Journal of Integer Sequences, Vol. 18 (2015),
Article 15.10.7

Solution Sequences for the Keyboard Problem and its Generalizations

Jonathan T. Rowell
University of North Carolina at Greensboro
Department of Mathematics and Statistics
116 Petty Building
317 College Avenue
Greensboro, NC 27412
USA
jtrowell@uncg.edu

Abstract

The keyboard problem is an optimization problem asking how many characters can be placed into a blank document using N keystrokes. The question is representative of a larger class of output maximization problems where there is the opportunity to expand output capacity by replicating the existing output as a single unit. Here I define a generalized keyboard sequence as an integer sequence representing the maximum output of such problems, explain the construction of optimal strings of operations leading to these outputs, and demonstrate that each sequence is linearly recursive for sufficiently large N. I then evaluate two competing solutions to the keyboard problem and connect additional integer sequences to this class. The article concludes with a brief overview of the crowd-sourcing involved in the keyboard problem's initial solution.

1 Introduction

Suppose that you are sitting at a computer using a word processing or text editing application, and that you are limited to the following four keystroke actions:

1. insertion by the typing of a single character (e.g., the letter "a");

Author	OEIS	sequence
Blewett	A178715	1, 2, 3, 4, 5, 6, 9, 12, 16, 20, 27, 36, 48, 64, 81, 108, 144, 192 ...
Derbyshire	A193286	1, 2, 3, 4, 5, 6, 7, 9, 12, 16, 20, 25, 30, 36, 48, 64, 80, 100, 125 ...

Table 1: Alternative solutions to the keyboard problem.

2. selection of all current text using the select-all command (Ctrl+A for Windows users, Cmd+A for Mac users);
3. copying the current selection to the clipboard (Ctrl+C or Cmd+C, respectively); and
4. pasting the clipboard contents into the document (Ctrl+V or Cmd+V, respectively).

Beginning with a blank document and an empty clipboard, what is the largest number of characters that can be produced by using only N keystrokes?

The question given above is called the typing or keyboard problem [1, 2, 3]. This is an easily conceptualized optimization problem, but the underlying feature of expanding output capacity is quite extensible. Blewett and Derbyshire offered two competing solutions, represented respectively by OEIS sequences A178715 and A193286 (Table 1). Both sequences are actually correct, but they solve different problems predicated on a subtle but important distinction regarding the de-selection of the workspace prior to pasting.

The two solutions are examples of a broader class of integer sequences that I term generalized keyboard sequences. These sequences represent solutions for maximizing output given the availability of a generalized select-and-copy operation that requires p steps (time, keystrokes, etc.) to package a new, larger unit of production (e.g., going from a single character insertion to pasting a block of characters, or being able to manufacture an integrated product). I present an analytic treatment of the solutions to these problems and the strings of operations which produce them. Generalized keyboard sequences are linearly recursive for sufficiently large string step-sizes, and both the amplification factor and lag time are functions of the cost of copying a new integrated unit into the buffer. I demonstrate that the Derbyshire sequence A193286 is equal to the solution sequence for a problem where copying costs $p = 2$ steps (namely the select-all/copy combination), while the Blewett sequence A178715, because it presumes the de-selection of the document text after the copy operation, instead is equivalent to the solution when content can be copied to the buffer in $p = 1$ step, with selection retained. I then describe solutions for more extensive copy operations ($p = 3$, 4, or 5) and identify two previously known sequences — the doubling sequence 2^{n-1} and the maximal size of an Abelian subgroup of the symmetric group S_n [4] — as members of this class of sequences corresponding to cost-free copying that, respectively, de-selects and retains selection of existing text.

2 The general keyboard problem

Consider a text-generation problem involving three operations. First there is a simple operation a that inserts a single character into the document. There is also a generalized copy operation C which both selects and copies the existing text to a memory buffer, while a generalized paste operation V appends buffered text to the document text. The copy operation is said to be *with replacement* if the first paste or simple insertion after copying eliminates, overwrites, or otherwise renders obsolete the currently existing text output. This is wholly analogous to retaining selection of a text field after copying the selection. If the existing text is immediately de-selected, copying is said to be *without replacement*.

