Selection Via the Bogo-Method – More on the Analysis of Perversely Awful Randomized Algorithms

Markus Holzer
Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
holzer@informatik.uni-giessen.de

Jan-Tobias Maurer
Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
jan.t.maurer@math.uni-giessen.de

Abstract

We continue our research on perversely awful randomized algorithms, which started nearly a decade ago. Based on the bogo-method we design a bogo-selection algorithm and variants thereof and analyse them with elementary methods. Moreover, practical experiments are performed.

2012 ACM Subject Classification Mathematics of computing → Combinatorics, Mathematics of computing → robability and statistics, Theory of computation → Algorithm design techniques

Keywords and phrases selection, bogo-method, combinatorial sums and series, inverse binomial coefficients, experimental result

Digital Object Identifier 10.4230/LIPIcs.FUN.2018.23

Acknowledgements Thanks goes to Gerrit Eichner for his help on the statistical analysis of our experimental data.

1 Introduction

Bogo-sort, also known as Monkey-sort, is according to [6] the archetypical perversely awful randomized algorithm. It is the equivalent to repeatedly throwing a deck of cards in the air, picking them up at random, and then testing whether they are in order. The analysis of bogo-sort carried out in [3] shows that this algorithm, while having best-case expected running time as low as $\mathrm{O}(n)$, achieves an asymptotic expected running time as high as $\Omega(n \cdot n!)$ already in the average case. Although there are other sorting algorithms known such as, e.g., Bogobogo-sort, Evil-sort, etc., with even higher inefficiency, the concept of bogusness was not considered as an algorithm design principle in general. Thus, the question arises, what is the concept of bogusness? In fact the design of Bogo-sort as stated in the beginning nicely generalizes to what we call the bogo-method by repeatedly throwing a deck of cards in the air and picking up sufficient cards at random in order to test whether a certain property is met. At first glance this design principle looks crazy, but it can be nicely applied to other problems than sorting such as, e.g., the selection problem or the two-element sum problem. In the former case the property that has to be met is that the kth largest element of a given array $a[1 \ldots n]$ is exactly on the kth position and that all the elements to the left of $a[k]$ are smaller while all the element to the right of $a[k]$ are larger – we obviously require that $1 \leq k \leq n$ holds. Thus, it is obvious that we have to pick up all cards at random in order to verify the stated property.


9th International Conference on Fun with Algorithms (FUN 2018).
Editors: Hiro Ito, Stefano Leonardi, Linda Pagli, and Giuseppe Prencipe; Article No. 23; pp. 23:1–23:23

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Here we will focus on the analysis of several selection algorithms following the bogo-method. The pseudo code for bogo-select that ensures the property mentioned above reads as follows:

Listing 1 Algorithm: Bogo-Select (by partitioning)

```
Input array a[1..n] and k with 1<=k<=n
while a[1..n] is not partitioned(k) do
        randomly permute a[1..n]
endwhile
Output a[k]
```

The test whether the array is partitioned such that $a[i] \leq a[k]$, for $1 \leq i < k$, and $a[k] \leq a[i]$, for $k < i \leq n$ as well as the permutation of the array have to be programmed with some care:

```
procedure partitioned (int: k):
// returns true if the array
// is partitioned according
// to a[k] and false otherwise
for i=1 to k-1 do
        if a[i]>a[k] then
                return false
        endif
endfor
for i=k+1 to n do
        if a[i]<a[k] then
                return false
        endif
endfor
return true
```

```
procedure randomly permute:
// permutes the array a[1..n]
for i=1 to n-1 do
        j := rand[i..n]
        swap a[i] and a[j]
endfor
```

The second algorithm was already used for the bogo-sort algorithm and is found, e.g., in [4, p.139]. The random permutation is done quickly by a single loop, where RAND gives a random value in the specified range. And the test for partitioning according to the element $a[k]$ is carried out from left to right.

In this work we present a detailed analysis of the bogo-select algorithm in several variations. It is worth mentioning that we still obtain a randomized selection algorithm if we relax the condition of being partitioned to a condition that only requires that there are exactly $k-1$ elements in $a[1 \ldots k-1, k+1, \ldots n]$ that are smaller or equal to $a[k]$; equivalently one requires that exactly $n-k$ elements in the array $a[1 \ldots k-1, k+1 \ldots n]$ are larger or equal to $a[k]$. This variant of the bogo-select algorithm will be analyzed later on. Our proofs require only a basic knowledge of probability and can be readily understood by non-specialists. This makes the analysis well-suited to be included as motivating example in courses on randomized algorithms. We will analyze the expected running time for bogo-select under the assumption that we are given an array $\overline{x} = x_1 x_2 \ldots x_n$ containing a permutation of the set of numbers $\{1, 2, \ldots, n\}$ with $n \geq 2$. To analyze the running time of the bogo-select algorithm and the variants thereof, we on the one hand count the number of comparisons, and on the other hand the number of swaps. This is similar to the analysis of the bogo-sort algorithm done in [3]. An immediate observation is that the algorithm isn't guaranteed to terminate at all. However, as we will prove that the *expectation* of the running time T is finite as we see by

Markov's inequality

$$\mathbb{P}[T \geq t] \leq \frac{\mathbb{E}[T]}{t}, \quad \text{for } t > 0,$$

that the probability of this event equals 0. There are essentially two different initial configurations: either the list $\overline{x}$ is initially partitioned according to the kth element or it is not. We have to make this distinction as the algorithm is smart enough to detect if the given list is initially well partitioned, and has much better running time in this case. This welcome built-in feature also makes the running time analysis of this case very easy: the number of total comparisons equals $n-1$, and the total number of swaps equals zero, since the while-loop is never entered.

We come to the case where the array is not initially well partitioned. Note that the first shuffle yields a randomly ordered list, so the behavior of the algorithm does no longer depend on the initial order – the number of comparisons before the first shuffle still does.

2 Bogo-Select: Selection By Partitioning

2.1 How Long Does it Take to Check Whether the kth Element is on its Correct Position?

Before the analysis of the procedure PARTITIONED we begin with a detour on inverse binomial coefficients, since they are mostly not too intensively covered by monographs on combinatorics – an exception is, e.g., [1]. Obviously, an inverse binomial coefficient obeys the identity $\binom{n}{k}^{-1} = \frac{k!(n-k)!}{n!}$. The first theorem is based on the elementary identity

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

on inverse binomial coefficients. The proof of the following theorem is literally taken from [7].

▶ **Theorem 1.** *For $n \geq 2$, we have*

$$\sum_{i=0}^{\infty} \binom{n+i}{i}^{-1} = \frac{n}{n-1}.$$

Proof. The proof is by induction on n. For $n = 2$, the sum equals

$$\sum_{i=0}^{\infty} \binom{2+i}{i}^{-1} = \sum_{i=0}^{\infty} \frac{i! \cdot (2+i-i)!}{(2+i)!} = 2\sum_{i=0}^{\infty} \frac{1}{(i+1)(i+2)} = 2\sum_{i=0}^{\infty} \left(\frac{1}{i+1} - \frac{1}{i+2}\right) = 2$$

for the terms pairwise cancel. For $n > 2$, we observe that

$$\sum_{i=0}^{\infty} \binom{n+i}{i}^{-1} = \binom{n+0}{0}^{-1} + \sum_{i=1}^{\infty} \binom{n+i}{i}^{-1} = 1 + \sum_{i=0}^{\infty} \binom{n+(i+1)}{i+1}^{-1}.$$

Applying the elementary equation on inverse binomial coefficients mentioned before the theorem to the rightmost sum, we have

$$\sum_{i=0}^{\infty} \binom{n+i}{i}^{-1} = 1 + \sum_{i=0}^{\infty} \left[\binom{n+i}{i}^{-1} - \frac{n}{n+1}\binom{n+(i+1)}{i}^{-1}\right].$$

Assuming $\sum_{i=0}^{\infty} \binom{n+i}{i}^{-1} = \frac{n}{n-1}$ and hence is finite, we obtain

$$\frac{n}{n+1} \sum_{i=0}^{\infty} \binom{(n+1)+i}{i}^{-1} = 1,$$

completing the proof. ◀

In the forthcoming we will come across more involved combinatorial sums and series containing inverse binomial coefficients, which will be evaluated with the help of a more general approach based on Euler's Gamma function. This will detailed in Subsection 3.1.

Now we are ready to analyse the procedure PARTITIONED. Recall that we demand n to be at least 2 as a prerequisite. Thus, we will not state $n \geq 2$ explicitly in all theorems and lemmata to come.

▶ **Theorem 2.** *Assume $\overline{x}$ is a random permutation of $\{1, 2, \ldots, n\}$, and let C denote the random variable counting the number of comparisons carried out in the test whether $\overline{x}$ is partitioned according to the kth element. Then*

$$\mathbb{E}[C] = \sum_{i=1}^{k-1} \frac{1}{i} + \frac{1}{k} \sum_{i=k}^{n-1} \binom{i}{k}^{-1}.$$

It holds $\mathbb{E}[C] = H_{n-1}$, if $k = 1$ or $k = n$, and $\mathbb{E}[C] \sim H_{k-1} + \frac{1}{k-1}$, otherwise.

