stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE STATISTIEK SW 51/77 OKTOBER
(DEPARTMENT OF MATHEMATICAL STATISTICS)

R. KAAS

THE COMPLEXITY OF DRAWING AN ORDERED RANDOM SAMPLE

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

AMS(MOS) subject classification scheme (1970): 62D05,62K99

The complexity of drawing an ordered random sample [*)]

by

R. Kaas

ABSTRACT

Every random sample of size k from {1,2,...,n} can be drawn in expected time O(k log k). We prove that, provided a logarithm costs one computation step, O(k) is sufficient to take a random sample without replacement.

Algorithms feasible for implementation on small hand-computers are given.

KEY WORDS & PHRASES: *random sampling, algorithms, complexity.*

---

*) This report will be submitted for publication elsewhere

1. INTRODUCTION

In this report we consider the average time complexity of drawing a sorted random sample (r.s.) of size k from an ordered population of known finite size n, represented by $\{1,2,\ldots,n\}$. The second section is devoted to algorithms that do the sorting only after the whole sample has been drawn. We prove that in this way $O(k \log k)$ average time is sufficient to draw a sample either with or without replacement. Since sorting the sample requires that the whole sample resides in memory, these algorithms are not suitable for small pocket calculators, unless k is very small. Quite often the relevant variables of the population can only be accessed sequentially: it takes time $O(|i-j|)$ to go from element i to element j, so collecting all the data in the sample alone requires a time at least proportional to n. If this is so it is rather pointless to draw the sample in a time shorter than $O(n)$.

The remainder of this report is mainly devoted to $O(n)$ algorithms, suitable also for small pocket calculators. They directly yield an ordered sample by deciding for each consecutive element how many times that element should be included in the sample. It is clear that by this method a sorted sample can be obtained. In section 3 we give a theorem supplying us with a method to draw a sorted random sample. BEBBINGTON's [2] algorithm for drawing a sorted r.s. without replacement is a direct consequence of this theorem. A slight simplification of Bebbington's algorithm results in an algorithm by HAMAKER [3]. He suggests a method resulting in a random sample-size following a binomial distribution. This disadvantage can be overcome by iterating the method until the desired sample-size is reached. The power of this algorithm lies in the fact, that a trick exists to let it run in an expected time of $O(k)$ rather than in time $O(n)$. We prove that ${}^2\log{}^2\log k$ iterations with an average length (in computing time) of $O(k)$ suffice to obtain a sorted r.s. of size $\underline{k}$*) with $E|\underline{k}-k| \leq 2$. This algorithm, however, supposes that a logarithm can be taken in constant time. If we change the procedure to the effect that, after an iteration resulting in a sample with size larger than k, we apply Bebbington's algorithm to reduce the sample to the right size, we obtain an $O(k)$ algorithm.

*) random variables are underlined

In the fifth section we return to sampling with replacement. According to Hamaker a slight modification of the sequential algorithm described in section 4 yields a way to obtain an ordered r.s. with replacement. We will show that the result is actually a random increasing row, cf. section 5. Theorem 3.1 however, directly provides a method of drawing a sorted random sample. In the last section we give ALGOL-68 source texts of those sequential algorithms that can be programmed on small hand computers, as they do not need more than a constant number of registers.

We conclude the introduction by giving some examples of practical situations to illustrate where which of our algorithms might be useful. Suppose you are at the beginning of a street containing n houses, k of which must be visited for an inquiry. Bebbington's algorithm, i.e. deciding with varying probability for every house if it will be included in the sample, is suggested if you have to walk the distances between the houses (sequential access). If you have a car at your disposal, the distances between the sample-elements cost a more or less constant amount of time: this is called random access. In this case the Hamaker algorithm is suggested, iterating the procedure, i.e. driving back to the beginning of the street if the sample is too small and drawing extra elements, if a fixed sample size must be obtained. The iteration is stopped as soon as after an iteration step the sample size obtained is not less than k. After this a part of the data is discarded by scanning the sample using Bebbington's algorithm (if necessary).

Another example with sequential access only occurs when the data are on a sequential file in a computer. The statistical program library SPSS provides a means of drawing a random sample from the data, using the O(n) version of Hamaker's algorithm. Though the non-fixed sample size is not a serious disadvantage for reasonably large k, as Hamaker pointed out, Bebbington's algorithm would, at the cost of n divisions, result in an r.s. of predetermined size.

