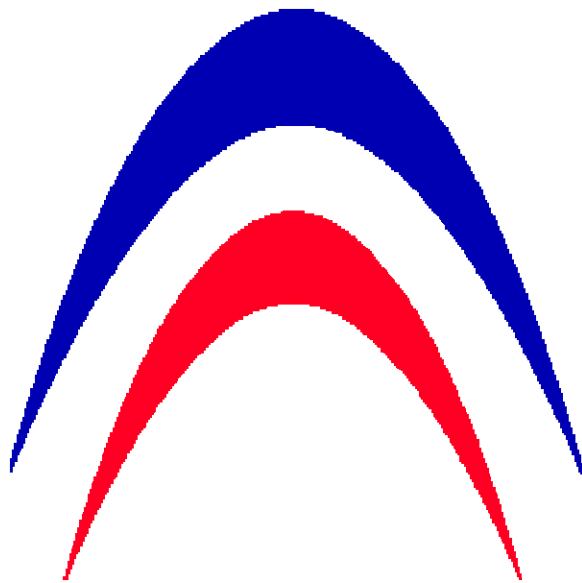


ALGORITHMS

An Approach Using Puzzles and Brainteasers

FOURTH EDITION



ALGANA ASSOCIATES

ALGORITHMS

An Approach Using Puzzles and Brainteasers

FOURTH EDITION

The exercises and applications presented in this book have been included for their instructional value. They have been tested but are not guaranteed for any particular purpose. The publisher neither offers any warranties or representations, nor accepts any liabilities with respect to the programs or applications.

Copyright © 2006 by Algana Associates.

All rights reserved. No part of this publication may be produced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States and in Great Britain.



CONTENTS

1. PUZZLES & BRAINTEASERS.....	5
1.1 INTRODUCTION	5
1.2 ABACI.....	6
1.3 BLOCKS & SHAPES	7
1.4 BOARDS & GRIDS.....	13
1.5 CHANCE & NATURE	18
1.6 LOGIC & MAZES.....	23
1.7 NUMBERS & WORDS	28
1.8 EXERCISES	32
1.9 BIBLIOGRAPHY & WEB REFERENCES.....	35
2. COMPRESSION	36
2.1 INTRODUCTION	36
2.2 HUFFMAN.....	37
2.3 JPEG	40
2.4 RUN-LENGTH.....	41
2.5 VARIABLE LENGTH.....	42
2.6 OTHER COMPRESSION ALGORITHMS.....	43
2.7 COMPARISON OF ASCII AND HUFFMAN.....	44
2.8 EXERCISES	45
2.9 BIBLIOGRAPHY & WEB REFERENCES.....	47
3. GRAPH ALGORITHMS.....	48
3.1 INTRODUCTION	48
3.2 DEPTH-FIRST SEARCH	48
3.3 BREADTH-FIRST SEARCH	49
3.4 DIJKSTRA.....	50
3.5 KRUSKAL.....	51
3.6 PRIM.....	51
3.7 COMPARISON OF DEPTH-FIRST AND BREADTH-FIRST SEARCH	52
3.8 EXERCISES	54
3.9 BIBLIOGRAPHY & REFERENCES	56
4. GRAPHICS	57
4.1 INTRODUCTION	57
4.2 BRESENHAM.....	58
4.3 DDA.....	59
4.4 MIDPOINT LINE	60
4.5 MIDPOINT CIRCLE	61
4.6 JULIA SETS	62
4.7 COMPARISON OF BRESENHAM AND DDA	63
4.7 CASE STUDY	65
4.8 BIBLIOGRAPHY & REFERENCES	66
4.8 BIBLIOGRAPHY & REFERENCES	66
5. NUMERICAL	67
5.1 INTRODUCTION	67
5.2 EUCLID.....	68
5.3 FIBONACCI.....	69
5.4 SIEVE OF ERATOSTHENES.....	71
5.5 COMPARISON OF NAÏVE AND EUCLID.....	72
5.6 COMPARISON OF FIBONACCI ITERATIVE, RECURSIVE AND MODIFIED RECURSIVE	74
5.7 EXERCISES	75
5.8 BIBLIOGRAPHY & REFERENCES	76

6. OPERATING SYSTEMS	77
6.1 INTRODUCTION	77
6.2 VARIOUS OS ALGORITHMS	77
6.3 CASE STUDY	79
6.4 BIBLIOGRAPHY & REFERENCES	80
7. SEARCHING	81
7.1 INTRODUCTION	81
7.2 SEQUENTIAL	82
7.3 BINARY	83
7.4 RED-BLACK TREES	84
7.4 RED-BLACK TREES	84
7.5 COMPARISON OF BINARY AND SEQUENTIAL	85
7.6 EXERCISES	87
7.7 BIBLIOGRAPHY & REFERENCES	89
8. SORTING	90
8.1 INTRODUCTION	90
8.2 QUICKSORT	91
8.3 BUBBLESORT	92
8.4 MERGESORT	93
8.5 INSERTIONSORT	94
8.6 COMPARISON OF BUBBLESORT AND QUICKSORT	95
8.6 COMPARISON OF OTHER SORTING ALGORITHMS	97
8.7 EXERCISES	98
8.8 BIBLIOGRAPHY & REFERENCES	99
9. STRINGS	100
9.1 INTRODUCTION	100
9.2 BOYER-MOORE	101
9.3 KNUTH-MORRIS-PRATT	102
9.4 PATTERN MATCHING	103
9.5 LEVENSCHTEIN'S ALGORITHM	104
9.6 COMPARISON OF BOYER-MOORE AND KNUTH-MORRIS-PRATT	105
9.6 EXERCISES	107
9.7 BIBLIOGRAPHY & REFERENCES	108
10. PAST EXAMINATION QUESTIONS & SOLUTIONS	109
11. GENERAL BOOKS ON ALGORITHMS	144
12. LIST OF WELL-KNOWN ALGORITHMS	145

1. PUZZLES & BRAINTEASERS

1.1 INTRODUCTION

The underlying ideas in many traditional recreational puzzles and challenging brainteasers can often be extended to areas much more than simple recreation. Indeed, they can find relevance in several branches of computing, engineering, management, mathematics and psychology. For example, the fundamentals of counting using ancient abaci have important applications in number systems in computer hardware, the arrangements of coloured blocks in Rubik's cube have interesting applications in combinatorial problems, the connections in popular grid puzzles have important analogies in management science and finding paths through mazes can relate to algorithms used in both robotics and databases. Certainly, the history of puzzles, dating back thousands of years, plays an important role in the interdisciplinary approach of this text book.

In this first section, the nature of puzzles and their solutions are described with an objective to highlight algorithms encountered in later sections. Categories include abaci, blocks, boards, chance, grids, logical, matches, mazes, numbers and words. Most of them can be explored interactively in the puzzles and brainteasers sections at the Algana¹ website www.algana.co.uk.

We will distinguish between a puzzle and a brainteaser as follows. A puzzle in its general sense tends to be an interesting solvable problem whose solution is normally achievable in a reasonable time and with reasonable effort. So examples could include playing tic tac toe, searching through a small maze or finding an anagram of a word. Often we can find appropriate solutions to puzzles such as these without the need for special aids such as a calculator, computer or machine. We will probably not count pencil and paper in this regard!

On the other hand, the term brainteaser is normally associated with a solvable problem whose solution might only be achievable in a relatively long time with significant effort, possibly with the support of a computer or machine. So examples could include playing multi-dimensional tic tac toe, searching through a very large maze or finding *all* of the anagrams of a long word.

Certainly, in either case, the problem in question is usually solvable and the step-by-step procedure for solving it is generally called a *solution algorithm*. However, it should be born in mind that there exist many interesting problems in computing and engineering for which we cannot say for sure if a solution exists or not, primarily because a solution has not yet been found². For instance, in number theory, the so-called *twin primes* include pairs such as (5,7) or (11,13) or (17,19) because each pair differs by precisely 2. However, to date no-one knows for sure if there exists a finite or infinite number of twin primes.

Most readers will recognise several of the puzzles included in this section as recreational. Many will have been encountered in school playgrounds or in toy stores. However, it is important to appreciate the development of *strategies* for solving them. These will probably be less recognizable. They can be generally defined in the context of *game theory* as one-person games or so-called *games against nature*. Also, the extension of the puzzles from the realms of recreation to a source of serious technical investigation needs to be thoroughly appreciated for an intrinsic understanding of subsequent sections.

¹ "ALGANA" is a contraction of the words "ALGORITHM" and "ANALYSIS".

² For a list of famous unsolved problems, see <http://mathworld.wolfram.com/UnsolvedProblems.html>

1.2 ABACI

The *abacus* is possibly the world's first calculating device. The word can be traced to the Phoenician *abak* (sand) and from the Hebrew *avak* (dust). There is evidence to suggest that ancient civilisations used a flat surface with sand spread evenly over it as a refreshable tool for both writing and counting and this could account for the connection.

Early abaci had grooves for small pebbles and then wires on which counters could freely move back and forth. Each wire corresponded to a digit in a positional number system. For many centuries, the Greeks and Romans, followed by the Western Europeans in the Middle Ages and early Renaissance, calculated on devices with this type of place-value system in which zero was represented by a wire. Yet the written notations did not have a symbol for zero until it was borrowed by Arabs from Hindus and eventually introduced into Europe in about 1200 by Leonardo Fibonacci of Piza in his *Liber Abaci* (The Book of Abacus). Clearly, at that time counting with abaci was considered so convenient and straightforward.

In contrast, the Chinese abacus consisted of a wire grid in a wooden frame with vertical rows of beads strung on wires (see figure 1). A centre bar divided the upper rows of two beads each from the lower rows of five beads. Starting from the right and moving to the left, one could designate a ones column to begin counting. The next column represented the tens column, the third the hundreds column, and so on. The counting process consisted of pushing beads toward the centre bar. Beads in the upper deck have a value of 5. Beads in the lower deck have a value of 1. Instead of counting the fifth unit by pushing the fifth bead up to the centre bar, we can substitute one of the top beads, by pushing it down toward the centre to represent the number five. In the hands of an experienced abacus user, this was (and still can be) done swiftly in an effortless, continuous motion.

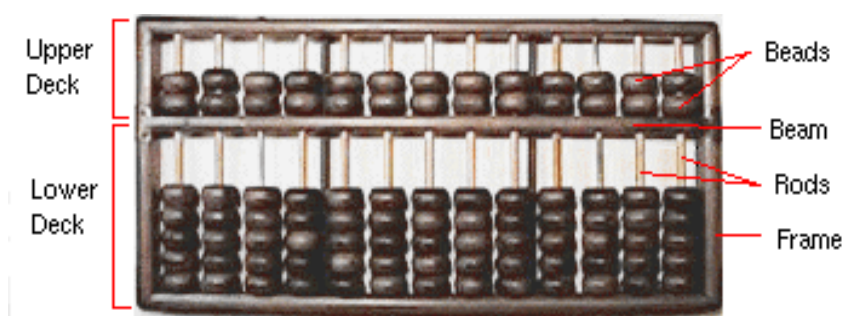


Figure 1. The frame, beam, beads and rods in the upper and lower decks of a typical Chinese abacus³.

Certainly, the abacus is still in use today by shopkeepers in Asia and elsewhere. The use of the abacus continues to be taught in Asian schools, and in several schools in Western Europe. Visually-impaired children are taught to use the abacus where their sighted counterparts would be taught to use paper and pencil to perform calculations. One particularly interesting use of the abacus is in the teaching of children to perform simple mathematics, especially multiplication; the abacus is an excellent substitute for memorization of multiplication tables, a particularly arduous task for young children. The abacus also continues to be an excellent tool for teaching alternative base numbering systems since it easily adapts to any base.

³ Source: An interactive link can be found in the *puzzles-abaci-Chinese abacus* section at the algana website.

1.3 BLOCKS & SHAPES

Blocks & shapes puzzles can be characterised as object-placement problems where the objects might be cubes, disks, dominos, rocks or tetrominos. The primary common feature is the placement of the objects according to some scheme or strategy. Specific examples of block puzzles include Asteroids, jigsaws, Rubik's cube⁴, sliding blocks, Tetris, Towers of Hanoi and many more.

Asteroids

The Asteroids game originated in about 1977 as an arcade game developed by Atari Corporation. The game developed the previously-successful space invader game into two-dimensions. A player's task was to blast incoming asteroids using repeated firepower, achieving the highest score possible before three lives were lost due to exploding asteroids (see figure 2). Notwithstanding many of the derivatives emanating from it, historical data confirms that Asteroids was probably the most successful arcade games of the twentieth century based on earnings.

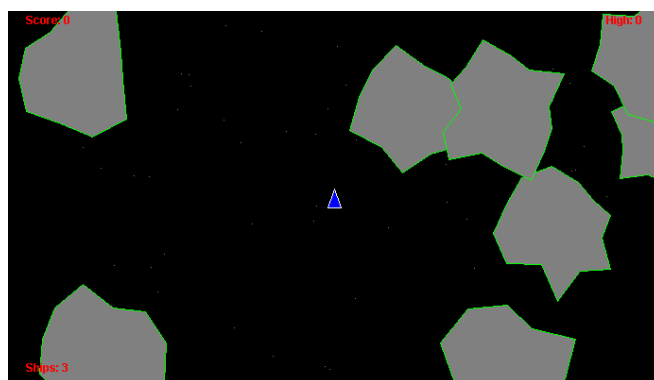


Figure 2. The spaceship blasts the incoming asteroids until all are obliterated. The game then moves to the next (more difficult) level until all three lives are lost, yielding a player's final score⁵.

All popular arcade games possess the common attributes of hook-ability, simplicity and survivability and the fact that Asteroids was relatively simple to understand how to play but so difficult to survive very long in terms of game time, made it a highly profitable game. Even experienced players would find it hard to last more than two minutes in a single game yet there would often be a queue of players waiting to feed in coins to play the next game.

Problem. Suppose that the game begins at each level with eight asteroids; four asteroids which break down into one sub-asteroid when hit by a bullet while the other four asteroids which break down into two sub-asteroids. Assume that each sub-asteroid is obliterated when hit by a bullet. How many bullets are needed to reach the second level of the game?

Solution. Twenty bullets are needed - eight for the one-break asteroids and twelve for the two-break asteroids.

⁴ Rubik's cube, copyright ©1998-2007, Seven Towns, Ltd. Source: An interactive link can be found in the *puzzles-blocks-Rubik's cube* section, at www.algana.co.uk.

⁵ Source: An interactive link can be found in the *brainteasers-blocks-solid asteroids* section at the algana website.

Jigsaws

The first jigsaw puzzle was introduced around 1760 by Englishman John Spilsbury, a London mapmaker. He mounted a map on a sheet of hardwood and cut around the borders of countries using a fine-bladed saw. The result was an educational pastime designed as an aid in teaching children geography. The idea caught on in schools and, until about 1820, jigsaw puzzles remained primarily educational tools.

Jigsaw puzzles further developed with illustrations glued or painted on the front of the wood and pencil tracings of where to cut were made on the back. These pencil tracings can still be found on some of these older puzzles. Cardboard versions of jigsaw puzzles were first introduced in the late 1800's, and were primarily used for children's puzzles (see figure 3). Thus, in the early 1900's, both wooden and cardboard jigsaw puzzles were widely available. Wooden puzzles still dominated, as manufacturers were convinced that customers would not be interested in "cheap" cardboard puzzles. Of course, a second motivation on the part of manufacturers and retailers of jigsaw puzzles was that the profit from a wooden puzzle, was far greater than for a cardboard jigsaw puzzle.



Figure 3. A simple jigsaw in which the six pieces (right) must be placed to form the image of cupid (left).

Jigsaws are a good example of a puzzle which requires placement in any order rather than adherence to a particular order. Clearly, once a jigsaw piece has been placed in its correct position, it does not essentially matter if it was placed first or last, as long as it is in the right place. This would not apply, for instance, in a maze search where following a particular ordered path might be critical to the solution. Of course, naively placing pieces at random is one strategy for solution, albeit longwinded. However, experienced jigsaw players are well aware that a speedier solution can be found by first correctly identifying and placing corner and side pieces, then the remaining pieces.

Problem. How many different ways are there of randomly placing six pieces into six places? Recognising the two straight edges of the four corner pieces and placing them, how many different ways do we have to randomly place the remaining two pieces? Accordingly, how many placements could we have saved by using a ‘first-place-corners’ strategy?

Solution. There are 6^6 (= 46656) different combinations of randomly placing six jigsaw pieces into six places? Having recognised the four corner pieces and placed them, there are just 2 ways of randomly placing the remaining two pieces. Therefore, 46654 placements would have been saved by using a ‘first-place-corners’ strategy - a dramatic saving.

Rubik's Cube

Hungarian Ernő Rubik invented the *Rubik's Cube* in 1974, when he discovered it was difficult to realign the colours to match on all six sides of a coloured spinning cube. He was not sure if he would ever be able to return his invention to its original position. Since then the standard 3 x 3 version (see figure 4, left) has been popularised worldwide.

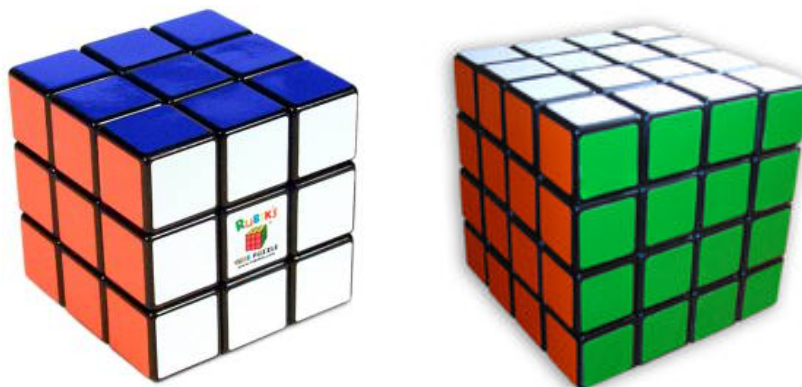


Figure 4. The standard 3 x 3 Rubik's cube puzzle (left) has six large faces, each of a different colour and is comprised of smaller spinning cubes. The 4 x 4 x 4 version (right) is called Rubik's Revenge.

It can be shown that there are approximately 10^{19} different arrangements of the standard Rubik's cube⁶ and of course only one of those is the desired solution. So it is not surprising that many enthusiasts have struggled for hours to solve a scrambled cube without ever finding a solution. However, experienced players are well aware that a speedier solution can be found which involves only about 30 turns⁷. Part of the strategy recognises that the smaller faces can be grouped respectively as corner-cube, side cube or centre-cube faces. These groups can then be considered to follow different rules.

In general, an $N \times N \times N$ spinning cube will require different solution strategies as the value of N increases. The so-called *Rubik's Revenge* is the case when $N = 4$. Developed by Péter Sebestény, it was briefly called the Sebestény Cube until a last-minute decision changed the puzzle's name to attract fans of the original Rubik's Cube⁸. The *Professor's Cube* (also known as *Rubik's Professor*) is a $5 \times 5 \times 5$ version developed by Udo Krell.

Problem. In the 3 x 3 version of the Rubik's cube, we only ever fully see nine similarly coloured faces of the smaller blocks and since there are six different colours, this makes 54 visible faces in total. Classify these as corner-cube, side cube or centre-cube faces, ensuring that they total 54.

Solution. There are eight corner-cubes each with three faces (total 24), twelve side-cubes each with two faces (total 24) and six centre-cubes each with one face (total 6), making a total 54 visible faces.

⁶ See for instance <http://williambader.com/museum/cubes/cubes.html>

⁷ See for instance http://en.wikibooks.org/wiki/How_to_solve_the_Rubik%27s_Cube

⁸ Solution strategies for Rubik's Revenge can be found at http://en.wikipedia.org/wiki/Rubik's_Revenge

Sliding Blocks

Sliding block puzzles typically involve sliding blocks around a two-dimensional frame similar to the ones shown in figure 5. One space in the frame is normally vacant to allow sliding to take place and the idea is to rearrange blocks to form a particular pattern⁹.

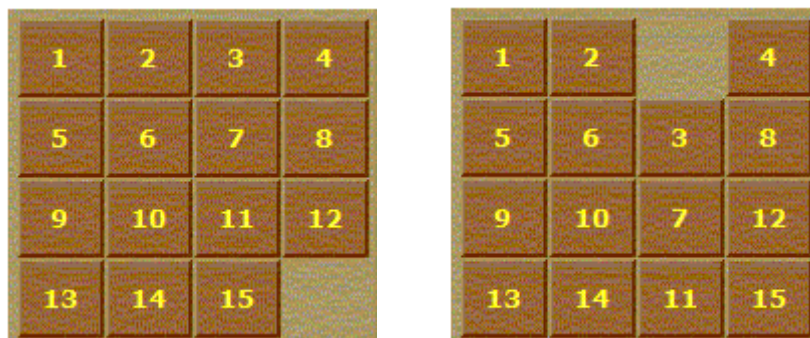


Figure 5. A simple sliders puzzle is shown in which the numbered blocks are ordered (left) and scrambled (right). The object is to slide blocks in the scrambled frame to recover the ordered frame arrangement.

It should be realised that not all randomly scrambled frames similar to figure 5 (right) can be rearranged to achieve the ordered frame in figure 5 (left).

The puzzle that really established huge popularity of the sliding block puzzles was the *15 Puzzle*. Invented in 1878 in the USA, its popularity spread through many other countries. The puzzle comprises 15 numbered square blocks which are almost ordered as in figure 5 (left) but with the 14-block and the 15-block transposed. The object was then to rearrange them into their correctly ordered sequence from 1 to 15 simply by sliding them around within the frame. It was marketed under various names, including *The Boss Puzzle*. The craze for this puzzle lasted about three years and several million people played it. Supply could hardly keep up with the demand, and it was reported that one New York store alone was selling more than 30,000 copies per day, very much like the craze for Rubik's Cube which took place 100 years later. Undoubtedly, one of the reasons for its longevity rested in the fact that this was precisely one of those scrambles which could never lead to a successful solution!

Problem. Show how the scramble in figure 5 (right) can be solved in just four slides.

Solution.



⁹ An interactive sliders puzzle can be found at the puzzles-slidingblocks-slip'n&slid'n link at Algana.

Tetris

Tetris is a falling-blocks puzzle video game designed by Alexey Pajitnov in 1984 while working in Moscow at the Academy of Science of the USSR. He is thought to have derived its name as a contraction of the Greek prefix *tetra*, as all of the pieces contain four segments, and *tennis*, one of his favourite sports. Nowadays, Tetris or one of its many variants is available for nearly every video game console and computer operating system in the world. It has even appeared as part of an art exhibition on the side of Brown University's sciences library and consistently appears on lists of greatest video games of all time.

As each four-segment object (in geometry, called a *tetromino*) drops with a particular speed, a player can move it either horizontally or rotationally (see figure 6). The main points are scored whenever a horizontal row of blocks is completely filled and it is then removed so that all blocks above it slip down to fill the empty row. The game progressively becomes harder as the dropping speed increases with time; eventually yielding the player's final points score.

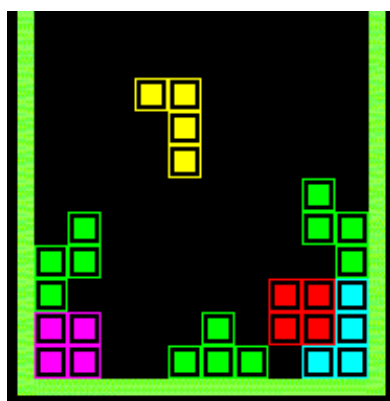
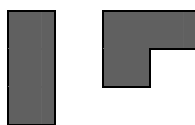


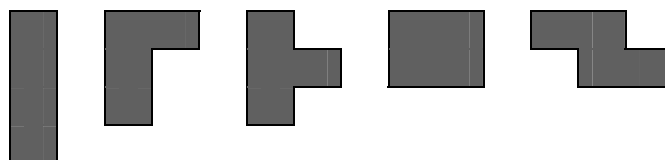
Figure 6. The initial stages in a game of Tetris are shown, where six tetrominos have already fallen and one is currently falling.

Problem. A “*n*-omino” is a generalization of the well-known domino to a collection of *n* squares of equal size arranged with coincident sides. Free *n*-ominoes can be flipped, so mirror image pieces are considered identical. A sketch of the two distinct free 3-ominoes is shown below.



Sketch the five distinct free 4-ominoes.

Solution.



Towers of Hanoi

The French mathematician, Edouard Lucas, is generally credited with inventing the Tower of Hanoi puzzle, also referred to as the Tower of Brahma, in 1883. He was supposedly inspired by a legend telling of a Hindu temple where the pyramid puzzle might have been used for the mental discipline of young priests. Legend says that the priests in the temple were given a stack of 64 gold disks, each one a little smaller than the one beneath it. Their assignment was to transfer the 64 disks from one of the three poles to another, with proviso that a large disk should never be placed on top of a smaller one.

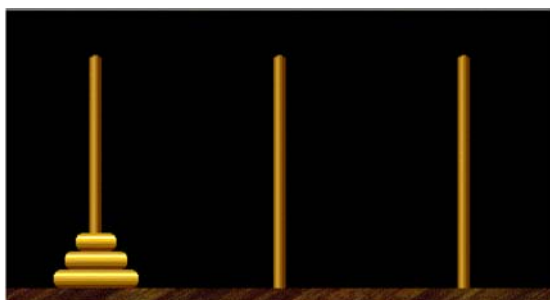


Figure 7(a). The Towers of Hanoi puzzle with three disks and three pegs.

A simpler version comprising three disks and three poles is shown in figure 7(a). This is solvable in seven moves and the moves are shown in the schematic in figure 7(b) below.