Definition 1. A *string* $\mathbf{x}$ is an ordered sequence of operations drawn from the set $\{a, C, V\}$.

The *step-length* or *cost* of the string, denoted $||\mathbf{x}||$, is equal to the cumulative number of steps necessary to implement the operations contained within the string. Simple insertion, a, and pasting, V, each cost a single step, while copying costs p steps.

The *output* of a string, $T(\mathbf{x})$, is the number of characters created within an initially blank document after reading the string from left to right and performing the corresponding operations. The clipboard buffer is likewise assumed to be initially empty. A string $\mathbf{x}^*$ is *optimal* if, for all other strings $\mathbf{x}$ of equal cost, $T(\mathbf{x}^*) \geq T(\mathbf{x})$.

As an example, if copying requires $p = 2$ steps, the string $\mathbf{s} = aaaCVVCVa$ has a cost $||\mathbf{x}|| = 11$. With replacement the string output equals $T(\mathbf{x}) = 7$, but without replacement, this string would output 19 characters. Generically, we can observe that by extending a given string $\mathbf{s}$, copying with replacement yields

$$T(\mathbf{s}CV) = T(\mathbf{s}C) = T(\mathbf{s}).$$

In contrast, if copying occurs without replacement, the same extended string satisfies

$$T(\mathbf{s}CV) = 2T(\mathbf{s}C) = 2T(\mathbf{s}).$$

This paper primarily considers those procedures with replacement; however, the section revisiting the original keyboard problem discusses the underlying relationship between the two types of copy operations and their corresponding solutions.

Definition 2. A *generalized keyboard sequence*, (S_N), is the sequence of integers representing the maximum output of a string in $\{a, C, V\}^*$ costing N steps to execute,

$$S_N = F(N) = \max_{||\mathbf{x}||=N} T(\mathbf{x}). \tag{1}$$

Optimal strings are not necessarily unique in giving the maximal output for the stated number of steps, e.g., if copying with replacement costs $p = 1$ steps, both $\mathbf{s}_1 = a^6$ and $\mathbf{s}_2 = a^3CV^2$ are optimal for strings of step-size 6. Further, note the following heuristic observations on the construction of optimal strings:

- Repeated copying does not increase the number of characters within the buffer, nor do they contribute to the output. Thus no optimal string contains adjacent copy operations, and no copying appears initially or terminally within the string.
- A useful copy operation loads into the buffer a number of bundled characters greater than 1. Therefore, once such a copy operation has been invoked, no single-character insertion will occur rightward within an optimal string. Consequently, simple insertions occur only in the earliest (leftmost) segment within the string, which seeds the entire process.
- No paste operation will occur prior to the first copy operation as the buffer is empty.
- If copying occurs with replacement, there must always be at least two pastes per copy operation, else there is no effective change to the string's output from that segment.

In light of these observations, we can restrict our attention to strings of the form ($k_i \geq 2$)

$$\mathbf{x} = a^{k_0} C V^{k_1} \cdots C V^{k_n}. \tag{2}$$

The initial seed of simple character insertions is additive, and the first copy buffers a text bundle equal to k_0 characters. The first paste operation then eliminates the original loose collection of characters and substitutes a unified collection of equal value. All subsequent pastings of the buffer content also contribute k_0 characters, so that just prior to the second copy operation, the string has produced $k_0 k_1$ total characters. This multiplicative effect is recapitulated after each copy and paste substring. Thus the total output generated by a string $\mathbf{x}$ of the form (2) is equal to the product of each substring's productive (non-copying) length,

$$T(\mathbf{x}) = k_0 k_1 \cdots k_n = \prod_{i=0}^{n} k_i. \tag{3}$$

The output of a string of the form (2) is invariant to the exact order of the k_i terms. Thus strings $\mathbf{x}_1 = a^3 PV^2 PV^4$ and $\mathbf{x}_2 = a^2 PV^4 PV^3$ both produce 24 characters. If there are H copy operations within a string of step-size N, the problem of maximum string output is equivalently restated as optimizing the product of positive integers $\prod_{i=0}^{H} k_i$ whose sum is $\sum_{i=0}^{H} k_i = N - pH$.

