CS–1990–21

The Duke Internet Programming Contest

Owen Astrachan
Vivek Khera
David Kotz

Department of Computer Science

Duke University

Durham, North Carolina 27708-0129

December 7, 1990

The Duke Internet Programming Contest

Owen Astrachan*
Vivek Khera
David Kotz

December 7, 1990

On the evening of October 23, 1990, electronic mail messages started to pour into the computers at the Duke University Computer Science Department. Teams of programmers from all over the world were registering to compete in the first global (as far as the authors are aware) programming contest to be held on the Internet. During the three-hour competition, modeled after the annual ACM scholastic programming contest, 60 teams from 37 institutions in 5 countries attempted to solve a set of six programming problems using C or Pascal. Their solutions were sent by electronic mail to Duke, where their programs were judged and the results returned by electronic mail. At the conclusion of the contest, 330 program submissions had been judged and 65 clarification requests were answered.

1 Introduction

Our contest was inspired by and modeled after the ACM scholastic programming contest. Regional ACM competitions take place every fall in most of the ACM regions around the world with finals held in the spring at the ACM Computer Science Conference. We decided that it would be fun to have a contest that did not involve any travel for the teams and would include people not usually allowed to compete in the ACM contest (e.g., faculty). We also wanted to allow an unlimited number of teams to participate. With these goals in mind we "advertised" the contest over one of the many electronic news groups distributed throughout the world (`comp.edu`). This notice was sent only one week before the contest was scheduled to take place, but local considerations precluded a longer interim. There was enthusiastic response from schools around the world: United States, Canada, Sweden (where the competition *began* at midnight!), Australia, and New Zealand. Indeed, far more teams responded than we had anticipated.

*The authors (in alphabetic order) are all from Duke University, and may be reached by electronic mail as `ola@cs.duke.edu`, `khera@cs.duke.edu`, and `dfk@cs.duke.edu`, respectively.

Each team consisted of up to four members using one computer terminal (of any type; the limit was one keyboard). Teams were given a set of six programming problems at the beginning of the contest, and were to solve as many as possible in the three hours allotted by writing programs in either C or Pascal.[1] When teams had a program ready, the source code was submitted to the judges at Duke via electronic mail. The judges at Duke ran the program on test data that was not known to the contestants, and examined only the output. The judges responded to each program submission with a message such as "correct", "incorrect output", or "runtime error". With a fast response from the judges, teams were able to fix "incorrect" programs and resubmit them, sometimes several times, before the contest ended. The team that solved the most problems in the three-hour contest won, with ties broken by the total amount of time used to solve the problems. The problems are summarized in Appendix A. The complete problem set, including full problem descriptions, rules, solutions, and the official test data, is available by anonymous `ftp` from `cs.duke.edu` in the file `dist/misc/acm_contest/problems.tar.Z`.

2 Logistics

Each site participating was expected to have a contest administrator to install the needed software and to make printed copies of the problems. The administrator was the contact person to whom all pre-contest mailings were sent. These mailings included full contest rules, the software for submitting problems, and the guidelines for participating. These items were sent several days prior to the date of the contest. The problem set was sent to the administrator the day before the contest, with instructions to keep it secret.

Prior to the start of the contest, the contestants knew only that they would be given three to ten problems and that the contest would last three hours.

We selected a start time that was convenient for us, since we initially had intended for the contest to include a few schools on the East Coast. Instead, the contest grew quickly to include schools from around the world. The contest start time posed a problem for a few sites in Europe, causing a team from the Netherlands to drop out. The team from Sweden, however, was enthusiastic about the late starting time.

[1] We supported the Sun C compiler, the Gnu C compiler, and the Sun (Berkeley) Pascal compiler.

3 Contest Results

The results of the contest are shown in Tables 1 and 2.[2] The top 10 overall teams are also shown in Table 3. No prizes were awarded, other than "bragging rights" to the winning team. The teams were divided into three divisions for scoring purposes:

I All team members are undergraduates or high school students.

II Current ACM rules:

- At least two undergraduates
- No member with more than two years in grad school
- No member with a Master's or Ph.D. degree

III Open (professors, staff, upper-level graduate students, others).

Teams are ranked first by the number of problems solved, and then by the amount of time used to solve the problems (not shown).

