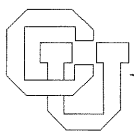
ON ULAM'S PROBLEM

By
Richard Dunn

CU-CS-011-73 January 1973



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

* Supported by NSF NYI #CCR-9357740, ONR #N00014-96-1-0720, and a Packard Fellowship in Science and Engineering from the David and Lucile Packard Foundation.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JAN 1973		**00-00-1973 to 00-00-1973**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
On Ulam's Problem	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Colorado at Boulder,Department of Computer Science,Boulder,CO,80301**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**14**	

ON ULAM'S PROBLEM *

by

Richard Dunn

Department of Computer Science
University of Colorado
Boulder, Colorado

Report #CU-CS-011-73 January 1973

* This work supported by N.S.F. Grant GJ-660.

ON ULAM'S PROBLEM

by Richard Dunn

ABSTRACT:

Use of a computer is described in an attempted solution of a currently unsolved problem known as "Ulam's Problem." Although no solution was found, several patterns and regularities were found which would be useful in an attempted solution.

I. The Problem

A sequence is to be generated by the following rules:

1. Choose the initial value n_0 as an integer greater than 1.
2. a) If an element of the sequence n_i is 1, terminate the sequence.
   b) Otherwise, if element n_i is odd, set element $n_{i+1} = 3n_i + 1$ and repeat step 2.
   c) If element n_i is even, set $n_{i+1} = \frac{n_i}{2}$ and repeat step 2.

Thus, some examples of sequences would be:

$n_0 = 3$: 3, 10, 5, 16, 8, 4, 2, 1

$n_0 = 7$: 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

The problem is then stated as: Is the sequence finite in length for any choice of n_0? The attempt for solution to the problem proceeds in both directions--a search for an n_0 which gives an infinite sequence, and a search for a proof that all sequences are finite.

II. Method 1 - Direct Solution

Before attempting any more elegant approaches, it seemed worthwhile to simply try a large number of n_0 values. Since the generating algorithm is so simple for the sequences, a computer can be programmed to check a large number of cases very quickly. Before actually writing the program, some simplifying information may be developed:

1. If we select n_0 values in increasing order, we need only verify that each n_0 value has some value in its sequence n_i which is

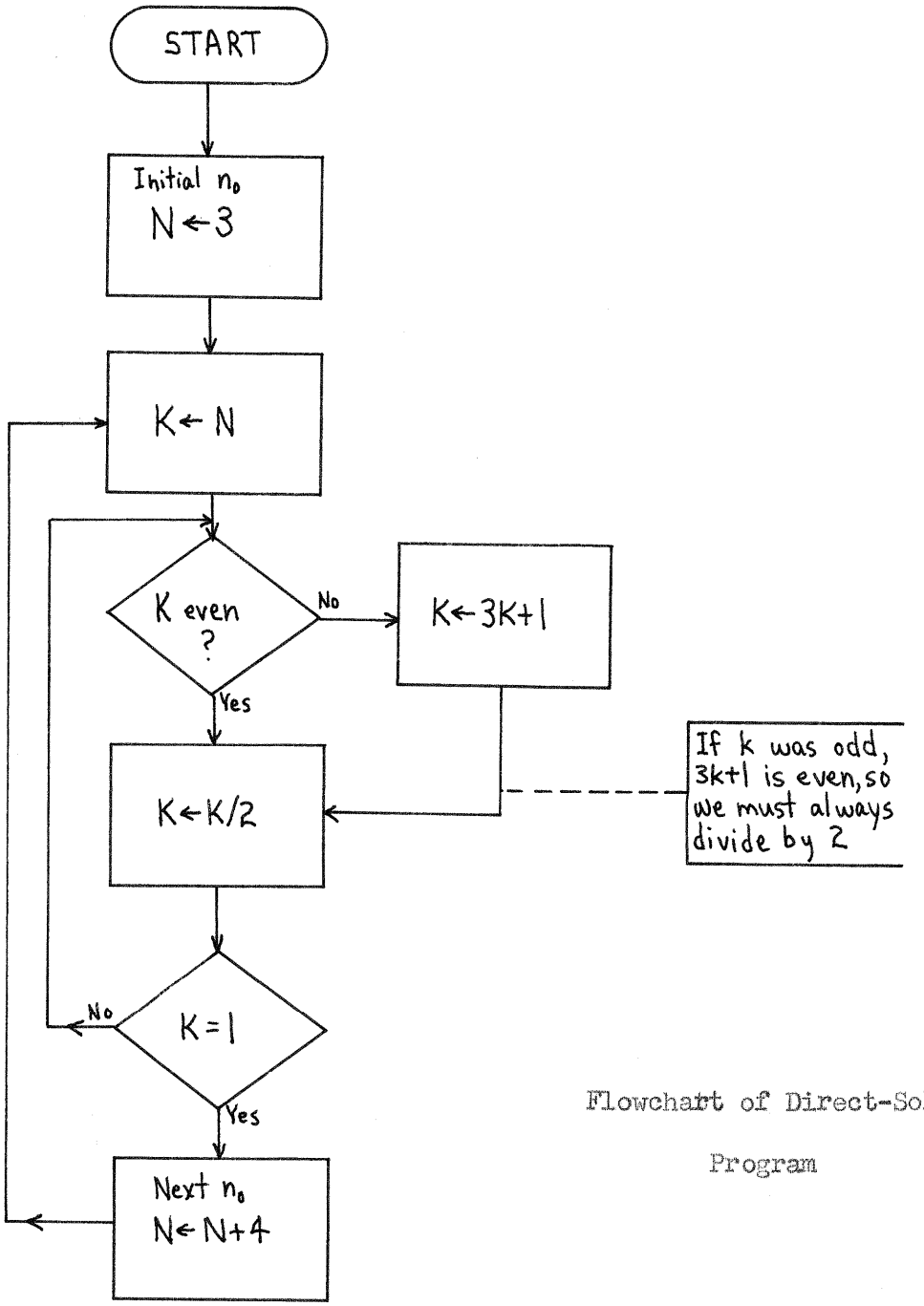
less than n_0 in order to guarantee that the sequence is finite. Since $n_i < n_0$, we have already tested the sequence beginning with n_i.