Lemma 3. *The number of characters that can be generated by an optimal string is of the form*

$$T(\mathbf{x}^*) = m^B (m+1)^E, \tag{4}$$

where, m, B, and E are nonnegative integers, and $B + E = H + 1$ is the number of substrings partitioned by copy operations.

Proof. Without loss of generality, assume that the optimal string $\mathbf{x}$ has H copy operations, producing a set of $H+1$ productive substring lengths $\{k_i\}$. Let $m = k_j$ and $M = k_k$ be the minimum and maximum values within that set, respectively.

Define a variant string $\mathbf{x}'$ as $k'_j = (m+1)$ and $k'_k = (M-1)$. All other substrings are the same as in the original string ($k'_i = k_i$), and the cost of the two strings are consequently identical, $||\mathbf{x}'|| = ||\mathbf{x}||$. The total output of the variant string $\mathbf{x}'$ is

$$\begin{aligned} T(\mathbf{x}') &= \prod k'_i \\ &= (m+1)(M-1)\prod_{i\neq j,k} k_i \\ &= mM\prod_{i\neq j,k} k_i + (M-(m+1))\prod_{i\neq j,k} k_i \\ &= T(\mathbf{x}) + (M-(m+1))\prod_{i\neq j,k} k_i \end{aligned} \tag{5}$$

Therefore, if $M > m+1$, $T(\mathbf{x}') > T(\mathbf{x})$, and $\mathbf{x}$ is not an optimal string. □

This lemma leads directly to the following theorem about the form of the solution.

Theorem 4. *For a generalized keyboard problem with copy cost p, the maximum output that can be generated over N steps is of the form*

$$F(N) = \max_H \left(m^{[(H+1)-E]}(m+1)^E \right), \tag{6}$$

where H is the number of copy operations contained within the associated string, and

$$\begin{aligned} m &= \lfloor (N+p)/(H+1) \rfloor - p, \text{ and} \\ E &= (N+p) \bmod (H+1). \end{aligned} \tag{7}$$

Proof. Assume that the optimal output is of the form given by (2). The total number of factors equals $E + B = H + 1$. Appending a phantom copy operation just prior to the seed string of simple insertions gives an extended string of step-size $N + p$. The terms $(m + p)$ and E can be calculated directly as the quotient and remainder resulting from the nearly equal division of the extended $N + p$ steps across $H + 1$ substrings. To compute m, we need only subtract those steps required for the operational cost of copying p from the quotient to obtain the final result in Eq. (7). □

Theorem 4 reduces the original problem of solving a particular generalized keyboard problem to the question of finding the optimal number of copy operations to invoke within the string. Unfortunately, the number of copies to use is not entirely predictable, at least for low N, but the minimum number of copies needed for an optimal string is a non-decreasing function of string step-size.

Theorem 5. *The least number of copies for an optimal string of step-size N is non-decreasing, i.e.,*

$$H(N) \leq H(N+1).$$

number of copies H	0	1	2	3	4	5	6	7
string step-size N	1	8	15	21	27	34	40	46

Table 2: First occurrences of numbers of copies in optimal strings ($p = 2$)

Proof. Suppose that for string step-size N, the smallest optimal copy number is h^*, with a corresponding total output $T_1 = d^B(d+1)^E$. The maximum output for a string of the same step-size but with fewer copy operations ($0 \leq h < h^*$) is $T_2 = \delta^\beta(\delta+1)^\eta$. $T_1 > T_2$ by optimality, and $\delta \geq d$ by division. Increasing the string step-size by one increases one least factor by one, so that the new strings have outputs $T_1' = ((d+1)/d)T_1$ and $T_2' = ((\delta+1)/\delta)T_2$, respectively. The string multipliers obey $(d+1)/d \geq (\delta+1)/\delta$, therefore $T_1' > T_2'$. Thus once a lower number of copies has been superseded by the next higher number, that number cannot subsequently generate a greater total output at larger string step-sizes. □

Theorem 5 assures a natural succession of the optimal number of copies beginning with $H = 0$ and incrementing upward as N increases; however, in most instances, the necessary number of additional string steps required to increment H is subject to some early exceptions to any proposed regularity (Table 2). For sufficiently large N, however, a substitution pattern develops among the factors making up the solution, and the switch to the next higher number of copies becomes regularly spaced with an optimal number of pastes per copy.