2. ORDERING THE SAMPLE AFTER HAVING DRAWN IT

In practice if we want a random sample with replacement we usually perform k random drawings from $\{1,2,\ldots,n\}$ and sort the results to obtain an ordered r.s. of size k from {1,2,...,n}. For an r.s. without replacement we could draw randomly first from {1,2,...,n} and renumber, then from {1,2,...,n-1} and renumber, etc., until we have a sample of k distinct elements. This is realized more conveniently by drawing from {1,2,...,n} each time, and re-drawing should the number drawn already be in the sample. A/H/U [1] describe a suitable data structure: the binary search tree,*) to represent the sample. It must be possible to insert elements in the sample, and also to test if an element is in the sample. The binary search tree supports these intructions, and also the intructions min (giving the minimal element currently in the sample) and delete (the inverse of insert). Using the instructions min and delete one can easily obtain an ordered r.s. from the tree. Theorem 4.5 in A/H/U [1] states that m instructions of this kind can be performed in average time O(m log m). We are now ready to state the following theorem:

THEOREM 2.1. *An ordered random sample of size k from the set* {1,2,...,n} *can be drawn in* O(k log k) *average time, either with or without replacement.*

PROOF. The case with replacement is easy: the drawings each take constant time, and k numbers can be sorted in time O(k log k). Let us now consider sampling without replacement. Consider the following algorithm:

```
(1)     while k1 < k
(2)     do u:= u;
(3)        if not member(u)
(4)        then insert(u); k1:= k1 + 1
(5)        fi
(6)     od;
```

Here u represents a random drawing from {1,2,...,n}, k1 is the number of

*) A binary search tree is a labelled binary tree such that for every node all labels in the left hand subtree are smaller and those in the right hand subtree are larger than its own label. Starting at the root of the tree we descend in the tree having a three way choice at every node: go into the left hand subtree, stop the process or go into the right hand subtree depending on whether the label of the node is larger, equal to or smaller than the element looked for.

elements already in the sample and is initially 0, and k is the required sample size. Member(u) and insert(u) are routines manipulating the initially empty binary search tree.

Now let $\underline{t}$ be the ~~running~~-time of this algorithm; $\underline{n}$ is the number of times the loop is executed. Assume without loss of generality that $k \le \frac{1}{2}n$ (if not, draw a sorted r.s. of size n-k and take its complement, which can be done in time O(k)). We want to prove $E\underline{t} = O(k \log k)$. Since $k/n \le \frac{1}{2}$ the expected running time of this algorithm certainly does not decrease if we replace the test at line 3 by a test failing with probability $\frac{1}{2}$ each time but otherwise taking the same time as a member instruction.

Suppose we change the algorithm in this way, obtaining a running-time $\underline{t}'$ and $\underline{n}'$ executions of the loop. Now $\underline{n}'$ has a negative binomial $(k,\frac{1}{2})$ distribution. The already mentioned theorem from A/H/U [1] implies the existence of constants c and d, only dependent of the machine and the manipulating routines, such that

$$E(\underline{t}' \mid \underline{n}' = n) \le c.n.\log n + d.$$

But

$$\sum_{n=k}^{\infty} n \log n \, P(\underline{n}' = n) =$$

$$= \sum_{n=k}^{\infty} n \log n \, \tfrac{1}{2}^{n} \binom{n-1}{k-1} =$$

$$= 2k \sum_{n=k}^{\infty} \log n \, \tfrac{1}{2}^{n+1} \binom{n}{k} =$$

$$= 2k \sum_{m=k+1}^{\infty} \log(m-1) \, \tfrac{1}{2}^{m} \binom{m-1}{k}.$$

This last sum equals $E(\log(\underline{m}-1))$, if the r.v. $\underline{m}$ has a negative binomial $(k+1,\frac{1}{2})$ distribution. Now applying Jensen's theorem one obtains directly

$$E(\log(\underline{m}-1)) \le \log(E(\underline{m}-1)) = \log(2k+1).$$

So

$$E\underline{t} \le E\underline{t}' \le \sum_{n=k}^{\infty} (cn.\log n + d) \, P(\underline{n}' = n) = O(k \log k).$$