4 Implementation issues

The support software for the contest was written as C-shell (`csh`) scripts and ran under SunOS 4.1 on a Sun 4 server. At the judges' machine, the receipt, unpacking, compiling, running, and initial testing of program submissions was completely automatic. Careful specification of the problems allowed our software to easily test a submitted program's output for correctness. Only when a submission appeared to be incorrect was it necessary for a human judge to intervene and decide on a score. Concurrency control was built into the software to allow several people to act as judges or to answer questions. Recording and reporting the score (again, via e-mail) was also automated.[3]

Contestants used `csh` programs provided by us to register, submit programs for judging, and submit questions for clarification. Thus, some form of Unix was run by all teams participating in the contest. Portability of our scripts was a problem at only one or two sites. Sites also needed a fast electronic-mail connection to the Internet (in particular, to Duke University.)

Since the time taken to solve a problem determined how teams placed, we needed to develop a method for estimating this time that was fair to all teams. To be fair to teams that had a slow mail

[2] Teams that submitted no problems are omitted from these charts, since some "teams" were from invalid or duplicate registrations. Teams in a tie are given the same rank and listed alphabetically.

[3] Our contest software can be obtained by anonymous `ftp` to `cs.duke.edu` in the file `dist/misc/acm_contest/run-a-contest.tar.Z`.

Table 1: Results for Duke Internet Programming Contest

Division I: Undergraduates Only

Rank	Problems Solved	Institution	Team Name
1	5	Carnegie-Mellon University	Brand X
2	3	Williams College	Brain Lecture
3	2	St. Olaf College	
4	2	Carnegie-Mellon University	Geeks
5	2	Old Dominion University	
6	2	Acadia University	
7	2	Case Western Reserve University	
8	2	University of Alabama	
9	2	Williams College	Random Nerds
10	1	University of Sydney	Gonfaloniers
11	1	Armstrong State College	
12	1	Rice University	Rice Harricana
13	1	University of California, Los Angeles	Team 4
14	1	University of California, Los Angeles	Team 3
15	1	University of Missouri, Rolla	
16	1	Emory University	Team 1
17	1	University of British Columbia	Robocoders
18	1	Carnegie-Mellon University	KDR+XO
19	0	Augusta College	
19	0	Bucknell University	
19	0	Carnegie-Mellon University	Freshman Desklamp Team
19	0	Rice University	Death by 212
19	0	University of Pennsylvania	Ugrads
20	0	Carnegie-Mellon University	Nemesis

Division II: Current ACM Rules

Rank	Problems Solved	Institution	Team Name
1	4	Duke University	Team 1
2	3	Michigan State University	Students
3	3	University of Virginia	
4	3	New Mexico Tech	Team 1
5	3	Stanford University	
6	3	University of Sydney	Hons
7	2	University of Oregon	Team 1
8	2	California State University, Sacramento	
9	1	Emory University	Team 2

Table 2: Results for Duke Internet Programming Contest, continued

Division III: Open teams

Rank	Problems Solved	Institution	Team Name
1	6	University of Maryland	Defending National Champs!
2	4	University of Tennessee	
3	4	University of Maryland	Terrapins
4	4	Carnegie-Mellon University	Saint James squad
5	4	University of Maryland	Kardiak Papa and Kids
6	4	University of California, Los Angeles	Team 1
7	3	Georgia Tech	BuzzNUGians
8	3	University of Colorado, Boulder	
9	3	Sun Microsystems, Rocky Mountain	
10	3	University of Southwestern Louisiana	
11	3	University of Canterbury, New Zealand	/dev/null
12	3	University of California, Los Angeles	Team 2
13	3	Williams College	Berkshire BugBusters
14	3	University of Sydney	LUDs
15	3	Duke University School of Engineering	Faculty/Staff Team
16	3	University of Oregon	Team 2
17	2	Linkoping University, Sweden	
18	1	Trinity College	
19	1	Michigan State	Faculty
20	1	Florida International University	Team 1
21	1	University of North Carolina, Greensboro	
22	1	Florida International University	Team 2
23	0	Duke University	Team 2
23	0	Naval Postgraduate School	
23	0	New Mexico Tech	Team 2
23	0	Ohio State University	
23	0	Rice University	Next Month's Rent