Proof. Let k with $1 \leq k \leq n$ be fixed. In order to calculate $\mathbb{E}[C] = \sum_{i \geq 1} \mathbb{P}[C \geq i]$ we determine the probability for every valid i, that is $1 \leq i \leq n-1$ since we can compare $a[k]$ with at most $n-1$ other elements. Observe that the ith comparison is reached if and only if the algorithm did not drop out at the $i-1$ comparisons before. Furthermore observe that the first comparison within the second loop equals the kth total comparison. Therefore we consider two cases:

1. Let $1 \leq i < k$. If the routine makes a minimum of i comparisons, the $i-1$ first comparisons needed to be successful in order for the ith one to be carried out, yielding us $x_1, x_2, \ldots, x_{i-1} \leq x_k$. Thus, the probability $\mathbb{P}[C \geq i]$ computes as

   $$\mathbb{P}[C \geq i] = \frac{\binom{n}{i} \cdot (i-1)! \cdot (n-i)!}{n!}.$$

   The numerator is the product of the number of possibilities to choose i elements to arrange the $(i-1)$ smaller ones at the beginning of the array and the number of possibilities to arrange them, place the maximum from the i chosen elements on position k, and the number of possibilities to arrange the remaining $n-i$ elements at the uncharted part of the array, and the denominator is just the total number of arrays of length n. Reducing this fraction, we obtain $\mathbb{P}[C \geq i] = \frac{1}{i}$.
2. For $k \leq i < n$ we argue in similar veins as above and again find that $\mathbb{P}[C \geq i]$ can be estimated as

   $$\mathbb{P}[C \geq i] = \frac{\binom{n}{i} \cdot (k-1)! \cdot (i-k)! \cdot (n-i)!}{n!}.$$

   The numerator is the product of the number of possibilities to choose i elements to arrange the $k-1$ smallest elements at the beginning of the array and the number of possibilities to arrange them, place the kth largest element on position k, and the remaining $i-k$ elements just after the kth position and the number of possibilities to arrange them, and

finally the number of possibilities to arrange the remaining $n-i$ elements at the end part of the array with the denominator again being the total number of arrays of length n. Simplifying this fraction, results in $\mathbb{P}[C \geq i] = \frac{1}{k}\binom{i}{k}^{-1}$.

As the range of C is non-negative, we combine those two cases for $\mathbb{P}[C \geq i]$ for the expected value of C as follows:

$$\mathbb{E}[C] = \sum_{i \geq 1} \mathbb{P}[C \geq i] = \sum_{i=1}^{k-1} \frac{1}{i} + \sum_{i=k}^{n-1} \frac{1}{k} \binom{i}{k}^{-1}.$$

The former sum can easily be identified as the harmonic number $H_{k-1} = \sum_{i=1}^{k-1} \frac{1}{i}$. For the second term of the expected value above we have to do a little bit more calculation and the help of Theorem 1 we get:[1]

$$\sum_{i=k}^{n-1} \frac{1}{k} \binom{i}{k}^{-1} = \frac{1}{k} \sum_{i=0}^{n-1-k} \binom{k+i}{k}^{-1} \leq \frac{1}{k} \sum_{i=0}^{\infty} \binom{k+i}{i}^{-1} = \frac{1}{k} \cdot \frac{k}{k-1} = \frac{1}{k-1},$$

if $k \geq 2$. In case $k = 1$ we find

$$\mathbb{E}[C] = \sum_{i=1}^{1-1} \frac{1}{i} + \sum_{i=1}^{n-1} \frac{1}{1} \binom{i}{1}^{-1} = \sum_{i=1}^{n-1} \frac{1}{i},$$

which equals the harmonic number H_{n-1} similar to above – observe, that for $k = 1$ and $k = n$ the values for $\mathbb{E}[C]$ are the same further stressing the highly symmetrical nature of this very algorithm. Putting things together we obtain $\mathbb{E}[C] = H_{n-1}$, if $k = 1$ or $k = n$, and $\mathbb{E}[C] \sim H_{k-1} + \frac{1}{k-1}$, otherwise, as n tends to infinity. This proves the stated claim on the asymptotics. ◀

Theorem 2 tells us that we can expect comparisons in the range of H_n to check if a large array is well partitioned, and for n large enough, this number is about $\ln n$, because H_n satisfies the asymptotic

$$H_n \sim \ln n + \gamma,$$

where $\gamma \approx 0.5772156649$ is the Euler-Mascheroni constant. Compare to the worst case, where we have to compare $n-1$ times. This is a noticeable departure from the check for being sorted from left to right as analysed in [3], where it was shown that on average one only needs a constant number of comparisons, namely $e - 1 \approx 1.718281828$ for large enough arrays.

2.2 The Expected Number of Swaps in Bogo-Select

When computing the expected number of iterations in bogo-select, we concentrate on the case where the input $\overline{x}$ is not partitioned according to the kth element; for the other case it

[1] With a little bit more effort one can show that

$$\sum_{i=0}^{n} \binom{k+i}{i}^{-1} = \frac{k}{k-1} \left[1 - \binom{n+k}{n+1}^{-1}\right],$$

for $k \geq 2$ and $n \geq 0$; e.g., see the alternative proof of Theorem 1 given on page 33:9 or [2, page 16, Equation (2.2)]. This allows us to refine the expected value of C to the precise expression $\mathbb{E}[C] = H_{k-1} + \frac{1}{k-1}\left[1 - \binom{n-1}{k-1}^{-1}\right]$, for $k \geq 2$.

equals 0, because of the intelligent design of the algorithm. In each iteration, the array is permuted uniformly at random, and we iterate until we hit a sequence that is well partitioned for the first time. As a well partitioned ordered sequence w.r.t. the kth element is hit with probability $\frac{(k-1)!(n-k)!}{n!}$ in each trial, the number of iterations I is a random variable with a probability as follows:

$$\mathbb{P}[I = i] = \left(1 - \frac{(k-1)!(n-k)!}{n!}\right)^{i-1} \cdot \frac{(k-1)!(n-k)!}{n!}$$

That is, I is a geometrically distributed random variable with hitting probability $p = \frac{(k-1)!(n-k)!}{n!}$. Simple calculations show that $p = \frac{1}{k}\binom{n}{k}^{-1}$ and that the expected value of I is equal to $\mathbb{E}[I] = p^{-1} = k \cdot \binom{n}{k}$

In each iteration, the array is shuffled and a shuffle costs $n-1$ swaps. As the algorithm operates kind of economically with respect to the number of swaps, these are *the only* swaps carried out while running the algorithm. If S denotes the random variable counting the number of swaps, when asking for the kth element, we have $S = (n-1) \cdot I$. By linearity of expectation, we derive:

▶ **Theorem 3.** *If S denotes the total number of swaps carried out for an input $\overline{x}$ of length n, when asking for the kth element, for $1 \leq k \leq n$, we have*

$$\mathbb{E}[S] = \begin{cases} 0 & \text{if } \overline{x} \text{ is well partitioned according to the } k\text{th element} \\ k(n-1)\binom{n}{k} & \text{otherwise.} \end{cases}$$

Thus we immediately obtain:

▶ **Corollary 4.** *Let S denote the number of swaps carried out by bogo-select on a given input $\overline{x}$ of length n and an integer k with $1 \leq k \leq n$. Then*

$$\mathbb{E}[S] = \begin{cases} 0 & \text{in the best case} \\ k(n-1)\binom{n}{k} & \text{in the worst and average case.} \end{cases}$$

If $k = 1$, that is, we are interested in obtaining the smallest element in the array, then we have $\mathbb{E}[S] = (n-1) \cdot n = n^2 - n$. This is not to bad for a bogo-algorithm. In case we are interested in the last element we come to the same polynomial of $\mathbb{E}[S] = n^2(n-1)$. On the other hand, since the binomial coefficients are uni-modal[2] the elements in the middle are the largest ones. Using the asymptotic estimate of $\binom{n}{k} \sim \frac{2^{n/2}}{\sqrt{\pi n}}$, if $k = \frac{n}{2} + \mathrm{O}(1)$ (taken from from [8]), we estimate that $\mathbb{E}[S] \sim \frac{n(n-1)\cdot 2^{n/2-1}}{\sqrt{\pi n}}$, if $k = \frac{n}{2} + \mathrm{O}(1)$. Therefore, computing the median induces an exponential number of swaps on average.

3 Two Variations on Bogo-Select

In this section we discuss two natural variants of the bogo-select algorithm by switching out the procedure PARTITIONED. This will influence the expected number of comparisons that are needed in order to verify that the kth element is in its correct position and also the expected number of swaps. We use the generic bogo-select algorithm and replace the probe algorithm in the second line by another appropriate verifier like, e.g., Z-PARTITIONED and P-COUNTED, which will be detailed in the forthcoming.

[2] A finite sequence of numbers is *uni-modal* if the sequence first increases and then decreases.