If after the previous algorithm we execute the following loop:

```
for k1:= k step -1 until 1
do intosample(min); delete(min) od;
```

we obtained an ordered r.s. without replacement in time O(k log k). □

3. DRAWING A SAMPLE ELEMENT BY ELEMENT; BEBBINGTON'S ALGORITHM

Suppose $\underline{f}^{k,n}$ denotes the frequency of element n in a random sample of size k from $\{1,2,\ldots,n\}$. Consider the following procedure for taking a sample:

1. Let f be the result of a drawing from the distribution of $\underline{f}^{k,n}$.
2. Draw a random sample of size k-f from $\{1,2,\ldots,n-1\}$ and sort it.
3. Concatenate the result of 2 with f times element n.

THEOREM 3.1. *This procedure results in an ordered random sample of size* k *from the population* $\{1,2,\ldots,n\}$.

PROOF. Because of 1 the probability of a sample with f occurrences of element n is what it should be. Because of 2 all unordered samples with f occurrences of element n have equal probability, and because of 2 and 3 the sample will be ordered. □

It does not make any difference if in step 2 we draw the sorted r.s. by considering at every stage the distribution of the highest numbered element: the proof then goes by induction on n. For a sample without replacement the distribution of $\underline{f}^{k,n}$ is given by:

$$P(\underline{f}^{k,n} = 1) = 1 - P(\underline{f}^{k,n} = 0) = \frac{k}{n}\,.$$

This follows by dividing the number of samples containing n: $\binom{n-1}{k-1}$ by the total number of samples $\binom{n}{k}$. Now direct application of the above theorem yields the correctness of the algorithm described by BEBBINGTON [2] for drawing an ordered r.s. without replacement:

ALGORITHM 3.1.

1. Draw from an alternative (k/n) distribution.
2. In case of a success, lower k by 1, and enter the element considered into the sample.
3. Lower n by 1, and proceed for the next element with 1., applying the new values of n and k, until k = 0.

4. DRAWING WITH FIXED PROBABILITY OF SELECTION

In algorithm 3.1 we selected elements with probability of selection equal to the proportion of elements still to be selected. In this section we consider what happens if we select every element with a fixed probability p. Only sampling wihtout replacement is dealt with. The following can be said about the performance of this modified scheme in drawing an ordered random sample.

THEOREM 4.1. *If we change algorithm* 3.1 *so that every element will be selected with fixed probability* p*, the result is a sample with a binomially* (n,p) *distributed size. Under this scheme every sample of size k has an equal probability.*

PROOF. The sample size is equal to the number of successes in n independent trials, each with probability p of sucess. For a sample size k exactly k successes must have occurred, and n-k failures. The probability of such a sample is equal to $p^k(1-p)^{n-k}$. □

If the population size is known this scheme in itself is of minor importance. It can simply be replaced by algorithm 3.1, and then one obtains a predetermined sample size. The disadvantage of a binomially $(n,\frac{k}{n})$ distributed sample size must not be overestimated. HAMAKER [3] showed, that the variance of the sample mean as an estimator of the population mean increases by only a factor $1 + \frac{1}{k}$, provided the sampling procedure is repeated until a non-empty sample is obtained. Assuming taking a logarithm can be done in one step, however, the mean running time of this algorithm can be improved from O(n) to O(k). In the sequel the random vector $(\underline{x}_1,\underline{x}_2,\ldots,\underline{x}_{\underline{k}})$ will denote the sample obtained with the algorithm of the preceding theorem with probability of selection p. Let the r.v. $\underline{x}_0$ satisfy $P(\underline{x}_0=0) = 1$ and define for $i = 1,2,..,\underline{k}$ the r.v.'s $\underline{d}_i$, the distances between sample elements, by

$$\underline{d}_i \overset{\text{def}}{=} \underline{x}_i - \underline{x}_{i-1}.$$

The following theorems hold.

THEOREM 4.2. *All* $\underline{d}_i$ *have a geometric*(p) *distribution.*

PROOF. For a distance $\underline{d}_i = d$ the (i-1)st success must be followed by a sequence of d-1 failures and a success. □

THEOREM 4.3. *If* $\underline{u}$ *has a uniform* (0,1) *distribution, the* r.v. $\underline{d}$, *defined by*

$$\underline{d} \overset{\text{def}}{=} \left[{}^{q}\log \underline{u}\right] + 1,$$

has a geometric (1-q) *distribution, with* $0 < q < 1$.