Table 3: Top ten teams, overall

Rank	Problems Solved	Institution	Team Name
1	6	University of Maryland	Defending National Champs!
2	5	Carnegie Mellon University	Brand X
3	4	University of Tennessee	
4	4	University of Maryland	Terrapins
5	4	Carnegie Mellon University	Saint James squad
6	4	Duke University	Team 1
7	4	University of Maryland	Kardiak Papa and Kids
8	4	University of California, Los Angeles	Team 1
9	3	Michigan State University	students
10	3	Georgia Tech	BuzzNUGians

connection to Duke, we used the submission time, rather than receipt time, in scoring programs. Since we assumed that each site started the contest when their local system clock reached the appointed start time, the use of the submission time is valid since the only measure that affects scoring is the elapsed time taken to solve a problem. We used Universal Time (GMT) for all time values, to handle multiple time zones. We ignored clock skew; although a message occasionally appeared to arrive before it had been sent, this was not a significant problem.

We depended on the integrity of the contestants for the success of the contest. We used only simple and relatively minor mechanisms to prevent people from cheating or disrupting the contest. Since we presumed that the participants were taking part in the contest for enjoyment, we did not think that strong protection mechanisms were warranted (nor did we want to take the time needed to develop such mechanisms.)

Although most of the contest ran smoothly, there were a few small problems. One team's program submissions were incorrectly timestamped, and another team's submissions were lost in the mail, due to incorrect porting of the submission software. One team entered an invalid electronic mail address, causing all mail to their team (and all mail sent to all teams simultaneously) to be lost. Some computers' clocks were faster than ours, which caused the scoring program to incorrectly apply penalties in certain cases. All of these minor problems were repaired manually during the contest and did not affect the final outcome.

5 Performance

There are two performance issues to consider when running a real-time contest on the Internet. First, electronic mail is a slow (and highly variable) communication mechanism. We chose it because of its wide-ranging availability and ease of use. Any mechanism that was more complex than mail would have required much more programming and development on our part, and would have been much less portable. Since the point of the contest was to have fun, and not to compete for large prizes, we were only concerned that the communication delays were short enough to make the contest feasible, if not fair.

The cumulative distribution of the mail propagation delays is plotted in Figure 1. Although there were a few messages that took a long time to arrive, nearly all of them arrived in less than five minutes; some sites had a mail delay of under 15 seconds. The negative times come from sites whose system clocks were ahead of ours. Two long times (more than two hours) are not shown, and are due to a system crash at the sender's site.

The other performance issue is in judging: a significant number of programs must be scored quickly by a few judges to provide a quick turnaround time for the contestants. Quick turnaround of incorrect programs is important to enable teams to fix their program and resubmit it. Once a program arrived, we were able to judge it quickly, as shown in Figure 2. Most of the long judging times came either from a user-submitted program that ran a long time (we cut them off at five wall-clock minutes, since we knew there was a fast solution to all of our problems), or at the end of the contest when we were swamped with last-minute submissions. Using several judges simultaneously, we could receive, judge, and return several programs per minute. Further automation could improve the judging time.

6 Conclusion

We received many positive comments about the contest and are gratified that it was successful. We thank the Department of Computer Science at Duke University for allowing us to use the computing resources we needed to run the contest.

In all, our Internet programming contest was fun for all involved, and ran smoothly, considering the diversity of people and machines involved. Post-contest enthusiasm ran high and several institutions have expressed an interest in "hosting" the next Internet contest. We expect there to be another such contest sometime in the near future.

Acknowledgements

Many thanks to Lars Nyland and Dania Egedi for their help during the contest.