3.1 Bogo-Select by Zig-Zag-Partition

Checking for being well partitioned w.r.t. the kth element is done from left to right, which corresponds to draw cards from the top of the deck. When using bottom dealing or base dealing, which is by the way considered as cheating in poker, we would end up in a similar situation as with drawing cards from the top (aka. top dealing). Thus, it is left to consider when top and bottom dealing is alternately used for checking well partitioning, i.e., $a[1]$ is checked against $a[k]$, then $a[n]$ against $a[k]$, followed by $a[2]$ against $a[k]$, etc. If alternation between top and bottom dealing is not possible anymore, then one continues with top or bottom dealing only. This results in the bogo-select (by zig-zag-partition), where the implementation of the checker Z-PARTITIONED reads as follows – the pseudo code if k belongs to the second half of the array is straight forward and therefore left to the interested reader:

```
1  procedure z-partitioned (int: k):
2  // returns true if the array is
3  // zig-zag partitioned according
4  // to a[k] and false otherwise
5  if k<=(n+1)/2 then
6          // k in the first half
7          ...
8  else
9          // k in the second half
10         ...
11 endif
12 return true
```

```
7.1  // k in the first half
7.2  for i=1 to k-1 do
7.3          if a[i]>a[k] then
7.4                  return false
7.5          endif
7.6          if a[n+1-i]<a[k] then
7.7                  return false
7.8          endif
7.9  endfor
7.10 for i=k+1 to n-k+1 do
7.11         if a[i]<a[k] then
7.12                 return false
7.13         endif
7.14 endfor
```

The analysis of this algorithm is quite similar to before. Again, we encounter a combinatorial sum and series with inverse binomial coefficients. A more general approach to evaluate these sums and series is based on Euler's well known Beta functions defined by

$$B(m,n) = \int_0^1 t^{m-1}(1-t)^{n-1}\,\mathrm{d}t,$$

for all complex numbers m and n with a positive real part. Since

$$B(m,n) = \frac{\Gamma(m)\Gamma(n)}{\Gamma(m+n)} = \frac{(m-1)!(n-1)!}{(m+n-1)!},$$

where Γ refers to the Gamma function satisfying $\Gamma(n+1) = n!$, we get

$$\binom{n}{k}^{-1} = (n+1)\int_0^1 t^k(1-t)^{n-k}\,\mathrm{d}t$$

for all non-negative integers n and k with $n \geq k$. Although this is an explicit formula for the inverse binomial coefficient, it is somehow more convenient to replace inverse binomial coefficients by factorials and in turn by the Gamma and Beta function and its integral representation when dealing with combinatorial sums and series. This allows to simplify the inner terms of a combinatorial sum taking extra factors into account. We demonstrate this strategy in the following theorem, which is known as Lehmer's identity [5], and uses basics on integrals and (inverse) trigonometric functions; the proof is literally taken from [10] to keep this presentation self contained.

▶ **Theorem 5** (Lehmer's Identity). *If* $|x| < 1$, *then*

$$\sum_{i=1}^{\infty} \frac{(2x)^{2i}}{i} \binom{2i}{i}^{-1} = \frac{2x}{\sqrt{1-x^2}} \arcsin(x).$$

Proof. We replace the inverse binomial coefficient and the factor $1/i$ in the infinite sum by the Gamma function and in turn by the Beta function and its integral definition and obtain

$$\begin{aligned}\sum_{i=1}^{\infty} \frac{(2x)^{2i}}{i} \binom{2i}{i}^{-1} &= \sum_{i=1}^{\infty} (2x)^{2i} \cdot \frac{i!(i-1)!}{(2i)!} \\ &= \sum_{i=1}^{\infty} (2x)^{2i} \cdot \frac{\Gamma(i+1)\Gamma(i)}{\Gamma(2i+1)} \\ &= \sum_{i=1}^{\infty} (2x)^{2i} B(i+1,i) = \sum_{i=1}^{\infty} (2x)^{2i} \int_0^1 t^i (1-t)^{i-1} \,\mathrm{d}t.\end{aligned}$$

After exchanging the sum and the integral and evaluating the geometric sum we find

$$\begin{aligned}\sum_{i=1}^{\infty} (2x)^{2i} \int_0^1 t^i (1-t)^{i-1} \,\mathrm{d}t &= \int_0^1 \sum_{i=1}^{\infty} (2x)^{2i} t^i (1-t)^{i-1} \,\mathrm{d}t \\ &= \int_0^1 \frac{1}{1-t} \sum_{i=1}^{\infty} (4x^2 t(1-t))^i \,\mathrm{d}t \\ &= \int_0^1 \frac{4x^2 t}{1-4x^2 t(1-t)} \,\mathrm{d}t.\end{aligned}$$

By considering the derivation of denominator we decide to use the substitution $s = x(2t-1)$ and hence $\frac{\mathrm{d}s}{\mathrm{d}t} = 2x$. Thus the above integral is further equal to

$$\int_{-x}^{x} \frac{s}{s^2 + (1-x^2)} \,\mathrm{d}s + x \int_{-x}^{x} \frac{1}{s^2 + (1-x^2)} \,\mathrm{d}s. \tag{1}$$

Since

$$\int \frac{1}{x} \,\mathrm{d}x = \ln|x|, \quad \text{and} \quad \int \frac{1}{x^2 + a^2} \,\mathrm{d}x = \frac{1}{a} \arctan\left(\frac{x}{a}\right)$$

as found in the theoretical computer science cheat sheet[3] one observes that the former term

[3] The theoretical computer science cheat sheet can be downloaded from, for instance, `https://tug.org/texshowcase/cheat.pdf`.

of Equation (1) evaluates to zero,[4] while the latter term of Equation 1 is equal to

$$x \int_{-x}^{x} \frac{1}{s^2 + (1-x^2)}\,\mathrm{d}s = x\left(\frac{1}{\sqrt{1-x^2}}\arctan\left(\frac{s}{\sqrt{1-x^2}}\right)\Bigg|_{-x}^{x}\right)$$
$$= \frac{2x}{\sqrt{1-x^2}}\arctan\left(\frac{x}{\sqrt{1-x^2}}\right),$$

because $\arctan(-x) = \arctan(x)$, for $|x| < 1$. Thus, we get

$$\sum_{i=1}^{\infty} \frac{(2x)^{2i}}{i}\binom{2i}{i}^{-1} = \frac{2x}{\sqrt{1-x^2}}\arctan\left(\frac{x}{\sqrt{1-x^2}}\right) = \frac{2x}{\sqrt{1-x^2}}\arcsin(x),$$

since the inverse trigonometry functions arcsin and arctan obey

$$\arcsin(x) = \arctan\left(\frac{x}{\sqrt{1-x^2}}\right) \quad \text{for } |x| < 1$$

as mentioned in the theoretical computer science cheat sheet. This proves the stated claim. ◀

It is worth mentioning that Theorem 1 can be shown in similar way as above by using the Beta function and its integral definition.

Alternative Proof of Theorem 1. We show a slightly stronger statement

$$\sum_{i=0}^{m}\binom{n+i}{i}^{-1} = \frac{n}{n-1}\left[1 - \binom{m+n}{n+1}^{-1}\right],$$

for $n \geq 2$ and $m \geq 0$. If m tends to infinity, we obtain the statement of Theorem 1, because the inverse binomial coefficient approaches 0.

We proceed as in the proof of Theorem 5. Thus, we replace the inverse binomial coefficient by its definition, then the factorials by the Gamma function and in turn by the Beta function and its integral definition. This results in the following calculation:

$$\sum_{i=0}^{m}\binom{n+i}{i}^{-1} = \sum_{i=0}^{m}\frac{i!n!}{(n+i)!}$$
$$= n\sum_{i=0}^{m}\frac{\Gamma(i+1)\Gamma(n)}{\Gamma(n+i+1)}$$
$$= n\sum_{i=0}^{m} B(i+1,n) = n\sum_{i=0}^{m}\int_0^1 t^i(1-t)^{n-1}\,\mathrm{d}t.$$

[4] A simpler argument that

$$\int_{-x}^{x} \frac{s}{s^2 + (1-x^2)}\,\mathrm{d}s = 0$$

is that the function $f(s) = s/(s^2 + (1-x^2))$ is symmetric to the origin, and thus the integral from $-x$ to x evaluates to zero.