PROOF. Let d be an arbitrary natural number. Then

$$P(\underline{d} = d) = P([{}^{q}\log \underline{u}] = d - 1) =$$

$$= P({}^{q}\log \underline{u} \in [d-1,d)) = P(\underline{u} \in (q^{d},q^{d-1}]) =$$

$$= q^{d-1}(1-q). \quad \square$$

It is clear that these two theorems yield a method of drawing random distances in one step, so we can draw an ordered r.s. of binomially (n,p) distributed size $\underline{k}$, in a time proportional to $\underline{k}$. So we have an algorithm with an O(k) average running time, if we take $k = E\underline{k} = n.p$.

We will now consider what happens if we try to remove the disadvantage of a non-fixed sample-size by using the same algorithm for either drawing extra elements of for removing elements from the sample, until the desired sample-size is obtained. Let us formalize this idea in the following algorithm.

ALGORITHM 4.1. (Drawing an r.s. of size k from $\{1,2,...,n\}$).

Step 1 Use the method of drawing random distances with parameter $q = 1 - \frac{k}{n}$ to obtain a sample of size $\underline{k}_1$.

Step i+1 Suppose $\underline{k}_i$, the sample size obtained by the first i steps has the value k_i. If $k_i = k$, we can stop the procedure. If $k_i < k$, just as in step 1 a sample is drawn from the remaining $n - k_i$ population elements, with parameter $q = (n-k)/(n-k_i)$. If $k_i > k$, we use the method of step 1, to remove elements from the k_i sample elements, taking $q = \frac{k}{k_i}$.

THEOREM 4.4. *Supppose* $\underline{k}_1, \underline{k}_2, \ldots$ *are defined as in the previous algorithm, then the following properties hold:*

a) $\underline{k}_1$ *has a binomial* $(n, \frac{k}{n})$ *distribution.*

b) $\underline{k}_{i+1} - \underline{k}_i | \underline{k}_i = k_i < k$ *has a binomial* $(n-k_i, \frac{k-k_i}{n-k_i})$ *distribution.*

c) $\underline{k}_i - \underline{k}_{i+1} | \underline{k}_i = k_i > k$ *has a binomial* $(k_i, \frac{k_i-k}{k_i})$ *distribution.*

PROOF. This follows directly from theorem 4.1, the equivalence of the sampling schemes and the choices of the parameter q. □

We have a row of binomially distributed r.v.'s, the expectation of every term being equal to the absolute deviation from the mean of the preceding term. Suppose $m = \lceil {}^2\log {}^2\log k \rceil$, where $\lceil t \rceil \overset{\text{def}}{=} -[-t]$. How close do m iterations of algorithm 4.1 bring us to our goal: a sample of size k. We will prove, that the resulting sample size $\underline{k}_m$ satisfies $E|\underline{k}_m - k| \le 2$. After the m-th step we proceed with an unspecified tail process, and expect to have to do only a few steps. First we need the following simple lemma:

LEMMA 4.1. *For every* r.v. $\underline{x}$ *the following inequalities hold:*

$$(4.1) \quad E\underline{x} \le \sqrt{E\underline{x}^2}$$

$$(4.2) \quad E|\underline{x} - E\underline{x}| \le \sqrt{\operatorname{var}(\underline{x})}.$$

PROOF. $0 \le E(\underline{x} - E\underline{x})^2 = E\underline{x}^2 - (E\underline{x})^2$, implying (4.1). (4.2) follows from applying (4.1) to the r.v. $\underline{y} = |\underline{x} - E\underline{x}|$. □

THEOREM 4.5. For $1 \le i \le m$ *the following inequality holds:*

$$\sigma^2_{\underline{k}_{i+1}} \le \sigma_{\underline{k}_i}.$$

PROOF.