2. Using the above, it is clear that the smallest n_0 which gives a solution will not be even, since an even n_0 immediately goes to $n_0/2$ according to rule 2c.

3. Checking a few values of n_0 will show that values of the form $4k+1$, $k \geq 0$, will always give a terminating sequence by the steps $4k+1$, $12k+4$, $6k+2$, $3k+1$. (At this point, a value less than the original $4k+1$ has been obtained.) Other categories of values can be found, but they are only productive up to a point in eliminating cases to be tried because it becomes complicated to check for each form.

Using the above simplifications, a program was written in assembly language for the CDC 6400 to check for solutions. A flowchart of the program used follows. As shown, there is no exit from the program; it was simply allotted an amount of time and run until the time was exhausted, at which point the values of variables were checked to ensure that no solution had been found. Using this program, it was determined that if any n_0 does give an infinite sequence, it must exceed 28 882 247. This was determined in less than four minutes of computer time. It would be possible to push the limit higher, of course, but it does not seem promising unless the lower bound for an n_0 could be raised by two orders of magnitude or so. This would require a faster computer.

The direct approach did not yield a solution, but it did give a hint that the answer to the problem might be negative--that all n_0 give finite sequences.



Flowchart of Direct-Solution Program

III. Method 2 - Construction of "Sieve"

In the preceding section, it was mentioned that there are certain "categories" of integers for which it can be stated that the sequences will always be finite. Those already mentioned were 2k and 4k+1 for non-negative integers k. These categories were discovered by the fact that they require a particular number of elements in the sequence to reach n_i less than the n_0 for the sequence. With this knowledge, it was possible to find further categories by classifying n_0 values according to the number of "steps" (application of the rules of section I) to reach a value less than the original. The numbers which fall into a given category may then be examined to attempt to find a rule for the category. As far as the search has been carried, it has been possible to fit values to categories. A few of the categories are listed here:

2k	128k+7	256k+95
4k+1	128k+15	256k+123
16k+3	128k+59	256k+175
32k+11	256k+39	256k+199
32k+23	256k+79	256k+219

The intent of this "category" system is to attempt to prove that all integers fall into such a category, and thus that there are none which can generate an infinite sequence. All categories which have been found so far have the form $2^j k + m$. We should hope to be able to derive a systematic form for giving the m values associated with each j. Given this, we might be able to find the behavior of the number of m values for each j. We can then take the set of all positive integers and eliminate them by using successively finer "sieves" formed by using higher and higher "j" values, [illegible] in the limit showing that the sieve closes (catches all numbers.)

The computer search for j and m values turned up some interesting (but not particularly useful) regularities. Not all j values are present.