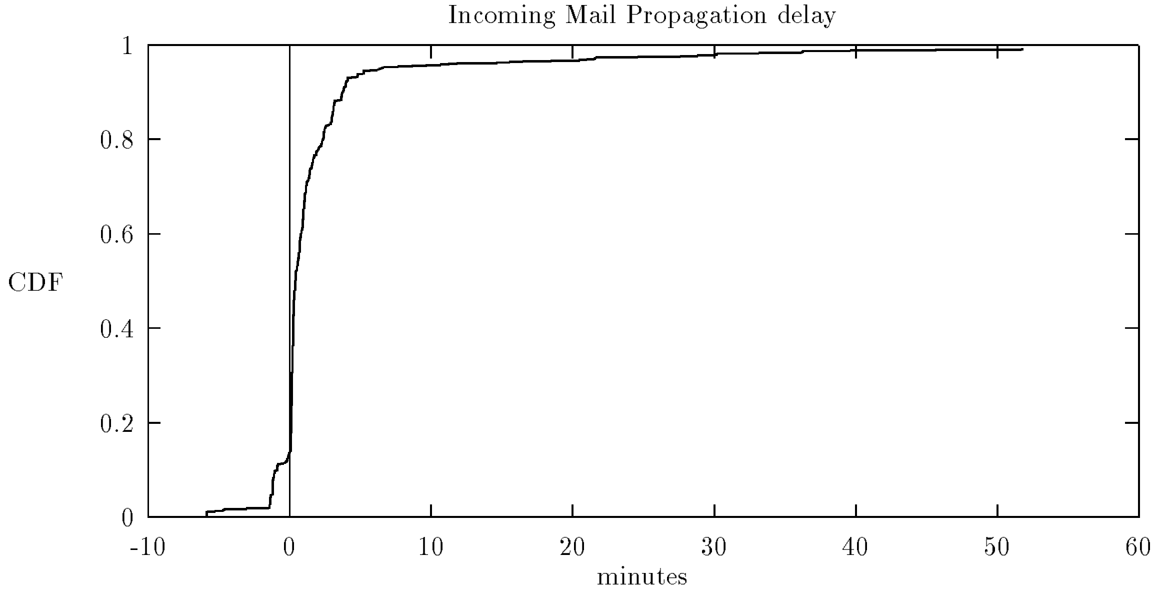


Figure 1: Time for a program or question submission to reach us. Over 90% of all mail reached us within five minutes. Two submissions took more than two hours (the sending machine crashed), and are not shown. Negative times are the result of clock differences.

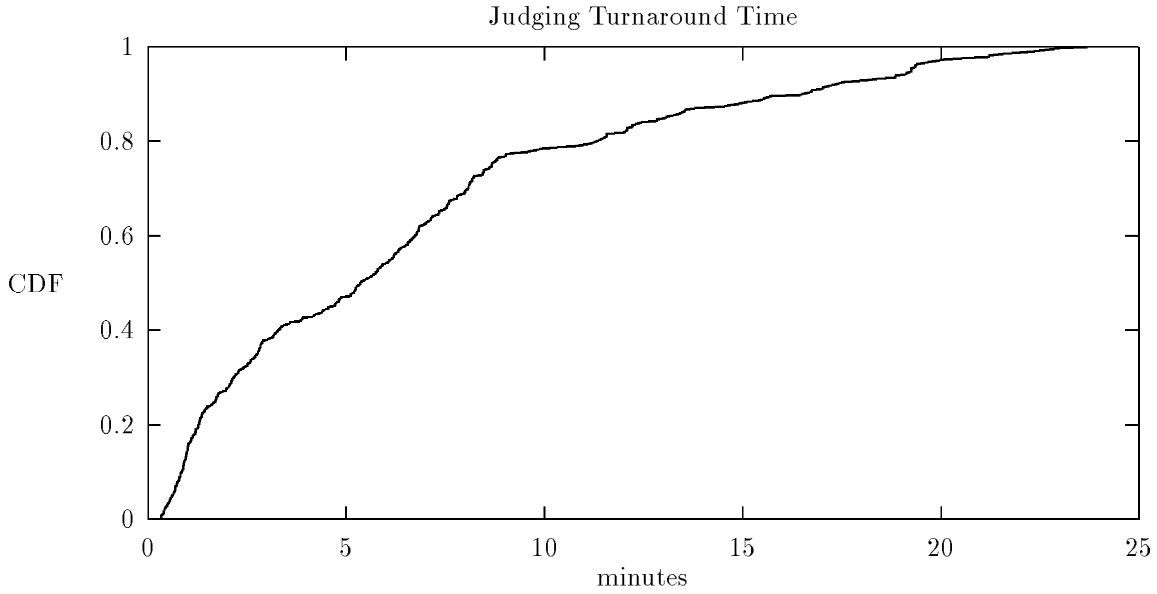


Figure 2: Time for us to judge a submitted program, once it arrived.

A The problems

A.1 The $3n + 1$ Problem

Consider the following algorithm:

1. input n
2. print n
3. if $n = 1$ then STOP
4. if n is odd then $n \longleftarrow 3n + 1$
5. else $n \longleftarrow n/2$
6. GOTO 2

Given the input 22, the following sequence of numbers will be printed

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value [1]. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers n such that $0 < n < 1,000,000$ (and, in fact, for many more numbers than this.)