Next we exchange the sum and the integral and evaluate the geometric sum. This leads us to

$$n\sum_{i=0}^{m}\int_0^1 t^i(1-t)^{n-1}\,\mathrm{d}t = n\int_0^1\sum_{i=0}^{m} t^i(1-t)^{n-1}\,\mathrm{d}t$$
$$= n\int_0^1 (1-t)^{n-1}\sum_{i=0}^{m} t^i\,\mathrm{d}t = n\int_0^1 (1-t)^{n-1}\frac{1-t^{m+1}}{1-t}\,\mathrm{d}t.$$

To keep the calculation simple we rewrite the above integral by its difference and obtain

$$n\int_0^1 (1-t)^{n-1}\frac{1-t^{m+1}}{1-t}\,\mathrm{d}t = n\int_0^1 (1-t)^{n-2}(1-t^{m+1})\,\mathrm{d}t$$
$$= n\int_0^1 (1-t)^{n-2}\,\mathrm{d}t - n\int_0^1 t^{m+1}(1-t)^{n-2}\,\mathrm{d}t.$$

Since $n \geq 2$ we find that the former term of the above given difference is equal to

$$n\int_0^1 (1-t)^{n-2}\,\mathrm{d}t = n\left(\frac{-1}{n-1}(1-t)^{n-1}\Big|_0^1\right)$$
$$= n\left(\frac{-1}{n-1}(1-1)^{n-1} - \frac{-1}{n-1}(1-0)^{n-1}\right) = \frac{n}{n-1},$$

while the latter term can be rewritten by the Beta function and in turn by the Gamma function or by factorials, which reads as

$$n\int_0^1 t^{m+1}(1-t)^{n-2}\,\mathrm{d}t = n\cdot B(m+2,n-1)$$
$$= n\cdot\frac{\Gamma(m+2)\Gamma(n-1)}{\Gamma(m+n+1)}$$
$$= n\cdot\frac{(m+1)!(n-2)!}{(m+n)!} = \frac{n}{n-1}\binom{m+n}{m+1}^{-1}$$

which gives the desired result by putting both terms together and factor $n/(n-1)$ out. ◀

We are interested in special values on inverse central binomial coefficients (with some particular factors). Both values are well known, e.g., see [9]. The value for the first combinatorial sum is obtained by evaluating Lehmer's identify for $x = 1/2$ taking into account that $\arcsin(1/2) = \pi/6$. In order to obtain the value for second combinatorial sum we differentiate Lehmer's identity, multiply it with x, integrate it, and evaluate it again at $x = 1/2$.

▶ **Theorem 6.** *It holds*

$$\sum_{i=1}^{\infty}\frac{1}{i}\binom{2i}{i}^{-1} = \frac{\pi\sqrt{3}}{9} \quad \textit{and} \quad \sum_{i=0}^{\infty}\frac{1}{2i+1}\binom{2i}{i}^{-1} = \frac{2\pi\sqrt{3}}{9}.$$

Proof. The value for the first combinatorial sum is obtained by evaluating Lehmer's identify for $x = 1/2$ taking into account that $\arcsin(1/2) = \pi/6$. Thus we have

$$\sum_{i=1}^{\infty} \frac{1}{i}\binom{2i}{i} = \frac{2}{\sqrt{3}}\frac{\pi}{6} = \frac{\pi\sqrt{3}}{9}.$$

In order to obtain the value for other combinatorial sum we start with the differentiation of Lehmer's identity. Differentiation of the polynomial on the left hand-side of Lehmer's identity is straight forward and gives

$$\frac{\mathrm{d}}{\mathrm{d}x}\sum_{i=1}^{\infty} \frac{(2x)^{2i}}{i}\binom{2i}{i}^{-1} = \sum_{i=1}^{\infty} 4(2x)^{2i-1}\binom{2i}{i}^{-1}. \tag{2}$$

On the other hand, the differentiation of the right hand-side involves the product of a quotient with an arcsin. We find

$$\frac{\mathrm{d}}{\mathrm{d}x}\frac{2x}{\sqrt{1-x^2}}\arcsin(x) = \frac{2\arcsin(x)}{\sqrt{1-x^2}} + \frac{2x^2\arcsin(x)}{(1-x^2)^{3/2}} + \frac{2x}{1-x^2}, \tag{3}$$

where we used

$$\frac{\mathrm{d}}{\mathrm{d}x}\arcsin(x) = \frac{1}{\sqrt{1-x^2}}, \quad \text{for } |x| < 1.$$

Equation (3) can be easily verified with a symbolic computation software at hand. Hence, plugging together (2) and (3) we have

$$\sum_{i=1}^{\infty} 4(2x)^{2i-1}\binom{2i}{i}^{-1} = \frac{2\arcsin(x)}{\sqrt{1-x^2}} + \frac{2x^2\arcsin(x)}{(1-x^2)^{3/2}} + \frac{2x}{1-x^2}, \tag{4}$$

Then value for the second combinatorial sum $\sum_{i=1}^{\infty}\frac{1}{2i+1}\binom{2i}{i}^{-1}$ is obtained by multiplying Equation (4) by x and integrating it. For the left hand-side of Equation (4) we thus get

$$\begin{aligned}\int \sum_{i=1}^{\infty} 2(2x)^{2i}\binom{2i}{i}^{-1}\mathrm{d}x &= \sum_{i=1}^{\infty}\int 2(2x)^{2i}\binom{2i}{i}^{-1}\mathrm{d}x \\ &= \sum_{i=1}^{\infty}\frac{2^{2i+1}x^{2i+1}}{2i+1}\binom{2i}{i}^{-1} = \sum_{i=1}^{\infty}\frac{(2x)^{2i+1}}{2i+1}\binom{2i}{i}^{-1}.\end{aligned} \tag{5}$$

The integration of the right hand-side of Equation (4) is quite tedious for it is done by parts. Hence we use a symbolic manipulation software, in order to minimize our work effort. We obtain

$$\begin{aligned}&\int x\left(\frac{2\arcsin(x)}{\sqrt{1-x^2}} + \frac{2x^2\arcsin(x)}{(1-x^2)^{3/2}} + \frac{2x}{1-x^2}\right)\mathrm{d}x \\ &\qquad = \frac{2\arcsin(x)}{\sqrt{1-x^2}} - 2\ln\left(\left|\frac{1}{\sqrt{1-x^2}} + \frac{x}{\sqrt{1-x^2}}\right|\right) - 2x - \ln(|x-1|) + \ln(|x+1|).\end{aligned} \tag{6}$$

Thus, by (5) and (6) we have

$$\begin{aligned}&\sum_{i=1}^{\infty}\frac{(2x)^{2i+1}}{2i+1}\binom{2i}{i}^{-1} \\ &\qquad = \frac{2\arcsin(x)}{\sqrt{1-x^2}} - 2\ln\left(\left|\frac{1}{\sqrt{1-x^2}} + \frac{x}{\sqrt{1-x^2}}\right|\right) - 2x - \ln(|x-1|) + \ln(|x+1|),\end{aligned}$$

which evaluated at $x = 1/2$ results in

$$\sum_{i=1}^{\infty} \frac{1}{2i+1}\binom{2i}{i}^{-1} = \frac{4\pi}{6\sqrt{3}} - 2\ln\left(\frac{3}{\sqrt{3}}\right) - 1 - \ln\left(\frac{1}{2}\right) + \ln\left(\frac{3}{2}\right)$$
$$= \frac{2\pi\sqrt{3}}{9} - \ln(3) + \ln(2) + \ln\left(\frac{3}{2}\right) - 1 = \frac{2\pi\sqrt{3}}{9} - 1.$$

At last we shift the sum down to $i = 0$ as we need this form later on:

$$\sum_{i=0}^{\infty} \frac{1}{2i+1}\binom{2i}{i}^{-1} = \frac{1}{2\cdot 0+1}\binom{2\cdot 0}{0}^{-1} + \sum_{i=1}^{\infty} \frac{1}{2i+1}\binom{2i}{i}^{-1} = 1 + \frac{2\pi\sqrt{3}}{9} - 1 = \frac{2\pi\sqrt{3}}{9}$$

This completes the proof. ◀

Now we are ready to determine the expected number of comparisons within the procedure Z-PARTITIONED.

▶ **Theorem 7.** *Assume $\overline{x}$ is a random permutation of $\{1, 2, \ldots, n\}$, and let C denote the random variable counting the number of comparisons carried out by the procedure* Z-PARTITIONED *in the test whether $\overline{x}$ is partitioned according to the kth element, for $1 \le k \le (n+1)/2$, then:*

$$\mathbb{E}[C] = \sum_{j=1}^{k-1}\left[\frac{1}{2j-1}\binom{2(j-1)}{j-1}^{-1} + \frac{1}{j}\binom{2j}{j}^{-1}\right] + \frac{1}{k}\sum_{i=2k-1}^{n-1}\binom{i}{k}^{-1}$$

It holds $\mathbb{E}[C] = H_{n-1}$, if $k = 1$, and $\mathbb{E}[C] \sim \frac{\pi\sqrt{3}}{3} + \frac{1}{k-1}\binom{2(k-1)}{k-1}^{-1}$, otherwise. The statement remains valid in case k satisfies $(n+1)/2 < k \le n$ by replacing k by $n-k+1$.

Proof. Let k with $1 \le k \le n$ be fixed. We analyse the procedure Z-PARTITIONED only for the case that k lies in the first half of the array, i.e., $k \le (n+1)/2$. It is easy to see that if k is in the second half of the array the case is symmetric to the above one by replacing k by $n-k+1$ in the formulas.

Recall that the ith comparison is reached if and only if the algorithm did not drop out at the $i-1$ comparisons before. We therefore consider two cases regarding the number of successful comparisons counting i – the probabilities are modelled similar to the proof of Theorem 2.