$$\mathrm{var}(\underline{k}_{i+1} \mid \underline{k}_i = k_i) = \begin{cases} (n-k_i)\,\dfrac{k-k_i}{n-k_i}\,\dfrac{n-k}{n-k_i} < k - k_i & \text{if } k > k_i, \\ 0 & \text{if } k = k_i, \\ k_i\,\dfrac{k_i-k}{k_i}\,\dfrac{k}{k_i} < k_i - k & \text{if } k < k_i. \end{cases}$$

so always $\mathrm{var}(\underline{k}_{i+1} \mid \underline{k}_i = k_i) \le |\underline{k}_i - k|$. As for every k_i

$$E(\underline{k}_{i+1} \mid \underline{k}_i = k_i) = k,$$

also

$$\sigma^2_{\underline{k}_{i+1}} = E(\mathrm{var}(\underline{k}_{i+1} \mid \underline{k}_i)) + \mathrm{var}(E(\underline{k}_{i+1} \mid \underline{k}_i)) \le$$

$$\le E|\underline{k}_i - k| \le \sigma_{\underline{k}_i} \text{ because of (4.2).} \quad \square$$

COROLLARY 4.1. $E|\underline{k}_m - k| \le 2$.

PROOF. Let $p = \frac{k}{n}$. Then $\sigma_{\underline{k}_1} = \sqrt{np(1-p)} < \sqrt{np} = \sqrt{k}$. By the definition of m,

$$2^{2^m} \ge k, \text{ so } k^{((\frac{1}{2})^m)} \le 2, \text{ and } (\sqrt{k})^{((\frac{1}{2})^{m-1})} \le 2.$$

Because of the preceding theorem, for every $i = 2,3,\ldots,m$

$$\sigma_{\underline{k}_{i+1}} \le \sqrt{\sigma_{\underline{k}_i}},$$

so

$$\sigma_{\underline{k}_m} \le \sigma_{\underline{k}_1}^{((\frac{1}{2})^{m-1})} \le 2,$$

so again with (4.2) $E|\underline{k}_m - k| \le 2$. $\square$

We have determined the average number of iterations algorithm 4.1 takes, but we have said nothing about their length. The first step is easily seen to have an average length of $O(k)$. Let $i > 1$ and let S_i be the "current" sample. If $|S_i| < k$, we have to draw elements from $\{1,2,\ldots,n\}\backslash S_i$; if $|S_i| > k$, we draw from S_i. Our method with jumps permits us to choose the

We will prove an interesting lemma, based on the following inequalities proven by UHLMANN [4]:

$$P(\underline{x} \le k | n, \frac{k}{n-1}) \ge \tfrac{1}{2} \ge P(\underline{x} \le k | n, \frac{k+1}{n+1}), \qquad k \le \frac{n-1}{2}, \ n > 1,$$

$$P(\underline{x} \le k | n, \frac{k}{n-1} \le \tfrac{1}{2} \le P(\underline{x} \le k | n, \frac{k+1}{n+1}), \qquad k \ge \frac{n-1}{2}, \ n > 1.$$

Since clearly $P(\underline{x} \le c|n,p_1) > P(\underline{x} \le c|n,p_2)$ if $p_1 < p_2$, and since $^k/n < \min\{^k/_{n-1}, ^{k+1}/_{n+1}\}$, $k = 1,2,\ldots,n-1$, $n = 2,3,\ldots$, we see that

$$P(\underline{x} \le k | n, \frac{k}{n}) > \tfrac{1}{2}$$

for $n = 2,3,\ldots$ and $k = 1,2,\ldots,n-1$. For $k = 0$ or $k = n$ and for $n = 1$ this also clearly holds. By looking at the complementary binomial $(n,\frac{n-k}{n})$ r.v. we find that also

$$P(\underline{x} \ge k | n, \frac{k}{n}) > \tfrac{1}{2}.$$

By the above discussion we have proven:

LEMMA 4.2. *The median of a binomial* $(n, \frac{k}{n})$ r.v. *is* k.

But this means that $P(\underline{i} > i) \le 2^{-i}$, and so $E\underline{i} \le 2$. We formulate the main result of this paragraph in the following theorem.