START MOVE 0	
MOVE 1	
MOVE 2	
MOVE 3	
MOVE 4	
MOVE 5	
MOVE 6	
FINISH MOVE 7	

Figure 7(b). A step-by-step solution of three-disk Towers of Hanoi.

Problem. In how many moves can the four disk version of the puzzle be solved? ¹⁰

Solution. Fifteen (for an animated solution, click ‘autosolve’ in the interactive puzzle).

¹⁰ An interactive Towers of Hanoi puzzle and step-by-step solution can be found at the *puzzles-blocks-towersofhanoi* link at Algana.

1.4 BOARDS & GRIDS

Board & grid puzzles can be characterised as arrangement-problems in a two-dimensional plane, where the plane might be a chess board, peg board, paper sketch or computer screen. The primary common feature is the rearrangement of objects in the plane to produce a particular pattern or end point. Specific examples of board & grid puzzles include chess, Fiver, pegs, Tactix, Tic Tac Toe and many more.

Chess

Probably the most famous popular board games include chess, backgammon, scrabble, monopoly and more. All have a rich history. The fact that there are so many different ways of playing a game of chess has contributed to its worldwide appeal. This is one reason why chess experts rely on applying strategies rather than trying to calculate every possible move. Interesting chess-related problems include N-queens, knights' tours, queens versus knights and board coverings. Queen chess pieces can attack horizontally, vertically and diagonally. The so-called N-queens problem requires placement of N queens on the board so that they do not attack each other. Figure 8 (right) shows a placement of five queens in a standard chess board in such a way that they do not attack each other.

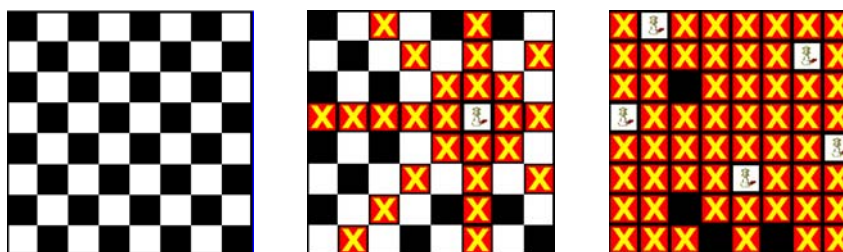


Figure 8. A clear standard 8 X 8 chessboard (left); a one-queen placement (middle) with squares marked with an “X” indicating lines of attack from the queen; a successful non-attacking five-queen placement (right).

Knight's tours are paths made by a knight as it traverses a chess board. Particularly interesting are those tours which pass through all sixty four squares without revisiting a square. Many exist and figure 8 shows one, having the additional property that it starts and ends at the same square.

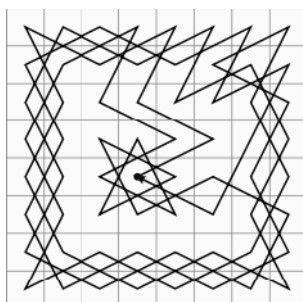


Figure 9. A closed knight's tour.

Problem. Covering a standard chess board with thirty two dominos is straightforward; all placed vertically, all placed horizontally and many combinations of both. Given a modified board with two diagonally-opposite corners removed, is it possible to find a covering using thirty one dominos?

Solution. No. For the simple reason that any domino covering would need to simultaneously cover both a black and a white square, but two squares of the same colour have been removed.

Fiver

Fiver is named after a character in *Watership Down* and the objective is to change the colour of all counters in a grid by clicking on them. Clicking on a counter with the mouse changes its colour, i.e. a white piece will turn black and a black piece will turn white. Also the counters above, below, to the right and to the left will also change the colour according to this rule¹¹.

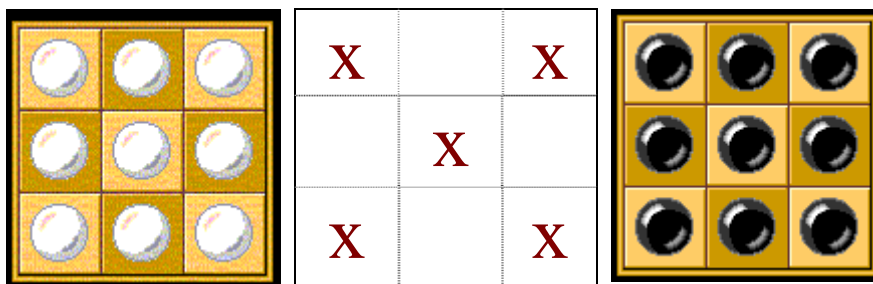


Figure 10. The initial position of the 3 x 3 Fiver puzzle (left); the places to click (centre) and the result (right) where all counters have changed colour.

Handheld toys based on a similar idea have been popularised by Tiger Toys¹² in the *Lights Out* series, including *Lights Out Deluxe* and *Lights Out Cube*. Symmetry is an interesting feature of solutions for larger versions of the puzzle. For consider the solution shown in figure 11 (left) for the 4 x 4 Fiver. A second four-click solution is available (shown right), which is a symmetrical reflection of the first solution about the leading diagonal. Furthermore, both solutions are rotationally symmetrical to each other. We shall see later how recognising various types of symmetry in a problem often leads to an efficient solution!

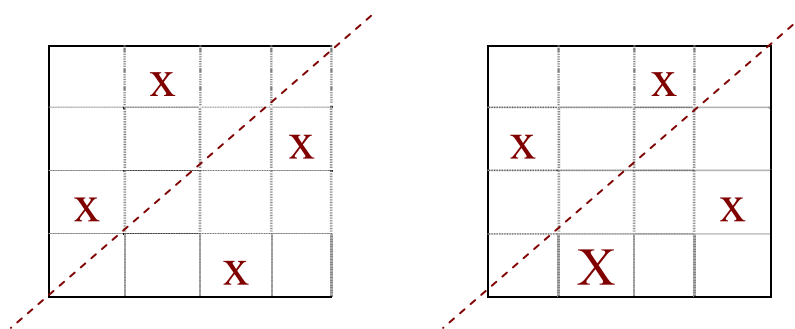


Figure 11. Two solutions of the 4 x 4 Fiver puzzle, symmetrical reflections of each other in the leading diagonal.

Problem. Check that the following represents a solution to the 5 x 5 version.

			X	X
X	X		X	X
X	X	X		X
	X	X	X	
X		X	X	

Solution. Use the interactive fiver puzzle.

¹¹ An interactive fiver puzzle can be found at the [puzzles-boards-fiver](#) link at Algana.

¹² Copyright Tiger Toys © 2007 Hasbro.

Pegs

There is varied account of the history of the *pegs puzzle*, also called *solitaire*. The French engraving (figure 12, left) shows Princess de Soubise (Anne Chabot de Rohan, 1663-1709) playing the game around the year 1697. Pieter van Delft & Jack Botermans¹³ suggest that the game was well-known by the time the engraving appeared, having reputedly been invented years earlier by a French nobleman while imprisoned in the Bastille. In desperation of boredom, this nobleman is said to have devised the game. However, since the famous German mathematician Leibnitz wrote about and described the puzzle in 1710, it is most likely to have originated sometime in the mid-seventeenth century. Several commercial versions have been created over the years¹⁴



Figure 12. Left: An engraving of Princess de Soubise playing solitaire, circa 1697. Right: *Puzzle-Peg*, 1929, Lubbens & Bell Manufacturing Company.

As the name *solitaire* would suggest, it is a one-person game where the goal is to repeatedly remove a peg from a pegboard by *jumping* with another peg; this removes the jumped peg (similar to checkers jumps). Only horizontal and vertical jumps are allowed. The game is over when no more jumps are possible and the game is successfully completed when all the pegs, except one, have been removed from the board.

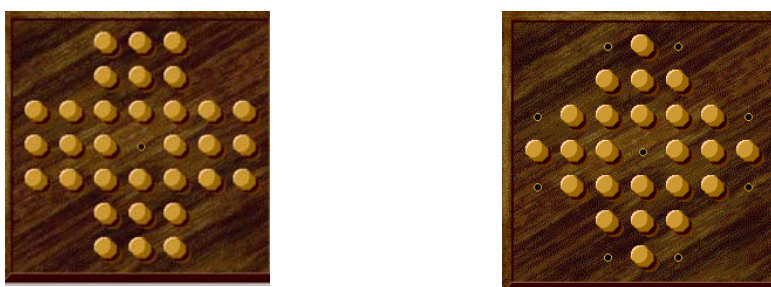


Figure 13. The pegs puzzle comprises several variations; *solitaire* and *diamond* are shown here.

Problem. Use ‘autosolve’ at the *puzzles-boards-pegs* link at Algana to check a thirty-one step solitaire solution and a twenty-three step diamond solution.

¹³ van Delft, P. & Botermans, J., *Creative Puzzles of the World*, Harry N. Abrams, Inc., 1978, p. 170.

¹⁴ A selection of boxed puzzles can be seen at *The Elliott Avedon Museum and Archive of Games*, University of Waterloo, Ontario, Canada.

Tactix

TacTix, created by Danish inventor Piet Hein, is a variation of Nim, one of the worlds's most analysed mathematical games.

Variants of Nim have been played since ancient times. The game is said to have originated in China since it closely resembles the Chinese game of "Jianshizi", or "picking stones" but the earliest European references date back to the beginning of the 16th century. Its current name was coined by Charles L. Bouton of Harvard University, who also developed an extensive theory of the game in 1901. The name is probably derived from German *nimm!* meaning "take!". Some have noted that turning the word NIM upside-down and backwards results in WIN. Nim is usually played as a *misère* game, in which the player to take the last object loses.

In 1951, a Nim-playing robot, called Nimrod, was exhibited at the Festival of Britain, and later at the Berlin trade fair. The story goes that the machine was so popular that spectators entirely ignored a free bar at the other end of the room!

There are many games closely related to Nim and three notable ones are Tactix, Last Match¹⁵ and Nimbi. Tactix is a game where one person plays against the computer. The player removes counters from the board, and the goal is to force the computer to remove the last counter. On any turn, it is permissible to remove counters from either a single row or a single column. The counters must be on adjacent squares, and must be as few as one or at most four. In the interactive version at the [puzzles-grids-tactix](#) link, one can use mouse clicks to remove the required number of counters from the board, and then press the 'move' button. The computer will make its move automatically after yours.

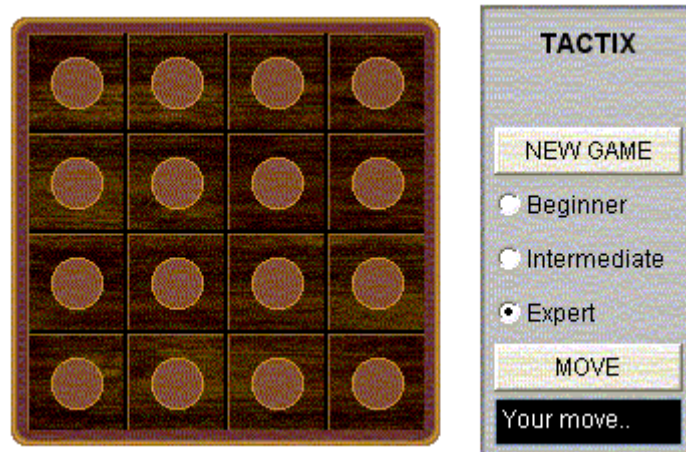


Figure 14. Tactix can be set to play an 'expert' computer, in which case a perfect game will be required to win.

Problem. Is it possible to move first and win against an 'expert' computer?

Solution. No. It has been proven that the player of Tactix who moves second can always win, if he or she plays perfectly. At the Expert level, the Tactix program will indeed play a perfect game. So a player wishing to win at the Expert level must make the computer move first!

¹⁵ The *Last Match* puzzle can be played at the [puzzles-matches-lastmatch](#) link at Algana.

Tic Tac Toe

Tic Tac Toe dates back to Roman times when it was known as *Terni Lapilli*. Grids have been found scratched and etched into stone surfaces all over the ancient Roman empire. More recently the game was known as *Three Men's Morris* and Three Men's Morris boards can be found in cathedrals in England dating back to 1300. In Europe, the game is more frequently called *Noughts and Crosses*.

The most popular form these days is played by one player against the computer; one is assigned “X” and the other “O”. Marks are alternately placed in an open board of squares which in a standard game is two-dimensional 3 x 3 but in other guises can be N x N or even multidimensional¹⁶. The object of the standard game is to get straight X's or O's in a row horizontally, vertically, or diagonally. Normally, the best play is to go first and choose the centre square.

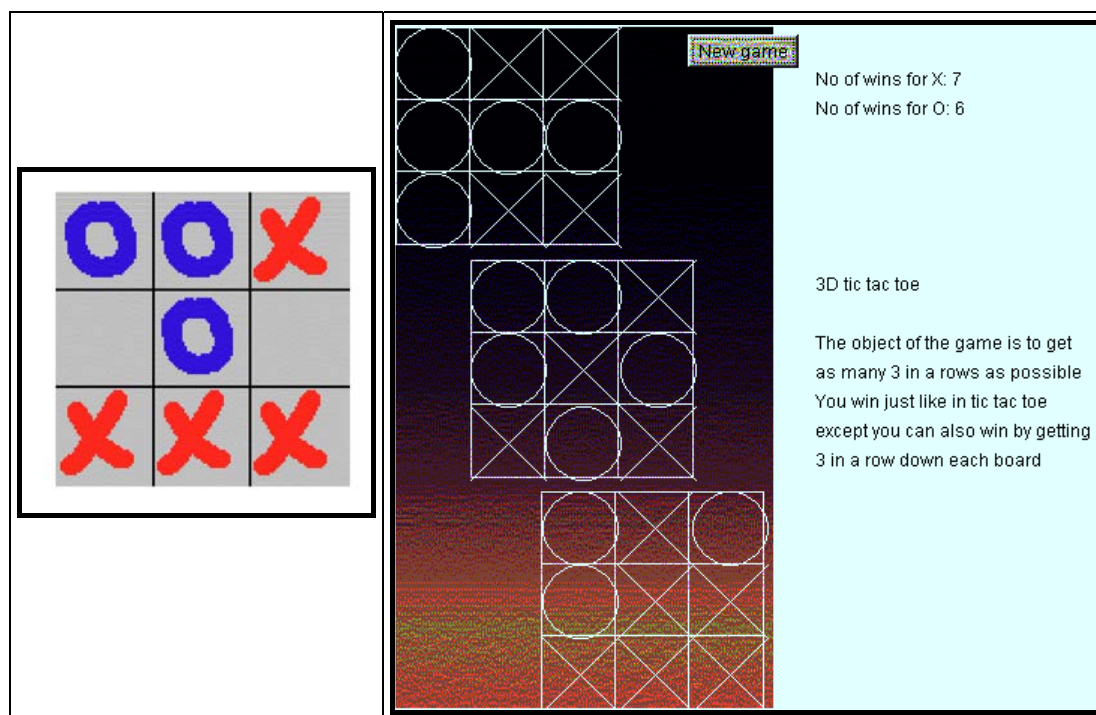


Figure 15. The two-dimensional 3 x 3 Tic Tac Toe puzzle (X wins) is shown left. The three-dimensional 3 x 3 x 3 Tic Tac Toe puzzle (X wins 7 vs 6) is shown right.

In the three-dimensional version shown in figure 15 (right), the three two-dimensional frames should be visualised as one on top of the other to produce an imaginary cube. Not surprisingly, research mathematicians have extended the idea to multi-dimensional versions of Tic Tac Toe¹⁷.

Problem. Identify and visualise the seven X wins and six O wins in figure 15 (right).

¹⁶ Three-dimensional Tic Tac Toe can be played at the brainteasers-boards-tictactoe3D link at Algana.

¹⁷ See, for examples, works of Jozsef Beck at Rutgers University or Dean Mah at Athabasca University.

1.5 CHANCE & NATURE

Chance & nature puzzles can be characterised as problems where probability plays a vital role. It can be argued for instance that naturally-encountered situations, such as genetic characteristics, physical health, likelihood of accidents, and odds of winning a lottery are all chance-related situations in which we can find ourselves. Specific examples of chance & nature puzzles include cards games, dice games, random problems, slot machines, magic tricks and many more.

Cards

One-person card games include solitaire, klondike, canfield, spider and others. One-person-versus-the-computer card games include blackjack and ziginette. Of course, if we have a guaranteed chance of winning any game, this is considered to be 100% chance. On the other hand, if we have precisely no chance of winning, this is considered 0% chance. Normally, most interesting positions of any game can be considered to have a chance somewhere between 0 and 100.

Solitaire or patience¹⁸ involves dealing cards from a shuffled deck into a prescribed arrangement on a tabletop, from which the player attempts to reorder the deck by suit and rank through a series of moves transferring cards from one place to another under prescribed restrictions. Their origins are largely unknown but some puzzlers think these kinds of games are French in origin since early English language books about patience games refer to French literature. Some of these games include *La Belle Lucie*, *Le Cadran*, *Le Loi Salique*, *La Nivernaise* and others.

In blackjack, the player competes against the computer to get a hand as close to 21 points as possible¹⁹. An ace card can optionally count as either one or eleven points while a picture card counts as ten points. American Professor Edward O. Thorp refined basic strategies of card counting techniques, publishing *Beat the Dealer*, a book on the 1963 New York Times best seller list.

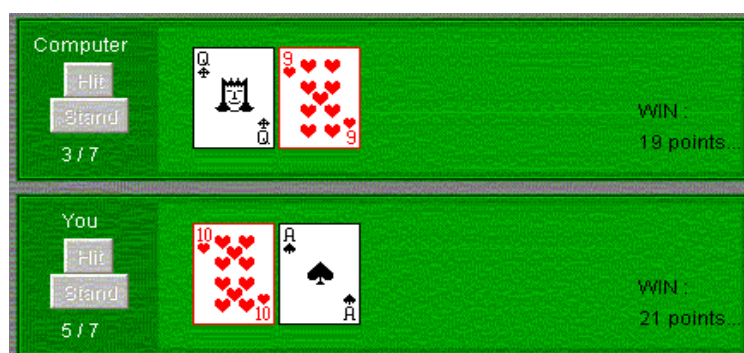


Figure 16. A game of blackjack is shown in which the computer is presently losing and the player's current hand is 'blackjack', viz. a ten-point card and an ace.

Problem. In a fair deal from a deck of fifty two cards, the chance of being dealt an ace is 4 out of 52, i.e. about 7%. Show that the chance of being dealt a 'blackjack' hand is approximately 5%.

Solution. A blackjack can be a ten-point card followed by an ace or vice versa. Now the chance of being dealt an ace is $1/13$ and the chance of this being followed by a ten-point card (from the remaining fifty one cards) is $16/51$, so the chance of both occurring is $(1/13)$ times $(16/51)$, i.e. $16/663$. The chance of being dealt a ten-point card followed by an ace is clearly also $16/663$. Hence, the chance of a blackjack is $32/663$, which is approximately 5%.

¹⁸ The games are generally referred to as "patience" in Europe or "solitaire" in North America.

¹⁹ An interactive version of blackjack can be found at the Algana [brainteasers-chance-blackjack](#) link.

Dice

One-person-versus-the-computer dice games include dice solitaire, craps, shibala and yahtzee.

Dice are often used in games of chance under the assumption that the die is fair²⁰. Craps is a game of several rounds where the player rolls two dice and if the total of the two dice is 2, 3, 7, 11 or 12, the round ends immediately. A result of 2, 3 or 12 is called 'craps' while a result of 7 or 11 is called a 'win' or a 'natural'. Craps is thought to be a simplification of an old English game *hazard* and it was introduced to New Orleans by Bernard Xavier Philippe de Marigny de Mandeville in the early 1800s.

Yahtzee is one of the most popular dice games. It can be played by one player or by a group. The game is played with five dice and consists of thirteen rounds. In each round, the dice is rolled and then scored in one of the categories. The player must score once in each category. The score is determined by a different rule for each category. The object of the game is to maximize the total score. The game ends once all 13 categories have been scored²¹.

A 'yahtzee' is a five-of-a-kind position where all the die faces are the same.

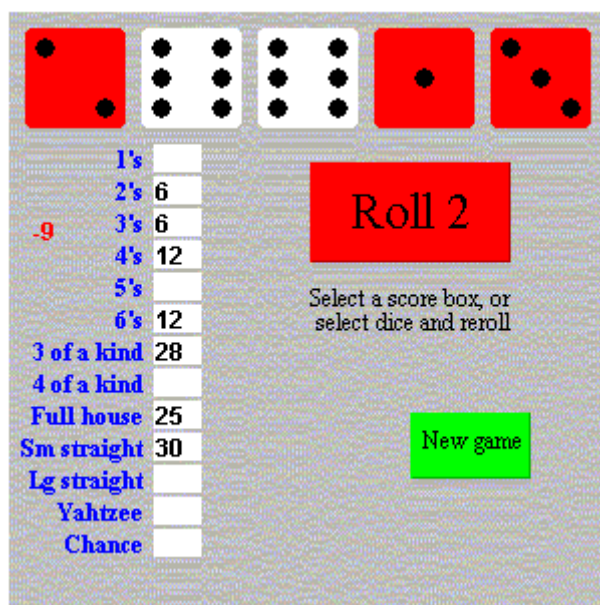


Figure 17 A game of yahtzee. Currently the player has rolled a pair of sixes and will re-roll the other three dice to try for a higher scoring category.

Problem. Calculate the chance of rolling a 'yahtzee' on the first roll.

Solution. There are six ways of rolling 'yahtzee', namely five ones, five twos, and so on. In all, there are 6^5 ($= 7776$) different rolls of the five dice. Therefore, the chance of a 'yahtzee' is 6 out of 7776, approximately 0.1%.

²⁰ Beware, loaded die are very easy to find in novelty shops so the 'fair' assumption is not always founded.

²¹ An interactive version of yahtzee can be found at the Algana [brainteasers-chance-yahtzee](#) link.

Random

The randomness factor in games of chance is very important. Random events have a whole mathematical theory devoted to them. Yet it should be noted that a truly random event is hard to create. It turns out that even supposedly random functions generated by computer can have interesting patterns which at best render them almost random (or so-called *pseudo-random*).

In the biological sciences, the theory of evolution ascribes the observed diversity of life to random genetic mutations some of which are retained in the gene pool due to the improved chance for survival and reproduction that those mutated genes confer on individuals who possess them. The characteristics of an organism arise to some extent deterministically (e.g., under the influence of genes and the environment) and to some extent randomly. For example, the density of freckles that appear on a person's skin is controlled by genes and exposure to light; whereas the exact location of individual freckles seems to be random.

In mathematics, the theory of probability arose from attempts to formulate mathematical descriptions of empirical observations. For the purposes of simulation it is necessary to have a large supply of random numbers, or a means to generate them on demand.

In the physical sciences, random motions of molecules have been modeled by statistical mechanics in order to explain various phenomena, such as Brownian motion (named in honor of the botanist Robert Brown) which represents random movement of particles suspended in a fluid²².

The central idea information theory is that a string of bits is random if and only if it is shorter than any computer program that can produce that string (so-called *Kolmogorov randomness*). This basically means that random strings are those that cannot be compressed. Pioneers of this field include Andrey Kolmogorov and his student Per Martin-Löf, Ray Solomonoff, Gregory Chaitin, and others.

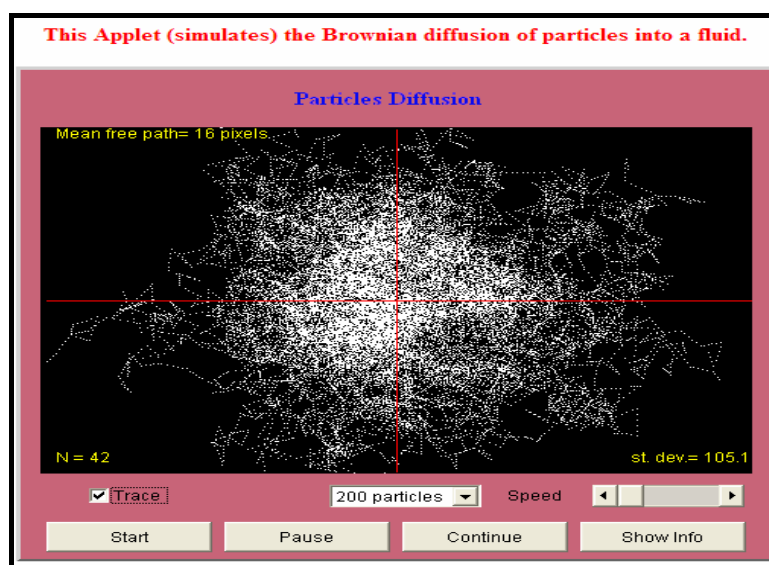


Figure 18. Brownian motion in a fluid.

Problem. Investigate the effect of increasing the number of particle in the interactive applet of figure 18. Note: You might need to adjust the speed slider slightly to see the effect.

²² An interactive version of Brownian motion can be found at the Algana [brainteasers-chance-diffusion](#) link.

Slots

A slot machine (American English), fruit machine (British English), or poker machine (Australian English) is a certain type of casino and bar game. Traditional modern slots are coin-operated machines with three or more spinning reels. The slot machine is also known informally as a ‘one-armed bandit’, primarily because of its traditional appearance and its ability to leave the player penniless. The machine typically pays off based on patterns of symbols visible on the front of the machine when it stops after a spin. Modern computer technology has resulted in many enhancements on the slot machine concept. Indeed, nowadays slot machines are the most popular gambling method in casinos and constitute about 70% of the average casino's income.

Historically, Sittman and Pitt of Brooklyn, New York developed a gambling machine in the 1890s that could be considered a precursor to the modern slot machine. It contained 5 drums holding a total of 50 card faces and was based on poker (see figure 19). This machine proved extremely popular with bars in the city. Players would insert a nickel and pull a lever, which would spin the drums and the cards they held, with the player hoping for a good poker hand. There was no direct payout mechanism, so a pair of kings might win the player a free beer, whereas a royal flush could pay out cigars or drinks, the prizes wholly dependent on what was on offer at the bar. To make the odds better for the house, two cards were typically removed from the "deck", viz. the ten of spades and jack of hearts, which cut the odds of winning a royal flush by half.

