binom{i}{s} \left(\frac{k}{n}\right)^{s} \left(\frac{n-k}{n}\right)^{i-s}$$

It can be shown that the previous expression equals to k/n for $i < n$, which means that the probability to select any item is exactly the same. Thus, *Alg_o* gives an unbiased result.

We have seen that *Alg_o* correctly gives an unbiased result, but it may not perform well under certain conditions such as when $k \ll n$. To see this, we examine the following example. Given $k = 1$ and $n = 100$, it may take 1 iteration in step 3 in the best case, or it may take 100 iterations if we are unlucky. In fact, we will see in the next section that on the average $\frac{k}{k+1}(n+1)$ drawings are needed to complete a selection of k candidates out of n, which says that, on the average, 50.5 trials are needed in a selection of 1 out of 100 candidates. Our next approach tries to remedy this.

2.2 Random Drawing with Replacement

The idea behind our second approach is similar to the (real) lottery drawing. In other words, we assign each candidate an identification (ID) number from 1 to n, put n balls labeled from 1 to n to an urn, and randomly draw k balls from the urn; A candidate is selected if one's ID number matches any of the labels of the k balls that have been drawn out of the urn.

The implementation of the above idea needs some attentions. Observe that to draw a ball is similar to get a random number between 1 and n. To draw the first ball from the urn can be simulated by generating a random number between 1 and n. However, to simulate the remaining drawing is not so straight forward. Since all the balls have different labels, we should have k distinct labels once we draw k balls. One question arises is that how can we simulate the i^{th} drawing such that the i^{th} random number is different from all the previous $i-1$ numbers. One way to handle this is to generate a random number in the range of $[1, n-i+1]$ for the i^{th} drawing and reallocate all the candidates to the first $n-i$ positions (relabel all the candidates from 1 to $n-i$) after the i^{th} drawing has been completed. This scheme may create a tremendous amount of work (reallocation) after each drawing when n is large.

To avoid the reallocation, we always generate random numbers in the range $[1, n]$ and record all the (ID) numbers that have been appeared. At each drawing we ignore the numbers if that have been drawn before, and repeatedly generate random numbers until a new one has been obtained. Formally, algorithm *Alg_w* is as follows.

Alg_w

```
1.   For i = 1 to n do
       Mark[i] = ' '.
2.   Set s = 0.
3.   While (s < k)
3.1    Generate an integer random number R,
       uniformly distributed in [1, n].
3.2    If Mark[R] = ' '  then
         Mark[R] = '*', s = s + 1.
4.   For i = 1 to n do
4.1    If Mark[i] = '*'  then select X[i].
```

The array `Mark[]` in *Alg_w* is used to record whether candidates have been selected before. With this, in step 3.2 of *Alg_w*, we ignore the random number R if it has been appeared before. One may also ask whether *Alg_w* gives an unbiased result. Since in step 3.2 we just ignore the selected ones, all the trials are still independent and the probability to select any one is still equally $1/n$. Therefore, *Alg_w* gives an unbiased result. In Section 2.1, we have seen that step 3 of *Alg_o* repeats no more than n times. Moreover, on the average, 50.5 trials are needed in a selection of 1 candidate out of 100. Here, it can be seen that, with *Alg_w*, exactly 1 trial is needed in a selection of 1 candidate out of 100. One interesting question is that how many iterations should be conducted in step 3 of *Alg_w* so as to select k candidates out of n. In the next section, we will show that on the average no more than $n(H_n - H_{n-k})$ trials are needed in a selection of k candidates out of n, where H_n is the n^{th} *Harmonic number*. Hence, *Alg_w* takes exactly 1 trial to select 1 candidate out of 100, which is quite an improvement over *Alg_o* under this exam-

ple. On the other hand, *Alg_w* takes about 290.4 ($100(H_{100} - H_5)$) trials to select 95 candidates out of 100. This is much worse than the result of *Alg_o*, which takes no more than 100 trials. Our third approach will give reasonable result in all cases.