In particular, note in the above examples that j=3 and 6 (8k+m and 64k+m) are absent. It was found that there is (empirically) a periodic rule which determines which "j" values do not occur. This rule relates to another quantity common to all elements of a category--the number of "steps" required by elements of that category to reach a value in the sequence less than the initial value. This value is common to all categories with a given "j" value, and increases monotonically with the j value. The following list of "j" values and the number of steps for each gives the correspondence which was found:

Steps	1	3	6	8	11	13	16	19	21	24	26	29	32	34	37	39	42
j	1	2	4	5	7	8	10	12	13	15	16	18	20	21	23	24	26

The relation is obtained from the difference in successive values. For the "Steps" values, this difference cycles through 2 3 2 3 2 3 3 2 3 2 3 3, and for the "j" values, it cycles through 1 2 1 2 1 2 2 1 2 1 2 2. This has been checked for "Steps" values up to 140. This relation has been one of the most interesting found in the problem, but to this point it has been of no help in solving the problem!

We should like to know if the fraction of all integers covered by categories of a given "j" or less will increase with increasing "j" in such a way as to eventually cover all integers. To investigate this, we need the number of m values which occur for a given j value. (At the outset, we can anticipate some problems; obviously we are not dealing with a "nice" smooth function since there are j values for which there are no m values.)

At this point, two new functions are introduced:

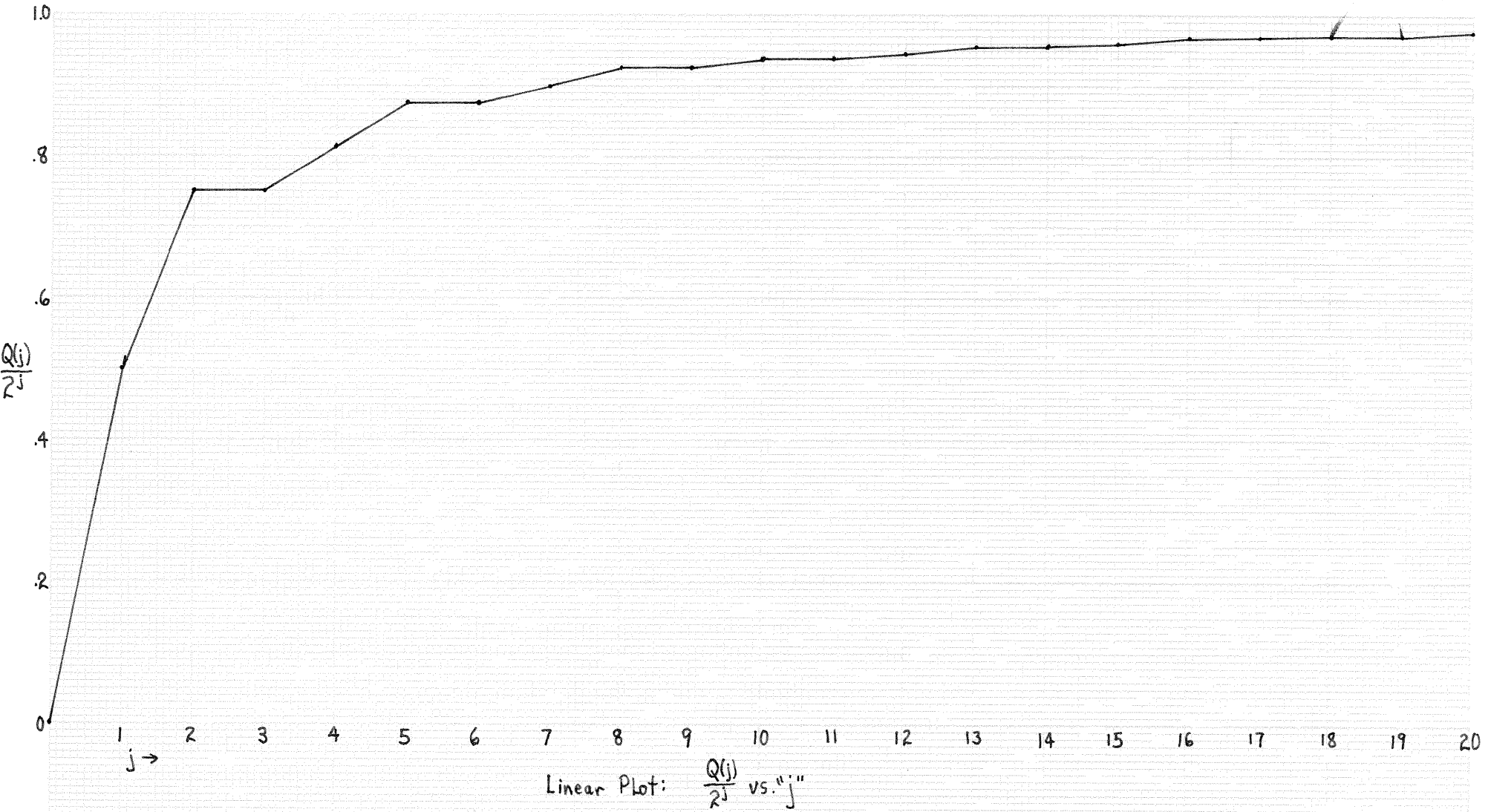
P(j) = the number of m values for this j