The first so-called ‘one-armed bandit’ was also invented in 1890s by Charles Fey of San Francisco, California, who devised a much simpler automatic mechanism. Due to the vast number of possible wins with the original poker card-based game, it proved practically impossible to come up with a way to make a machine capable of making an automatic pay-out for all possible winning combinations. Charles Fey devised a machine with three spinning reels containing a total of five symbols - horseshoes, diamonds, spades, hearts and a Liberty Bell, which also gave the machine its name. By replacing ten cards with five symbols and using three reels instead of five drums, the complexity of reading a win was considerably reduced, allowing Fey to devise an effective automatic payout mechanism. Three bells in a row produced the biggest payoff - ten nickels!

Of course, the way many probability-based gambling games succeed is by assuming that in the long run, the amount paid out is less than the amount received. So for instance, a traditional slot machine these days will average a pay-out of, say, 80% of the total coins put in. Though, to an onlooker, a player could put in just a few coins and win a jackpot – a case of right place, right time.



Figure 19. A typical Sittman & Pitt slot machine (left) and a modern slot machine (right).

Problem. Why was the chance of a royal flush halved in the Sittman & Pitt slot machine?

Solution. With the ten of spades and the jack of hearts removed, only two royal flushes (club or diamond) were possible, compared to the normal four.

Tricks

It is generally thought that playing cards originated at around the same time in China and India, leading to its introduction to Europe in the 14th century by Arabs who had travelled from the Orient. Mediterranean countries Italy, Spain and France feature in the first literary mentions of playing cards and card magic. Two of the earliest references to card tricks²³ are a 15th century article by Leonardo da Vinci, which describes a card trick performed by Giovanni de Jasone de Ferara, and a 16th century article by Cardanus, describing a performance by Spanish magician Dalmau, who performed for Emperor Charles V in Milan. In the 18th century, Italian magician Giovanni Guiseppe Pinetti performed for audiences in Germany, Russia and England, where he entertained King George III at Windsor Castle.

The 19th century witnessed two of the world's greatest contributors to card magic; Frenchman Jean Eugene Robert-Houdin (1805-1871) and Austrian Johann Nepomuk Hofzinser (1806-1875). Robert-Houdin performed mainly in Paris and he is generally recognised as the first real performer of magic tricks. Hofzinser performed mainly in Vienna and he is often referred to as the *Father of Card Magic*.

The 'magic' card trick shown in figure 20 can identify a card chosen from twenty one possibilities and requires only three clicks to do it. The identity of the chosen card is certain after three clicks, each respectively indicating a pile containing the card. This is because the first click limits the options to seven cards, the second click limits to at most three cards and the third click limits to just one card, assuming careful relaying of the cards into piles.

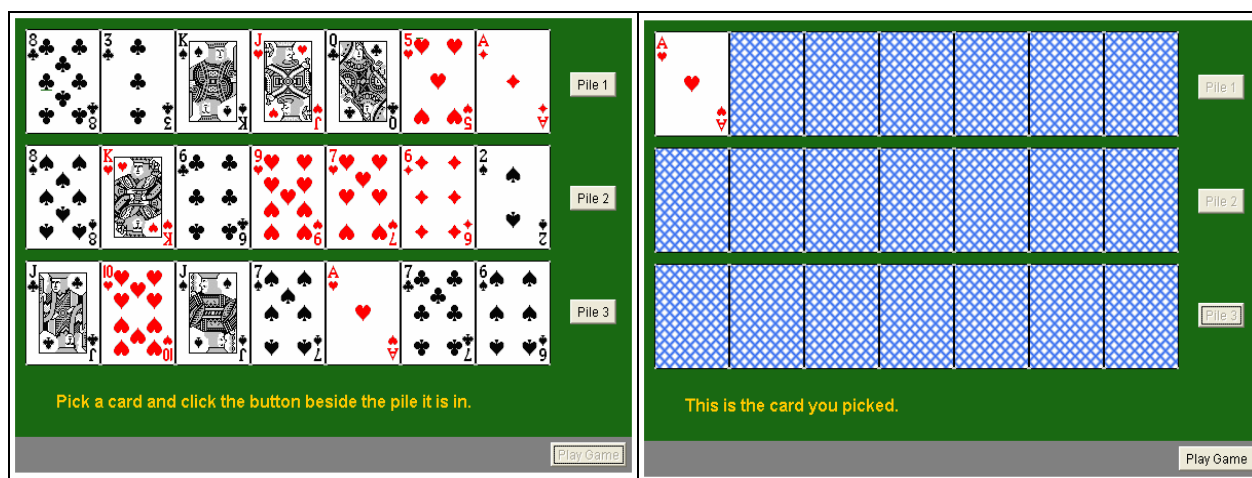


Figure 20. A 'magic' trick requiring just three clicks to identify a chosen card.

Problem. Although the popular version of the magic trick shown in figure 20 uses twenty one cards, it is possible to carry out the trick successfully with more cards. Still assuming that there are three clicks available to identify the chosen card from three piles, what do you think is the maximum number of cards that can be used?

Solution. Twenty seven in three piles of nine. The first click reduces to nine, the second click to three and the third to one.

²³ See, for example, http://www.card-trick.com/card_trick_history.htm

1.6 LOGIC & MAZES

Logic & maze puzzles can be characterised as problems where reason plays a vital role. Sometimes the challenge might be to reason through information identifying a candidate for solution; other times it might be better to start by eliminating those candidates that cannot possibly represent a solution. Of course, everyday we encounter situations where reasoning is needed, although we frequently consider our analysis to be more *commonsense* than formal reasoning. Planning a travel route, solving riddles, organising files in a computer, interpreting conversations and searching through large amounts of information are all common reasoning situations. Specific examples of logic & maze puzzles include analytical puzzles, intelligence problems, plain maze puzzles, scientific maze problems and transport puzzles.

Analytical puzzles

Analytical puzzles are often solved by reasoning through all available possibilities. Indeed, the word *analysis* can be defined as “an investigation of the component parts of a whole and their relations in making up the whole”²⁴. In many cases, an analytically-solvable puzzle will involve several participants with reasoning capability and the solution to the puzzle may be based on identifying what would happen in an obvious case, and then repeating the reasoning to less obvious cases. Consider the following *King's Wise Men* puzzle:

To decide who would become his new advisor, the King called the three wisest men in the country to his palace. He placed a hat on each of their heads, such that each wise man could see all of the other hats, but none of them could see their own. Each hat was either yellow or blue. The king gave his word to the wise men that at least one of them was wearing a blue hat. The King also announced that the contest would be fair to all three men. The wise men were also forbidden to speak to each other. The King declared that whichever man stood up first and correctly announced the colour of his own hat would become his new advisor. The wise men sat for a long time before one stood up and correctly announced the answer. What did he say, and how did he work it out?

To understand the solution, suppose that you are one of the wise men and, looking at the other wise men, you see they are both wearing yellow hats. Since the King specified that there were at most two yellow hats, you would immediately know that your own hat must be blue. So you would immediately stand up and announce it. Now suppose that you see the other wise men, and one is wearing a yellow hat and the other is wearing a blue hat. If your own hat was yellow, then the man you can see wearing the blue hat would be seeing two yellow hats and would have promptly stood up and declared his hat colour. Since a long time has elapsed, it can only be because your hat isn't yellow. Therefore it must be blue. Now suppose that you see the other wise men and both are wearing blue hats. If your own hat was yellow, then one of the two other wise men would be seeing a blue and a yellow hat, and would have declared his hat colour. Thus, your hat must be blue.

Problem. Suppose that you are trapped in a room with two doors, one leading to certain death and the other leading to freedom. You don't know which is which. On each door there is a guard and each guard will let you choose their door but upon doing so, you must go through it. The difficulty is that one guard always tells the truth, the other always lies and you don't know which is which. You can, however, ask one guard one question. What is the question you ask to guarantee freedom?

Solution. Ask any guard “what the other guard say if asked which door is safe?” On hearing the reply, go through the other door.

²⁴ See for instance <http://wordnet.princeton.edu/>

Intelligence problems

Intelligence problems are generally used to test or illustrate facets of animal intelligence, often in the hope of better modelling them in a machine or computer. The *Monkey & Banana Problem* and the *Turing Test* are two well-known intelligence problems.

The Monkey & Banana problem can be summarised as follows:

A monkey is in a room. Suspended from the ceiling is a bunch of bananas, beyond the monkey's reach. In the corner of the room is a box. How can the monkey get the bananas?

The main purpose of the problem²⁵ is to raise the question: Are monkeys intelligent? The answer is of course that the monkey must push the box under the bananas, then stand on the box and take the bananas. However, working this out does seem to require intelligence. Both humans and monkeys have the ability to use mental maps to remember things like where to go to gather food but they also have the ability not only to remember how to hunt and gather, but to create solutions to new unseen problems, as is the case here. Despite the fact that a monkey may never have been in an identical situation, with the same artefacts at hand, most appear capable of concluding that the box should be moved across the floor. The degree to which such abilities should be ascribed to instinct or learning is a matter of debate among natural scientists.

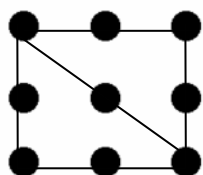
The Turing Test²⁶ is a proposal for testing a machine's apparent capability to demonstrate thought. It proceeds with a human judge engaging in a natural language conversation with two other parties, one a human and the other a machine. If the judge cannot reliably discern which is which, then the machine is said to have passed the test. It is generally assumed that both the human and the machine try to appear human. In order to keep the test setting simple and universal, the conversation is usually limited to a text-only channel such as a teletype machine as Turing suggested or, more recently, IRC or instant messaging system.

Logic forms the basis of most rational thought. In its simplest form, propositional logic is based on values of two values of “true” and “false”, although there exist logical systems using three values (ternary logic) and many continuous values (fuzzy logic). These logical systems also have a role in a wide range of interesting problems. Consider the following logical problem:

Which (if any) of the following four statements is true?

- Exactly one of these four statements is false.
- Exactly two of these four statements are false.
- Exactly three of these four statements are false.
- Exactly four of these four statements are false.

Since all contradict each other, at most one statement can be true. So the only consistent true statement is “exactly three of these four statements are false” and the other three are false statements.

Problem.	Suppose that you are given nine dots on paper. The figure (right) shows five straight lines connecting the dots. Show how you can draw just <u>four</u> straight lines connecting the dots without your pencil point leaving the paper. Hint: Creative thinking is sometimes referred to as “thinking outside the box”.	
-----------------	---	---

²⁵ In other variants of the problem, the bananas are in a chest and the monkey first has to open the chest using a key.

²⁶ Described by Professor Alan Turing in the 1950 paper "Computing machinery and intelligence."

Plain maze problems

A maze is a so-called *tour puzzle* in the form of a complex branching passage through which the solver must find a route. This is different from a labyrinth which typically has an unambiguous through-route and is not designed to be difficult to navigate. The pathways and walls in a plain maze are fixed whereas other mazes might allow interchangeable walls and variable pathways.

Throughout history, real plain mazes have been built with walls and rooms made from hedges²⁷, turf, paving stones, fields of crops and many more²⁸. Natural maize mazes for instance can be very large but obviously are only kept for one growing season, so they can be different every year and promoted as seasonal tourist attractions. Numerous mazes of different kinds have been painted, published in books, used in advertising, simulated in software and sold as art. In the 18th century, Leonhard Euler was one of the first mathematicians to analyse plane mazes mathematically, and in doing so made significant contributions to the related study of graph theory.

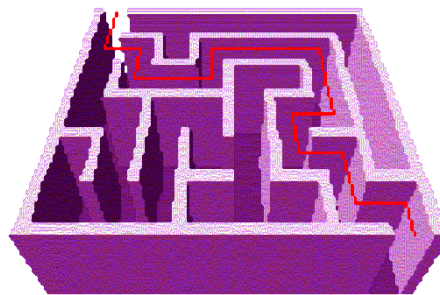


Figure 21. A simple plain maze is shown. Starting from the top entrance, the solution path can be represented by the string SLRLLRRLLRLR, where “L”, “R” and “S” respectively mean a movement left, right or straight on at each junction option.

Various strategies can be used to find a way through a maze by a traveller with no prior knowledge of the wall arrangements. The *Wall Follower algorithm*, also known as either the left-hand maze rule or the right-hand maze rule, is a strategy that works in simple connected mazes, i.e. where all of the walls are connected together. By keeping a hand in contact with one wall of the maze the traveller is guaranteed not to get lost and will reach a different exit if there is one. Otherwise the traveller will return to the entrance. Other strategies include the *Pledge algorithm*, *Random mouse* and *Tremaux's algorithm*.

Problem. Check that the (left-hand) wall follower algorithm applied to the maze shown in figure 21, starting from the top entrance, can be represented as LRRLLRLLLRLLRRRLLRLR.

²⁷ Famous mazes open to the public include: Aschaffenburg (Park Schönbusch), Germany; Castlewellan, Northern Ireland; Chatsworth House, England; Dole Plantation, Wahiawa, Hawaii; Hampton Court Palace, England; Leeds Castle, Maidstone, England; Longleat, England; Maize Quest, New Park, Pennsylvania, USA (mazes made of fence, rope, stone, turf, corn and straw); Maze Mania, Garden City, South Carolina, USA; Mohonk Mountain House hedge maze, New Paltz, New York; Parc del Laberint d'Horta, Barcelona, Spain; Sachsen-Anhalt, nr Dessau, Germany; Serendipity Maze, Mouille Point, Cape Town, South Africa; Shakopee, Minnesota, USA; Soekershof Walkabout Mazes, Western Cape, South Africa; The Crystal Palace, England.

²⁸ One of the short stories of Jorge Luis Borges featured a book called “The Garden of Forking Paths”, a literary maze.

Scientific mazes

Mazes have been regularly used in scientific studies where animals are given various challenges to test spatial memory and problem-solving skills.

In the neurosciences, the *Morris water maze* is a behavioural procedure designed to test spatial memory. It was developed by neuroscientist Richard G. Morris in 1984, and is commonly used today to explore the role of the hippocampus in the formation of spatial memories. A typical scenario involves mouse or rat placed into a small pool of water facing pool-side to avoid bias and back-end first to avoid stress. The pool contains an escape platform hidden a few millimetres below the water surface and visual cues, such as coloured objects, are placed around the pool in clear sight for the animal. The pool is usually approximately 2 metres in diameter and 1 metre deep. The pool can also be half-filled with water to 1 foot in depth. A sidewall above the waterline prevents the rat from being distracted by laboratory activity. When released, the animal swims around the pool in search of an exit while various parameters are recorded, including the time spent in each quadrant of the pool, the time taken to reach the platform (latency), and total distance travelled. The animal's desire to escape from the water reinforces its desire to find the platform efficiently, and on subsequent trials (with the platform in the same position) it is able to locate the platform increasingly more rapidly. This improvement in performance occurs because it has learned where the hidden platform is located relative to the conspicuous visual cues. After enough practice, a capable animal can swim directly from any release point to the platform.

The blind mole rat is naturally lacking in eyesight and it spends the majority of its life underground. However, scientific findings indicate that the animal possesses a rare talent: the ability to exploit the earth's magnetic field to find its way home. Blind mole rats have been observed using both their sense of smell and their balance to navigate over short distances. Tali Kimchi of Tel Aviv University²⁹ and her colleagues tested the creature's ability to stay on course during longer treks from home. The researchers brought wild mole rats into the laboratory and tested them in two types of mazes. In the first, a central hub surrounded by eight spokes, the animals were required to return to a specific starting point. When the scientists altered the surrounding magnetic fields using external magnets the animals were less likely to perform the task correctly. In the second test, the animals were placed in a rectangular maze. Under normal conditions, the animals effectively sought out a shortcut. Once the magnetic field was altered, however, their attempts to find the shortcut were less successful.

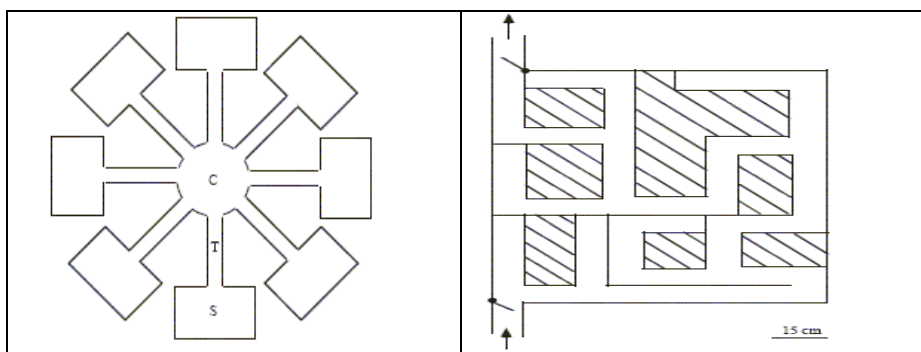


Figure 22. Two mazes used in the scientific study of blind mole rats.

Problem. Check that the shortest shortcut (avoiding any unnecessary dead-ends) for the maze in figure 22 (right) can be represented as RSLRLRLRLRLR.

²⁹ Kimchi, T & Terkel, J., “Magnetic Compass orientation in the Blind Mole Rat”, Journal of Experimental Biology, 204, 2001,p. 751-758

Transport puzzles

Transport puzzles are logical puzzles which normally represent real-life problems requiring movement of objects in a restricted space. For example, in sliding puzzles, the objects are labelled blocks. Other transport puzzles might require that all objects have to follow certain rules of movement. Examples include so-called *river-crossing problems*.

Consider the *Fox, Goose & Beans* problem.

Once upon a time a farmer went to market and purchased a fox, a goose, and a bag of beans. On his way home, the farmer came to the bank of a river and hired a boat to make it to the other bank of the river. In crossing the river by boat, the farmer could carry only himself and a single one of his purchases - the fox, the goose, or the bag of the beans. If left alone, the fox would eat the goose, and the goose would eat the beans. How could the farmer safely transport himself and his purchases to the far bank of the river?

The first step must be to bring the goose across the river, as any other will result in the goose or the beans being eaten. When the farmer returns to the original side, he has the choice of bringing either the fox or the beans across. If he brings the fox across, he must then return to bring the beans over, resulting in the fox eating the goose. If he brings the beans across, he will need to return to get the fox, resulting in the beans being eaten. Here he has a dilemma, solved by bringing the fox (or the beans) over and bringing the goose back. Now he can bring the beans (or the fox) over, leaving the goose, and finally return to fetch the goose. His actions in the solution are summarised in the following step-by-step procedure:

1. Take goose over
2. Return
3. Take fox or beans over
4. Bring goose back
5. Take beans or fox over
6. Return
7. Take goose over

Thus there are seven crossings, four forward and three back, at the end of which the farmer has safely transported himself and the purchases to the far bank. Variations of essentially the same problem include the three objects wolf, goat and cabbage or fox, duck and sack of corn, but the central logic remains similar.

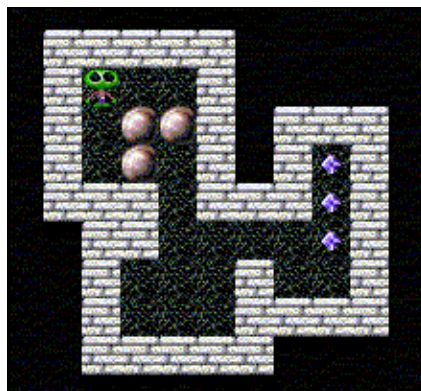


Figure 23. Sokoban is a popular object pushing game which tests the abilities of a player to transport groups of objects from one position to another, preferably in as few moves as possible.

Problem. Check that the three disks in figure 23 can be pushed into position using as few as thirty one pushes³⁰.

³⁰ An interactive version can be found at <http://www.algana.co.uk/downloads/sokoban/Sokoban.htm>.

1.7 NUMBERS & WORDS

Numbers & words puzzles often involve placing patterns in order according to various rules and requirements. Sometimes the challenge might be to place words into a grid, as in crosswords or a game of Scrabble; other times the challenge might be to place numbers into a magic square or a Sudoku puzzle. A good grasp of an appropriate dictionary of words and operations on integer numbers is essential. Specific examples of number & words puzzles include crosswords, magic squares, number-guessing puzzles, word-finding puzzles and word-spotting problems.

Crosswords

Although they have only a relatively short history, crossword puzzles are generally considered to be one of the most popular and widespread games in the world. The first rudimentary puzzles with appeared in the United Kingdom during the 19th century. They were derived from the word square, a group of words arranged so the letters read alike vertically and horizontally, and printed in children's puzzle books. In the United States, however, they later developed into a serious adult pastime. The first published crossword puzzle was created by Arthur Wynne, a journalist from Liverpool, England and he is often credited as the inventor of the popular crossword. It appeared in a Sunday newspaper, the New York World, in 1913. Wynne's original puzzle is shown in figure 24 (left) and it differed from today's crosswords in that it was diamond shaped and contained no internal black squares. Popular variations on the crossword theme include those similar to figure 24 (right) in which keywords are spelled in a vertical column.

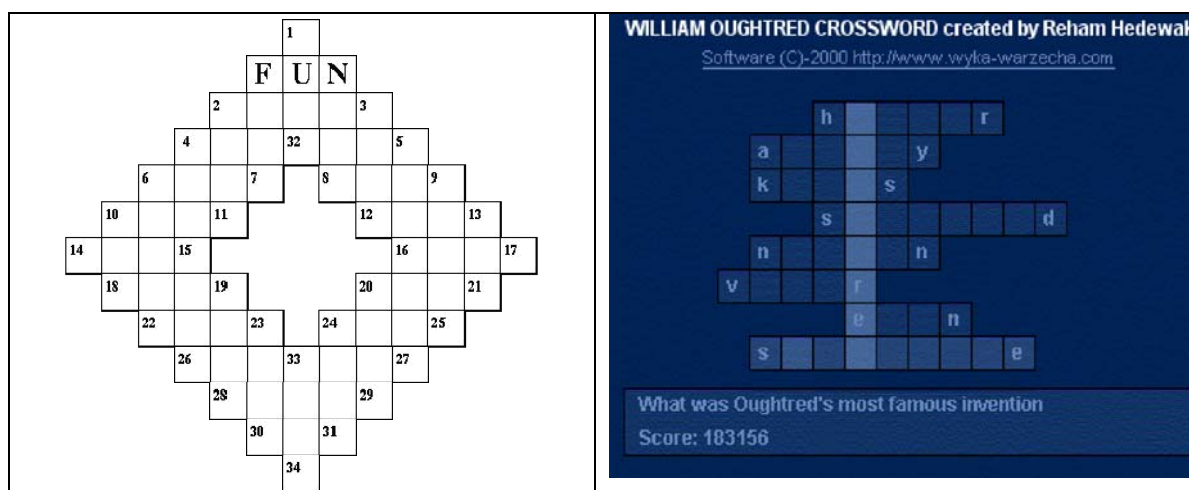


Figure 24. Wynne's original 1913 crossword puzzle is shown in figure 24 (left). A William Oughtred puzzle is shown (right) in which the word “Oughtred” should be spelled down the vertical shaded column.

Problem. Given the following clues for Figure 24 (left), complete the crossword solution.

2-3.	What bargain hunters enjoy.	12-13.	A bar of wood or iron.	1-32.	To govern.
4-5.	A written acknowledgment.	16-17.	What artists learn to do.	33-34.	An aromatic plant.
6-7.	Such and nothing more.	20-21.	Fastened.	N-8.	A fist.
10-11.	A bird.	24-25.	Found on the seashore.	24-31.	To agree with.
14-15.	Opposed to less.	10-18.	The fibre of the gomuti palm.	3-12.	Part of a ship.
18-19.	What this puzzle is.	6-22.	What we all should be.	20-29.	One.
22-23.	An animal of prey.	4-26.	A day dream.	5-27.	Exchanging.
26-27.	The close of a day.	2-11.	A talon.	9-25.	To sink in mud.
28-29.	To elude.	19-28.	A pigeon.	13-21.	A boy.
30-31.	The plural of is.	F-7.	Part of your head.		
8-9.	To cultivate.	23-30.	A river in Russia.		

Magic Squares

A normal magic square is an arrangement of the numbers from 1 to n^2 in an $n \times n$ grid, with each number occurring exactly once, such that the sum of the entries of any row, any column, or any main diagonal is the same. It can be shown mathematically that this sum must be $n(n^2 + 1)/2$. The simplest magic square is the 1×1 magic square whose only entry is the number 1. The next simplest is the 3×3 magic square (see the first table in figure 25). Magic squares have fascinated humanity throughout the ages, and have been around for over 4,000 years. They are found in a number of cultures, including Egypt and India, engraved on stone or metal and worn as talismans, the belief being that magic squares had astrological and divinatory qualities, their usage ensuring longevity and prevention of diseases. In about 1510, Heinrich Cornelius Agrippa wrote *De Occulta Philosophia*, drawing on the Hermetic and magical works of Marsilio Ficino and Pico della Mirandola, and in it he expounded on the magical virtues of seven magical squares of orders 3 to 9, each associated with one of the astrological planets. This book was very influential throughout Europe until the counter-reformation. A general magic square can use numbers other than 1 to n^2 .