2.3 Random Permutation

A different approach to tackle the problem is to *arrange* the n items in a certain order and then select the first k in the line. One issue remains to be solved is: how to properly arrange the items. It is known that there are $n!$ different ways to order n items; any of these could be our choice. With the above observation, an immediate solution is to choose anyone of the $n!$ permutations. Therefore, our third method is to generate a random permutation of the n items and then select the first k in that permutation. To produce a random permutation is similar to shuffle the items. The problem of shuffling has been studied before. Assume that initially we place the n items in any order. It is known, see *e.g.*, [3], that a random permutation of n items can be obtained by a sequence of $n-1$ exchanges; In the i^{th} $(1 \le i \le n-1)$ iteration we simply exchange the items placed at the i^{th} and at the r^{th} positions, where r is a random number in the range $[i, n]$. Formally, *Alg_p* is as follows.

Alg_p

1. For $i = 1$ to $n-1$ do
1.1 Generate an *integer* random number R, uniformly distributed in $[i, n]$.
1.2 Exchange X[i] and X[R].
2. Select X[1], X[2], $\cdots$, X[k].

Again, one may ask whether *Alg_p* gives an unbiased result. To answer this, we demonstrate that step 1 in *Alg_p* gives an unbiased random permutation. For $i = 1$, by step 1.1, any of the n items could be selected (with probability $1/n$) and permanently placed at the first position. In general, one can see that for $i = t$, any of the remaining $n-t$ items could be selected (with probability $1/(n-t)$) and permanently placed at the t^{th} position. Therefore, *Alg_p* gives an unbiased result.

3 Performance Analysis

In this section, we formally analyze the performance the three algorithms. We use the number of trials (random variates) needed (to draw k objects out of the n candidates) as our metrics, since the execution time of any algorithm is primarily determined on this factor. Due to the nondeterministic characteristic of the *Alg_o* and the *Alg_w*, the numbers of trials will vary for different drawings. We compute the expected numbers and the variance of the expected numbers for the *Alg_o* and the *Alg_w*. On the other hand, a random permutation can be obtained with a fixed number of random variates. We give the exact number for the *Alg_p*.

3.1 Random Drawing without Replacement

The approach taken by the *Alg_o* can be modeled as follows. Suppose that we have an urn containing n balls of which k are red and others (*i.e.*, $n-k$) are black. Balls are randomly drawn one by one *without replacement* until exactly all k red balls are drawn. Let $P(s)$ denote the probability that the k^{th} red ball will be drawn on the s^{th} trial, *i.e.*, it needs s trials to complete a drawing. Let $E(S)$ denote the expected number of trials to complete a drwaing. We compute $P(s)$ first, then obtain the $E(S)$.

Since there is no way to have k red balls in less than k trials, we have

$$P(s) = 0 \quad \text{for } s < k.$$

For $k \le s \le n$, exactly $k-1$ red balls and $s-k$ black balls are drawn in the first $s-1$ trials and the k^{th} red ball is drawn on the s^{th} trial. Hence, the probability of completing a drawing in exactly s, where $k \le s \le n$, trial is

$$P(s) = \frac{\binom{k}{k-1}\binom{n-k}{s-k}}{\binom{n}{s-1}} \frac{1}{n-s+1}$$

With some algebraic operations, the probability can be expressed as

$$P(s) = \binom{s-1}{s-k} \Big/ \binom{n}{k} \quad \text{for } k \le s \le n \quad (1)$$

It is known that $E(S) = \sum_s sP(s)$. Hence,

$$E(S) = \sum_{s=k}^{n} s \left(\begin{array}{c} s-1 \\ s-k \end{array} \right) / \left(\begin{array}{c} n \\ k \end{array} \right)$$

With some algebraic operations, the expected number of trials can be expressed as

$$E(S) = \left(\begin{array}{c} n \\ k \end{array} \right)^{-1} k \sum_{s=k}^{n} \left(\begin{array}{c} s \\ k \end{array} \right)$$