$Q(j) = \sum_{i=1}^{j} P(i) \cdot 2^{j-i}$ (This will give the number of integers out of 2^j covered by categories up to j.)

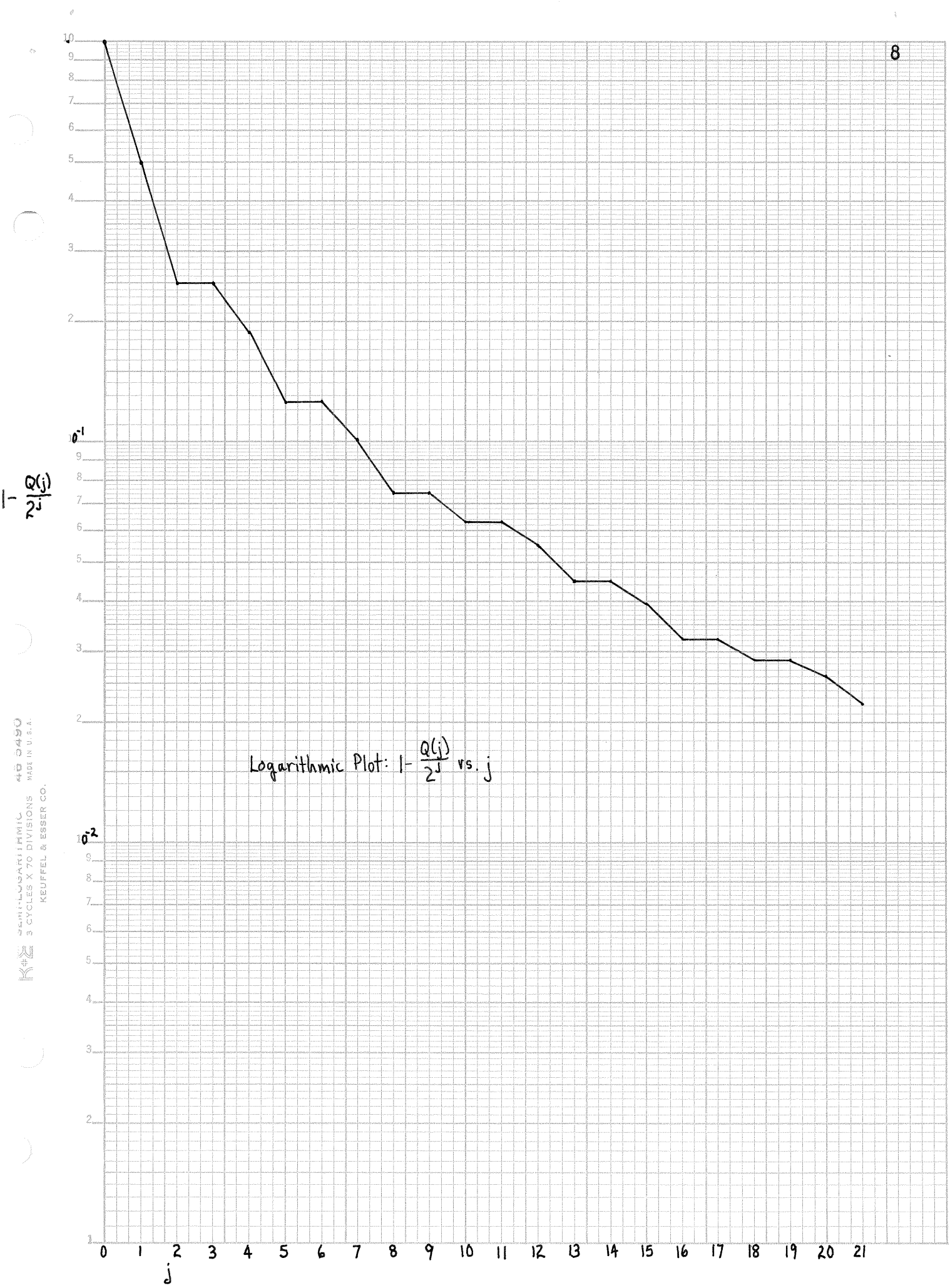
The function Q will eventually allow us to compute the quantity we are interested in--the fraction of all integers covered by categories up to a given "j" value. This is just $Q(j)/2^j$. The table which follows gives this function. Also, since it is difficult to get the "feel" of the meaning of this function for "j" values over 10 or so, the function $1 - Q(j)/2^j$ is also tabulated. This function represents the fraction of all integers not covered by categories up to j.