Given an input n, it is possible to determine the number of numbers printed before and including the 1 is printed. For a given n this is called the *cycle-length* of n. In the example above, the cycle length of 22 is 16.

For any two positive integers i and j you are to determine the maximum cycle length over all integers between and including both i and j.

Number of teams attempting (submitting a possible solution to) this problem: 55. Number of teams solving this problem correctly: 40. Many teams had trouble when they incorrectly assumed $i \leq j$.

A.2 The Blocks Problem

The problem is to parse a series of commands that instruct a robot arm in how to manipulate blocks that lie on a flat table. Initially there are n blocks on the table (numbered from 0 to $n - 1$) with block b_i adjacent to block b_{i+1} for all $0 \leq i < n - 1$ as shown in the diagram below:

The valid commands for the robot arm that manipulates blocks are:

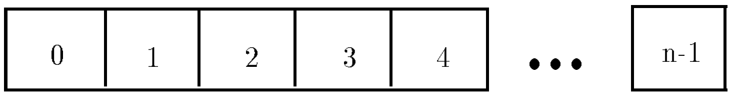


Figure 3: Initial Blocks World

- move a onto b

  where a and b are block numbers, puts block a onto block b after returning any blocks that are stacked on top of blocks a and b to their initial positions.

- move a over b

  where a and b are block numbers, puts block a onto the top of the stack containing block b, after returning any blocks that are stacked on top of block a to their initial positions.

- pile a onto b

  where a and b are block numbers, moves the pile of blocks consisting of block a, and any blocks that are stacked above block a, onto block b. All blocks on top of block b are moved to their initial positions prior to the pile taking place. The blocks stacked above block a retain their order when moved.

- pile a over b

  where a and b are block numbers, puts the pile of blocks consisting of block a, and any blocks that are stacked above block a, onto the top of the stack containing block b. The blocks stacked above block a retain their original order when moved.

- quit

  terminates manipulations in the block world.

Any command in which $a = b$ or in which a and b are in the same stack of blocks is an illegal command. All illegal commands should be ignored and should have no affect on the configuration of blocks.

Number of teams attempting this problem: 23. Number of teams solving this problem correctly: 18. Source: [2].

A.3 Ecological Bin Packing

Recycling glass requires that the glass be separated by color into one of three categories: brown glass, green glass, and clear glass. In this problem you will be given three recycling bins, each containing a specified number of brown, green and clear bottles. In order to be recycled, the bottles will need to be moved so that each bin contains bottles of only one color.

The problem is to minimize the number of bottles that are moved. You may assume that the only problem is to minimize the number of movements between boxes.

For the purposes of this problem, each bin has infinite capacity and the only constraint is moving the bottles so that each bin contains bottles of a single color.

Number of teams attempting this problem: 42. Number of teams solving this problem correctly: 35.

A.4 Stacking Boxes

Consider an n-dimensional "box" given by its dimensions. In two dimensions the box (2,3) might represent a box with length 2 units and width 3 units. In three dimensions the box (4,8,9) can represent a box $4 \times 8 \times 9$ (length, width, and height). In 6 dimensions it is, perhaps, unclear what the box (4,5,6,7,8,9) represents; but we can analyze properties of the box such as the sum of its dimensions.

In this problem you will analyze a property of a group of n-dimensional boxes. You are to determine the longest *nesting string* of boxes, that is, a sequence of boxes $b_1, b_2, \ldots, b_k$ such that each box b_i nests in box b_{i+1} $(1 \leq i < k)$.

A box D = $(d_1, d_2, \ldots, d_n)$ nests in a box E = $(e_1, e_2, \ldots, e_n)$ if there is some rearrangement of the d_i such that when rearranged each dimension is less than the corresponding dimension in box E. This loosely corresponds to turning box D to see if it will fit in box E. However, since any rearrangement suffices, box D can be contorted, not just turned. For example, the box D = (2,6) nests in the box E = (7,3) since D can be rearranged as (6,2) so that each dimension is less than the corresponding dimension in E.