The summation term in the previous equation is exactly $\left(\begin{array}{c} n+1 \\ k+1 \end{array} \right)$ (see *e.g.*, [1, pp.159]). Therefore, we have

$$E(S) = \frac{k}{k+1}(n+1) \tag{2}$$

It can be shown that $E(S)$ is an increasing function of k for any fixed n. Furthermore, the expected number of trials for the *Alg_o* is at least $0.5(n+1)$ (which occurs when $k = 1$) and never exceeds n (when $k \to n$), *i.e.*, it is in the range of $[0.5(n+1), n]$. To investigate how well is the distribution of s, we further compute its variance (denoted by $V(S)$).

Define $E(S(S+1)) = \sum_s s(s+1)P(s)$. It can be seen that $E(S(S+1)) = E(S^2) + E(S)$. In a way similar to our computation of $E(S)$, it can be shown that

$$E(S(S+1)) = \frac{k}{k+2}(n+1)(n+2)$$

By definition, $V(S) = E(S^2) - E^2(S)$. Hence,

$$\begin{aligned} V(S) &= E(S^2) - E^2(S) \\ &= E(S(S+1)) - E(S) - E^2(S) \\ &= \frac{k(n-k)(n+1)}{(k+2)(k+1)^2} \end{aligned} \tag{3}$$

3.2 Random Drawing with Replacement

The approach taken by the *Alg_w* can be modeled as follows. Suppose that we have an urn containing n balls that are labeled $1, 2, \cdots, n$. Balls are randomly drawn one by one; the label of each ball drawn is recorded and then put back to the urn (*with replacement*) until exactly k distinct balls have been obtained. Let $P(s)$ and $E(S)$ be defined as in Section 3.1. We compute $P(s)$ first, then obtain the $E(S)$.

To compute $P(s)$, we consider the problem of random distribution of s balls into n boxes and seek the probability $q_k(s)$ that exactly k boxed are *empty*. The random distribution problem is an example of the occupancy problems (see *e.g.*, [2]), which has various reformulations such as the coupon problem. The probability for $q_k(s)$ is known [2, pp.45] to be

$$q_k(s) = \left(\begin{array}{c} n \\ k \end{array} \right) \sum_{j=0}^{n-k} (-1)^j \left(\begin{array}{c} n-k \\ j \end{array} \right) \left(\frac{n-k-j}{n} \right)^s$$

It can be shown that $P(s)$ is exactly the probability of the instance that the k^{th} distinct box been *filled* on the s^{th} trial. This instance is equivalent to the conjunction of the following two conditions: (A) exactly $k-1$ boxed are *filled* in $s-1$ trials, and (B) the s^{th} ball is distributed to one of the remaining $n-k+1$ empty boxes. Let $p_{k-1}(s-1)$ denote the probability of condition (A), *i.e.*, after the random distribution of $s-1$ balls, there are $n-k+1$ empty boxes. With the definition of $q_k(s)$, it can be seen that

$$p_{k-1}(s-1) = q_{n-k+1}(s-1) \tag{4}$$

The probability of condition (B) is simply $\frac{n-k+1}{n}$. Therefore, we have

$$P(s) = p_{k-1}(s-1)\frac{n-k+1}{n} \tag{5}$$

We have seen that the number of trials to complete a drawing in *Alg_o* varies in the range of $[k, n]$. Equations 4 and 5 show that the number of trials to complete a drawing in *Alg_w* is also a random variable. Indeed, it takes at least k trials to complete a drawing of k distinct balls. However, if we were unlucky, we may never complete a drawing. In other words, the number of trials to complete a drawing in *Alg_w* is in the range of $[k, \infty)$.