SATURN=15			JUPITER=34				MARS=65					SOL=111					
4	9	2	4	14	15	1	11	24	7	20	3	6	32	3	34	35	1
3	5	7	9	7	6	12	4	12	25	8	16	7	11	27	28	8	30
8	1	6	5	11	10	8	17	5	13	21	9	19	14	16	15	23	24
			16	2	3	13	10	18	1	14	22	18	20	22	21	17	13
							23	6	19	2	15	25	29	10	9	26	12
												36	5	33	4	2	31

VENUS=175							MERCURY=260								LUNA=369								
22	47	16	41	10	35	4	8	58	59	5	4	62	63	1	37	78	29	70	21	62	13	54	5
5	23	48	17	42	11	29	49	15	14	52	53	11	10	56	6	38	79	30	71	22	63	14	46
30	6	24	49	18	36	12	41	23	22	44	45	19	18	48	47	7	39	80	31	72	23	55	15
13	31	7	25	43	19	37	32	34	35	29	28	38	39	25	16	48	8	40	81	32	64	24	56
38	14	32	1	26	44	20	40	26	27	37	36	30	31	33	57	17	49	9	41	73	33	65	25
21	39	8	33	2	27	45	17	47	46	20	21	43	42	24	26	58	18	50	1	42	74	34	66
46	15	40	9	34	3	28	9	55	54	12	13	51	50	16	67	27	59	10	51	2	43	75	35
							64	2	3	61	60	6	7	57	36	68	19	60	11	52	3	44	76
															77	28	69	20	61	12	53	4	45

Figure 25. Magic squares for $n=3$ to $n=9$, each related to an astrological planet.

Problem. Verify in the figure below that it is possible to find a 3×3 general magic square (i.e. one in which the numbers are not restricted to 1 through 9) constructed so that the letter count of the written numbers (ignoring spaces) also form the entries in a magic square?

SUM = 45			WORDS			SUM = 21		
5	22	18	five	twentytwo	eighteen	4	9	8
28	15	2	twentyeight	fifteen	two	11	7	3
12	8	25	twelve	eight	twentyfive	6	5	10

Number-guessing puzzles

Number-guessing puzzles are interesting because we usually know that a pattern or strategy can be used and realising the strategy is sometimes even more enjoyable than eventually guessing a particular number. Number sequences, Sudoku problems and cross-figures, an extension of crosswords to numbers, are three examples.

Sequences of integers (whole numbers) play an important role in engineering and mathematics. As one example, consider the Fibonacci numbers credited to the Italian mathematician Leonardo of Pisa (c.1170-1240), also known as Fibonacci. The first few Fibonacci numbers are 1, 1, 2, 3, 5, 8, 13, 21, 34 ... Each new number in the series is generated by simply adding the two previous numbers. Fibonacci numbers are found in many problems in nature, including the spiral growth of shells, branching of plants, numbers of ancestors of a honey bee, spiral patterns in the heads of sunflowers and characteristics of pinecones.

An integer greater than one is called a prime number provided that its only positive divisors (factors) are 1 and itself. For example, the prime divisors of 10 are 2 and 5; and the first six primes are 2, 3, 5, 7, 11 and 13. A fundamental theorem in arithmetic confirms that the primes are the building blocks of the positive integers: every positive integer is a product of prime numbers in one and only one way, except for the order of the factors. This is the key to their importance: the prime factors of an integer determine its properties. As we shall see later, the effort involved in finding prime factors of very large numbers³¹ is used in cryptography.

Sudoku puzzles are popular number guessing problems, usually in the form of a 9x9 grid where the numbers 1 through 9 must be placed in every row and column without repetition. Additionally, the number should not be repeated in each 3x3 sub-grid (see figure 26). The name *Sudoku* means "single digits" and it is a trademark of puzzle publisher Nikoli Co. Ltd. in Japan. There are many strategies for solving Sudoku puzzles.



Figure 26. A typical Sudoku puzzle (left) and its solution with solved number shown (right) in red.

Problem. Crossfigure.																													
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Across 1 1 down plus 346 4 Three times 20 down 7 21 down minus nineteen 8 20 down divided by five 9 26 down plus eleven 10 A prime number 11 Two times 10 down 13 6 down plus 1214 15 4 down minus 778 17 28 down times 8 across 20 20 down plus eighty-four 23 14 down divided by four 24 7 across plus nine 25 14 down divided by four																													
Down 1 29 across plus eighty-two 2 A prime number 3 20 down minus 591 4 4 across minus 117 5 Five times 28 down 6 Eight times 8 across 10 10 across plus four 12 11 across minus seven																													
14 Eight times 12 down 16 6 down divided by six 17 15 across plus 311 18 9 across plus twenty-eight 19 6 down plus 264 20 2 down times 5 down 21 28 across times two 22 4 down plus eight 26 24 across plus eighteen 28 28 across minus one HINT! The answers to 2 down and 10 across are 43 and 23, respectively.																													

³¹ An interactive applet for finding all prime numbers up to a given number can be found in the algorithms-numerical-prime section at algana.

Word-guessing puzzles

Word-guessing puzzles are interesting because we usually know that a pattern or strategy can be used and realising the strategy is sometimes more enjoyable than guessing a particular number. Hangman and Wordsearch are two examples.

Hangman is thought to have originated in Victorian England. The game is mentioned in Alice Bertha Gomme's "Traditional Games" in 1894 under the name "Birds, Beasts and Fishes." The rules are simple; a player writes down the first and last letters of a word for an animal, and the other player guesses the letters in between. The game has been known under various names, including *Gallows*, *Game of Hangin'* and *Hanger*.



Figure 27. A Hangman history of computing puzzle – solution name “ARTIN”.

Wordsearch is an interesting puzzle because many of us use different strategies. For instance, figure 28 shows a scrambled grid of letters in which the names of animals can be found horizontally, vertically, diagonally and even in reverse! Some might find “ALLIGATOR” by looking first for the “A” and then for the “L”. Others might notice a double letter “LL” and focus first on looking for that, primarily since there are fewer possibilities.

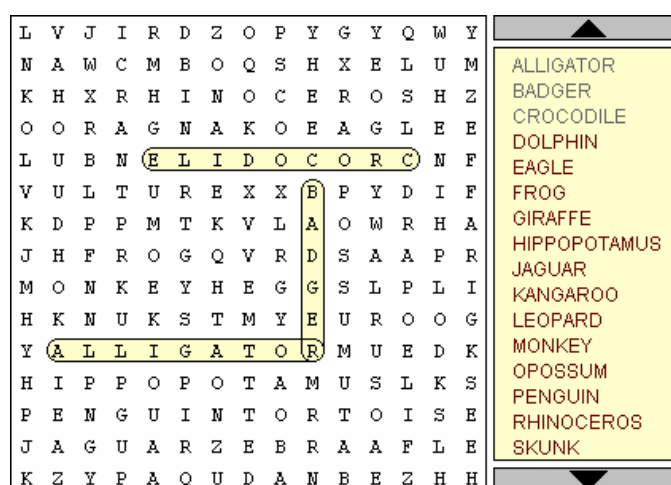


Figure 28. A Wordsearch puzzle – three solutions “CROCODILE”, “BADGER” and “ALLIGATOR” are shown.

Problem. Find the remaining solutions for the ANIMAL Wordsearch puzzle in figure 28.

1.8 EXERCISES

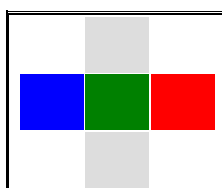
1. The popular 3 x 3 version of Rubik's cube is comprised of twenty-six smaller blocks. How many smaller blocks would you expect to comprise the 4 x 4 version?

As it turns out, we never fully see some of the faces of the smaller blocks in the 3 x 3 version. For instance, corner blocks show only three of their faces while blocks in the middle only show one. In the 3 x 3 version, how many faces in total are never fully shown?

2. The following are famous names associated with puzzles. Prepare a concise web biography of at least six of the following in a format similar to that shown for Albert Einstein at <http://www.algana.co.uk/FamousNames/E/einstein.htm>

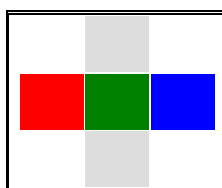
- (a) Edouard Lucas (Tower of Hanoi),
- (b) Enro Rubik (Rubik's cube),
- (c) Sam Loyd (15 Puzzle),
- (d) Leonardo Fibonacci of Piza (The Book of Abacus).
- (e) Emperor Claudius (Tabula).
- (f) Howard Staunton (Chess)
- (g) Richard Swiveller (Cribbage)
- (h) Arthur Wynne (Crossword).
- (i) John Spilsbury (Jigsaw)
- (j) Goro Hasegawa (Othello)
- (k) Alfred Butts (Scrabble)

3. A certain sliding block puzzle comprises five compartments containing three blocks, coloured blue, green and red. Sliding a block to a vacant compartment enables the order of the blocks to be changed. A sketch of the initial horizontal position of the blocks is shown in the following figure.



Start position

What is the minimum number of sliding moves needed to reverse-order the blocks as shown in the following figure?



End position

4. In an 8X8 chess board:

- (a) place 8 Queens on the board without them threatening each other,
- (b) place 4 Queens and 4 Knights in positions that do not threaten each other.

You might wish to use the interactive applet at

<http://www.algana.co.uk/Puzzles/Grids/qvk/qvk.html>

5. In a “Last Match” two-player game, the object of the game is to make your opponent take the last match from a pile of matches on a table. Each player alternates in taking one, two or three matches from the pile in turn. Explain concisely a guaranteed winning strategy assuming you take the first turn.

You might wish to use the interactive applet at:

<http://www.algana.co.uk/Puzzles/matches/lastmatch/lastmatch.htm>

6. For the word search puzzle in figure Q6, find all the “units of measure”.

V	J	R	Y	P	D	Q	C	F	V	K	X	S	D	P
R	E	T	E	M	O	L	I	K	F	B	G	J	Q	J
F	E	E	T	O	H	M	S	Q	U	A	E	Y	Q	Q
I	W	C	Q	T	T	A	W	S	L	V	X	X	P	U
W	P	N	C	K	I	V	H	L	I	T	E	R	O	A
J	O	U	U	N	A	E	O	K	K	E	N	D	U	R
I	E	O	X	I	L	N	B	L	R	K	O	D	N	T
A	A	Y	D	N	O	C	E	S	T	Y	T	O	D	L
C	L	O	C	J	B	Z	C	E	L	F	W	Y	H	M
A	E	R	T	R	D	W	L	E	E	U	E	A	H	U
J	D	T	E	G	V	V	I	K	E	R	N	R	Y	E
H	N	X	O	L	C	A	I	K	Y	E	G	D	H	E
Y	A	O	D	C	I	N	C	J	E	A	R	E	N	V
Z	C	F	X	I	C	M	G	Y	F	D	O	C	D	X
E	H	T	U	H	O	U	J	H	E	R	T	Z	A	M

Figure Q6: A “units of measure” word search puzzle (kilometer is already labeled).

7. Valid moves in a *Jumping Counters* board game involve moving one space forward or jumping over a counter of a different type. There are two different types of counter, **O** and **X**, respectively, and **O** counters move only to the right while **X** counters move only to the left. The initial position of the board has two vacant spaces between the n **O** counters on the left and the m **X** counters on the right, as shown in the first row of the diagrams in figure Q4. The end game position shows all **O** counters on the right-hand side of the board and all **X** counters on the left. Figure Q4 shows minimal moves for the particular Jumping Counters (n,m) games for which $(n,m) = (1,1)$ and $(n,m) = (1,2)$.

O			X	
	O		X	
		O	X	
	X	O		
	X		O	
X			O	

O			X	X
	O		X	X
		O	X	X
	X	O		X
	X		O	X
X			O	X
X		X	O	
X	X		O	
X	X			O

Figure Q7. Minimal moves for the Jumping Counters (n,m) board game when $(n,m) = (1,1)$ (figure left) and when $(n,m) = (1,2)$ (figure right). The initial position of the board has two vacant spaces.

- (a) Reason through the various steps in the Jumping Counters (n,m) game for which (n,m) takes values in the list $[(1,1), (1,2), (2,1), (2,2), (2,3), (3,2), (3,3)]$ and create a table of values for the minimal number of moves needed.
- (b) Suggest by conjecture a possible generalised explicit function in terms of n and m for the (n,m) game. Comment briefly on your expectation that this function should be symmetric or otherwise.
- (c) Check whether your solution in part (b) satisfies the $(3,4)$ game.

1.9 BIBLIOGRAPHY & WEB REFERENCES

1. <http://www.algana.co.uk/Puzzles/abaci/chineseabacus/abacus.htm>
2. <http://www.algana.co.uk/Brainteasers/blocks/TowersofHanoi/TowersofHanoi.html>
3. <http://www.algana.co.uk/Puzzles/blocks/rubikscube/rubikscube.html>
4. <http://www.algana.co.uk/Puzzles/boards/TicTacToe/tictactoe.html>
5. <http://www.algana.co.uk/Puzzles/boards/Pegs/pegs.html>
6. <http://www.algana.co.uk/Brainteasers/chance/hilo/hilo.html>
7. <http://www.algana.co.uk/Brainteasers/chance/yahtzee/yahtzee.htm>
8. <http://www.algana.co.uk/Puzzles/Grids/qvk/qvk.html>
9. <http://eluzions.com/Puzzles/Logic/>
10. <http://www.algana.co.uk/Puzzles/matches/christmastree/christmastree.htm>
11. <http://www.algana.co.uk/Brainteasers/LargeMaze/MazesLargeMazeMain.html>
12. <http://www.algana.co.uk/Puzzles/numbers/magicsquares/magicsquares.htm>
13. <http://www.algana.co.uk/Puzzles/words/Wordsearch/wordsearch.html>

2. COMPRESSION

2.1 INTRODUCTION

Data compression is the process of deriving more efficient ways of representing and storing data. This often means compressing similar information into a smaller storage space. In the case of text, the Huffman code can be used to store alphanumeric data into less space than a fixed-length code such as ASCII. In the case of graphic files, the JPEG format can be used to store images into less space than a format such as BMAP or GIF.

Some of the algorithms for data compression considered in this section are:

1. Huffman Compression
2. JPEG Encoding
3. Run Length Encoding
4. Variable Length Encoding

References for these and other related algorithms are given at the end of the section.

2.2 HUFFMAN

In standard 8-bit ASCII code, text characters (including punctuations) are represented by a unique sequence of 8 bits. Since an n -bit binary code provides 2^n different combinations, 8-bit ASCII allows us to represent 256 different text characters. This is sufficient for most computer keyboard usage today. Unicode is an extension of ASCII that allows a much wider range of languages to be represented and which allocates 16 bits to each character. This is used for example by the language Java, and any ASCII character can be converted to Unicode by prefixing it with the zero byte.

To calculate total storage space used by a large text file using 8-bit ASCII, we need to calculate the number of text characters in the file multiplied by eight. Thus a text file containing 1000 characters will need approximately 8000 bits using ASCII, disregarding a relatively small number of formatting and file properties. Although these codes are simple, there are obvious inefficiencies; clearly Unicode wastes at least half of the available space when storing plain ASCII.

Originally developed by David Huffman [1], the Huffman code can be used to store a large text file taking approximately 40% less storage space than 8-bit ASCII. Given the huge amount of text information transmitted and stored by all of us everyday, this can represent a valuable cost saving. In fact, the Huffman code uses detailed knowledge of the language concerned to come up with a “cheaper” means of representing the data. It takes into account the fact that not all letters in the associated alphabet occur with the same frequency. The Huffman code assigns a smaller bit pattern to the most frequently occurring letters, and essentially longer bit patterns to the least frequent. In the end, the total number of needed compared to standard ASCII is significantly reduced. To decide the bit patterns, a binary tree can be built from the characters present in the given piece of text. The tree is built in such a manner that the more frequent characters are found near to the top of the tree and the least frequent near the bottom. The resulting code also has a prefix property such that no character code is the prefix, or start of the code for another character. Clearly a code with the prefix property avoids any need to have additional separators in the original characters, while permitting variable lengths. To allocate a code to a particular character, it is necessary to work down the tree consistently assigning either a 0 or a 1 to a left branch or a right branch.

An algorithm for building a Huffman Tree comprises the following steps:

1. Read a file of text characters.
2. Count the occurrence of each character.
3. Form a frequency table.
4. Assign small number of bits to high frequency characters
5. Assign larger number of bits to low frequency characters
6. Encode the text file

Consider a small text file containing the eleven characters: **SAVES SPACE**. Note that the space between the words counts as a character. A simple count of the frequencies yields the table shown in figure 1.

Character	Frequency
S	3
A	2
E	2
C	1
P	1
V	1
<space>	1
Total	11

Figure 1. A count of the frequency corresponding to each character in the text file **SAVES SPACE**. Characters in the table are arranged in descending order according to frequency.

This information is then used to build the Huffman tree shown in figure 2. Generally, the points of connection in a tree are known as *forks* and the segments as *branches*. Final segments and the nodes at their ends are called tree *leaves*. A tree with two branches at each fork and with one or two tree leaves at the end of each branch is called a *binary tree*. Begin with a collection (a forest) of very simple trees, one for each symbol to be coded, with each consisting of a single node, labelled by that symbol, and the frequency with which it occurs in the string. The construction is recursive: at each stage the two trees which account for the least total frequency in their root nodes are selected, and used to produce a new binary tree. This has, as its children the two trees just chosen: the root is then labelled with the total frequency accounted for by both subtrees, and the original subtrees are removed from the forest. The construction continues in this way until only one tree remains; that is then the Huffman encoding tree.

Thus it can be seen in figure 2 that the eleven characters are divided into a group of four and seven, respectively. The binary division into further groups continues until each member of the set of text characters {S, A, E, C, P, V, <space>} is a leaf in the tree [2]. The six forks in this example are labelled T1, T2, T3, T4, T5 and T6, respectively.

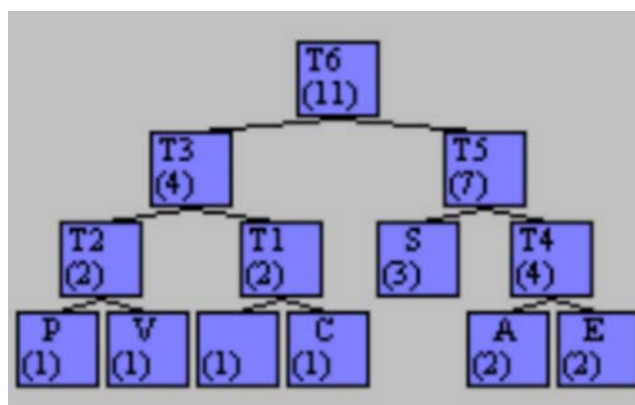


Figure 2. A Huffman tree for the text file **SAVES SPACE**.

Each symbol is encoded according to its position in the tree. Choosing a branching to the right as a “0” and branching to the left as a “1”, the symbol **A** is encoded as 001 and so on.

Symbol	Encoding	Frequency	Bits required for each symbol (Huffman)	Bits required for each symbol (ASCII)
A	001	2	3	8
C	100	1	3	8
E	000	2	3	8
P	111	1	3	8
S	01	3	2	8
V	110	1	3	8
<space>	101	1	3	8
Total Bits (all symbols)			30	88

Figure 3. Encoding the text file **SAVES SPACE**.

A comparison of the total bits required in the encoding of **SAVES SPACE** yields 30 bits for Huffman compared to 88 bits in ASCII, a relative saving of 66% for Huffman over ASCII. In this case, the number of text characters involved (n) is small, viz. $n = 11$. When n is larger, say 300 or 3000 or 30000, a relative saving of approximately 40% is more likely.

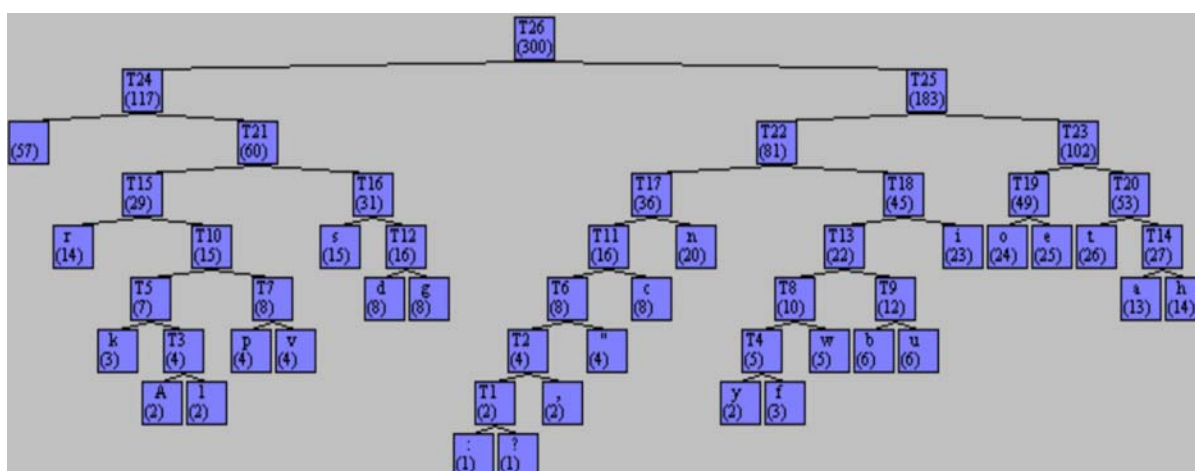


Figure 4. A Huffman tree for the text file³²:

Alice was beginning to get very tired of sitting by her sister on the bank and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, “and what is the use of a book” thought Alice “without pictures or conversation?”

Practice 2.1 For the text file in figure 4, construct an encoding table and calculate the relative saving of Huffman over 8-bit ASCII.

³² An extract from “Alice in Wonderland” by Lewis Carroll

2.3 JPEG

JPEG, named after the Joint Photographic Experts Group [3], is a standard for image compression recognised by the ISO (International Standards Organization) [4]. JPEG gives users the ability to take an image and compress it with little or no noticeable quality degradation. The original image is taken through a series of steps, (initial input, Discrete Cosine Transform, quantized ion, and encoding) each of which contributes to the overall compression of the image. Compression ratios for a 24-bit colour image can be as high as 100:1. At that ratio, however, there is noticeable image degradation, even at normal magnification. Usual image compression ranges from 10:1 to as high as 20:1 without any visible image degradation. The actual file format defined in the JPEG standard is the Still Picture Interchange File Format (SPIFF).

The first step of the JPEG compression process is straightforward. Source image samples are grouped into an 8x8 matrix or block. The initial image data is usually converted from normal RGB colour space to a luminance/chrominance colour space, such as YUV, a colour space scheme that stores information about an image's luminance (brightness) and chrominance (hue). YUV signals are created from an original RGB (red, green and blue) source. The weighted values of R, G and B are added together to produce a single Y signal, representing the overall brightness, or luminance, of that spot. The U signal is then created by subtracting the Y from the blue signal of the original RGB, and V by subtracting the Y from the red. This can be accomplished easily with analog circuitry. In a digital system, the following equations can be used to derive Y, U and V from R, G and B:

$$Y = 0.299R + 0.587G + 0.114B \quad (1)$$

$$U = -0.147R - 0.289G + 0.437B + 0.5 \quad (2)$$

$$V = 0.615R - 0.515G - 0.100B + 0.5 \quad (3)$$

Here, R, G and B are assumed to range from 0 to 1, with 0 representing the minimum intensity and 1 the maximum. Since the human eye is much more sensitive to luminance than chrominance, you can afford to discard much more information about an image's chrominance, especially the higher frequencies. By using a YUV colour space scheme, it is much easier to identify chrominance and thus eliminate unnecessary image data that cannot be seen by the human eye. Another optional manipulation to the image data is to down sample the chrominance components by averaging groups of pixels together. Once the image data in the 8x8 matrix has been satisfactorily manipulated, the matrix is inputted into a Discrete Cosine Transform (DCT). The ultimate goal of the DCT is to represent the image data in a different domain using the cosine function. The image data is transformed into numerous curves of different sizes. When these curves are put back together, through inverse Discrete Cosine Transform, the original image (or an extremely close approximation to it) is restored.

The next step is to prepare the matrix for further compression by quantizing each element in the matrix. The JPEG standard defines two tables of quantization constants, one for luminance and one for chrominance. These constants are calculated based on the JPEG image quality, which can be selected by the user.

The final step in the JPEG compression algorithm (prior to writing the image information to file) is to encode the data using a run-length encoding scheme. Before the matrix is encoded, it is arranged in a zigzag order. Also, the elements representing low frequencies are moved to the beginning of the matrix and elements representing high frequencies are put towards the end of the matrix. By placing the higher frequencies (which are more likely to be zeros) at the end of the matrix, longer sequences of zeros are more likely to occur; thus, better overall compression can be achieved.

More technical information about the JPEG algorithm can be found at [\(5\)](#).

2.4 RUN-LENGTH

This compression algorithm replaces sequences or runs of consecutive repeated characters with a single character and the length of the run. This can either be applied to all input characters, including runs of length one or a special character can be used to introduce a run-length encoded group. The longer and more frequent the runs are, the greater the compression that will be achieved. This technique is particularly useful for encoding black and white images where the data units would be single bit pixels. For example, simple graphic images such as icons and line drawings. It is a very simple form of data compression in which runs of data are stored as a single data value and count, rather than as the original run.

Consider a hypothetical single scan line, with B representing a black pixel and W representing white:

[illegible]

If we apply a simple run-length code to the above hypothetical scan line, the result could be:

W12B1W12B3W24B1W14

This can then be interpreted as twelve W's, one B, twelve W's, three B's, twenty four W's, one B and fourteen W's. This run-length code represents the original 67 characters using only 16 characters. It is evident from the algorithm that only sequences of 4 or more repeating characters are worth encoding. This is because it takes at least 3 characters to define the encoding, viz. a control character, the frequency of the character & the character itself. Hence coding two repeating characters would actually lead to an increase in the file size.

Note that this example is primarily for explanatory purposes and the actual format used for the storage of images is generally binary rather than ASCII, but the essential principle remains the same.