Productive segments have a multiplicative effect on the output of optimal strings (see Eq. (3)), so the ideal number of pastes k^* per copy operation with cost p gives the greatest geometric growth rate over the $k + p$ substring,

$$G(k) = \sqrt[k+p]{k}. \tag{8}$$

If the optimal number of pastes per copy is k^*, we can define $I = p + k^*$ as the step-size of the corresponding substring CV^{k^*}. For sufficiently large N, this generates the recursive rule

$$F(N) = k^*F(N - I). \tag{9}$$

Let N^* be the string step-size such that $F(N)$ is recursively defined for all $N \geq N^*$. Equation (9) implies that k^* will be a factor in the solution to $F(N)$ when $N \geq N^*$, and thus either $m = k^*$ or $m + 1 = k^*$ in the solution given in (6). Although it is tempting to assume $(H + 1) = \lfloor (N + p)/I \rfloor$, thus implying $m = k^*$, we cannot presume k^* to be the larger or smaller factor. The optimal number of pastes per copy can be either factor, or even alternate between them once the sequence becomes predictable.

Theorem 6. *If N^* is the minimum necessary number of steps for a generalized keyboard sequence with copy cost p to become recursive, i.e., $F(N) = k^*F(N - I)$ for all $N \geq N^*$, then there are non-negative parameters V and Q, such that for all $N \geq N^* - I$,*

$$F(N) = (k^* - 1)^{\langle Z \rangle}(k^*)^{(H+1)-|Z|}(k^* + 1)^{\langle -Z \rangle} \tag{10}$$

where, $(H+1) = \max\{1, \lfloor (N+Q)/I \rfloor\}$ *is the number of partitioned substrings,* $R = (N+Q) \bmod I$*,* $Z = V - R$*, and* $\langle Z \rangle = \max\{0, Z\}$*.*

Proof. Let k^* be the optimal number of pastes per copy given a copy cost p, and let N^* be the string step-size at which the solution becomes recursive (6). There are at most two distinct factors in $F(N)$ for any given N. For all $N \geq N^* - I$, these factors are either the pair (k^*-1) and k^* or the pair k^* and (k^*+1). Accordingly, the solution is of the generalized form $(k^*-1)^W (k^*)^X (k^*+1)^Y$, with W and $Y \geq 0$, but not both positive. Note we allow for the possibility that k^* has a trivial exponent for integers $N^* - I \leq N < N^*$.

Let $N_Q = MI + \xi$, with $0 \leq \xi < I$, be the smallest N greater than N^* such that $F(N_Q)$ has exactly one more factor than $F(N^*)$. Define V as the number of (k^*-1) factors in $F(N_Q)$ and $\tilde{H}+1$ the total number of factors. Finally define the parameter Q as

$$Q = (\tilde{H} + 1 - M)I - \xi. \tag{11}$$

Note that $N_Q + Q = (\tilde{H}+1)I$.

Between $N_Q \leq N < N_Q + I$, the number of factors in $F(N)$ is constant (else N_Q would not be the minimum $N > N^*$ with more factors in $F(N)$). As N increments within this range, all increases in the product $F(N)$ must be reflected in a change of factors: first by incrementing the (k^*-1) factors and then incrementing k factors to $(k+1)$.