THEOREM 4.6. *A sorted random sample of size* k *from the population* $\{1,2,\ldots,n\}$ *can be drawn in time* $O(k)$, *provided we assume taking a logarithm is a basic computation step.*

5. SAMPLING WITH REPLACEMENT

A sample with replacement arises with a sampling method admitting an element already in the sample again to be chosen in the sample. To obtain such a sample with methods similar to those of algorithm 3.1 and theorem 4.1 after having selected an element we do not proceed with the next element, but decide by a random drawing if the same element is to be included again. We start to process an element only after the rejection of the previous element. Proceeding this way we get a non-decreasing row of elements from the population. Suppose $\underline{k}$ is the length of the row we get by each time rejecting with fixed probability q. Now the r.v. $\underline{k} + n$ has a negative binomial (n,q) distribution, since it equals the number of independent experiments we have to perform to obtain the n-th success, a rejection being a success. By a suitable choice of q a row of expected size k results. If $\underline{d}$ is the difference between two successive row elements, the r.v. $\underline{d} + 1$ has a geometric(q) distribution. Just as in the preceding section we may simulate the random drawing procedure by drawing random distances $\underline{d}' = [{}^{q}\log \underline{u}]$, $\underline{u}$ being uniformly (0,1) distributed. A distance $\underline{d}' = 0$ means that the same element has to be put in the sample once again.

After $\lceil {}^{2}\log {}^{2}\log k \rceil$ iterations we have a non-decreasing row of a length $\underline{k}$ with $E|\underline{k} - k| \leq 2$, in a total computing time $O(k)$ on the average. It is also possible to obtain a non-decreasing row of a predetermined length k directly by rejecting in the j^{th} drawing with probability

$$\frac{n - \underline{n}_j - 1}{n - \underline{n}_j - 1 + k - \underline{r}_j},$$

if before the j^{th} drawing $\underline{r}_j$ elements are in the row and $\underline{n}_j$ elements have been rejected. It can be proved that with the methods described above all rows of a given length have the same probability.

Unjustly HAMAKER [3] presents the above method as a way to draw an r.s. with replacement. It is clear that, given the resulting length, a random element of the set of non-decreasing rows of population elements results. The definition of an ordered r.s., however, requires that a sorted random sample is delivered. It may occur that a non-decreasing row has more than one original (under the operation of sorting) in the set of unordered samples. What is wrong can be understood from the following simple example:

<u>EXAMPLE 5.1</u>. Suppose we want an ordered r.s. of size 2 from the set {1,2}. The following unordered samples may arise:

(1,1), (1,2), (2,1), and (2,2).

So the probability of the ordered random sample 1,2 equals ½. Hamaker's method gives all non-decreasing rows of length 2

1,1 , 1,2 and 2,2

equal probability. □

This difficulty does not exist for drawing without replacement, since in this case every increasing row of length k has exactly k! originals under the operation of ordering in the sample space.

It is possible to derive a correct algorithm from our theorem 3.1. The frequency $\underline{f}^{k,n}$ of element n in an r.s. with replacement of size k has a binomial $(k, \frac{1}{n})$ distribution. So our problem has been reduced to generating random drawings from binomial distributions. This might be done by performing alternative $(\frac{1}{n})$ drawings k times and counting the number of successes. In this way, however, O(nk) computing time is used. A more efficient way to do it is by computing the inverse of the binomial d.f. in an argument that is obtained by a random drawing from (0,1), using a recurrence relation between binomial probabilities. The correctness of the following algorithm is a direct consequence of the already mentioned theorem 3.1 and the above discussion.

ALGORITHM 5.1 (Drawing an r.s. with replacement from $\{1,\ldots,n\}$).

1. Let u be a pseudo random number between 0 and 1.
2. Let i = 0 and compute $p = q_0 = (1 - \frac{1}{n})^k = p(\underline{f}^{k,n} = i)$.
3. if $p > u$, proceed with 6.
4. Add 1 to i, compute $q_i = P(\underline{f}^{k,n} = i)$ by using the binomial recurrent relation

$$q_i = q_{i-1} \cdot \frac{(k-i+1)}{i(n-1)} .$$

5. Add q_i to p, so $p = \sum_{j \le i} q_j$, and proceed with 3.
6. Include the current element i times in the sample, replace k by k-i and n by n-1 and continue with 1, using the new values of k and n.

6. SOURCE-TEXTS OF ALGORITHMS

In this section we give ALGOL-68 source-texts of those algorithms described in sections 3, 4 and 5, that are suitable for small hand computers. The following identifiers must have been declared in an outer block:

- f, n, and k are integer variables; n initially equals the size of the population, k equals the required sample-size;
- u, p and q are real variables;
- intosample(n,f) is a procedure that causes element n to be included in the sample with frequency f; the second parameter is omitted in the case of sampling without replacement, and then it is supposed to be 1;
- random is a real procedure computing the next number in a row of pseudo independently uniformly (0,1) distributed numbers.