Run-length encoding performs loss-less data compression and is well suited to palette-based iconic images where a significant amount of repetition can be expected. It does not normally work well continuous-tone images such as photographs in which repetition is limited. However JPEG does use it quite effectively on the coefficients that remain after transforming and quantizing image blocks.

2.5 VARIABLE LENGTH

Variable-length encoding is a loss-less compression technique where bits used to encode a character differ, or vary, according to the occurrence frequency of that character. Variable-length encoding uses the idea that frequently-represented characters are assigned shorter bit codes than less-frequent codes, each code being uniquely identifiable. So the Huffman code falls in to the broad category of variable-length encoding algorithms.

Variable-length encoding is also called ‘entropy encoding’. Samuel Morse noticed the advantages of this idea in the mid-19th century when inventing the telegraph. He assigned shorter sequences to letters that occur more frequently, for instances [e (.), a (. -)], and longer sequences to less frequently occurring letters, for instances [q (- - .-), j (. - - -)].

The most important characteristics for Variable-Length coding are:

1. The more frequently-occurring the character, the shorter the code.
2. The less frequent the character, the longer the code.
3. Each code can be uniquely decoded. In order for this to be true, no character's encoding is a prefix of any other preserving the prefix property.

This algorithm can save a substantial amount of space on text files, and also appears in such applications such as fax machines, video compression, and computer security.

More information can be found at [\(6\)](#)

2.6 OTHER COMPRESSION ALGORITHMS

The LZW compression algorithm, named after Lempel, Ziv and Welch, is a lossless data compression algorithm. Most of this method was invented and published by Lempel and Ziv in 1978; this is known as the LZ78 algorithm. Welch later gave a few details and improvements in 1984. A key feature of the algorithm is that it is possible to automatically build a dictionary of previously seen strings in the text being compressed. The dictionary does not have to be transmitted with the compressed text, since the de-compressor can build it in the same way as the compressor, and if coded correctly, will have exactly the same strings that the compressor dictionary had at the same point in the text. The dictionary initially contains 256 entries, one for each possible single byte string. Every time a string not already in the dictionary is seen, a longer string, consisting of that string with the single character following it, is stored in the dictionary.

The output consists of integer indices into the dictionary. These initially are 9 bits each, and as the dictionary grows, can increase to up to 16 bits. A special symbol is reserved for "flush the dictionary" which takes the dictionary back to the original 256 entries, and 9 bit indices. This is useful if compressing a text file that has variable characteristics, so that a dictionary of early material is no longer much use later in the text. This use of variable increasing index sizes is one of Welch's contributions. Another was to specify an efficient data structure to store the dictionary.

The method became moderately widely used in the program "compress" which became a more or less standard utility in Unix systems circa 1986. (It has since disappeared from many for both legal and technical reasons.) Several other popular compression utilities also used the method, or closely related ones. It became very widely used after it became part of the GIF image format in 1987. It can also be used in TIFF-files.

On large English texts, LZW typically compresses to about half the original size. It has now been superseded for many purposes by algorithms such as Deflate and Burrows-Wheeler transform methods. This is partly because these newer methods offer better compression ratios in most cases, and partly for legal reasons. However the GIF image format with LZW compression is still widely used.

Most "zip" compression programs use a variation of the LZ adaptive dictionary-based algorithm to shrink files. "LZ" refers to Lempel and Ziv, the algorithm's creators, and "dictionary" refers to the method of cataloguing pieces of data (see ref. [7](#).)

2.7 COMPARISON OF ASCII AND HUFFMAN

The Huffman code comes out as clearly the more effective of the two in terms of space saving based on the experimental results shown in figure 5. This is achieved by allocating the size of the bits according to the frequency, thereby saving space by assigning shorter code lengths to the more frequent characters. Another advantage of the Huffman code is that we do not have the limit of represent 256 characters only, since the tree can be infinitely large to accommodate however many characters are present in the text.

The Huffman code is limited because the representation changes from text to text, by simply adding one extra occurrence of a character in the text we can completely change the code assignment required. This also necessitates that the tree be saved together with the text so that it can be translated correctly, since without the tree it would not be possible to interpret the text. In the present era of computing storage, space is no longer expensive, and 256 characters are normally sufficient to capture the set of characters required by most users, for this reason the ASCII code is used. Unicode is also available as standard for larger sets of characters. Plus the easy of implementation and constant codes make it ideal for everyday use.

The Huffman code on the other hand is more suited to customised compression implementations, where space rather than ease of interpreting the file, is more important. It can also be used as a means of increased security for one-time transmission of a sensitive text document. Based on the experimental results shown in figure 5, the ASCII number of bits (Y) follows a straight line $Y = 8n$, where n is the size of the text file. This compares with an approximate straight line $Y = 4.4n$ for the number of Huffman bits required for the same text string. This corresponds to an approximate space saving of 45%.

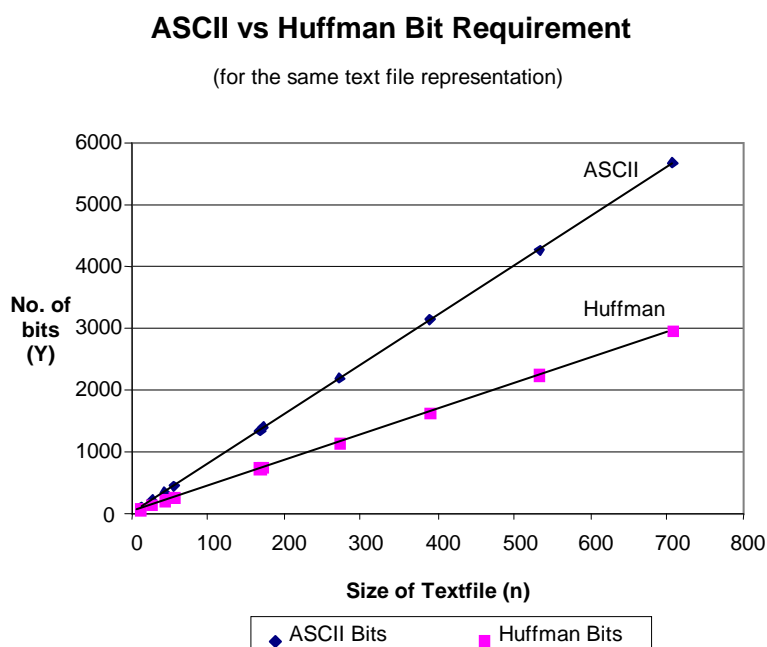


Figure 5. A plot of the number of bits required for the same text string using ASCII and Huffman.

2.8 EXERCISES

1. For the small alphabet { c, d, e, n, o }, consider the two binary codes shown in figure Q1.

Symbol	Code 1	Code 2
c	001	000
d	001	11
e	010	01
n	011	001
o	100	10

Figure Q1: Code 1 is a fixed length code and Code 2 has the prefix property.

- Encode the characters of the word **ENCODE** using each code.
- Using the answer in part (a), calculate the percentage saving for Code 2 compared to Code1.
- Assuming a right-to-left reading and no errors, explain at least two reasons why the string:

1101000101101

must have used Code 2.

- Decode the string in part (c).
2. The following are famous names associated with data compression. Prepare a concise web biography of each in a format similar to that shown for Albert Einstein at <http://www.algana.co.uk/FamousNames/E/einstein.htm>
- David Huffman (Huffman code),
 - Samuel Morse (Morse code),
 - Abraham Lempel (LZW algorithm)
 - David John Wheeler (Burrows-Wheeler transform).
3. Carry out an experimental investigation to test the assertion that “Compared with ASCII, the Huffman code saves approximately 50% storage space for large files containing natural English text characters, such as a chapter from English novel or a large *readme* file”. In the course of your investigation, you should create an appropriate computer program containing the following features:
- The capacity to read characters from a large text input file,
 - The capacity to count and record the frequency of each character,
 - The means of generating the associated Huffman code,
 - The means of summarizing the calculation of the relative space saving.

4. Consider a hypothetical single graphics scan line, with B representing a black pixel and W representing white:

WWWWWWWWWWWWBBBBBBBBWW

- (a) How many binary bits are contained in the full binary equivalent of the scan line using the appropriate ASCII codes for **w** and **B**.
 - (b) Write down the result of applying a simple run-length code to the scan line, in the form $\mathbf{Wn_1Bn_2Wn_3}$, where n_i ($i = 1, 2, 3$) represents decimal numbers.
 - (c) Converting the **w** and **B** characters to 8-bit ASCII and the decimal numbers to 4-bit binary, write down binary string for the answer in part (b).
 - (d) Calculate the percentage saving for the binary run-length code in (c) compared to the full binary equivalent found in (a).
5. Discuss briefly the fundamental idea behind **zipping** files. Carry out an experimental investigation to test the relative disk storage space saved using a zipped and unzipped version of several word-processed documents. Choose unzipped word-processed documents of approximately 1K, 10K, 100K, and 1000K sizes.
6. Discuss briefly the fundamental idea behind **JPEG** files. Carry out an experimental investigation to test the relative disk storage space saved using a JPEG and Bitmap version of several graphics files. Choose graphics files of approximately 10K, 100K, 1000K and 10000K sizes.
7. How would you expect the results obtained in question 3 to change if the frequency of words, rather than individual characters, was used for the compression? Assume the Huffman code assignment is made to each word.
8. By visiting the International Standards website at www.iso.ch or otherwise, identify the specific ISO/IEC standards that apply to each of the following:
 - (i) Information technology -- Digital compression and coding of continuous-tone still images: Requirements and guidelines
 - (ii) Information technology -- Digital compression and coding of continuous-tone still images: Registration of JPEG profiles, SPIFF profiles, SPIFF tags, SPIFF colour spaces, APPn markers, SPIFF compression types and Registration Authorities (REGAUT)
 - (iii) Information technology -- Coded representation of picture and audio information -- Progressive bi-level image compression
 - (iv) Information technology -- Data compression for information interchange -- Adaptive coding with embedded dictionary -- DCLZ Algorithm
 - (v) Information technology -- Procedure for the registration of algorithms for the lossless compression of data.

2.9 BIBLIOGRAPHY & WEB REFERENCES

1. <http://www.algana.co.uk/FamousNames/H/huffman.htm>
2. <http://www.algana.co.uk/Algorithms/Compression/Huffman/challenge.html>
3. <http://www.jpeg.org/>
4. <http://www.iso.org/iso/en/ISOOnline.frontpage>
5. <http://www.cs.sfu.ca/CourseCentral/365/li/material/notes/Chap4/Chap4.2/Chap4.2.html>
6. <http://www.cs.sfu.ca/CourseCentral/365/li/material/cgi-bin/whichjpeg.cgi>
7. <http://computer.howstuffworks.com/file-compression1.htm>
8. <http://www.lyvedon-way.fsnet.co.uk/compression/compress.html>
9. <http://datacompression.info/RLE.shtml>
10. http://www.campusprogram.com/reference/en/wikipedia/l/lz/lzw_compression_algorithm.html
11. [Sayood, K., "Introduction to Data Compression", Morgan Kaufmann, 2nd edition, 2000, ISBN: 1558605584](#)
12. [Miano, J., "Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP", ACM Press, Pearson Education, 1999, ISBN: 0201604434](#)
13. [Salomon, D., "Data Compression: The Complete Reference", Springer-Verlag, 3rd edition, 2004, ISBN: 0387406972](#)

3. GRAPH ALGORITHMS

3.1 INTRODUCTION

A graph search or a graph traversal algorithm is an algorithm which systematically passes through some or all the nodes in a graph, often with the goal of finding a particular node, or one with a given property. Searching a graph can be a very complex task as the process requires us to visit the root nodes and sub nodes but thankfully a digital computer can be programmed relatively easily.

3.2 DEPTH-FIRST SEARCH

In the depth first search the algorithm progress through the graph by first selecting a node and expanding and progressing down along that node path until either the goal is reached, or not, at which point it backtracks to the previous node and selects another path. For this reason it is often known as the backtracking algorithm. Figure 1 below shows an example of a DFS tree. The numbers represent the sequence in which the nodes are visited.

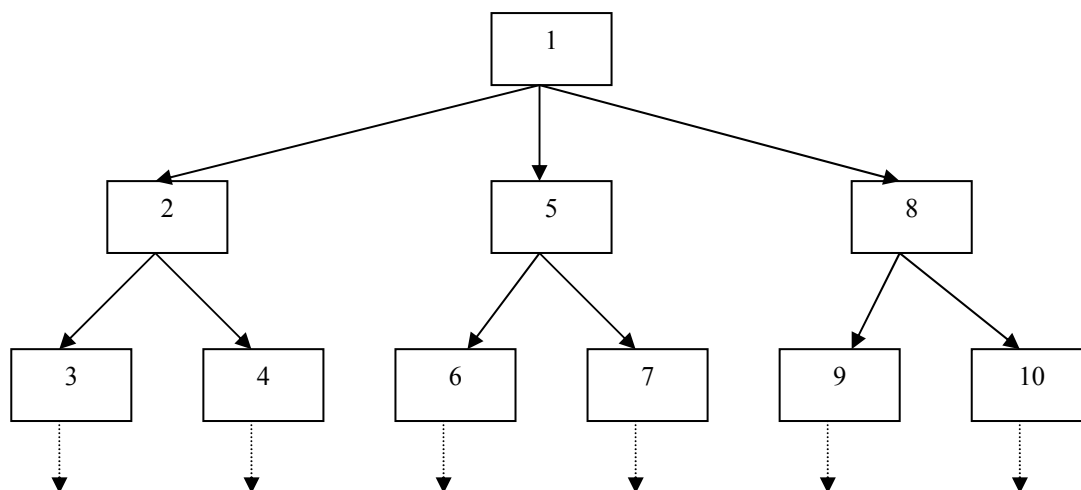


Figure 1. A DFS search tree.

In a weighted graph, the decision of which node to follow down would be based on the weight of the node. For example, in the above tree if node 5 had a lower weight than node 2, then node 5 would have been expanded first and followed through to its end, before backtracking back to expand nodes 2 and 8.

3.3 BREADTH-FIRST SEARCH

In the BFS the algorithm progress through the graph by first expanding all the nodes reachable from the current node, then, checks to see if the goal has been reached from all newly created nodes. If not it then expands all the nodes on the next level down, below is tree showing this progression, and again the numbers represent the sequence the nodes are visited (see figure 2).

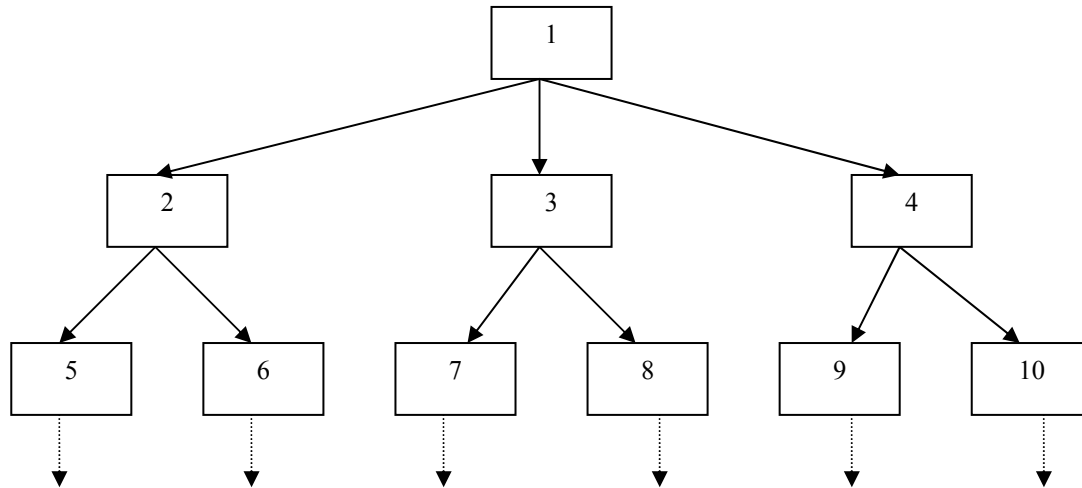


Figure 2. A BFS search tree.

Just as in the weighted DFS, in the weighted BFS the decision of which node to follow through would be based on the weight of each individual node. As an example in the above node, if node 3 had a lower weight than node 2, then nodes 5 and 6 would be descendants of Node 3. After which the algorithm would expand the nodes 2 and 4 according to their weights.

3.4 DIJKSTRA

Dijkstra's algorithm is a tree-traversing algorithm that finds the single source shortest path problem on a directed, weighted graph (provided all weights are above zero). The algorithm progresses by maintaining a set of vertices (S) from the source (s), whose distance has already been calculated. To begin with the set is empty. It further progresses by adding the vertex (u) with the shortest distance from s into set S . Next all edges leaving u are followed, and the distances from the origin are calculated. If the new distance is found to be smaller than that already placed in S then it is updated. This is repeated until all the vertices reachable have been inserted in S . The progression of the algorithm ensures that the shortest path is always the one found.

Due to its nature of always selecting the shortest path, this is an example of a greedy algorithm, and although greedy algorithms do not always ensure the best results, this particular algorithm leads to the best result. This is guaranteed since every time node u is added to S , it has the smallest distance to s compared to the rest of the nodes not in s .

Shortest Path Problem

Given a connected graph $G=(V,E)$, a weight $d:E \rightarrow \mathbb{R}^+$ and a fixed vertex s in V , find a shortest path from s to each vertex v in V .

Dijkstra's algorithm is known to be a good algorithm to find a shortest path.

1. Set $i=0$, $S_0 = \{u_0=s\}$, $L(u_0)=0$, and $L(v)=\text{infinity}$ for $v \neq u_0$. If $|V| = 1$ then stop, otherwise go to step 2.
2. For each v in $V \setminus S_i$, replace $L(v)$ by $\min\{L(v), L(u_i)+d_{v u_i}\}$. If $L(v)$ is replaced, put a label $(L(v), u_i)$ on v .
3. Find a vertex v which minimizes $\{L(v): v \in V \setminus S_i\}$, say u_{i+1} .
4. Let $S_{i+1} = S_i \cup \{u_{i+1}\}$.
5. Replace i by $i+1$. If $i=|V|-1$ then stop, otherwise go to step 2.

The time required by Dijkstra's algorithm is $O(|V|^2)$. It will be reduced to $O(|E|\log|V|)$ if a heap data structure is used to keep $\{v \in V \setminus S_i : L(v) < \text{infinity}\}$.

3.5 KRUSKAL

Minimum Spanning Tree Problem:

Given a connected graph $G=(V,E)$ and a weight $d:E \rightarrow \mathbb{R}^+$, find a minimum spanning tree T .

Kruskal's Algorithm

1. Set $i=1$ and let $E_0=\{\}$
2. Select an edge e_i of minimum value not in E_{i-1} such that $T_i=\langle E_{i-1} \cup \{e_i\} \rangle$ is acyclic and define $E_i=E_{i-1} \cup \{e_i\}$. If no such edge exists, Let $T=\langle E_i \rangle$ and stop.
3. Replace i by $i+1$. Return to Step 2.

The time required by Kruskal's algorithm is $O(|E|\log|V|)$.

3.6 PRIM

Prim's algorithm is also known to be a good algorithm to find a minimum spanning tree.

1. Set $i=0$, $S_0= \{u_0=s\}$, $L(u_0)=0$, and $L(v)=\text{infinity}$ for $v \neq u_0$. If $|V| = 1$ then stop, otherwise go to step 2.
2. For each v in $V \setminus S_i$, replace $L(v)$ by $\min\{L(v), d_v^{u_i}\}$. If $L(v)$ is replaced, put a label $(L(v), u_i)$ on v .
3. Find a vertex v which minimizes $\{L(v): v \text{ in } V \setminus S_i\}$, say u_{i+1} .
4. Let $S_{i+1} = S_i \cup \{u_{i+1}\}$.
5. Replace i by $i+1$. If $i=|V|-1$ then stop, otherwise go to step 2.

The time required by Prim's algorithm is $O(|V|^2)$. It will be reduced to $O(|E|\log|V|)$ if heap is used to keep $\{v \text{ in } V \setminus S_i : L(v) < \text{infinity}\}$.

3.7 COMPARISON OF DEPTH-FIRST AND BREADTH-FIRST SEARCH

Experimental Analysis

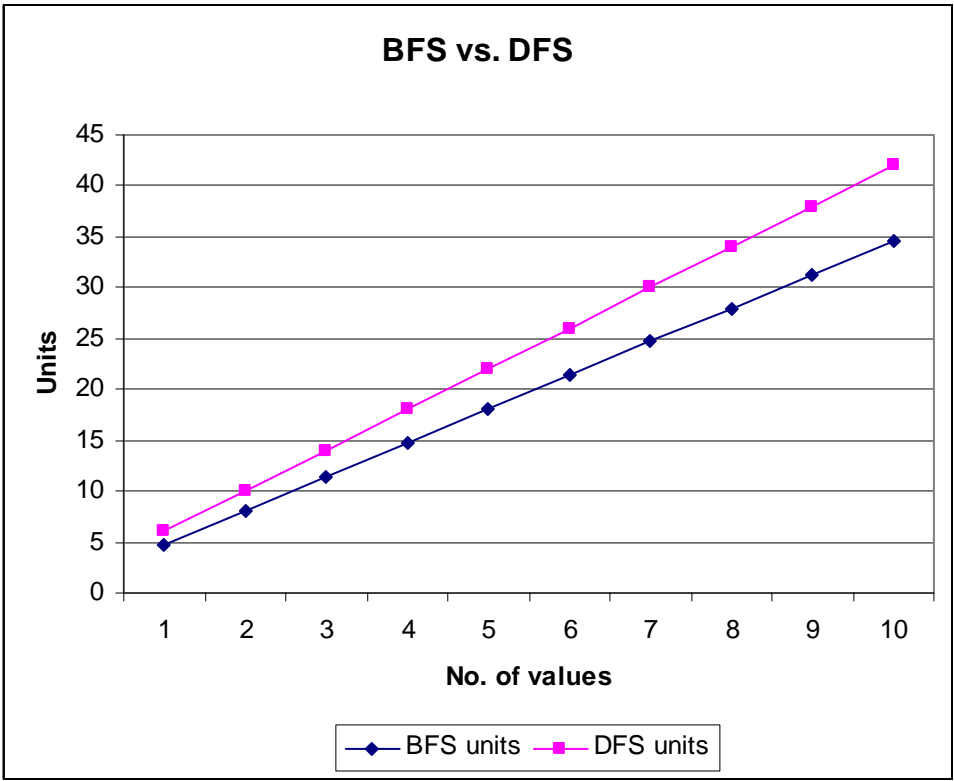
In the experiment to compare the performance of the two algorithms an increasing number of items to search through was assigned. The number of items to search through varied from 4 to 22. A significant difference in performance and efficiency was not noted.

For the DFS its efficiency is hugely dependent on how well the nodes to follow through are selected. If the wrong node is selected first, you could end up spending a huge time exploring the tree when the node lies on another path or you can get a much longer path when a shorter path exists. For the BFS search you eliminate these factors, meaning time is not wasted following fruitless paths, and you are guaranteed to always find the shortest path. Another downfall of the DFS is the time it spends backtracking through the tree, this can also lead to lengthy searches.

On the other hand the BFS might take a longer if the search tree is broad or the goal lies real deep. The selection of which node to expand first will also affect its efficiency, although not as greatly as the DFS. Another thing to keep in mind is that with the BFS we will always need to have a set of current nodes, whilst the DFS only keep one current node, meaning there is a memory space factor with the BFS.

N	BFS units	DFS units
4	4.67	6.00
6	8.00	10.00
8	11.33	14.00
10	14.67	18.00
12	18.00	22.00
14	21.33	26.00
16	24.67	30.00
18	28.00	34.00
20	31.33	38.00
22	34.67	42.00

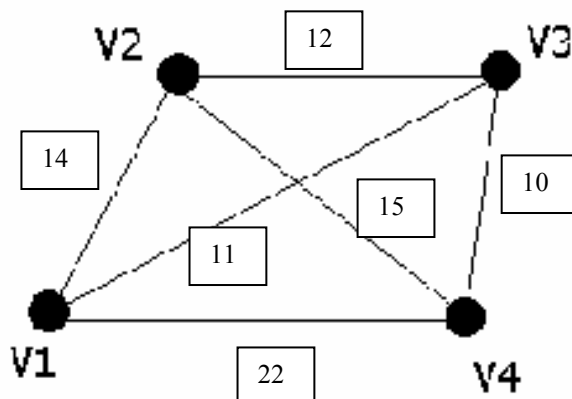
Table 3. Table for time measure.



3.8 EXERCISES

- For the weighted complete graph over four vertices shown in figure given below, explain clearly the steps involved in finding the shortest path from V1 to V4 using the following algorithms:

- Exhaustive depth-first search with ordering (V1, V2, V3, V4)
- Exhaustive breadth-first search with ordering (V1, V2, V3, V4)



- Explain the terms brute-force and greedy as used to describe certain algorithms. For the weighted graph in figure Q2, show clearly the steps involved in finding the shortest path from B to E using the following algorithms:

- Exhaustive depth-first search with ordering (B,R,U,T,E)
- Exhaustive breadth-first search with ordering (B,R,U,T,E)
- Best-first search (expand best one node according to weight)

Classify each of the algorithms in parts (a), (b) and (c) according to either brute-force or greedy.