Now, all positive integers can be expressed in reference to N_Q as $N = N_Q + \delta I + R$, with $0 \leq R < I$, and thus

$$\begin{aligned} N + Q &= N_Q + Q + \delta I + R \\ &= (\tilde{H} + 1 + \delta)I + R. \end{aligned} \tag{12}$$

In the range $N_Q \leq N < N_Q + I$, R tracks the position of step N away from the starting point N_Q so that $\langle Z \rangle$ correctly accounts for the number of (k^*-1) factors in $F(N)$. Likewise for $\langle -Z \rangle$ and the (k^*+1) factors. The remainder position is preserved as one moves up or down the index N by I steps, i.e., $(N+Q) \bmod I = R$. Thus for $N \geq N^* - I$, the number of $(k^* \pm 1)$ terms is predictable. For the substring count, we see that

$$\lfloor (N+Q)/I \rfloor = (\tilde{H}+1) + \delta. \tag{13}$$

Setting $H = \tilde{H} + \delta$ correctly accounts for the loss or gain of k^* factors (δ) through the recursive formula Eq. (6).

□

Although exceptions do exist, Theorem 6 also provides a reasonable estimate for the number of partitioned substrings $H+1$ in shorter strings $(N < N^* - I)$ which can then be used in conjunction with Theorem 4 to evaluate different possibilities for $F(N)$.

3 The keyboard problem revisited

Returning to the original keyboard problem, the astute reader will have noticed an apparent discrepancy wherein the keyboard problem is posed with four, not three, potential actions.

From a practical consideration, the select-all and copy commands must occur in tandem as an undivided pair. Any optimal string of keystrokes would necessarily have this characteristic, otherwise, the string would at best waste a step (e.g., copying an empty insertion cursor) or even be counter-productive (e.g., inserting or pasting text immediately after selection). Restriction to this two-keystroke combination reduces the set of operations to three elements with a single select-and-copy operation which costs $p = 2$ steps.

Using Theorem 4 with copy cost $p = 2$, the resulting solution sequence matches OEIS A193286, confirming that the Derbyshire sequence [1, 3] was generated under the assumption that selection was retained during copying. For $N \geq 34$, this sequence settles into the recursive formula

$$f(N) = 4f(N-6).$$

In contrast, Blewett generated his solution, A178715, with the assumption that the copy command released the selection (Blewett, personal communication), and any subsequent pasting or character insertion would therefore not overwrite existing text but be immediately additive. Among the effects of this interpretation, the ideal number of pastes per copy must maximize the geometric growth rate

$$G(k) = \sqrt[k+2]{k+1}, \tag{14}$$

where the radicand reflects the retention of the preceding output (cf. Eq. (8)). By a simple linear substitution, solving (14) is identical to optimizing $\sqrt[k'+1]{k'}$. In fact for all string step-lengths N, the solution to the original keyboard problem where copying automatically de-selects text is identical to the generalized keyboard sequence obtained when copying with replacement bears a reduced cost of $p = 1$ step.

Moreover, any generalized keyboard problem characterized by a copy operation that costs p steps and automatically releases selected text produces a solution sequence equal to that obtained for a problem where copying with replacement costs $p-1$ steps. Strings under each interpretation are in a one-to-one equivalence through the relationship $C = \widehat{C}V$, where C is copying without replacement and has cost p and $\widehat{C}$ is copying with replacement and has cost $p-1$, for any $p \geq 1$. The only distinction between the two interpretations is that one would technically report the number of optimal pastes per copy without replacement as $k^* - 1$, not k^*.

4 Other sequences

Other construction scenarios may require more involved copying procedures, e.g., in many photo-manipulation and graphic programs, one can select starting and ending actions in a history palette and create a single command to execute the entire range of actions, all in sequence. This process can be replicated ad nauseum to create a very elaborate treatment of multiple images. Similarly, copying and pasting on tablets often require the use of multiple gestures, e.g., an iOS device (prior to version 8.4) required long press, select all, copy, tap, long press, paste. At the opposite extreme, copying could be freely applied, perhaps as an ongoing background update process which does not require a loss of user productivity (e.g., the cost of string steps) to maintain. Table 3 provides five additional sequences that belong to this class of solution sequences. Three sequences correspond to copy costs $p = 3$, 4, or 5, while two others are for cost-free copying either with or without replacement.

Consider first the case of generalized keyboard sequences where copying is more costly. OEIS sequences A193455, A193456, and A193457 are the output solutions for $p = 3$, 4, and 5, respectively. The previous generalized keyboard sequences had the ideal number of pastes per copy, k^*, either always the greater ($p = 1$) or always the lesser ($p = 2$) factor in Eq. (6); however these new sequences make use of three distinct factors once the sequence is recursive, and k^* occurs as both the maximum and minimum factor for different N.