1. Let $1 \le i \le 2k-2$, i.e., the algorithm has yet to leave the zig-zag-comparison. Then we distinguish two subcases, namely whether i is odd or even. In the former case we have completed $j := (i-1)/2$ pairs of comparisons with $0 \le j \le k-2$, which yields the probability

   $$\mathbb{P}[C \ge i] = \frac{\binom{n}{i} j!j!(n-i)!}{n!} = \frac{1}{2j+1}\binom{2j}{j}^{-1}.$$

   In the latter case the $(i-1)$th comparison is carried out at the beginning of the array, and we find

   $$\mathbb{P}[C \ge i] = \frac{\binom{n}{i} j!(j-1)!(n-i)!}{n!} = \frac{1}{j}\binom{2j}{j}^{-1},$$

   where $j := i/2$ with $1 \le j \le k-1$.

2. Let $2k-1 \leq i < n-1$, i.e., the algorithm is in the second for-loop and therefore outside the zig-zag-comparison. Then we have

$$\mathbb{P}[C \geq i] = \frac{\binom{n}{i}(k-1)!(i-1-(k-1))!(n-i)!}{n!} = \frac{1}{k}\binom{i}{k}^{-1}.$$

We obtain the expected value of C by summing up the probabilities in every case, since the range of C is non-negative, and we get

$$\begin{aligned}\mathbb{E}[C] = \sum_{i\geq 1}\mathbb{P}[C \geq i] &= \sum_{j=0}^{k-2}\frac{1}{2j+1}\binom{2j}{j}^{-1} + \sum_{j=1}^{k-1}\frac{1}{j}\binom{2j}{j}^{-1} + \sum_{i=2k-1}^{n-1}\frac{1}{k}\binom{i}{k}^{-1} \\ &= \sum_{j=1}^{k-1}\left[\frac{1}{2j-1}\binom{2(j-1)}{j-1}^{-1} + \frac{1}{j}\binom{2j}{j}^{-1}\right] + \frac{1}{k}\sum_{i=2k-1}^{n-1}\binom{i}{k}^{-1}.\end{aligned} \tag{7}$$

Each of the three sums in Equation (7) can be bounded from above. For the first two sums we use the estimates from Theorem 6, while for the last one we need a more sophisticated estimate than the one encountered within the proof of Theorem 2, because the sum is truncated at the beginning. Consider

$$\sum_{i=2k-1}^{n-1}\binom{i}{k}^{-1} = \sum_{i=k-1}^{n-k-1}\binom{i+k}{k}^{-1} = \sum_{i=0}^{n-k-1}\binom{i+k}{k}^{-1} - \sum_{i=0}^{k-2}\binom{i+k}{k}^{-1},$$

which by the equation given in the footnote on page 5 is equal to

$$\frac{k}{k-1}\left[1-\binom{n-1}{n-k}^{-1}\right] - \frac{k}{k-1}\left[1-\binom{2k-2}{k-1}^{-1}\right] = \frac{k}{k-1}\left[\binom{2(k-1)}{k-1}^{-1} - \binom{n-1}{n-k}^{-1}\right]$$

and in the limit is $\frac{k}{k-1}\binom{2(k-1)}{k-1}^{-1}$ as n tends to infinity. Therefore $\mathbb{E}[C]$ is at most

$$\frac{2\pi\sqrt{3}}{9} + \frac{\pi\sqrt{3}}{9} + \frac{1}{k}\cdot\frac{k}{k-1}\binom{2(k-1)}{k-1}^{-1} = \frac{\pi\sqrt{3}}{3} + \frac{1}{k-1}\binom{2(k-1)}{k-1}^{-1},$$

for $k \geq 2$. In case $k = 1$ the former two combinatorial sums are zero and the remaining terms sum up to $\sum_{i=1}^{n-1}\frac{1}{i}$, which again equals the harmonic number of H_{n-1}. This completes our proof. ◀

Wasn't that awesome? Theorem 7 tells us that we need only a constant number of comparisons on the average to check if a large array is well partitioned w.r.t. k, if $k \geq 2$, and for n large enough, and this number is about

$$\frac{\pi\sqrt{3}}{3} \approx 1.81379936423.$$

Compare this to Theorem 2 and to to the worst case, where we have to compare $n-1$ times.

When considering the expected number of swaps, we find an identical result as in the case of the original bogo-select (by partition) algorithm, which we state without proof.

▶ **Theorem 8.** *If S denotes the total number of swaps carried out by the bogo-select (by zig-zag partition) for an input $\overline{x}$ of length n, when asking for the kth element, for $1 \leq k \leq n$, we have*

$$\mathbb{E}[S] = \begin{cases} 0 & \text{if } \overline{x} \text{ is well partitioned according to the } k\text{th element} \\ k(n-1)\binom{n}{k} & \text{otherwise.} \end{cases}$$

3.2 Bogo-Selection by Counting

As already mentioned in the introduction we also consider a variant of bogo-select relaxing the condition of being partitioned to a condition that only requires that there are exactly $k-1$ elements in $a[1 \dots k-1, k+1, \dots n]$ smaller or equal to $a[k]$; equivalently one requires that exactly $n-k$ elements in the array $a[1 \dots k-1, k+1 \dots n]$ are larger or equal to $a[k]$. This slight change of the condition alters the running time of the algorithm significantly. For instance, in case $n = 7$ and $k = 3$ the original bogo-select and its newly introduced variant would terminate on input $\overline{x} = 1\,2\,3\,4\,5\,6\,7$: the values 1 and 2 are smaller or equal than 3 while the remaining values are strictly larger and so the input is well partitioned according to the 3rd element of $\overline{x}$. On the other hand, on input $\overline{x} = 5\,6\,3\,1\,2\,4\,7$ the original bogo-select algorithm starts with randomly permuting the input, because in this case $\overline{x}$ is not well partitioned w.r.t. the 3rd element. For the newly introduced variant this second input $\overline{x} = 5\,6\,3\,1\,2\,4\,7$ leads to immediate termination in contrast. The new variant of bogo-select (now by counting) gives rise to a new procedure named P-COUNTED that requires an additional counter c initially set to 0 in order to collect the number of elements that are smaller or equal to $a[k]$. Observe, that the value of c is only compared to k in the second for-loop (and once after the for-loop is finished). The procedure P-COUNTED reads as follows:

```
1  procedure p-counted (int: k):
2  // returns true if exactly k-1 elements
3  // of a[1..k-1,k+1..n] are smaller or
4  // equal than a[k] and false otherwise
5  c := 0
6  for i=1 to k-1 do
7          if a[i]<=a[k] then
8                  c := c+1
9          endif
10 endfor
11 for i=k+1 to n do
12         if a[i]<=a[k] then
13                 c := c+1
14                 if c>=k then
15                         return false
16                 endif
17         endif
18 endfor
19 if c<k-1 then
20         return false
21 endif
22 return true
```

Initially, enumerating the number of comparisons of the bogo-select (by counting) algorithm looks more involved, since one has also to take the counter values into consideration. In fact, the analysis is surprisingly easy.

▶ **Theorem 9.** *Assume $\overline{x}$ is a random permutation of $\{1, 2, \dots, n\}$, and let C denote the random variable counting the number of comparisons of array elements carried out by the procedure* P-COUNTED *in the test whether $\overline{x}$ is partitioned according to the kth element. Then*

$$\mathbb{E}[C] = \begin{cases} H_{n-1} & \text{if } k = 1, \\ (k-1) + k(H_{n-1} - H_{k-1}) & \text{otherwise.} \end{cases}$$

Proof. Let k with $1 \leq k \leq n$ be fixed. We proceed as in the previous two proofs. Therefore we again consider two cases:

1. Let $1 \leq i < k$. Then by the careful design of the algorithm we have

$$\mathbb{P}[C \geq i] = 1,$$

because in the first for-loop the procedure P-COUNTED in no case terminates. Furthermore the first comparison within the second loop is always carried out totalling to a minimum of k comparisons.