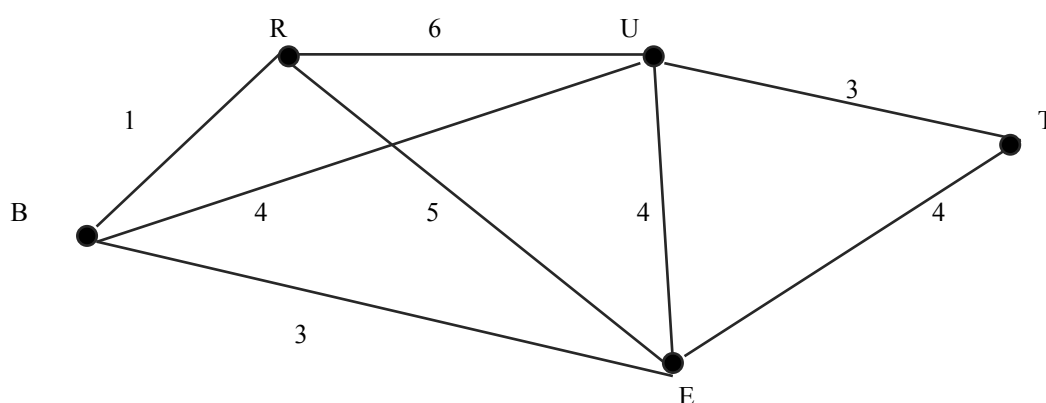


Figure Q2. A weighted graph over vertices B, R, U, T and E.

Table 5. The following are famous names associated with graphical algorithms. Prepare a concise web biography of each in a format similar to that shown for Albert Einstein at <http://www.algana.co.uk/FamousNames/E/einstein.htm>

- (b) Edsger Wybe Dijkstra (Dijkstra's algorithm),
- (c) Joseph B. Kruskal (Kruskal's algorithm)
- (d) Robert Prim (Prim's algorithm).

3.9 BIBLIOGRAPHY & REFERENCES

1. <http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Dijkstra.shtml>
2. <http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/kruskal/Kruskal.shtml>
3. <http://www.algana.co.uk/Algorithms/Graphs/Prim/Prim.html>
4. <http://www.algana.co.uk/Algorithms/Graphs/Prim/challenge.html>

4. GRAPHICS

4.1 INTRODUCTION

Graphics algorithms are algorithms primarily used in the process of generating computer graphics, although they can have a variety of secondary applications in various branches of computer science and engineering. There are many variations of algorithms in graphics and these include primitive drawing enhancement, transformation pipeline, linear decoration, non-linear decoration, hidden line removal and surface removal algorithms. In this section we discuss line drawing algorithms and also introduce main midpoint algorithms.

In geometrical computing, it is important to draw a line effectively and quickly, as it is a very basic geometric shape. The objective of any line drawing algorithm is always to construct a good possible approximation of an ideal line. The qualities that are considered in order to examine the accuracy and efficiency of such an algorithm are:

- Continuous appearance
- Uniform thickness and brightness
- Fast generation of the line

References for related algorithms are given at the end of the section.

Several interactive graphics applets are available at:-

<http://www.algana.co.uk/Algorithms/Graphics/GraphicsFrameset.htm>

These include:

- 3D Rotation
- Boundary Fill
- Cubic Bézier curve
- Cohen-Sutherland Clipping
- DDA
- Graph Transformation
- Julia Set Fractal
- Lighting Model Midpoint (Line)
- Midpoint (Arc)
- Newton's Fractal

4.2 BRESENHAM

This is a famous scan conversion algorithm used to draw raster graphic line on a computer screen. The algorithm draws lines by choosing appropriate pixels at specific points on the screen to realise a straight line. To draw curves and other shapes it requires some modification of formula used to calculate which position to draw the next pixel.

The algorithm treats the screen like a grid, identifies two points in the grid, one marking the start, the other marking the end, and attempts to put pixels between these two points in the best possible way to fit a line. To find the best fit points to put the pixels it decides by choosing whether to put the next pixel at either position $(x+l, y)$ or $(x+l, y+l)$, where position (x, y) is the current position, and l is the slope of the line. The decision of which of these points should be used as the next pixel, is made by finding which of the two options is closer to the actual line that we wish to draw. The decision is made by calculating how much error lies at position $(x+l, y)$. If this error is greater than 0.5, then position $(x+l, y+l)$ is chosen, else position $(x+l, y)$ is used. A pseudo-code description is as follows.

```

Current = (x, y)
Repeat until Current = (xend, yend)
Plot (Current)
If (error (xCurrent+l, yCurrent) < 0.5) then
    Current = (xCurrent+l, yCurrent)
Else
    Current = (xCurrent+l, yCurrent+l)

```


4.3 DDA

This algorithm is also a scan conversion algorithm, just as Bresenham's algorithm. This works as well by selecting which pixels to put on between two points, but instead of calculating the next point by looking at the errors between the possible choices, this algorithm always increments the value along one axis and calculates the value of the other co-ordinate. The decision to which co-ordinate to move at a constant step is by finding which of the two have a greater gradient, i.e., if $d_y < d_x$, then co-ordinate x will move in steps, and co-ordinate y will be calculated.

```

Current = (x, y)
Calculate  $d_y$  and  $d_x$ 
If ( $d_y > d_x$ ) then
  Steps =  $d_x$ 
Else
  Steps =  $d_y$ 
 $x_{inc} = d_x / \text{steps}$ 
 $y_{inc} = d_y / \text{steps}$ 
Repeat until Current = ( $x_{end}$ ,  $y_{end}$ )
Plot (Current)
 $x_{Current} = x_{Current} + x_{inc}$ 
 $y_{Current} = (x_{Current} + l, y_{Current} + y_{inc})$ 

```

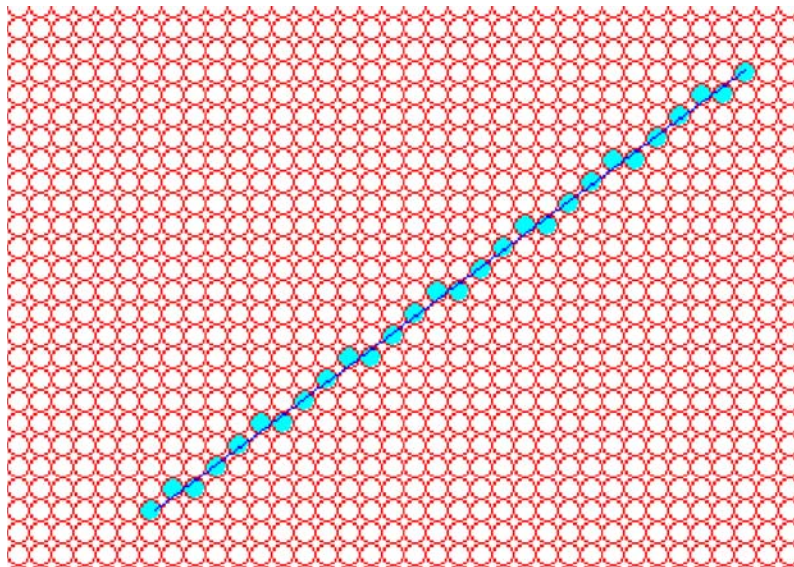


Figure 1. The applet at ref [1] demonstrates the line drawing algorithm using the Digital Differential Analyser (DDA) method. Select any two points on the pseudo screen with mouse clicks. The 'ideal line' is first drawn as thin blue straight line between the two selected points. Determining appropriate intermediate pixels using the DDA algorithm enables the line to be drawn. Click 'clear' to refresh the screen, and to select a new pair of points.

4.4 MIDPOINT LINE

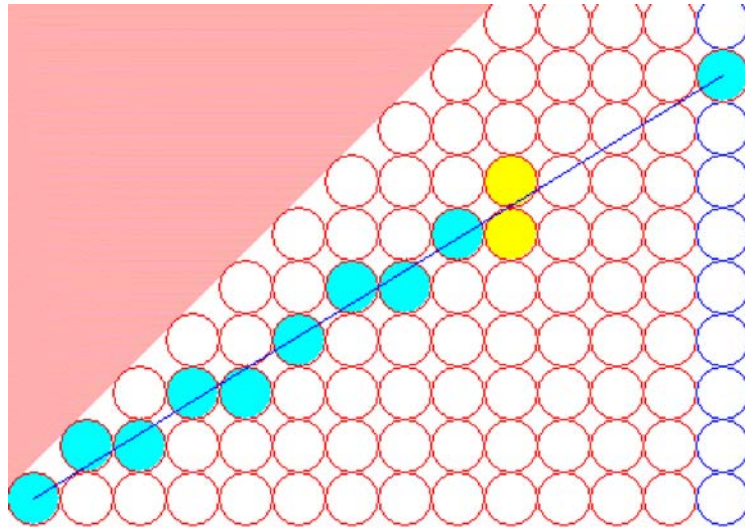


Figure 2. The applet at ref [2] demonstrates the line drawing algorithm using the midpoint line algorithm in drawing a line of slope less than 1. Circular regions represent the pixels. Select any pixel on the right hand side of the window. A line connecting this point and the origin is then drawn using the midpoint algorithm. The candidate pixels at each stage are also shown.

4.5 MIDPOINT CIRCLE

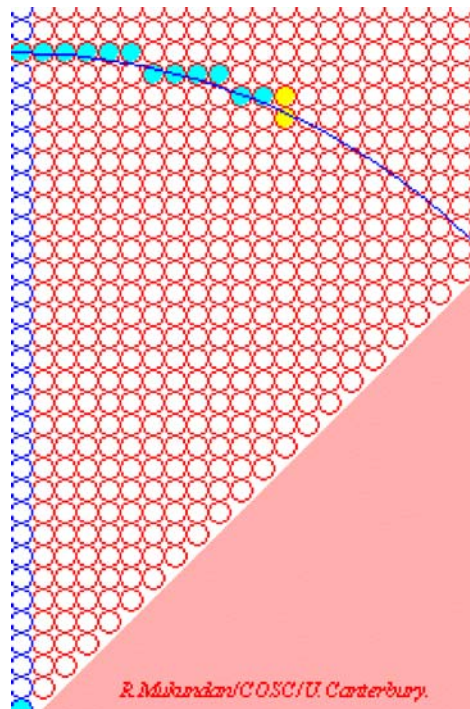


Figure 3. The applet at ref [3] demonstrates the working of the midpoint circle algorithm, in drawing a circular arc. A pseudo screen is used for an enlarged representation of the pixels. Select any pixel on the left hand side of the window. A circular arc from this point, with the origin at the centre, is then generated using the midpoint algorithm. The candidate pixels at each stage are also shown.

4.6 JULIA SETS

We can create fractal images by plotting Julia sets derived from iterating complex functions of the form $z^2 + c$, varying z over the complex plane for a given value of c . For each complex point z that is not in the set, the “escape velocity” of the point’s orbit is represented by a shade of grey – the more iterations, the darker the shade. For each point z that is in the set, its color is determined by the modulus of the final computed orbit value.

To run the demo at <http://www.apropos-logic.com/nc/JuliaFractal.html>:

1. Enter the real and imaginary parts of the complex value c , and then press the **Manual** button. Some values of c that generate interesting fractal images include:

Real	Imaginary	Image
0.30900264	-0.0339787	Claws
0.33843684	-0.4211402	Snowflakes
-0.3985014	0.5848901	Turtles
-0.8184639	-0.2129812	Serpents
-0.3346212	0.6340579	Amoebas
-0.5530404	0.5933997	Sparklers

2. You can choose one of these from the drop-down list.
3. Or, press the **Random** button to generate a random value for c . Only 5-10% of random values produce interesting fractal images, so you may need to press this button repeatedly.
4. Because the fractal images are recursive, you can use the mouse to drag a rectangle around an area of an image that you want to zoom into.

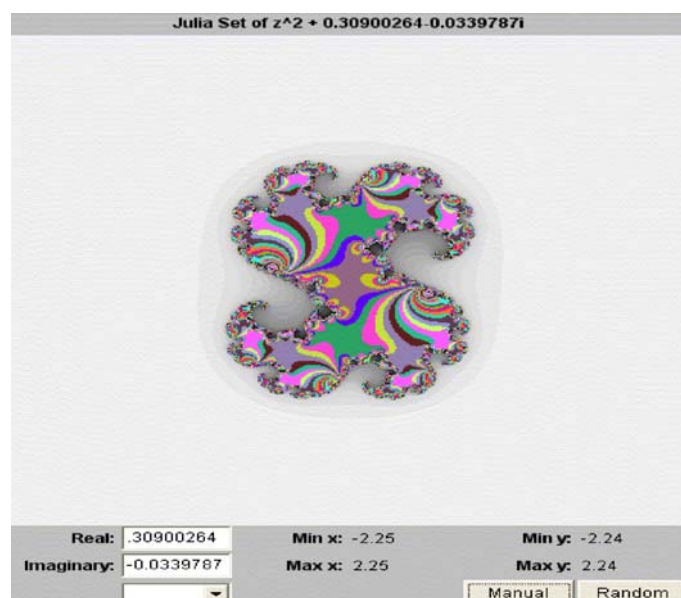


Figure 4. The “claws” fractal.

4.7 COMPARISON OF BRESENHAM AND DDA

Experimental Analysis

The experiment was done giving both algorithms to draw a line; the line has an origin at point (0, 0) and a randomly generated end point. Adding an extra digit increased the number of digits in the second point.

The graph plotted was for log (time) since the values obtained were too big. Both the algorithms are exponential $\{O(e^n)\}$, noted by the straight line, but the only difference is that Bresenham's algorithm increases at a faster rate. Looking at the results the obvious superior algorithm is the DDA Algorithm. The reason for this is that the DDA algorithm performs less calculation. Hence it is generally faster.

While when we analyse Bresenham's algorithm, we see for each point it has to plot, it must calculate the error value. This small extra step increases the time taken to run. It should also be noted that Bresenham's algorithm is the one that will draw a more accurate line out of the two algorithms, this is due to the fact that at each point it takes into consideration the error present due to the fact that you can't draw true lines on a computer screen.

Digits	Xend	Yend	LineDDA	LineBres
1	9	4	13	12
2	98	12	13	17
3	622	717	51	70
4	1871	9951	603	811
5	92942	55387	5567	5961
6	251355	298614	17869	27000
7	6065758	1268361	371498	426592
8	39888628	16478338	2426965	2762782
9	174287673	215901763	13562633	17325658
10	1648819367	1589183427	100360342	108193715

Table 5. A table of the computation steps required for the same line drawn using DDA and Bresenham's algorithm.

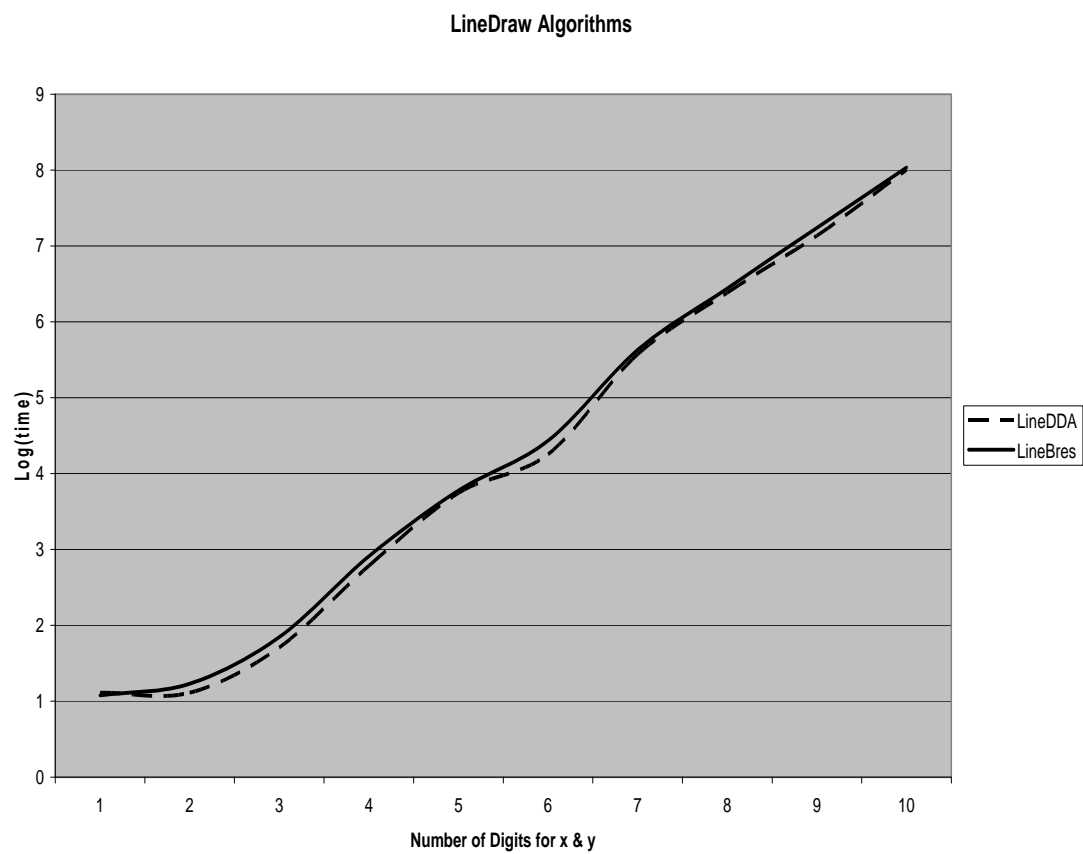


Figure 6. A plot of values shown in table 5.

A brief biography of Jack Bresenham can be found at ref [\[4\]](#).

A good introduction to other line drawing algorithms can be found at ref [\[5\]](#).

4.7 CASE STUDY

The “Design and Implementation of an Educational Java Module” senior project of Hassan Chahrour (2006) provides an interactive module for use in an algorithms course (see figure 7).

Figure 7. A screen shot of the final running program (resized).

Treating this project as a case study, analyse the module design and adapt it to include the following graphics algorithms:

- (i) 3D Rotation
- (ii) Boundary Fill
- (iii) Cubic Bézier curve
- (iv) Cohen-Sutherland Clipping
- (v) DDA
- (vi) Graph Transformation
- (vii) Julia Set Fractal
- (viii) Lighting Model Midpoint (Line)
- (ix) Midpoint (Arc)
- (x) Newton's Fractal

4.8 BIBLIOGRAPHY & REFERENCES

1. <http://www.cosc.canterbury.ac.nz/people/mukundan/cogr/DDA.html>
2. <http://www.cosc.canterbury.ac.nz/people/mukundan/cogr/LineMP.html>
3. <http://www.cosc.canterbury.ac.nz/people/mukundan/cogr/CircleMP.html>
4. <http://www.algana.co.uk/FamousNames/B/bresenham.htm>
5. <http://graphics.lcs.mit.edu/~mcmillan/comp136/Lecture6/Lines.html>

5. NUMERICAL

5.1 INTRODUCTION

Number theory and the algorithms relating to the properties of numbers occupy an important place in computer science and engineering. In this section, we study some of these algorithms but to be sure there are many others. Of course, number types (such as power, prime, Mersenne and Sophie Germain) and number sequences (such as arithmetic, geometric and Fibonacci) have all been widely investigated for many years so it is not surprising that some related algorithms date back hundreds of years.

Some of the algorithms for numerical computation considered in this section are:

1. Naïve Greatest Common Divisor
2. Euclid
3. Fibonacci iterative
4. Fibonacci recursive
5. Sieve of Eratosthenes

References for these and other related algorithms are given at the end of the section.

Interactive applets for the following numerical algorithms are available at www.algana.co.uk via the “algorithms”, “numerical” links.

- Euclid
- Fibonacci
- Genetic
- Magic
- Prime
- Prime largest

5.2 EUCLID

Finding the Greatest Common Divisor is an important problem in computer science. Naïve and Euclid algorithms are most commonly used algorithms to find the Greatest Common Divisor.

Naive algorithm

To find the Greatest Common Divisor (or the GCD) of two numbers using a naïve algorithm we need to find the prime factors of the two numbers. Then, the common factor of them is the GCD of these two numbers.

For example: Find $\text{GCD}(48, 27)$.

$$\text{Factorisation for } 48 = 2 \times 2 \times 2 \times 2 \times 3 = 2^4 \times 3^1$$

$$\text{Factorisation for } 27 = 3 \times 3 \times 3 = 3^3$$

From observation, we can see that 3^1 is present as a greatest common factor in both numbers. Therefore, $\text{GCD}(48, 27) = 3$.

We can see that the major task involved in this approach is to carry out the prime factorisation. Typically, we achieve the factorisation by repeated division by 2 until a non-integer remainder is obtained. Then divide by 3, then by 5 and so on. Clearly, finding $\text{GCD}(N, M)$, where N and M are large numbers, by the naïve algorithm involves many computations.

Practice 5.1 Find $\text{GCD}(64, 28)$.

Euclid's algorithm

The writings of Euclid (ref. [1](#)) contain references to an alternative algorithm for finding the GCD of two numbers. The so-called Euclid's algorithm is a relatively simple means of achieving the GCD compared to the naïve algorithm and certainly involves fewer computations. To calculate the GCD of two numbers, we repeatedly calculate the remainder after dividing the smaller number into the larger number. If the remainder turns out to be zero, then the final smaller number is the GCD.

For example: Find $\text{GCD}(168, 132)$.

$$168 \bmod 132 = 36$$

$$132 \bmod 36 = 24$$

$$36 \bmod 24 = 12$$

$$24 \bmod 12 = 0$$

$$\text{Therefore, } \text{GCD}(168, 132) = 12$$

Observe that the penultimate step, before getting zero for the remainder, yields the value of GCD for the two integers.

Practice 5.2 Find $\text{GCD}(64, 28)$ using Euclid's algorithm.

5.3 FIBONACCI

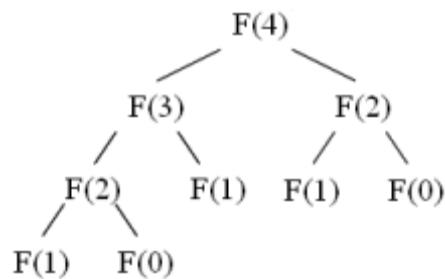
The Fibonacci sequence is a set of recursively defined numbers. The n^{th} number is defined by adding the previous two numbers.

Recursive Fibonacci

Fibonacci series could be computed recursively using the following function.

Fib (n)

```
{
    if ( n > 2 )
        return Fib( n - 1 ) + Fib ( n - 2 )
    else
        return 1
}
```



Improved Fibonacci

As we have noticed that in recursive definition, some values are computed over and over. In order to avoid that we could use dynamic programming concepts. Since we know that the algorithm always depends on the previously calculated numbers, we add a form of learning to the algorithm so that we store the numbers already calculated to prevent repetition of calculating numbers we have already come across. E.g. in the example above if we store the $F(3)$ when we calculate $F(4)$, we don't have to regenerate $F(3)$ when we try to calculate $F(3) + F(4)$. Compute $F(2)$, then $F(3)$, then $F(4)$, etc. using stored values for smaller $F(n)$ values to compute next value. Each $F(n)$ value is computed just once.

e.g.

We can calculate $F(n)$ in linear time by storing small values.

$$F[0] = 0;$$
$$F[1] = 1;$$

```

for i = 2 to n

```

$$F[i] = F[i - 1] + F[i - 2];$$

```

return F[ n ]

```

Iterative Fibonacci

This works to speed up the algorithm since it prevents repetition of calculations already performed earlier. This sort of algorithm is one that gets more efficient with time. The limitation of this algorithm is its need for storage space to keep track of the numbers that it has encountered.

Explicit formula could be used to calculate the Fibonacci series.

$$\text{Fib}(n) = \frac{1}{\sqrt{5}} * a^n - \frac{1}{\sqrt{5}} * b^n$$

where

$$a = (1 + \sqrt{5}) / 2$$

$$b = (1 - \sqrt{5}) / 2$$

5.4 SIEVE OF ERATOSTHENES

An integer greater than one is called a prime number if its only positive divisors (factors) are one and itself. For example, the prime divisors of 10 are 2 and 5; and the first six primes are 2, 3, 5, 7, 11 and 13. In the nineteenth century it was shown that the number of primes less than or equal to n approaches $n/(\log n)$ as n gets very large; so a rough estimate for the n th prime is $n \log n$.

For finding all the small primes, say all those less than 10,000,000,000; one of the most efficient ways is by using the Sieve of Eratosthenes (ca 240 BC):

- make a list of all the integers less than or equal to n (greater than one)
- strike out the multiples of all primes less than or equal to the square root of n
- numbers that are left are the primes.

For example, to find all the odd primes less than or equal to 100 we first list the odd numbers from 3 to 100 (why even list the evens?) The first number is 3 so it is the first odd prime--cross out all of its multiples. Now the first number left is 5, the second odd prime--cross out all of its multiples. Repeat with 7 and then since the first number left, 11, is larger than the square root of 100, all of the numbers left are primes. This method is so fast that there is no reason to store a large list of primes on a computer--an efficient implementation can find them faster than a computer can read from a disk.

To find individual small primes trial division works well. To test n for primality (to see if it is prime), just divide by all of the primes less than the square root of n . For example, to show 211 is a prime, we just divide by 2, 3, 5, 7, 11, and 13. Sometimes the form of the number n makes this especially effective.

Rather than divide by just the primes, it is sometimes more practical to divide by 2, 3 and 5; and then by all the numbers congruent to 1, 7, 11, 13, 17, 19, 23, and 29 modulo 30 - again stopping when you reach the square root. This type of factorization is sometimes called wheel factorization. It requires more divisions (because some of the divisors will be composite), but does not require us to have a list of primes available.

Suppose n has twenty-five or more digits, then it is impractical to divide by the primes less than its square root. If n has two hundred digits, then trial division is impossible--so we need much faster tests. We discuss several such tests below.

Mersenne primes are primes of the form $2^p - 1$. These are the easiest type of number to check for primality on a binary computer so they usually are also the largest primes known.

Twin primes are primes of the form p and $p+2$, i.e., they differ by two. It is conjectured, but not yet proven, that there are infinitely many twin primes (the same is true for all of the following forms of primes). Because discovering a twin prime actually involves finding two primes, the largest known twin primes are substantially smaller than the largest known primes of most other forms.