One way to compute $E(S)$, as we did in the previous subsection, is to multiply Eq.5 with s and sum over all possible values of s. This approach leads us to work on an infinite series, which is rather complicated. Instead of doing this, we take the following approach: For $k = 1$, we need exactly one trial to draw one (distinct) ball. For $k \geq 2$,

let x_i (where $i = 1, 2, \cdots, k-1$) denote the number of trials that must be performed after the i^{th} new ball enters the sample and until the $(i+1)^{th}$ new ball enters the sample. It can be seen that $s = 1 + x_1 + \cdots + x_{k-1}$.

We now focus on the $x_i's$. The event $x_i = j$ occurs if and only if the first $j-1$ balls drawn after the i^{th} distinct ball enters the sample duplicates anyone of the previous i distinct balls, and the j^{th} ball drawn is different from the previous i balls. Since all the drawings are independent and the trial continues until we obtain a distinct ball, x_i is in a geometric distribution with parameter $\frac{n-i}{n}$. Thus, $E(X_i) = \frac{n}{n-i}$. Furthermore, all the trials on $x_i's$ are also independent. Therefore,

$$\begin{aligned} E(S) &= E(1) + E(x_1) + \cdots + E(x_{k-1}) \\ &= 1 + \sum_{i=1}^{k-1} (\frac{n}{n-i}) \\ &= n[\frac{1}{n} + \frac{1}{(n-1)} + \cdots + \frac{1}{(n-k+1)}] \\ &= n(H_n - H_{n-k}) \end{aligned} \quad (6)$$

where H_n is the n^{th} *Harmonic number*. Similar to the case of the *Alg_o*, Eq.6 shows that $E(S)$ is an increasing function of k for any fixed n. More precisely, the expected number of trials for the *Alg_w* is at least k and never exceeds $n(H_n - H_{n-k})$, To investigate how well is the distribution of s for the *Alg_w*, we compute its variance (denoted by $V(S)$).

We have defined that $s = 1 + x_1 + \cdots + x_{k-1}$. By definition,

$$V(S) = V(1 + X_1 + X_2 + \cdots + X_{k-1})$$

Since all the $x_i's$ are independent, we have

$$V(S) = V(X_1) + V(X_2) + \cdots + V(X_{k-1})$$

Recall that x_i is in a geometric distribution, its variance is $\frac{ni}{(n-i)^2}$. Therefore,

$$\begin{aligned} V(S) &= \sum_{i=1}^{k-1} \frac{ni}{(n-i)^2} \\ &= n[\frac{1}{(n-1)^2} + \cdots + \frac{k-1}{(n-k+1)^2}] \end{aligned} \quad (7)$$

3.3 Random Permutation

We have seen that a random permutation of n objects can be completed in $n-1$ exchanges. In our problem only the first k objects are selected. Indeed, it is not necessary to order the rest $n-k$ objects. Therefore, we can modify the procedure for random permutation and terminate the (permutation) procedure after making the first k exchanges. With this modification, we have the following result:

$$E(S) = k \quad (8)$$

$$V(S) = 0 \quad (9)$$

3.4 Comparison

To summarize, the expected numbers of trials for *Alg_o* *Alg_w* and *Alg_p* are captured in Eqs. 2, 6 and 8, and their variances are captured in Eqs. 3, 7 and 9, respectively. Table 1 shows some values of expected number of trials and standard deviation (square root of variance) for the three algorithms for $n = 10,000$.

It can be seen that *Alg_p* needs exactly k trials and its standard deviation is always 0, which is the best possible and the best choice of the three. The performance of *Alg_o* is acceptable only when $k \approx n$, under that condition the expected number of trials is about n; however, the expected number of trials is at least $0.5(n+1)$, even when we just need to pick up a single one (*i.e.*, $k = 1$) out of the n candidates; this is not acceptable for large n. On the other hand, *Alg_w* performs reasonably well when $k \ll n$; however, Eq. 6 shows that as k approaches n, the the expected number of trials goes up to about nH_n, which can be approximated (see *e.g.*, [1, pp.262]) by $n \log n$ for large n; this is quite different from the other two algorithms, where both are bounded by n. How do these factors translate into computer execution time? In the next section, we will further examine the performance of the three algorithms by computer simulation.