If the generalized keyboard problem is instead distinguished by a free copy procedure that could be applied at any time without cost ($p = 0$), the solutions would maximize the product $\prod_i k_i$ subject to the constraint $\sum_i k_i = N$. This famous puzzle problem and its solution (OEIS sequence A000792) have been known for some time [5, 6, 8]; also see [7, pp. 30–31, 188]. This sequence is notable also for describing the maximal size of an Abelian subgroup of the symmetric group S_n [4].

Now extend the idea of free copying further by assuming that copying occurs without replacement. Per the discussion in the previous section, the solution to $p = 0$ copying without replacement can be equivalently identified as the solution when copying with replacement has a "cost" of $p = -1$. The optimal number of pastes per copy has a computed value of $k^* = 2$, and the lag is $I = 1$. The simple recursion formula, $F(N) = 2F(N - 1)$, is valid for all $N > 1$, and the sequence is simply the powers of 2 beginning at 1 (OEIS sequence A131577). For this particular problem, the string's initial two-bit segment could be either aa or $a(C)V$, which has a carryover effect wherein both $H + 1 = N$ and $H + 1 = N - 1$ are valid numbers of substrings in optimal solutions.

Table 3 provides the ideal number of pastes per copy for several different costs p, as well as the recursive rules and parameters for use with (10). The table also gives how many exceptions there are for the computed number of substrings for $N < N^* - I$ and lists the associated OEIS sequence number.

5 Discussion

Generalized keyboard sequences provide solutions for a class of optimization problems that have taken their inspiration from the keyboard problem. These problems feature the idea of increasing the rate of production by bundling pre-existing output as an integrated unit, e.g., copying and pasting all current text in a document. This study distinguished two competing solutions to the original keyboard problem and led to the discovery of several new sequences. Additionally, two well-established sequences were shown also to belong to this class of sequences and to correspond to cost-free copying.

All generalized keyboard sequences with a single-step paste implementation share two

p	k^*	I	N^*	$F(N)$	$N^* - I$	V	Q	Except.	OEIS
-1	2	1	2	$2F(N-1)$	1	1	0	0	A131577
0	3	3	5	$3F(N-3)$	2	1	1	0	A000792
1	4	5	16	$4F(N-5)$	11	4	5	2	A178715
2	4	6	34	$4F(N-6)$	28	0	2	5	A193286
3	5	8	34	$5F(N-8)$	26	3	6	1	A193455
4	6	10	69	$6F(N-10)$	59	8	12	11	A193456
5	6	11	80	$6F(N-11)$	69	3	8	6	A193457

Table 3: Paradigm shift sequence results for different copy costs. Items listed are copy cost, (p); optimal pastes per copy (k^*); period or recursion lag (I); step where recursion formula begins to hold (N^*); recursion formula ($F(N)$); first step where Theorem 2 fully applies ($N^* - I$); maximum number of ($k^* - 1$) factors (V); and procedure increment marker (Q). Also listed are the number of exceptions to the expected number of procedures (H) or substrings ($H + 1$) over $1 \leq N < N^* - I$ and the OEIS sequence number.

common factors. The maximum output for each string step-size N can be expressed as a product of at most two distinct consecutive factors, each raised to a power. Second, for sufficiently large N, each sequence settles down to a linear recursion formula where the multiplier is the number of pastes per copy that yields the greatest geometric growth rate, and the lag is the sum of that number and the copy cost.

The ideal number of pastes per copy, k^*, is merely a non-decreasing function of copy cost, but the lag or innovation cycle, I, strictly increases with the cost of copying. Parameters V and Q in Theorem 6 do not correlate with copy costs, nor does the number of exceptions to the predicted number of optimal pastes for low N. For example, the sequence for copy cost $p = 3$ has a single exception, and that for $p = 5$ has six, but the sequence for $p = 4$ has the most exceptions among the tested costs.