5.5 COMPARISON OF NAÏVE AND EUCLID

Experimental Analysis

In the naïve algorithm all the factors of each number need to be calculated. However with the Euclid algorithm we just divide one number with the other. The complexity of this algorithm is $O(N)$.

In order to examine the efficiency of both algorithms, values of twenty integer pairs are used to find the gcd as well as the total calculation time for both algorithms. Collected data is shown below.

Number A	Number B	Naïve (ms)	Euclid(ms)
1581754094	310732020	60	1
74562812	6212983	100	1
1658459435	1330442124	380	1
536781234	87654932	410	1
348970141	211422209	921	1
603372504	185704437	2183	1
884747213	741564928	4276	1
622443169	135183739	4617	1
805824185	378392636	5468	1
799762693	403897678	6810	1
1688149592	17648573	7180	1
987435978	436451954	7360	1
639674474	587671490	10936	1
1619710367	1537685005	15602	1
625924561	354622932	21661	1
2063572617	1817638072	23564	1
719035631	667016144	26217	1
765095473	473708001	26228	1
1211693033	380057865	41800	1
1234567891	129876598	41890	1

Table 1. Computation data for Euclid vs Naïve.

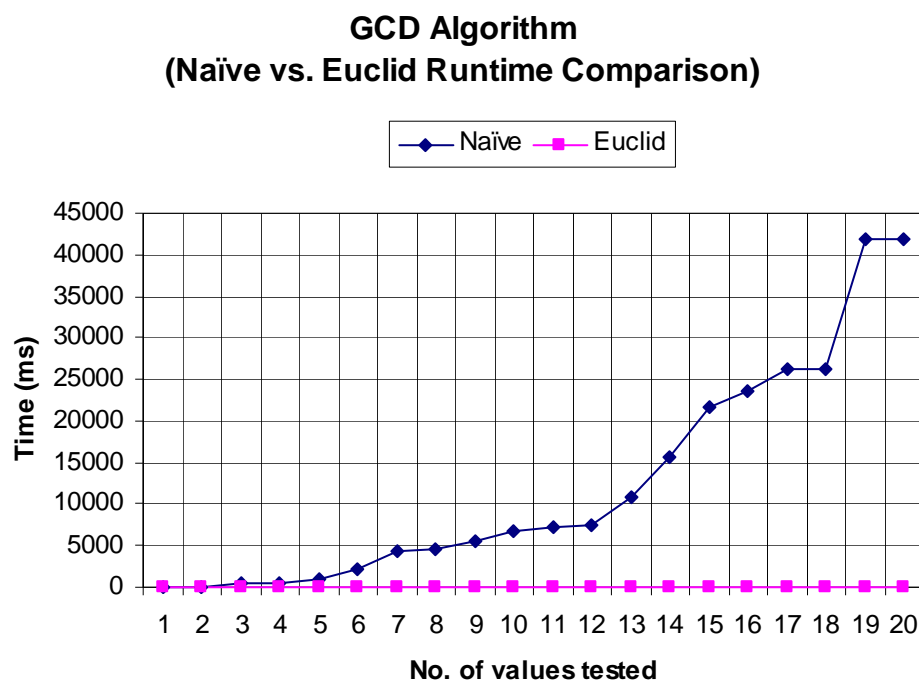


Figure 1. A plot of data shown in table 1.

Conclusions

From graphical results, it is apparent that Euclid's algorithm is a significant time-wise improvement over the Naïve algorithm. If we even keep increasing the multitude of numbers involved, the runtime for Euclid stays closer to 0 ms while runtime for Naïve could range anywhere between 1000 – 45000 ms. Less computing resources are required in order to get immediate result with Euclid algorithm while in most cases, Naïve algorithm makes substantial use of processor time as well as other computing power.

5.6 COMPARISON OF FIBONACCI ITERATIVE, RECURSIVE AND MODIFIED RECURSIVE

Experimental Analysis

N	Recursive(ms)	Iterative(ms)	Improved(ms)
20	8	1	0
25	10	1	45
30	230	1	230
35	930	1	400
37	2494	2	800
40	10285	2	1600

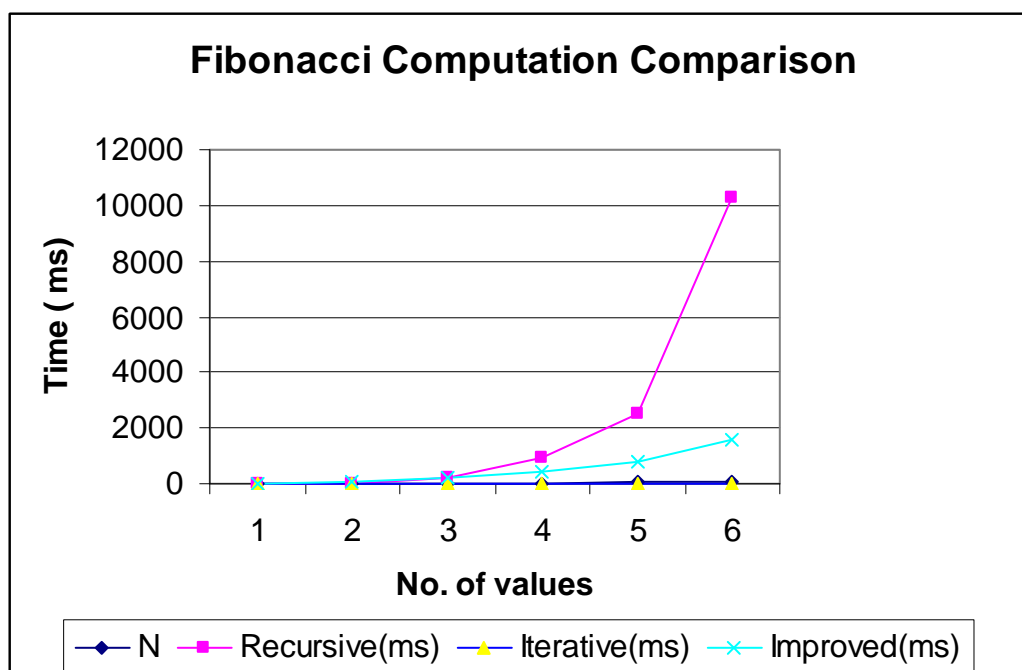


Figure 3. Three implementations of Fibonacci: iterative, recursive, and modified recursive.

Conclusions

As the results show, the iterative version of the Fibonacci algorithm is the most effective, giving us practically constant time calculation. Whilst the direct implementation is the least efficient, time wise. The reason for such variation in the times is that the recursive algorithm has to calculate the preceding numbers, not once, but at least twice. The modified recursive is a bit smarter, and stores the past values to help it work faster. The iterative is extremely efficient since it is just a continuous loop. The larger the number being calculated, the more space needed to stored the values already calculated. The iterative algorithm doesn't use require any storage space, apart from keeping track of the last two values. While the modified recursive and recursive will each continuously allocate new memory space as they call themselves recursively.

5.7 EXERCISES

1. Show clearly how Euclid's algorithm can be used to find the greatest common divisor of the two numbers 1244 and 792.
2. The following are famous names associated with numerical algorithms. Prepare a concise web biography of each in a format similar to that shown for Albert Einstein at <http://www.algana.co.uk/FamousNames/E/einstein.htm>
 - (a) Euclid (Euclid's algorithm),
 - (b) Hudalricus Regius (prime numbers),
 - (c) Marin Mersenne (Mersenne numbers),
 - (d) Marie-Sophie Germain (Sophie Germain numbers),
 - (e) Eratosthenes of Cyrene (Sieves algorithm).
3. Write down the first six terms of the following recurrence formulae:
 - (a) $T(N) = T(N/2) + 1$, given that $T(1) = 1$,
 - (b) $T(N) = T(N/2) + N$, given that $T(1) = 1$,
 - (c) $T(N) = 2 T(N/2) + 2$, given that $T(1) = 2$,
 - (d) $T(N) = 2 T(N/2) + N$, given that $T(1) = 2$.

where N is a power of 2.

4. For the three comparison pairs of time complexities of two algorithms A and B shown in table Q4, find the least positive integer N for which algorithm B is better than algorithm A.

Comparison	A	B
1	$32N$	N^2
2	$N^3 + 16$	$N^2 + 64$
3	$2N - 1$	$\lg(64N)$

Table Q4.

5. Given that k_1 , k_2 , k_3 are constants, find the orders of the following functions for large N :
 - (i) $F_1 = 2 * (k_1 * (k_2 + N^2)) + (k_3 * \lg N)$,
 - (ii) $F_2 = (k_1 * \lg N * (k_2 + N)) + (k_3 * N) * \lg N$.
6. Research the largest known prime number and summarize how it was found. Find the next prime number after that (just kidding!)

5.8 BIBLIOGRAPHY & REFERENCES

1. <http://www.algana.co.uk/FamousNames/E/euclid.htm>
2. <http://www.algana.co.uk/Algorithms/numerical/gcd/gcd.html>
3. <http://www.utm.edu/research/primes/largest.html>
4. <http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/fib.html>

6. OPERATING SYSTEMS

6.1 INTRODUCTION

Operating systems are an important component of computer systems. In this section, we study some of these algorithms.

Some of the algorithms for operating systems considered in this section are:

References for these and other related algorithms are given at the end of the section.

6.2 VARIOUS OS ALGORITHMS

Election

A network of processors has to solve a problem for which a certain coordination is needed. Many distributed algorithms require one process to act as coordinator, initiator, sequencer, or otherwise perform some special role. This unique processor known by all the others and that typically, knows the others, coordinates the work of the others by assigning them sub-tasks. This coordinator will act as the leader who distributes to the processors under its coordination some information defining the status of the problem to be solved.

Election algorithms are for electing a coordinator. If all processors are exactly the same, with no distinguishing characteristics, there is no way to select one of them to be special. Consequently, it is assumed that each process has a unique number, for example its network address. In general, election algorithms attempt to locate the process with the highest process number and designate it as coordinator.

Furthermore, we also assume that every process knows the process number of every other process. What the processors do not know is which ones are currently up and which ones are currently down. The goal of an election algorithm is to ensure that when an election starts, it concludes with all processors agreeing on who the new coordinator is to be.

Producer Consumer

Generally there are two entities in Producer-Consumer algorithm:

- Producer - an entity that generates data,
- Consumer - an entity that uses the data that was generated.

There always is a relationship between these two entities as they have to know about the state of each other, e.g. producer has to know, when a consumer is free to consume data, thus it is allowed to generate data. At the same time consumer has to know, when new data has been produced so consumer could consume it.

In real life it often happens that producers and consumers have different efficiencies. It is usually not a problem if consumer is highly efficient, as there is less data loss during data consumption (consumer consumes data when it is produced, and does nothing, when there is nothing to produce). However if producer is more efficient than consumer, then there is potential problem with loss of data as the consumer is given too much data with which to work. In this case some sort of storing device should be used, e.g. a buffer (or a queue). When some data is produced and no consumer is available, then that data goes to the buffer. When the consumer becomes available, the next piece of data it will consume will be the one from the buffer.

The most common real-life example of the Producer-Consumer algorithm is a process, called print spooling. Print spooling refers to putting jobs(in this case documents that are about to be printed) into a special location(buffer) in either computer memory, or hard disk, so that a printer could access this document whenever the printer is ready. There are couple of advantages of spooling. First of all the printer can access data from the buffer at any rate that is suitable for the printer. Secondly the work of computer is not interrupted while printing, thus a user can perform other tasks.

As it was mentioned mentioned above, producer and consumer have to know about the state of each other, so they would know when each one of them can produce/consume. If we have one producer and consumer, what would be the producer-consumer algorithm, so both would know when it is possible to producer/consume and when one of these should wait? HINT: Incorporate some sort of wait/signal messages so producer and consumer would inform each other when they are ready for production/consumption.

Round Robin Scheduling

A round robin is an arrangement of choosing all elements in a group equally in some rational order, usually from the top to the bottom of a list and then starting again at the top of the list and so on. A simple way to think of round robin is that it is about "taking turns." Used as an adjective, round robin becomes "round-robin."

In computer operation, one method of having different program process take turns using the resources of the computer is to limit each process to a certain short time period, then suspending that process to give another process a turn (or "time-slice"). This is often described as round-robin process scheduling. In sports tournaments and other games, round-robin scheduling arranges to have all teams or players take turns playing each other, with the winner emerging from the succession of events.

A round-robin story is one that is started by one person and then continued successively by others in turn. Whether an author can get additional turns, how many lines each person can contribute, and how the story can be ended depend on the rules. Some Web sites have been created for the telling of round robin stories by each person posting the next part of the story as part of an online conference thread.

Other algorithms for which interactive applets are available at www.algana.co.uk are:

- Banker
- Bully
- Priority Scheduling
- Resource Request
- Virtual Memory Clock

6.3 CASE STUDY

The “Design and Implementation of an Educational Java Module” senior project of Hassan Chahrour (2006) provides an interactive module for use in an algorithms course (see figure 7).

Figure 1. A screen shot of the final running program (resized).

Treating this project as a case study, analyse the module design and adapt it to include the following operating systems algorithms:

- (i) Banker
- (ii) Bully
- (iii) Election
- (iv) Priority Scheduling
- (v) Producer Consumer
- (vi) Resource Request
- (vii) RR Scheduling
- (viii) Virtual Memory Clock

6.4 BIBLIOGRAPHY & REFERENCES

1. <http://www.algana.co.uk/Algorithms/OperatingSystems/OSFrameset.htm>
2. http://www.cwu.edu/~cs_dept/hilights.html
3. <http://www.cis.ksu.edu/~dan/despot/pdpta01.pdf>
4. http://www.cs.nott.ac.uk/Seminars/seminars_2003.htm
5. <http://www-db.stanford.edu/people/hector.html>

7. SEARCHING

7.1 INTRODUCTION

The retrieval of a particular piece or pieces of information from large volumes of previously stored data is a fundamental searching operation that is intrinsic to a great many computational tasks. Often, the goal of the search is to find the items with keys matching a given search key. In this section we discuss two fundamental and widely used search methods, viz. sequential (or linear) search and binary search. These two algorithms are intrinsic to searching through arrays. They determine whether or not any of a sequence of objects appears among a set of previously stored objects.

Other searching algorithms of interest include:-

- Binary Tree
- Breadth-first/Depth-first³³
- RedBlack Tree

³³ BFS and DFS have previously been considered in the graph algorithms section.

7.2 SEQUENTIAL

The sequential search algorithm is a widely-used algorithm. It starts from the beginning of the array and moves up sequentially. This algorithm uses a brute force approach. At each step it checks whether the object at the current position is the one it is searching for, if it is, it returns the object, if its not, it move to the next position in the array. This is repeated until the object wanted is found, or the end of the array if reached.

1. For pos = start(array) to end(array)
 - 1.1. If array[pos] = target
 - 1.1.1 Return array[pos]

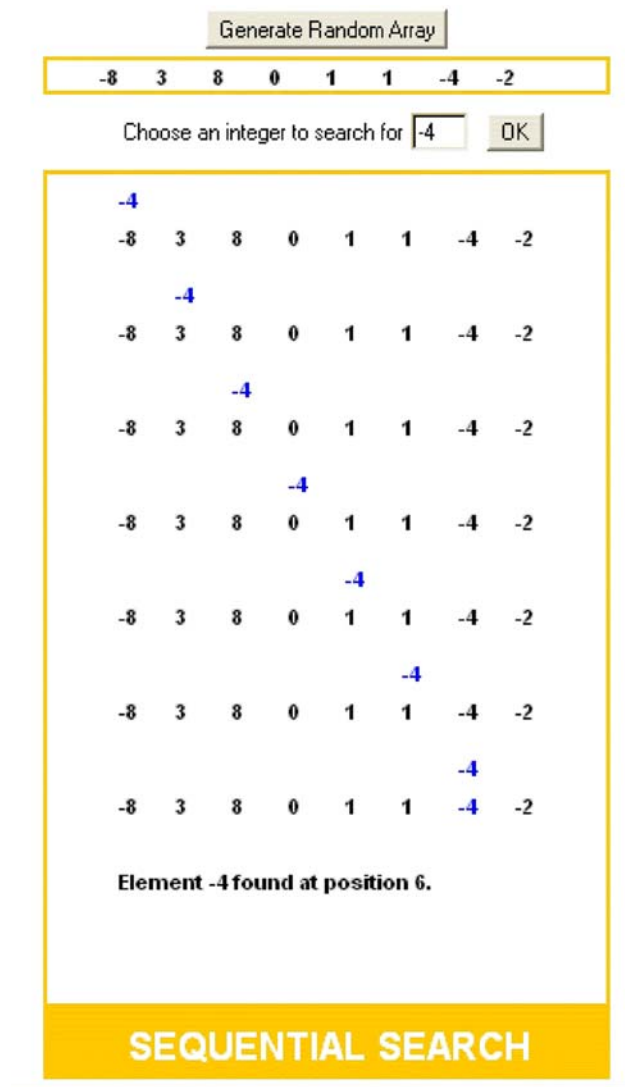


Figure 1. The applet at ref [1] demonstrates the sequential search algorithm. Select an integer to search and the position at which it will be found is identified. Here the positions are 0 through 7 for an array of eight integers and element -4 is found at position 6.

7.3 BINARY

This is a divide and conquer algorithm for an ordered array. This algorithm takes the array and goes to the middle position, if the object at the middle position is the item required it returns it, otherwise it checks to see if the middle object is greater or less than the desired object. If it is greater it limits its search to the upper half of the array, if it is smaller then it limits its search to the lower half of the array. The same process is carried out in whichever half of the array that is entered to find the desired object. This is repeated until the object is found, or the size of the array to be searched is zero, at which point it means the object is not present in the array

- 1) size(array) = 0, return not found;
- 2) Go to middle position in array
- 3) Compare object at the middle with the desired object
 - i) If middle object equal to desired object, return
 - ii) If middle object less than desired object, repeat steps 1-3 for upper half of array
 - iii) If middle object greater than desired object, repeat steps 1-3 for lower half of array

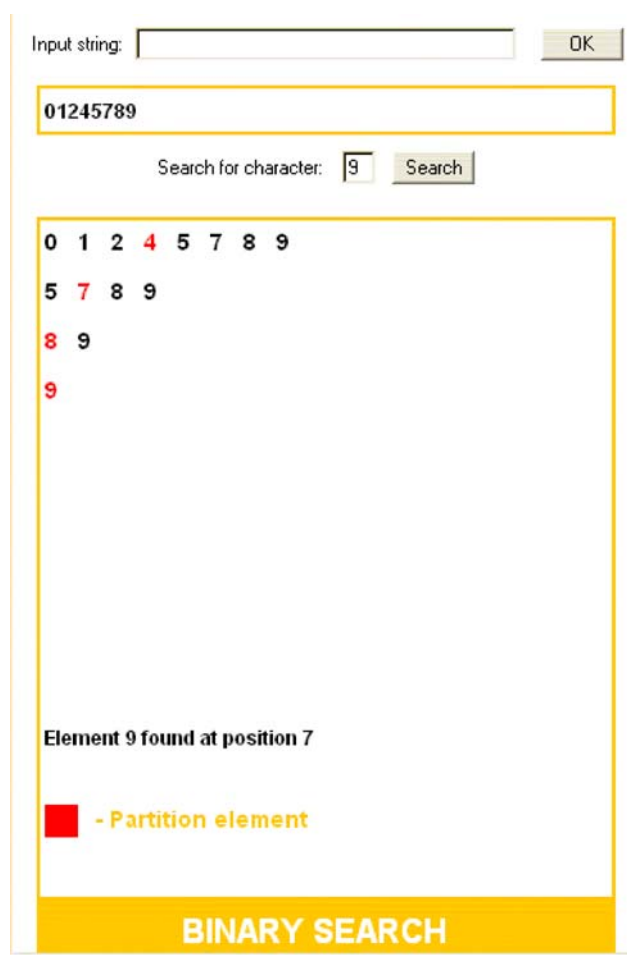


Figure 2.

The applet at ref [2] demonstrates the binary search algorithm. Select an integer to search and the position at which it will be found is identified. Here the positions are 0 through 7 for an array of eight integers and element 9 is found at position 7.

7.4 RED-BLACK TREES

Red-Black Tree Description

A red-black tree is a balanced binary tree which has colour as an extra feature for each node. It was invented by a computer scientist called Rudolf Bayer in 1972. The purpose of this algorithm is ensuring that the longest path from the root to a leaf is never longer than twice the shortest path. As a result, every red-black tree must have the following properties:

1. Every node must have a value.
2. The value of any node must be greater than the value of its left child and less than the value of its right child.
3. Every node must be coloured either red or black.
4. Every red node that is not a leaf must have only black children.
5. Every path from the root to a leaf contains the same number of black nodes.
6. The root node must be black.
7. Red nodes can only have black children.

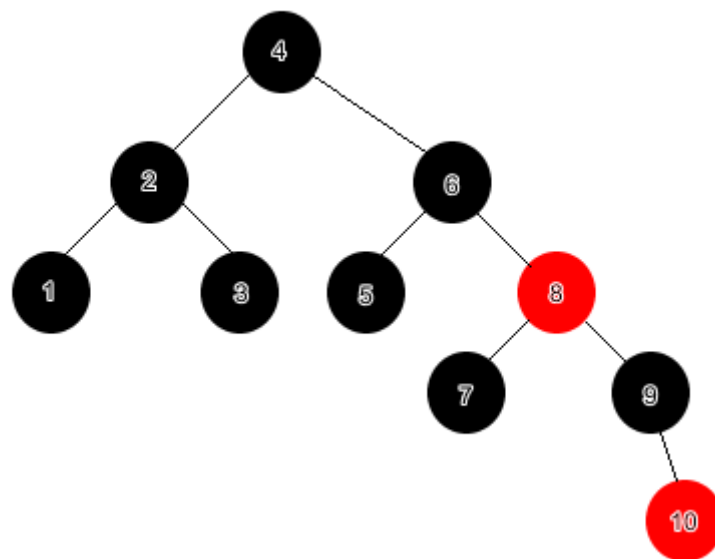


Figure 3. A red-black tree.

Check out the challenge question posted at:-

<http://www.algana.co.uk/Algorithms/Trees/RedBlack/challenge.html>

where a red-black tree algorithm is demonstrated using an interactive applet.

7.5 COMPARISON OF BINARY AND SEQUENTIAL

Binary search takes $O(\log N)$ time to find an item while the sequential search is an order of $O(n)$.

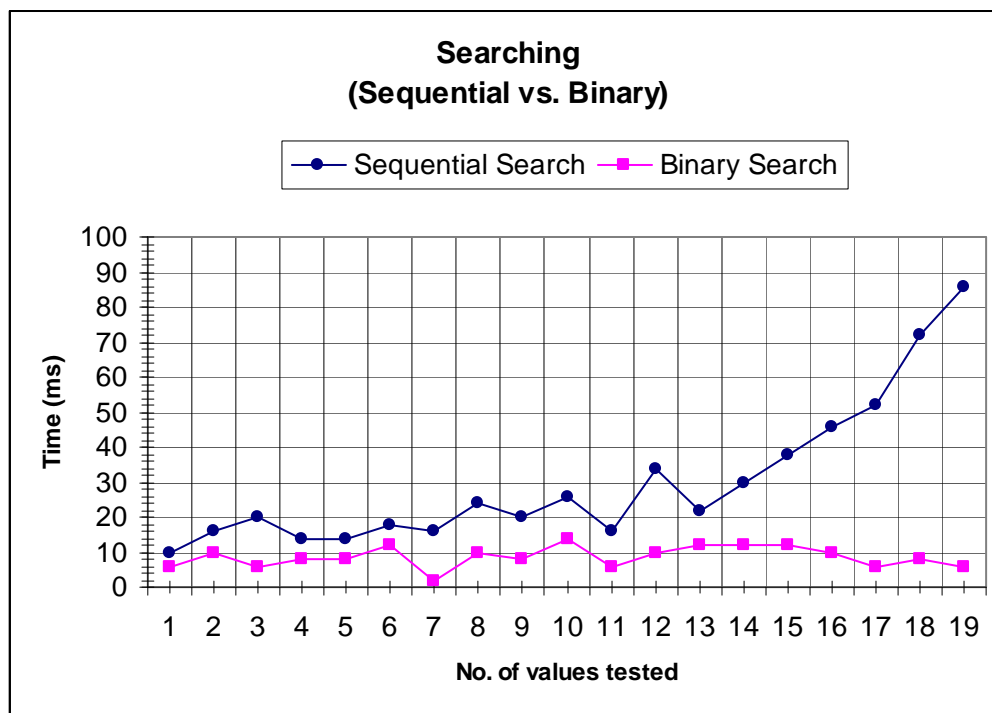
Experimental Analysis

The algorithms were compared for increasing sizes of arrays from 100000 to 10000000. The algorithms were then compared by randomly choosing a number to search for. The clear winner was the binary search that took approximately constant time for any search, while the performance of the sequential search was definitely linked to how far it had to go into the array. The sequential search is of order n $\{O(n)\}$, while the binary search has an order of 0 $\{O(k)\}$, where k is a constant

The binary search algorithm makes searching easier by dividing the array into halves, meaning it always reduces the search space, this proves to be extremely efficient as the results speak for themselves. The main difference is that the position of the target in the array, so depending on the position of the target and where the search commences hugely affect the performance of the sequential search, the sequential search will be affected variably. The binary search on the other hand is always reducing the search space it has to look through to find the target, while the sequential search reduces its search space by a single unit, the binary search reduces it by $n/2$, where n is the size of the current section.

Array Size	Sequential Search (ms)	Binary Search (ms)
100000	10	6
200000	16	10
300000	20	6
500000	14	8
700000	14	8
900000	18	12
1000000	16	2
1200000	24	10
1400000	20	8
1600000	26	14
2000000	16	6
3000000	34	10
4000000	22	12
5000000	30	12
6000000	38	12
7000000	46	10
8000000	52	6
9000000	72	8
10000000	86	6

Table 4 Table for time measures.