j	P(j)	Q(j)	$Q(j)/2^j$	$1-Q(j)/2^j$
1	1	1	.500	.500
2	1	3	.750	.250
3	0	6	.750	.250
4	1	13	.813	.187
5	2	28	.875	.125
6	0	56	.875	.125
7	3	115	.898	.102
8	7	237	.926	.074
9	0	474	.926	.074
10	12	960	.938	.062
11	0	1920	.938	.062
12	30	3870	.945	.055
13	85	7825	.955	.045
14	0	15650	.955	.045
15	173	31473	.960	.040
16	476	63422	.968	.032
17	0	126844	.968	.032
18	961	254649	.971	.029
19	0	509298	.971	.029
20	2652	1021248	.974	.026
21	8045	2050541	.978	.022

The two graphs which follow give some idea of the behavior. Because of the irregularities ("j" values with P(j)=0), it is difficult to say very much about the possibilities of convergence. If more points were available, it might be possible to analyze the form of the function $1 - Q(j)/2^j$. However, another ten points would be a reasonable number to ask for, but it is doubtful that these points could be obtained. The 21 given values required checking about 2^{19} values, and each additional data point doubles the



Linear Plot: $\frac{Q(j)}{2^j}$ vs. "j"



number of values which must be checked. The program which calculated the 21 values for the table required about a minute of time on the CDC 6400.

At this point, it seems unlikely that much further computer effort would help in the construction of a "sieve". It would be interesting to obtain a few more points on $1 - Q(j)/2^j$ to try to infer whether or not it would converge, although this would not constitute a proof, of course.

IV. Method 3 - Binary Representation

Some additional information can be gained by considering the progressive values in a sequence in their binary representation. Notice that when we reach a power of two in the sequence, the sequence will terminate after only a series of divisions by two. We could therefore examine the generating rules to see if they will always generate a power of two. Another way to look at this approach is to note that a power of two has only one "1" in its binary representation. We could therefore ask for a proof that the generating rules will eventually reduce the number of "1" bits in the binary representation of the number.

It is clear that division of an even number by two does not change the number of "1" bits. We need to examine the number of "1" bits which appear or disappear in the operation $n \rightarrow 3n+1$. After checking a few initial values, it became apparent that there was no simple relation. It was decided to examine the sequences for values of the form 2^q-1, since these values have all "1" bits in a number of a given length. This led to another interesting (but as yet unproductive) pattern. All such values have terminating sequences which follow a particular pattern. Consecutive q values group into pairs with sequences of the same length.

This last approach to the problem really consists of a few scattered

observations without much direction. It seems somewhat promising if one were to work further on this problem.

V. The Programs

It would have been impossible to try several approaches without the use of a computer. The programs were by no means intended to produce a finished solution to the problem; they merely helped with the drudgery of repeated calculations and ensured that the frequent errors in hand calculation would not crop up. The use of a computer in several cases allowed a sufficient number of calculations to be made to point up patterns in the results which would not have been evident otherwise.

All of the programs were run on the University of Colorado CDC 6400. They were written in a mixture of FORTRAN and assembly language, and none of them were over a page long. The smallest--the direct-solution search--required only twelve words of code.

Conclusion

The fact that no solution was found seems to be the least important aspect of this exercise. Considerable insight was gained into the problem, including the discovery of some patterns which might be useful to others investigating the problem. Of much greater importance was the development of techniques for approaching problems in pure mathematics with a computer. The most successful attack was to distill each question carefully before it was programmed. This kept the programs short and simple, allowing the time to be spent on the problem itself rather than debugging.