Formally, we define nesting as follows: box D = $(d_1, d_2, \ldots, d_n)$ *nests* in box E = $(e_1, e_2, \ldots, e_n)$ if there is a permutation π of $1 \ldots n$ such that $(d_{\pi(1)}, d_{\pi(2)}, \ldots, d_{\pi(n)})$ "fits" in $(e_1, e_2, \ldots, e_n)$ i.e., if $d_{\pi(i)} < e_i$ for all $1 \leq i \leq n$.

Number of teams attempting this problem: 9. Number of teams solving this problem correctly: 6. Source: [3].

A.5 Arbitrage

Arbitrage is the trading of one currency for another with the hopes of taking advantage of small differences in conversion rates among several currencies in order to achieve a profit. For example, if \$1.00 in U.S. currency buys 0.7 British pounds currency, £1 in British currency buys 9.5 French francs, and 1 French franc buys 0.16 in U.S. dollars, then an arbitrage trader can start with \$1.00 and earn $1 \times 0.7 \times 9.5 \times 0.16 = 1.064$ dollars thus earning a profit of 6.4 percent.

You will write a program that determines whether a sequence of currency exchanges can yield a profit as described above, given a table of exchange rates.

To result in successful arbitrage, a sequence of exchanges must begin and end with the same currency, but any starting currency may be considered.

Number of teams attempting this problem: 8. Number of teams solving this problem correctly: 1. Source: [4].

A.6 The Skyline Problem

You are to design a program to assist an architect in drawing the skyline of a city given the locations of the buildings in the city. To make the problem tractable, all buildings are rectangular in shape and they share a common bottom (the city they are built in is very flat). The city is also viewed as two-dimensional. A building is specified by an ordered triple (L_i, H_i, R_i) where L_i and R_i are left and right coordinates, respectively, of building i and H_i is the height of the building. In the diagram below, the buildings shown on the left are from the triples

$$(1,11,5),(2,6,7),(3,13,9),(12,7,16),(14,3,25),(19,18,22),(23,13,29),(24,4,28).$$

The skyline, shown on the right, is represented by the sequence:

(1, 11, 3, 13, 9, 0, 12, 7, 16, 3, 19, 18, 22, 3, 23, 13, 29, 0)

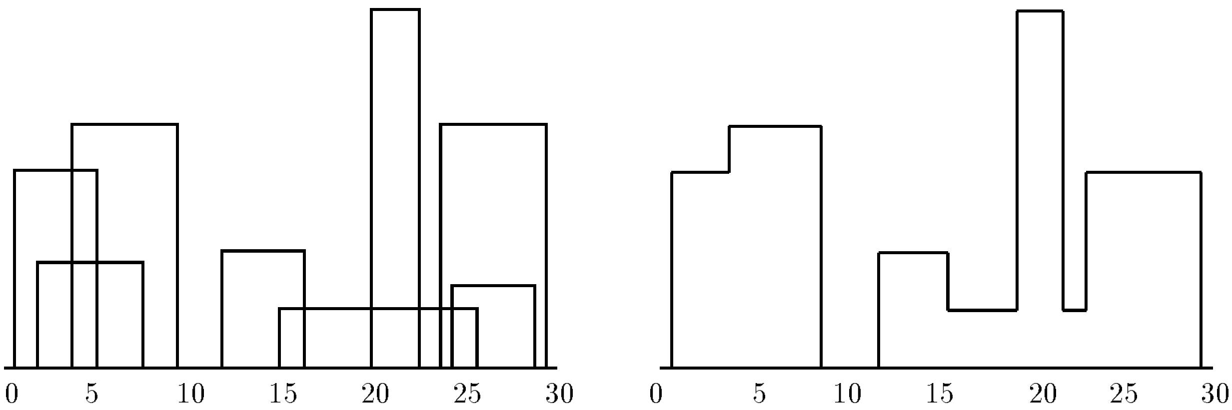
This represents the positions encountered while tracing the skyline from left to right, alternating x and y values.

Number of teams attempting this problem: 36. Number of teams solving this problem correctly: 18. Source: [5].

References

[1] Also known as Ulam's problem (although he apparently did not invent it). This is part of computer science folklore, and appears in many books.

[2] Clancy and Harrison, Spring 1990. Modified from a programming assignment in CS 60C, University of California at Berkeley.



[3] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press, 1990. Problem number 25-2.

[4] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press, 1990. Based on problem number 25-3.

[5] Udi Manber. *Introduction to Algorithms: A Creative Approach.* Addison-Wesley Publishing Company, 1989. Pages 102–103.