2. Next assume $k \leq i < n$. Up until the ith comparison the value of the counter variable c ranges from 0 to at most $k-1$, since otherwise the procedure P-COUNTED would have already returned *false*. Thus, in order to determine the value of $\mathbb{P}[C \geq i]$ we first consider the following example: let $n = 10$ and $k = 4$. Assume we are about to reach $i = 7$ comparisons with counter c set to 3. Then we have hit i elements out of n, that is, for instance, $\{1, 3, 4, 7, 8, 9, 10\}$. Now c indicates that exactly three elements from that sublist are already smaller than our given element x_k, thus yielding us 7 to be x_k – in general x_k then equals the $(c+1)$st element. The remaining $i-1$ elements from the selected set can be freely permuted as well as the rest with a size of $n-i$. A last ith comparison is then carried out that is allowed to also fail now thus setting no further restrictions on the input. From this consideration we can derive in general

$$\mathbb{P}[C \geq i] = k \cdot \frac{\binom{n}{i}(i-1)!(n-i)!}{n!} = \frac{k}{i}.$$

Summing up gives

$$\mathbb{E}[C] = \sum_{i>0} \mathbb{P}[C \geq i] = \sum_{i=1}^{k-1} 1 + \sum_{i=k}^{n-1} \frac{k}{i} = (k-1) + k \sum_{i=k}^{n-1} \frac{1}{i}.$$

The remaining sum is then equal to $H_{n-1} - H_{k-1}$, yielding

$$\mathbb{E}[C] = (k-1) + k(H_{n-1} - H_{k-1}),$$

if $k \geq 2$. In case $k = 1$ the former sum disappears and we end up with $\sum_{i=1}^{n-1} \frac{1}{i}$, which is the $(n-1)$th harmonic number H_{n-1}. This proves the claimed bounds. ◀

A more detailed analysis of the procedure P-COUNTED also taking into account the comparisons of the counter c at lines 14 and 19 is given next. While the upper theorem offers a complete analysis of P-COUNTED regarding the comparison of array elements it ignores that counter c is also monitored *via* a comparison (in line 14). As it turns out this does not factor into the asymptotics in a meaningful way. Going back to the procedure P-COUNTED we can see that the only repeated comparison not covered by the analysis above happens in line 14 where the counter c is checked for overflow. This very comparison is only carried out when c itself has been incremented one line earlier, i.e., the comparison of array elements within the second loop was successful. We can therefore deduce that the number of comparisons within the second loop is at maximum double the comparisons we counted in Theorem 9. Furthermore a last comparison can happen afterwards summing up to an upper bound. Thus, we have

$$\mathbb{E}[C] \leq \mathbb{E}[C_{\text{tot}}] \leq \mathbb{E}[C] + \begin{cases} H_{n-1} + 1 & \text{if } k = 1 \\ k(H_{n-1} - H_{k-1}) + 1 & \text{otherwise,} \end{cases}$$

where C_{tot} denotes the random variable counting the *total* number of comparisons (lines 12, 14, and 19) carried out by the procedure P-COUNTED in the test whether $\overline{x}$ is partitioned according to the kth element. The next theorem gives a more precise estimate on $\mathbb{E}[C_{\text{tot}}]$, by a more detailed analysis of the P-COUNTED procedure.

▶ **Theorem 10.** *Assume $\overline{x}$ is a random permutation of $\{1, 2, \ldots, n\}$, and let C_{tot} denote the random variable counting the* total *number of comparisons (lines 12, 14, and 19) carried out by the procedure* P-COUNTED *in the test whether $\overline{x}$ is partitioned according to the kth element. Then*

$$\mathbb{E}[C_{tot}] = \mathbb{E}[C] + \frac{k+1}{2} - \frac{k(k-1)}{2n}.$$

It holds $\mathbb{E}[C_{tot}] \sim \mathbb{E}[C] + \frac{k+1}{2}$.

Proof. Let R denote the random variable counting the raises of counter c within the procedure P-COUNTED. We calculate the expected value using the ansatz $\mathbb{E}[R] = \sum_{i \geq 0} i \cdot \mathbb{P}[R = i]$. Since we drop out for $c = k$ we can deduce $0 \leq i \leq k$. We consider two cases:

1. When c is raised at most $0 \leq i \leq k-1$ times there are a total of i smaller elements within the array and therefore x_k has to be the $(i+1)$st element which we hit with a probability of
$$\mathbb{P}[R = i] = \frac{1}{n}.$$
2. Otherwise $c = k$, which can be modeled using the complementary probability to the case above:
$$\mathbb{P}[R = k] = 1 - \sum_{i=0}^{k-1} \frac{1}{n} = 1 - \frac{k}{n}$$

Summing it up we get

$$\begin{aligned}\mathbb{E}[R] = \sum_{i \geq 0} i \cdot \mathbb{P}[R = i] &= \sum_{i=0}^{k-1} i\frac{1}{n} + k\left(1 - \frac{k}{n}\right) = \frac{1}{n}\sum_{i=1}^{k-1} i + k - \frac{k^2}{n} \\ &= \frac{1}{n}\left(\frac{(k-1)k}{2}\right) + k - \frac{k^2}{n} = \frac{k^2 - k + 2kn - 2k^2}{2n} = \frac{2kn - k - k^2}{2n}.\end{aligned}$$

Up next let the random variable R_1 count the raises of counter c only within the first loop. Going over the first k elements we can at most find $k-1$ smaller elements. Each possible counter value fixes us a specific element again, allowing us to calculate the expected value as follows:

$$\mathbb{E}[R_1] = \sum_{i \geq 0} i \cdot \mathbb{P}[R = i] = \sum_{i=0}^{k-1} i\frac{1}{k} = \frac{1}{k}\sum_{i=1}^{k-1} i = \frac{1}{k}\left(\frac{(k-1)k}{2}\right) = \frac{k-1}{2}$$

Lastly let random variable F indicate whether the final comparison in line 19 is carried out. We reach this comparison when we have not dropped out before, that is if and only if there were a maximum of $k-1$ smaller elements to x_k. In other words x_k is one of the k smallest elements within the array yielding us an expected value of

$$\mathbb{E}[F] = \sum_{i \geq 0} i \cdot \mathbb{P}[F = i] = 0 \cdot \mathbb{P}[F = 0] + 1 \cdot \mathbb{P}[F = 1] = \frac{k}{n}.$$

Now observe that every comparison within procedure P-COUNTED that was not covered by random variable C of the theorem 9 happens either when the counter is raised within the second loop or in line 19 after the complete array has been covered. So the random variable C_{tot} counting the total number of comparison within procedure P-COUNTED is given by

$$C_{\text{tot}} = C + (R - R_1) + F.$$

By linearity of the expected value we can calculate

$$\mathbb{E}[C_{\text{tot}}] = \mathbb{E}[C] + \mathbb{E}[R] - \mathbb{E}[R_1] + \mathbb{E}[F],$$

where

$$\mathbb{E}[R] - \mathbb{E}[R_1] + \mathbb{E}[F] = \frac{2kn - k - k^2}{2n} - \frac{k-1}{2} + \frac{k}{n} = \frac{kn + k - k^2 + n}{2n}$$

and therefore

$$\mathbb{E}[C_{\text{tot}}] = \mathbb{E}[C] + \frac{kn + k - k^2 + n}{2n} \qquad = \mathbb{E}[C] + \frac{k+1}{2} - \frac{k(k-1)}{2n},$$

yielding the stated result. Thus, asymptotically $\mathbb{E}[C_{\text{tot}}] \sim \mathbb{E}[C] + \frac{k+1}{2}$, as n tends to infinity. ◀

This completes the more detail analysis of the procedure P-COUNTED.

Finally, for the total number of swaps of the bogo-select (by counting) algorithm we obtain the following result. A permutation (or array) $\overline{x}$ satisfying

$$|\{\, i \mid x_i \leq x_k, \text{ for } 1 \leq i \leq n \text{ and } i \neq k \,\}| = k - 1$$

is said to be *p-counted w.r.t. the kth element.*

▶ **Theorem 11.** *If S denotes the total number of swaps carried out by bogo-select by counting for an input $\overline{x}$ of length n, when asking for the kth element, for $1 \leq k \leq n$, we have*

$$\mathbb{E}[S] = \begin{cases} 0 & \text{if } \overline{x} \text{ is p-counted w.r.t the } k\text{th element} \\ n(n-1) & \text{otherwise.} \end{cases}$$

Proof. We argue as in the case of the bogo-select algorithm. The hitting probability for a permutation to be p-counted by the kth element is $\frac{(n-1)!}{n!}$ in each trial; the kth element is fixed and the remaining $n-1$ elements can be arbitrarily arranged. The number of iterations I is a geometrically distributed variable with hitting complexity $p = \frac{1}{n}$ and $\mathbb{E}[I] = p^{-1} = n$. Since each iteration costs $n-1$ swaps the expected number of swaps counted by the random variable S is equal to $\mathbb{E}[S] = n(n-1)$, if the input is not p-counted w.r.t. the kth element of the array, and $\mathbb{E}[S] = 0$, otherwise. ◀

4 Experimental Results

We have implemented the considered algorithms in LUA[5] and have performed some experiments. The source code as well as the test scripts are available on request by email to one of

[5] LUA is a powerful, efficient, lightweight, embeddable scripting language that can be downloaded form `https://www.lua.org` as a small package that builds out-of-the-box on all platforms that have a standard C compiler.

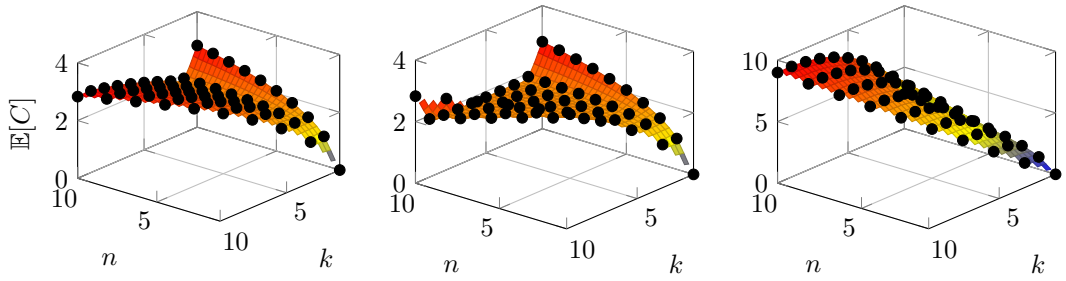

Figure 1 The expected number of comparisons carried out by the three considered probe procedures (i) PARTITIONED, (ii) Z-PARTITIONED, and (iii) P-COUNTED – diagrams are from left to right. Observe, that the vertical axis is scaled differently on the rightmost diagram.