PROGRAM 6.1. (Source-text of algorithm 3.1, Bebbington's algorithm for drawing an r.s. wihtout replacement)

```
begin for n:= n step -1 until 1 while k > 0
      do if random < k/n
         then intosample(n); k:= k - 1
         fi
      od
end
```

This algorithm is recommended if the data organization is sequential; it takes O(n) time, but the result always is an ordered r.s. of size k. For programming simplicity all programs in this section deliver the sample in reversed order. The following two programs are described theoretically in section 4; the first one should never be used, since it is hardly simpler than the previous one, it has the same O(n) running time, but it results in a sample with binomially $(n,\frac{k}{n})$ distributed size. The second program 6.3 has the same disadvantage, but its expected running time is O(k).

PROGRAM 6.2. (Drawing an r.s. without replacement of expected size k).

```
begin p:= k/n;
      for n:= n step -1 until 1
      do if random < p
         then intosample(n)
         fi
      od
end
```

PROGRAM 6.3. (Faster implementation of program 6.2)

```
begin f:= n + 1; q:= 1 - k/n;
      while f > n do f:= entier(log(1 - random)/log(q)) + 1 od;
      while f ≤ n
      do intosample(f);
         f:= f + entier(log(1 - random)/log(q)) + 1
      od
end
```

The second line of the program ensures that the resulting sample-size is unequal to zero. Instead of random we use 1 - random, since random may be zero. This program works best if the data-organization permits easy forward access, by which we mean that the time needed to go from data-item ℓ to data-item $\ell+m$ lies between $O(1)$ and $O(m)$. If the resulting sample-size is $\underline{k}$, the running time of this algorithm is $O(\underline{k})$, so the average running time equals $E(O(\underline{k})) = O(E(\underline{k})) = O(k)$.

Our last program, described theoretically in section 5, draws an r.s. of size k with replacement. The running time is $O(n+k)$, as we will show, so it requires a sequential organization of the data.

PROGRAM 6.4. (Drawing a sample with replacement of fixed size)

```
begin while k > 0 & n > 1                                    ( 1)
      do f:= 0; p:= q:= $(1 - \frac{1}{n})^k$; u:= random;   ( 2)
         while p < u                                         ( 3)
         do q:= q * (k-f) / ((f+1) * (n-1));                 ( 4)
            f:= f + 1;                                       ( 5)
            p:= p + q                                        ( 6)
         od;                                                 ( 7)
         intosample(n,f); k:= k - f; n:= n - 1               ( 8)
      od;                                                    ( 9)
      intosample(1,k)                                        (10)
end                                                          (11)
```

Line 1 of this program ensures that the sampling is stopped when either the sample is full (k=0) or only one element is left to be considered (n=1). In the second line the frequency f, the left-hand tail probability u and p and q are initialized, with

$$q = P(\underline{f}^{k,n} = f), \text{ and } p = \sum_{j \leq f} q_j,$$

with $\underline{f}^{k,n}$ as in the previous section, f as in the program. In the loop of lines 4-8 the inverse binomial $(k, \frac{1}{n})$ distribution function is computed for argument u, using a recurrence relation between binomial probabilities (line 4). The number of times the check in line 3 is performed is equal to one plus the resulting f. So the total running time of the algorithm is proportional to n plus the sum of the resulting f-values, so the algorithm runs in time O(n+k).

We conclude by remarking that all programs of this section can easily be adapted to run on small pocket calculators. Random number generators are often available as a subroutine, but are easily programmed anyhow. The into-sample procedure can be implemented by e.g. displaying n followed by its frequency, whenever it is positive.

REFERENCES

[1] AHO, A.V., J.E. HOPCROFT & J.D. ULLMAN, *The design and Analysis of Computer Algorithms*, Addison Wesley, Reading Mass. (1974).

[2] BEBBINGTON, A.C., *A simple method of Drawing a Sample Without Replacement*, Applied Statistics 24 (1975), 1, p.136.

[3] HAMAKER, H., *An Alternative Procedure for taking a Random Sample*, Statistica Neerlandica 29 nr. 2 (1975).

[4] UHLMANN, W., *Vergleich der hypergeometrischen mit der Binomial-Verteilung* Metrika, 10 (1966), p.145-148.