4 Simulation Results

Although the expected number of trials needed is a good measurement for the performance of the three algorithms, there are other factors involved. In fact, the *exchange* operation in *Alg_p* (line 1.2)

k	*Alg_o*		*Alg_w*		*Alg_p*	
	mean	std	mean	std	mean	std
100	9902.0	97.5	100.5	0.7	100	0
500	9981.0	19.4	512.9	3.7	500	0
1000	9991.0	9.5	1053.5	7.6	1000	0
2000	9996.0	4.5	2231.3	16.4	2000	0
3000	9997.7	2.8	3566.5	26.8	3000	0
4000	9998.5	1.9	5107.9	39.5	4000	0
5000	9999.0	1.4	6931.0	55.4	5000	0
6000	9999.3	1.1	9162.2	76.4	6000	0
7000	9999.6	0.8	12038.6	106.3	7000	0
8000	9999.8	0.6	16092.4	154.6	8000	0
9000	9999.9	0.4	23021.4	258.7	9000	0
9500	9999.9	0.2	29947.8	399.8	9500	0
9900	10000.0	0.1	46002.3	969.0	9900	0

Table 1. Mean and Standard deviation for number of trials for $n = 10,000$.

is also a deciding factor in determining the execution time, since it repeats k times. In this section, we use computer simulation to do further study.

We have implemented the three algorithms in C language, and run them on a Sun Ultra 1 workstation with 128 MB RAM. It is seen that k and n are two determining factors in execution time. Different set-ups for k and n are tried. Since all three algorithms are nondeterministic, the execution times were averaged over 400 runs. In our previous analysis, we have observed that *Alg_p* would be the best choice of three, if the expected number of trials is considered along. However, based on our simulation experiment, this is not always true. The simulation result for $n = 10,000$ is shown in table 2, from which we can observe the following: (1) *Alg_o* performs best for $k/n \geq 0.95$, (2) *Alg_w* performs best for $k/n \leq 0.1$, and (3) *Alg_p* outperforms others for $0.2 \leq k/n \leq 0.9$. To see the above observations, we focus on different k/n ratios. For small k/n (*i.e.*, $k \ll n$), Eqs. 2, 6 and 8 show that the expected number of trials is: (a) at least $0.5(n+1)$ for *Alg_o*, (b) approximately k for *Alg_w*, and (c) exactly k for *Alg_p*. It is clear that *Alg_o* needs to do much work compared to others. Step 3 in *Alg_w* is iterated for approximate k times, where most time is spent on random number generation. Step 1 in *Alg_p* is also iterated for k times, where in each iteration we need to do one *exchange*

k/n %	*Alg_o*	*Alg_w*	*Alg_p*
1	149150	13700	13750
5	150325	20025	20325
10	151325	28200	28525
20	153425	46000	44850
30	155250	66175	61175
40	157000	89250	77450
50	158425	116125	93775
60	159900	148725	109975
70	161075	190175	126400
80	162150	247850	142450
90	163275	344325	158750
95	163700	439600	166825
99	164175	659525	173025

Table 2. Execution time in μ-sec for $n = 100,000$

in addtion to generate a random number. This explains obervation (2). For large k/n (*i.e.*, $k \approx n$), Eqs. 2, 6 and 8 show that the expected number of trials is: (a) approximately k for *Alg_o*, (b) approximately nH_n for *Alg_w*, and (c) exactly k for *Alg_p*. Again, k *exchanges* are needed for *Alg_p*, but not in *Alg_o*. This explains observation (1). For medium k/n, the expected number of trials becomes the determinant factor of the execution time. Hence, *Alg_p* will be our best choice, which

explains observation (3).