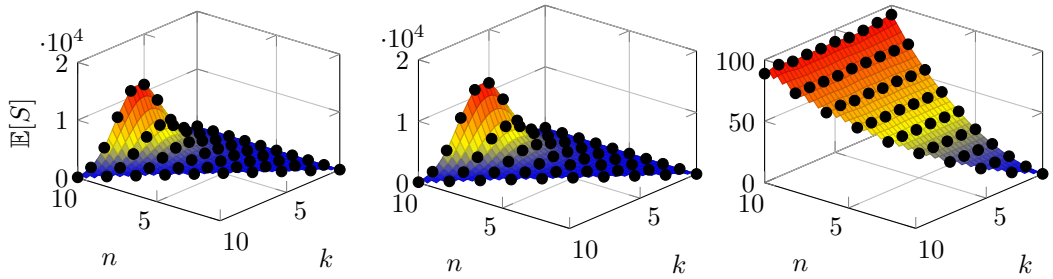

Figure 2 The expected number of swaps carried out by the three considered randomized selection algorithms bogo-select (i) by partition, (ii) by zig-zag-partition, and (iii) by counting – diagrams are from left to right. Observe, that the vertical axis is differently scaled on the rightmost diagram.

the authors. The experiments were conducted on an iMac (mid 2011 version) with an Intel Core i5 processor (2.7 GHz) and 4GB main memory. It took quite some time to collect our results, but this was no problem, since we started in early January with our experiments. We first implemented every investigated variant of the bogo-select as-is in order to observe the performance on the given machine depending on different input sizes.

We ran all three probe procedures on every possible permutation of input array $a[1 \dots n]$ up to length $n = 10$ in order to track the arithmetic mean of the number of comparisons. As expected our experimental values for every valid combination of n, k exactly match the expected value calculated earlier. The results on the number of comparisons are shown in Figure 1. Since the number of possible inputs grows superexponentially, exact values for higher input sizes become quite costly and would necessitate further experiments.

It is important to note that our calculations for the number of swaps are based on the premise that the input does not comply to the property at hand initially. Furthermore neither the number of swaps nor the state of the input from iteration 2 onward in any way depend on the original input – as we already stated in the introduction. We therefore take an array that meets the given property – i.e., a fully ordered one – and swap entries x_k, x_{k+1} yielding us an input as required. *Via* the insertion of a counter variable we tracked the swaps carried out in 10.000 runs for every valid combination of n, k up to $n = 10$. The experimental results depicted in Figure 2 nicely line up with the theoretical values. A more detailed statistical analysis for the number of swaps is given next. When we consider the swaps the analysis is not that simple for we counted them over $r = 10000$ randomized runs of each algorithm. First up we use their arithmetic mean $\overline{S}$ as an unbiased estimate for the expected value due to the number of runs r being sufficiently large – as stated earlier the values lined

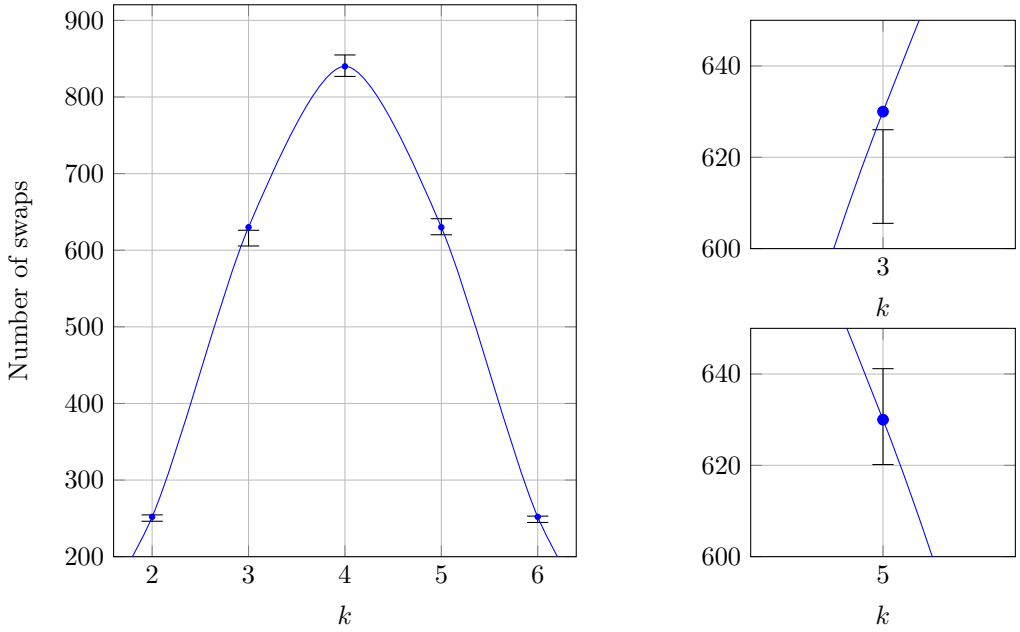

Figure 3 Confidence intervals for the number of swaps performed in bogo-select (by zig-zag-partition) with $n = 7$ and $k \in \{2, 3, 4, 5, 6\}$. The expected number of swaps is drawn blue.

up reasonably well with the theoretical results. We also calculated the sample standard deviation via $\sigma = \sqrt{\frac{1}{r-1}\sum_{i=1}^{r}(S_i - \overline{S})^2}$ to quantify how far the experimental results are dispersed in relation to the arithmetic mean. Having this information we now perform some statistical analysis upon the experimentally produced data sets – we do not factor in our theoretical results to reach a second (and independent) conclusion.

The generally known hypothesis test does not offer further insight here since it is only significant when it comes to rejecting an hypothesis while our objective is to verify the theoretical results as calculated earlier. Based on our experimental data it is good practise to calculate a confidence interval for the expected value of the number of swaps $\mathbb{E}[S]$ instead. It is a statistical technique that gives an interval covering the underlying expected value of a given probe with a fixed probability α. In our case we make no preconception about the distribution at hand and have a probe size r sufficiently large. Therefore we can calculate the confidence interval as $[\overline{S} \pm t_{\infty,1-\alpha/2}\frac{\sigma}{\sqrt{r}}]$ with $t_{\infty,1-\alpha/2}$ being the $1 - \alpha/2$ quantile of the t-distribution with unlimited degrees of freedom.

With this method we are now able to check whether our theoretically calculated expected value falls into the range that covers the expected value within the experiments (with probability $\alpha = 95\%$). E.g. looking into the results for bogo-select (by zig-zag-partition), we can see that for $n = 7$ and $k = 5$ we have an expected value of $\mathbb{E}[S] = 630$ that comfortably sits in the confidence interval of $I \approx 630.68 \pm 10.5$ we have generated experimentally. On the other hand for $n = 7$ and $k = 3$ the expected value of $\mathbb{E}[S] = 630$ misses the confidence interval $I \approx 615.79 \pm 10.253$ by a noticeable margin. Still the overall fit for $n = 7$ is quite good as we can see in Figure 3. Deviations like the above are to be expected since the confidence interval itself depends upon the randomness inherent to the experiments. In order to further quantify how often the individual confidence intervals match with our theoretical results we

Table 1 We cumulated the ratio whether the confidence intervals cover the theoretical values (Conf) for all three bogo-select variants as well as the maximum deviation between arithmetic mean and theoretical value relative to the latter (maxDev $= \max\left(\frac{|\mathbb{E}[S]-\overline{S}|}{\mathbb{E}[S]}\right)$).

Bogo-select by...	Conf	maxDev
partitioning	$81.\overline{81}\%$	2.507619%
z-partitioning	$89.\overline{09}\%$	2.255524%
p-counting	$94.\overline{54}\%$	2.73%

analyse the generated data sets with the powerful statistical programming language R.[6] First we calculated the percentage for said matches regarding every bogo-select variant that was analysed within this paper. As you can see (in the first column of the Table 1) the coverage is quite high. Please note that this ratio is not directly related to the significance of the confidence intervals. The given level only ensures that a specific interval under identical conditions covers the real value in 95% of all cases. In our experiments on the other hand we only calculated a single confidence interval for every valid pair of n and k. Additionally we are interested how much the experimental value differs from the theoretical results in general. For this reason we also included the biggest distance between the two (relatively to the theoretical expected value) in Table 1.

As one can see we stay well below a 3% discrepancy for every algorithm. In summary it is safe to say that our experiments also support our theoretical analysis regarding the number of swaps.