The most direct extension of this model is to consider problems where the implementation of a single paste requires multiple steps to complete. Strings with "lengthy" implementations share many similarities with the optimal strings presented here, and some results will parallel those in this paper, yet there are important differences such as the relationship of the seed length to the ideal number of pastes per copy (unpublished manuscript). Other extensions include copying or bundling procedures with increasingly inefficient paste implementations where the buffer value degrades over time, and manufacturing problems with multiple tracks of production or a stackable memory buffer.

6 An informal history of the keyboard problem

The solution to the keyboard problem and its early dissemination was a product of crowd-sourcing that reflects the work of both mathematicians and hobbyists, and this history serves as a reminder that simple questions can inspire a broader idea. It is unknown when or by whom this problem was first posed, but given its simplicity and the ubiquity of its context, it would be unsurprising to discover that it had been previously asked, or even answered, by math enthusiasts in isolated circumstances. The history documented here concerns the connective power of the internet in the form of message boards, web articles and blogs, and online encyclopedias.

Late in 2010, an unidentified individual posted a version of the keyboard problem to an intra-company message board at Microsoft Corporation. Bill Blewett, a software design engineer with the company, responded to that challenge (Blewett, personal correspondence), and by early 2011, he had submitted his solution to the On-Line Encyclopedia of Integer Sequences as A178715. A few months later in the summer of 2011, John Derbyshire, a writer and author of two general readership books on mathematics, gave the keyboard problem greater visibility when he posed the problem as a monthly brainteaser to his online readership [1]. He himself had been forwarded the problem by a correspondent who had encountered the puzzle in a job interview with another software company (Derbyshire, personal correspondence). Derbyshire subsequently generated many solutions via a brute force search from whence he detected the sequence's long-term recursive property. (His updated solution page [3] has expanded its list of solutions and reports some of the analytic ideas presented here — which were discussed in correspondence — commingled with those of his own industry.) N. J. A. Sloane volunteered the sequence's OEIS entry A193286, while blogger Jonathan Campbell further publicized the problem and its dueling solutions [2]. Following his readers' observation of the distinction between the competing solutions, Campbell contributed python scripts that eventually corroborated both the Blewett and Derbyshire sequences for their respective assumptions [2].

Concurrently I had become aware of the problem and was inspired to provide a formalized description of not only the original problem, but also its generalized form, which led to the development of the idea of the generalized keyboard sequence (originally termed a "paradigm shift sequence" in OEIS entries), the contribution of new sequences for extended copy processes, and the identification of connections to existing sequences.

References

[1] J. Derbyshire, June Diary: On pessimism, Western Civilization, and more, *National Review Online*. Published electronically at http://tinyurl.com/q9wgf43, 2011.

[2] J. Campbell. Typing puzzle, in *Questions about Politics, Philosophy, and Math*. Published electronically at http://www.acouplequestions.com/?p=468, 2011.

[3] J. Derbyshire, June 2011 Solutions to puzzles in my National Review Online Diary. Published electronically at http://tinyurl.com/ntun8z6, 2011.

[4] R. Bercov and L. Moser, On Abelian permutation groups, *Canad. Math. Bull.* **8** (1965) 627–630.

[5] B. R. Barwell, Cutting string and arranging counters, *J. Rec. Math.* **4** (1971), 164–168.

[6] B. R. Barwell, Maximum product: solution to problem 2004, *J. Rec. Math.* **25** (1993), 313.

[7] P. R. Halmos, *Problems for Mathematicians Young and Old*, Mathematical Association of America, 1991.

[8] E. F. Krause, Maximizing the product of summands, *Math. Mag.*, **69** (1996), 270–271.

2010 *Mathematics Subject Classification*: Primary 11N64; Secondary 11N37, 00A08, 11K65, 11B99.
Keywords: keyboard problem, generalized keyboard sequence, doubling sequence, output optimization.

(Concerned with sequences A000792, A131577, A178715, A193286, A193455, A193456, and A193457.)

Received June 16 2014; revised versions received March 18 2015; August 22 2015. Published in *Journal of Integer Sequences*, October 10 2015.

Return to Journal of Integer Sequences home page.