To quantify the cut-off points for small, medium and large, we did further experiment on $n = 1,000$ and $n = 10,000$. The result is shown in Figures 1 and 2, where the execution time (denoted by t) is displayed as a function of k/n. Figure 1 illustrates

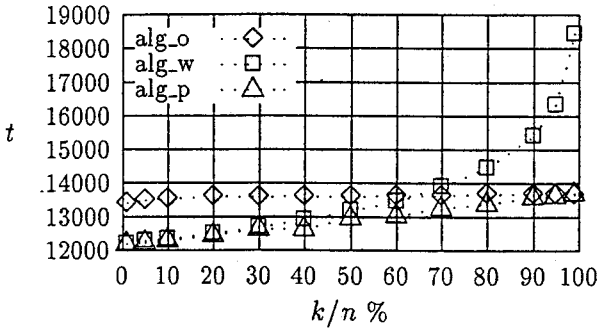

Figure 1. Execution time in μ-sec for $n = 1,000$

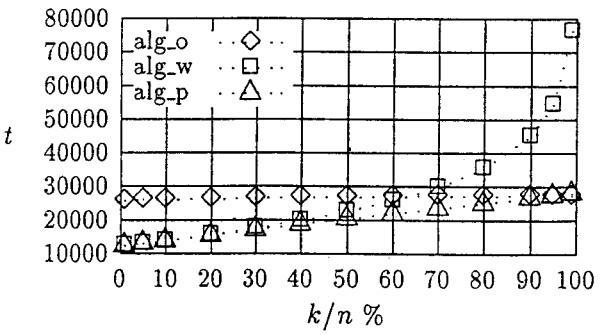

Figure 2. Execution time in μ-sec for $n = 10,000$

the performance of the three algorithms, which confirms our analysis result obtained in the previous section. More precisely, the execution time for both *Alg_o* and *Alg_p* are bounded above by a linear function of n, whereas the execution time for *Alg_w* can be approximated by a fucntion of $n \log n$ when $k \approx n$. The cut-off points also depend on the value of n. From Table 2, Figure 1 and 2, the cut-off points for small, medium and large are: (1) 0.05, 0.99 for $n = 1,000$, (2) 0.05, 0.95 for $n = 10,000$, and (3) 0.1, 0.95 for $n = 100,000$, respectively. It can be seen that the cut-off point for small and medium moves up as n increases. On the other hand, the cut-off point for medium and large goes down as n increases. Both are due to the exchange operations invloved in *Alg_p* but not in others. As n increases, the time spent for exchange operations becomes an important part in the performance. Hence, despite of more trials, *Alg_w* and *Alg_o* outperforms *Alg_p* for small k/n and large k/n, respectively. In summary, all three algorithms have their own merit. In general, for different k/n ratio, a guideline would be: (1) *Alg_w*, if $k/n \leq 0.1$, (2) *Alg_p*, if $0.1 < k/n < 0.9$, and (3) *Alg_o*, if $k/n \geq 0.9$.

5 Concluding Remarks

In this work, we have studied the problem of efficient lottery drawing using computer, *i.e.*, an unbiased choice of k items at random from a total of n candidates, where $k \leq n$. We have presented three algorithms to tackle the problem. Our analysis shows that the expected number of random numbers required for the three algorithms are approximately n, $n \log n$, and k, respectively. However, there are other factors involved. Each algorithm has its strength for certain values of k and n. For different k/n ratio, a general guideline would be: choose (1) *Alg_w*, if $k/n \leq 0.1$, (2) *Alg_p*, if $0.1 < k/n < 0.9$, and (3) *Alg_o*, if $k/n \geq 0.9$. In fact, *Alg_p* has been used for years at out University in providing an unbiased student enrollment for certain courses.

References

1. GRAHAM, R. L., KNUTH, D. E., AND PATASHNIK, O. *Concret Mathematics.* Addison Wesley, 1989.

2. HOEL, P. G., PORT, S. C., AND STONE, C. J. *Introduction to Probability Theory.* Houghton Mifflin Co., 1971.

3. KNUTH, D. E. *The Art of Computer Programming*, second ed., vol. 2. Addison-Wesley, 1981.