5 Conclusions

We continued our research on the still unexplored research field of pessimal algorithm design with a theoretical and experimental study of bogo-select and variants thereof, which are archetypical perversely awful algorithms. Remarkably, the expected running time of these algorithms in terms of the number of swaps and comparisons can be determined exactly using only elementary methods in probability and combinatorics. Though optimizing the running time seems somewhat out of place in the field of *pessimal* algorithm design, it can be quite revealing for beginners in both fields of optimal and pessimal algorithm design to see how a single optimization step can yield a dramatic speed-up. The very first obvious optimization step in all aforementioned algorithms is to swap two elements only if this makes sense. That is, before swapping a pair, we check if it is an inversion: a pair of positions (i, j) in the array $a[1 \dots n]$ is an *inversion* if $i < j$ and $a[i] > a[j]$. This leads to an optimized variant of bogo-select (by partitioning or zig-zag partition), which we refer to as bogo-select$_{opt}$. As there are at most $\binom{n}{2}$ inversions, this number gives an immediate upper bound on the number of swaps for the optimized bogo-select variants. Thus a single optimization step yields *polynomial* running time. This can be shown with a similar proof based on the coupon collectors' problem as given in [3] for an appropriate optimized inversion based bogo-sort variant.

[6] The statistical programming language R is a industry standard to analyse huge amounts of data with a vast library of statistical functions available out of the box.

References

1. L. Comtet. *Advanced Combinatorics—The Art of Finite and Infinite Expansions*. Reidel Publishing, 1974.
2. H. W. Gould. *Combinatorial Identities*. Morgantown Printing and Binding, 1972.
3. Hermann Gruber, Markus Holzer, and Oliver Ruepp. Sorting the slow way: An analysis of perversely awful randomized sorting algorithms. In Pierluigi Crescenzi, Giuseppe Prencipe, and Geppino Pucci, editors, *Fun with Algorithms, 4th International Conference, FUN 2007, Castiglioncello, Italy, June 3-5, 2007, Proceedings*, volume 4475 of *Lecture Notes in Computer Science*, pages 183–197. Springer, 2007. doi:10.1007/978-3-540-72914-3_17.
4. D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 1969.
5. D. H. Lehmer. Interesting series involving the central binomial coefficient. *Amer. Math. Mon.*, 92(7):449–457, – 1985. doi:10.2307/2322496.
6. E. S. Raymond. *The New Hacker's Dictionary*. MIT Press, 1996.
7. A. M. Rockett. Sums of inverses of binomial coefficients. *Fibonacci Quart.*, 19(5):433–437, 1981.
8. R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms*. Pearson Education, 2013.
9. R. Sprugnoli. Sum of reciprocals of the central binomial coefficients. *Integers*, 6:A27, 2006.
10. B. Sury, T. Wang, and F.-Z. Zhao. Identities involving reciprocals of binomial coefficients. *Integer Seq.*, 7:04.2.8, 2004.

Correction of Theorem 7

In Theorem 7 the expected value of the random variable C that counts the number of comparisons carried out by the procedure Z-PARTITIONED was determined to be

$$\mathbb{E}[C] = \sum_{j=1}^{k-1} \left[\frac{1}{2j-1} \binom{2(j-1)}{j-1}^{-1} + \frac{1}{j} \binom{2j}{j}^{-1} \right] + \frac{1}{k} \sum_{i=2k-1}^{n-1} \binom{i}{k}^{-1}. \tag{8}$$

Then the asymptotic value of $\mathbb{E}[C]$ that comes from the first sum was overestimated under the constraints that k is fixed and that n, the number of elements, tends to infinite. Under these prerequisites Theorem 7 must read as follows, where (8) is also simplified:

▶ **Theorem 7*.** *Assume $\overline{x}$ is a random permutation of $\{1, 2, \ldots, n\}$, and let C denote the random variable counting the number of comparisons carried out by the procedure* Z-PARTITIONED *in the test whether $\overline{x}$ is partitioned according to the kth element, for $1 \leq k \leq (n+1)/2$, then:*

$$\mathbb{E}[C] = 3 \sum_{j=1}^{k-1} \frac{1}{j} \binom{2j}{j}^{-1} + \frac{1}{k} \sum_{i=2k-1}^{n-1} \binom{i}{k}^{-1}$$

It holds $\mathbb{E}[C] = H_{n-1}$, if $k = 1$, and

$$\frac{\pi\sqrt{3}}{3} - \frac{1}{k-1} \binom{2(k-1)}{k-1}^{-1} \leq \mathbb{E}[C] \leq \frac{\pi\sqrt{3}}{3} + \frac{1}{k-1} \binom{2(k-1)}{k-1}^{-1},$$

otherwise. The statement remains valid in case k satisfies $(n+1)/2 < k \leq n$ by replacing k by $n - k + 1$.

Observe, that the left term of the first sum in (8) can be rewritten as

$$\frac{1}{2j-1} \binom{2(j-1)}{j-1}^{-1} = \frac{1}{2j-1} \cdot \frac{(j-1)!(j-1)!}{(2j-2)!} = \frac{2j}{j \cdot j} \cdot \frac{j!j!}{(2j)!} = \frac{2}{j} \binom{2j}{j}^{-1}$$

and therefore the former sum simplifies to

$$\sum_{j=1}^{k-1} \left[\frac{1}{2j-1} \binom{2(j-1)}{j-1}^{-1} + \frac{1}{j} \binom{2j}{j}^{-1} \right] = 3 \sum_{j=1}^{k-1} \frac{1}{j} \binom{2j}{j}^{-1}.$$

Next, in order to determine the bounds on $\mathbb{E}[C]$ we have to estimate the error in terms of k that is induced by partial summation. To this end we use the following upper and lower bound result from [1] on approximation of finite sums: let $S = \sum_{i=1}^{\infty} a_i$ and let the kth partial sum be $S_k = \sum_{i=1}^{k} a_i$. Suppose that the sequence $a_1, a_2, \ldots$ is a positive decreasing sequence and $\lim_{i \to \infty} \frac{a_{i+1}}{a_i} = q < 1$. Then we distinguish two cases:

1. If $\frac{a_{i+1}}{a_i}$ decreases to the limit q, then

$$S - \frac{a_{k+1}}{1 - \frac{a_{k+1}}{a_k}} \leq S_k \leq S - a_k \frac{q}{1-q}.$$

2. If $\frac{a_{i+1}}{a_i}$ increases to the limit q, then

$$S - a_k \frac{q}{1-q} \leq S_k \leq S - \frac{a_{k+1}}{1 - \frac{a_{k+1}}{a_k}}.$$

In other words, the kth partial sum S_k can be bounded from above and below in terms of the last element from the partial sum and its successor element, that is already outside the partial sum.

Consider the first sum

$$3\sum_{j=1}^{k-1}\frac{1}{j}\binom{2j}{j}^{-1}=\sum_{j=1}^{k-1}a_i,$$

in Theorem 7*, where $a_i=\frac{3}{i}\binom{2i}{i}^{-1}$. Obviously, the infinite sequence $a_1,a_2,\ldots$ is positive and decreasing and

$$\lim_{i\to\infty}\frac{a_{i+1}}{a_i}=\lim_{i\to\infty}\frac{3}{(i+1)}\cdot\frac{(i+1)!(i+1)!}{(2i+2)!}\cdot\frac{i}{3}\cdot\frac{(2i)!}{i!i!}=\lim_{i\to\infty}\frac{i(i+1)}{(2i+1)(2i+2)}=\frac{1}{4},$$

By inspection, the sequence $\frac{a_{i+1}}{a_i}$ increases to the limit $q:=\frac{1}{4}$ leading to the second case for bounding the partial summation. Since $\frac{q}{1-q}=\frac{1}{3}$, we therefore bound the sum of the $(k-1)$st partial sum of the a_i's by

$$\frac{\pi\sqrt{3}}{9}-a_{k-1}\cdot\frac{1}{3}\le\sum_{j=1}^{k-1}a_j\le\sum_{j=1}^{\infty}a_j=\frac{\pi\sqrt{3}}{9},$$

where the last equality follows from Lehmer's identity. Since $a_{k-1}=\frac{3}{k-1}\binom{2(k-1)}{k-1}^{-1}$, together with the already correctly stated upper bound $\frac{1}{k-1}\binom{2(k-1)}{k-1}^{-1}$ of the second sum in Theorem 7* in the original proof, one observes that $\mathbb{E}[C]$ lies in between

$$\frac{\pi\sqrt{3}}{3}-\frac{1}{k-1}\binom{2(k-1)}{k-1}^{-1}\le\mathbb{E}[C]\le\frac{\pi\sqrt{3}}{3}+\frac{1}{k-1}\binom{2(k-1)}{k-1}^{-1}.$$

This proves the stated result.

References

1 B. Braden. Calculating sums of infinite series. *American Mathematical Monthly*, 99(7):649–655, August–September 1992.

Revision Notice

This is a revised version of the eponymous paper appeared in the proceedings of FUN 2018 (LIPIcs, volume 100, `http://www.dagstuhl.de/dagpub/978-3-95977-067-5`, published in May, 2018), in which Theorem 7 was revised to correct an overestimation of the asymptotic bound of the expected number of comparisons for one of the methods.

Dagstuhl Publishing – October 15, 2018.