Conclusions

From experimental data, it is clear that Binary search is more efficient as compared to sequential search and a primary reason is that the array is pre-ordered. This efficiency is more apparent when the size of array is very large. Theoretical results also confirm our experimental results.

7.6 EXERCISES

1. External Searching

- (a) Explain the terms page and probe used in searching algorithms appropriate for accessing items from huge files of data.
- (b) With the aid of an example, explain the fact that "a B tree of order M constructed from N random items is expected to have about $1.44N/M$ pages".
- (c) With the aid of an example, explain the fact that "a search or an insertion in a B tree of order M with N items requires between $\log_M N$ and $\log_{M/2} N$ probes, a constant number for practical purposes".
- (d) Explain a straightforward approach for building an index in index sequential access

2. Array Searching

Explain clearly, with the aid of a trace of the steps similar to figures 1 and 2, how you would search the following array of positive integers to find the target 4096:

1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
---	---	---	---	----	----	----	-----	-----	-----	------	------	------	------	-------	-------

using (a) sequential search and (b) binary search.

3. (a) The recursive Java pseudo-code to search for the definition of a word in a dictionary is shown in figure Q3 and the following call is made to the dictionary:

```

1      public static void main(String args[])
2      {
3          String definition = dict.search("Computer");
4      }
```

Use stack diagrams to illustrate the operation of this program. Assume that the word "Computer" is found after the third call to the search method. Include one recursive call to the left half of the dictionary and one to the right half.

- (b) The Dictionary class explicitly maintains a tree structure for the left and right halves of the dictionary. Rewrite the pseudo-code so that, instead, it uses a binary search tree interface and an associated binary search tree implementation class. Note that the items stored in the tree must be comparable. Also, the binary search tree interface defines a method called *find*. You need not provide the code for the interface and its implementation class or any other class you need to use.

```
1public class Dictionary
2{
3    private Dictionary leftHalf;
4    private Dictionary rightHalf;
5
6    public String search(String word)
7{
8    if (word found)
9    {
10       return foundWord.definition;
11    }
12    else if (this.leftHalf not empty)
13    {
14       return this.leftHalf.search(word);
15    }
16    else if (this.rightHalf not empty)
17    {
18       return this.rightHalf.search(word);
19    }
20    else
21    {
22       throw new NotFoundException();
23    }
24}
```

Figure Q3. Recursive Java pseudo-code to search for the definition of a word in a dictionary.

4. Investigate the ways in which binary trees work by experimenting at the “how binary tree works” link at <http://www.algana.co.uk/Algorithms/Searching/searchingFrameset.htm>. Summarize some of the most important applications of binary trees in computer science.

7.7 BIBLIOGRAPHY & REFERENCES

1. <http://www.algana.co.uk/Algorithms/Searching/sequential/sequential.htm>
2. <http://www.algana.co.uk/Algorithms/Searching/binary/binary.htm>
3. <http://www.faqs.org/docs/javap/c8/s4.html>
4. <http://www.smartarrays.com/new/downloads/manual/arraymethods.html>

8. SORTING

8.1 INTRODUCTION

One objective of a sorting algorithm is to rearrange items (or their associated labels) according to some well-defined ordering rule that is usually either a numerical or alphabetical. The abstract notion of putting keys and associated information into order characterizes the sorting problem.

Popular sorting algorithms described in the following pages include:

- bubblesort,
- heapsort,
- insertionsort,
- mergesort,
- quicksort,
- selectionsort

Other sorting algorithms of interest include:

- bucketsort,
- postmansort
- radixsort,
- shearsort,
- shellsort

In this section, an experimental comparison of bubblesort and quicksort is described in section 8.6.

8.2 QUICKSORT

Quicksort is a form of a divide and conquer algorithm. What quick sort does is collect the array of objects that need to be sorted, selects a point which it calls a pivot, and puts all the objects smaller than it to one side, and those greater than it to the other side. Next it splits the array into two sections, one half of objects greater than pivot, and other of objects less than pivot. To each of these halves the quick-sort algorithm is applied again. This recursion carries on until an array passed to the quick-sort algorithm has no items (since such an array is already sorted). The quicksort algorithm is an excellent way to get integers ordered quickly, but there are some cases where other options may be better. This algorithm is best used with large numbers of elements.

The steps are:

1. Select a point as a pivot
2. Put all object larger than pivot to one side, all those smaller to the other side of the pivot.
3. Recursively apply steps 1-3 to each sub-array to other side of pivot until the size of array to process = 0;

Example 7.2

The following data are stored in an array and need to be sorted in ascending order from left to right:

14 23 12 24 33 18 16 20

Explain concisely the steps involved in using the quicksort algorithm, starting with the last item on each pass. Solution:

14	23	12	24	33	18	16	20
14	16	12	18	20	24	23	33
14	16	12	18	20	23	24	33
12	16	14	18	20	23	24	33
12	14	16	18	20	23	24	33
No more swaps							

A demonstration quicksort applet is available at ref [\[1\]](#)

8.3 BUBBLESORT

Using the Brute Force approach, Bubble-sort uses a completely different technique to sort the arrays. Bubble sort checks all the elements of an array that needs to be sorted with each other. Then it moves the lowest value to the top of the array like a bubble. Thus it sorts the whole array of numbers. The algorithm repeats this process until it makes a pass all the way through the list without swapping any items, which is an indication that all items are in the correct order. This causes larger values to "bubble" to the end of the list while smaller values "sink" towards the beginning of the list. The method used to start at the beginning of the array and compare consecutive members, if position $p > p + 1$ the two objects are swapped. This process is continued, working the way through out the array until the end. At this point the largest object will have been placed in its position. This process is repeated for $n - 1$ times (n is the length of the array), with each subsequent pass through the array there is no need to compare the whole array, instead you only perform the comparison until the point where you ended in the previous pass.

- For $k = 0$ to $n-1$
 - For $j = 0$ to $j - k$
 - If $\text{item}_{(j)} > \text{item}_{(j+1)}$, swap items

Example 8.3

The following data are stored in an array and need to be sorted in ascending order from left to right:

14 23 12 24 33 18 16 20

Explain concisely the steps involved in using the bubblesort algorithm, starting with the last item on each pass.

14	23	12	24	33	18	16	20
12	14	23	16	24	33	18	20
12	14	16	23	18	24	33	20
12	14	16	18	23	20	24	33
12	14	16	18	20	23	24	33
No more swaps							

8.4 MERGESORT

In the mergesort algorithm in order to sort a given file, it is divided in half, then the two halves are sorted recursively and merged. This algorithm is one of the best-known examples for the divide and conquer algorithm. It is analogous to a top-down management style, where a manager gets an organization to take on a big task by dividing it into pieces to be solved independently by underlings. If each manager operated by simply dividing the given task in half, then putting together the solutions that the subordinates develop and passing the result up to a superior, the result is a process like mergesort. Not much real work gets done until someone with no subordinates gets a task; but managements does much of the work putting together solutions. Mergesort is important because it is a straightforward optimal sorting method. It runs in time proportional to $N \log N$ to sort any file of N elements.

Example 8.4

Show that Mergesort is $O(n \lg n)$ by finding and solving the recurrence formula for $T(n)$, the running time for mergesorting n data,

$$T(n) = 2 T(n/2) + n, \text{ given that } T(1) = 1,$$

where n is a power of 2. Solution:

$$\begin{aligned} T(1) &= 1 \\ T(2) &= 4 \\ T(4) &= 12 \\ T(8) &= 32 \\ T(16) &= 80 \\ T(n) &= N((\lg N) + 1) \\ T(n) &= O(N \lg N) \end{aligned}$$

8.5 INSERTIONSORT

The method that people often use to sort bridge hands is to consider the elements one at a time, inserting each into its proper place among those already considered. In a computer implementation in order to make space for the element being inserted by moving larger elements one position to the right, and then inserting the element into vacated position. In insertion sort, the elements to the left of the current index are in sorted order during the sort, but they are not in their final position, as they may have to be moved to make room for smaller elements encountered later. The array is, however, fully sorted when the index reaches the right end.

The implementation of insertion sort is straightforward, but inefficient. The running time of insertion sort primarily depends on the initial order of the keys in the input. If the file is large and the keys are in order or even are nearly in order, then insertion sort is quick.

8.6 COMPARISON OF BUBBLESORT AND QUICKSORT

Experimental Analysis

For these algorithms the comparison was based on giving the algorithms arrays with randomly generated numbers in a random order. The sizes of the arrays were incremented to measure performance against size. The results put the quick-sort algorithm as the clear winner; it progresses linearly $\{O(n)\}$, whilst the bubble sort algorithm definitely has a quadratic progression $\{O(n^2)\}$.

To understand the difference in behaviour we must analyse the algorithm. For the bubble-sort the algorithm traverses the array n times, and in each path it has to perform $n - p$ comparison (p = number of pass). On the other hand the quick-sort takes a different approach of dividing the array into two sections, and in each it traverses that section of the array once, performing k comparisons, where k is the size of the section. This means the largest pass will be the first pass through the array which will contain n comparisons; all the other passes through the subsections of the array will be less than n , making n the most significant figure in the time taken to sort the array. The divide and conquer method is therefore a very efficient method to perform the sorting of an array.

Data for analysis is collected under the following conditions

1. For each run, random numbers are generated in the range of 'four times the size of an array'. E.g. size = 100 then the random numbers are from 0 to $100 * 4$. It is done to have distinguished values in an array as much possible.
2. Same array of integers is provided to both algorithms to sort.
3. For each size of the array, five values for time taken are found and then the average value is used to formulate a runtime value for graph. This procedure is used because each runtime generates different set of random numbers and each run for the same size of an array could take different time.

Length of Array	BubbleSort (ms)	QuickSort(ms)
100	0	0
1000	12	2
2000	34	2
3000	65	6
4000	106	8
5000	164	2
6000	226	8
7000	325	8
8000	413	8
9000	509	10
10000	629	10
20000	2549	10
30000	6194	11
40000	10291	18
50000	16147	20
60000	23324	22
70000	32076	25
80000	41650	31
90000	52936	33
100000	65455	40

Table 1 – Table for time measures

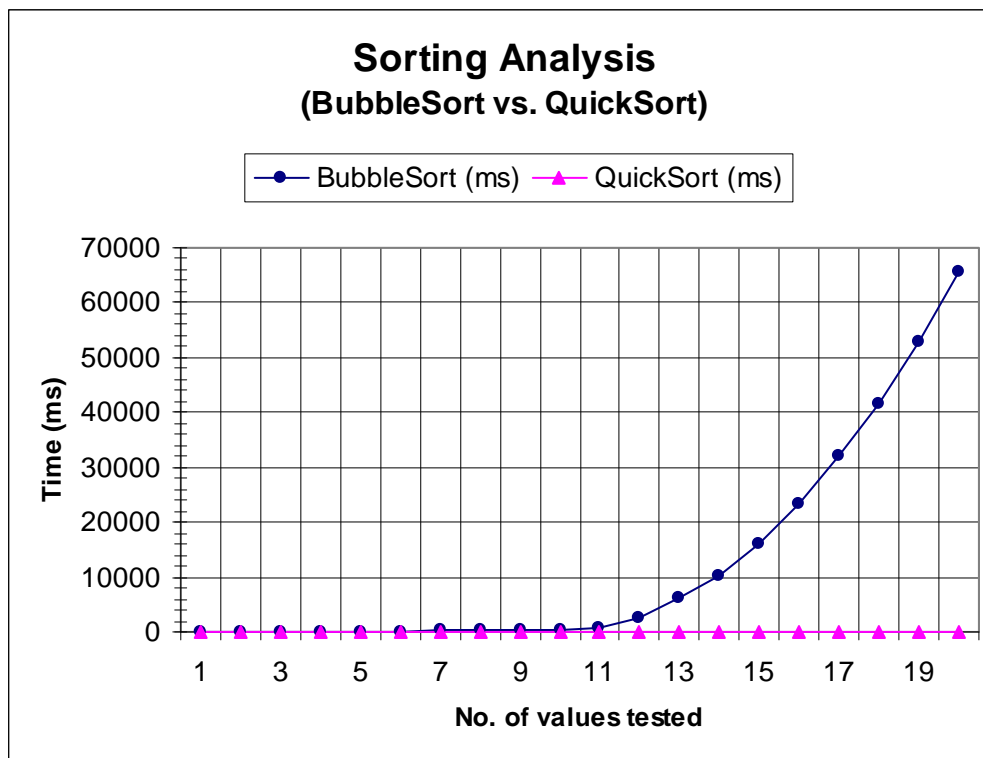


Table 2 – Experimental graph for bubblesort vs. quicksort.

8.6 COMPARISON OF OTHER SORTING ALGORITHMS

Experimental Analysis

Choose the “sort comparison” link at:-

<http://www.algana.co.uk/Algorithms/Sorting/SortingFrameset.htm>

Observe the relative progress of insertionsort, heapsort, quicksort and mergesort over the same data set. Figure 3 shows typical output.

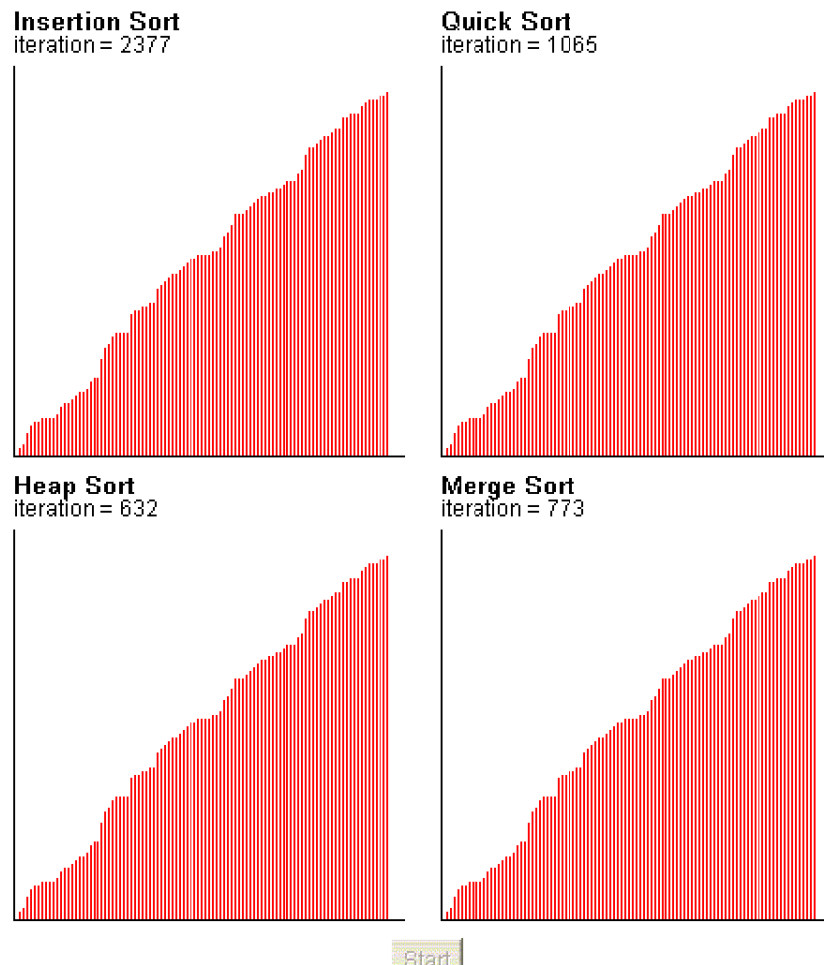


Figure 3 – A comparison of insertionsort, heapsort, quicksort and mergesort.

8.7 EXERCISES

1. Show how bubble sort sorts the following sample file alphabetically:

F A C E T I O U S

2. The following are famous names associated with data compression. Prepare a concise web biography of each in a format similar to that shown for Albert Einstein at <http://www.algana.co.uk/FamousNames/E/einstein.htm>

- (a) Anthony Hoare (quicksort),
- (b) Donald Shell (shellsort),

3. Give an example of a file for which the number of exchanges done by bubble sort is maximized.
4. The following data are stored in an array and need to be sorted in ascending order from left to right:

14 23 12 24 33 18 16 20

- (a) Explain concisely the steps involved in using the bubblesort algorithm, starting with the last item on each pass.
 - (b) Explain concisely the steps involved in using the quicksort algorithm, where the pivot is chosen to be the last item.
 - (c) Show the first four passes for insertionsort and mergesort and for each of the pass write down the number of comparisons required.
5. Another variant of the bubble sorting algorithm is the so-called *cocktail shaker sort*. Like bubble sort, this algorithm executes a total of $n-1$ passes through the array. However, alternating passes go in opposite directions. For example, during the first pass the largest item bubbles to the end of the array and during the second pass the smallest item bubbles to the beginning of the array.
 - (a) Show that the array is guaranteed sorted after $n-1$ passes.
 - (b) What is the running time of this algorithm in terms of n ?

8.8 BIBLIOGRAPHY & REFERENCES

1. <http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/qsort.html>
2. <http://atschool.eduweb.co.uk/mbaker/sorts.html>
3. <http://www.cs.rut.edu/~atk/Java/Sorting/sorting.html>
4. <http://www.developerfusion.com/show/3824/>
5. <http://www.cs.umass.edu/~immerman/cs311/applets.html>

9. STRINGS

9.1 INTRODUCTION

String algorithms are often used to search a piece of text for a sub-string or a binary string for a pattern consisting of 0 and 1 symbols. Searching of this type is a common activity in computer science and finding new and more efficient methods of performing it can greatly influence the performance of a computer program or software system.

9.2 BOYER-MOORE

In this algorithm the pattern is moved far to the right as possible. But it searches the pattern from right to left, not left to right, matching the character at the end of the substring. In deciding how far to move the pattern, an observation is used to find out where in the pattern the current text character occurs. The pattern is then moved to the right by the appropriate distance.

The algorithm structure

```
while ( !matched && !exhausted)
{
    while( pattern char != text char)
    {
        shift pattern as far right as possible;
        // Amount to shift pattern to the right is
        // obtained from two tables which are
        // calculated by pre-processing the pattern
        start search at right hand end of the pattern;
    }
    decrement indices of pattern and text by one;}
```

Example 9.1

R	a	V	b	K	x	a	B	c	d	e	f	g
A	b	C	d									

Since d is not equal to b, but b occurs at the second spot in the substring, so the text has to be shifted 2 (4 -2) spaces along 2

R	a	V	b	K	x	a	B	c	d	e	f	g
		a	b	C	d							

X is not found anywhere in the substring so the we shift along 4 (length of substring) spaces along

R	A	v	b	K	x	a	B	c	d	e	f	g
				a	B	c	d					

We start matching from d; we find that the substring is found.

Example 9.2

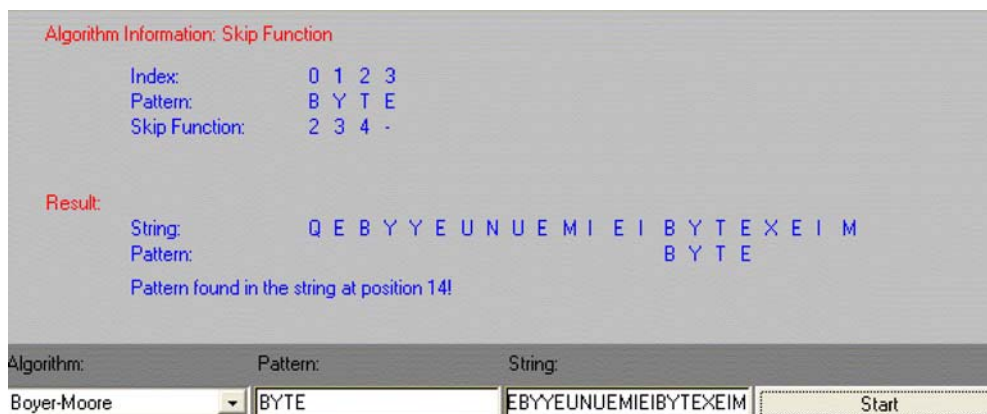
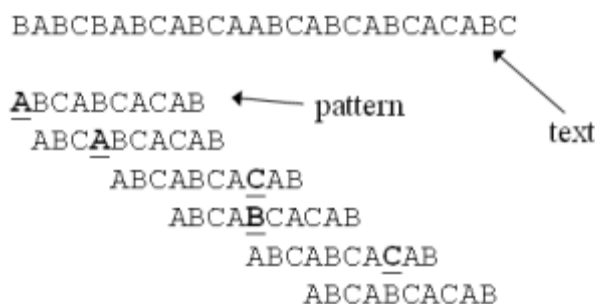


Figure 1. The applet at ref [1] demonstrates the Boyer-Moore algorithm. Choose Boyer-Moore from the drop-down menu.

9.3 KNUTH-MORRIS-PRATT

Instead of moving only one character at a time as in Naïve approach, The KMP algorithm uses information about the characters in the string we're looking for to determine how much to 'move along' that string after a mismatch occurs. The objective of the KMP algorithm is to find a matching string in a given string using fewer comparisons than the brute-force method. One advantage of KMP over the other two algorithms is that it always moves forward in the text string, which is particularly useful if we are reading text from a file or dealing with linked lists of chunks. For example:



The algorithm structure

```
while ( !matched && !exhausted )
{
    while ( pattern char != text char)
    {
        shift pattern as far right as possible
        // Amount to shift pattern to the right is obtained from a
        // table which is calculated by pre-processing the pattern
        if pattern has been moved past the current position of the text
            start search one position to the right;
        else
            start search at current position of the text;
    }
    increment indices of pattern and text by one;
}
```

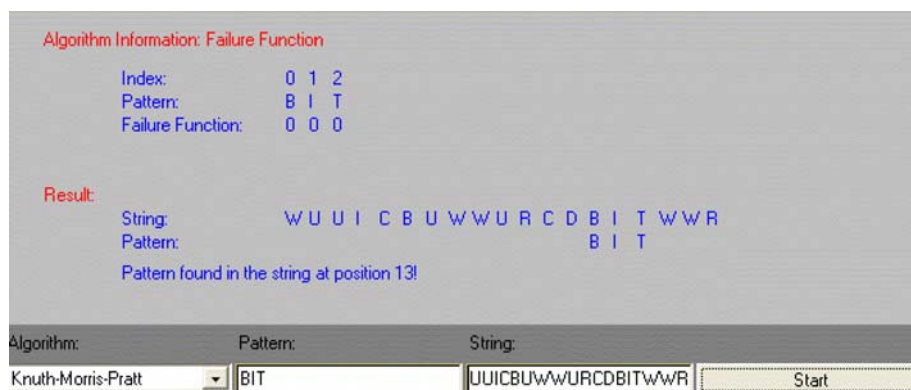
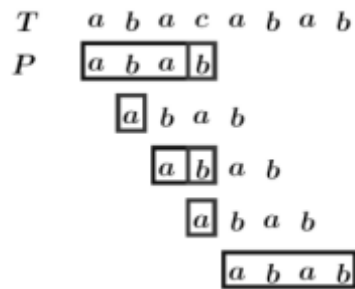


Figure 2. The applet at ref [1] demonstrates the Knuth-Morris-Pratt algorithm. Choose Knuth-Morris-Pratt from the drop-down menu.

9.4 PATTERN MATCHING

This is the most intuitive form of searching that is performed whenever we search. Starting from the beginning of the piece of the text, you traverse it sequentially, moving in steps of one. You start by trying to match the first character of the sub string, with the current position in the text; if they don't match we move the current position by one. If they match we move compare the proceeding character until the whole sub string is matched, if at any time the characters don't match, the current position is moved by one. For example:



The algorithm structure

```

n = length of text, T
m = length of pattern, P
for s = 0 to ( n - m )
    do j = 1
        while j <= m and P[j] = T[ s + j]
            do j = j + 1
        if j > m
            then print "pattern is found".
        else
            print "pattern is not found".

```

let m = length of the text

n = length of the pattern

There are two loops involved in the algorithm. Inner loop will take 'm' steps to confirm the pattern matches. Outer loop will take 'n' steps. Therefore, worst case is $O(m * n)$.

9.5 LEVENSCHTEIN'S ALGORITHM

Levenshtein distance (LD) is a measure of the similarity between two strings. The distance between two strings is calculated by comparing two strings, labelled the source string (s) and the target string (t). The actual distance between two strings is the number of steps (such as deletion, substitution and insertion) that are needed to transform s into t. For example, if s is "test" and t is "test", then $LD(s,t) = 0$, because no transformations are needed. The strings are already identical. If s is "test" and t is "tent", then $LD(s,t) = 1$, because one substitution (changing "s" to "n") is sufficient to transform s into t.

In order to achieve a result, the Levenshtein distance algorithm involves a number of steps:

1. Set n to be the length of s. Set m to be the length of t. If $n = 0$, return m and exit. If $m = 0$, return n and exit. Construct a matrix containing $0..m$ rows and $0..n$ columns.
2. Initialize the first row to $0..n$. Initialize the first column to $0..m$.
3. Examine each character of s (i from 1 to n).
4. Examine each character of t (j from 1 to m).
5. If $s[i]$ equals $t[j]$, the cost is 0. If $s[i]$ doesn't equal $t[j]$, the cost is 1.
6. Set cell $d[i,j]$ of the matrix equal to the minimum of:
 - a. The cell immediately above plus 1: $d[i-1,j] + 1$.
 - b. The cell immediately to the left plus 1: $d[i,j-1] + 1$.
 - c. The cell diagonally above and to the left plus the cost: $d[i-1,j-1] + \text{cost}$.
7. After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell $d[n,m]$.

Practice 9.5 Calculate the Levenshtein distance between the strings "work" and "world"

9.6 COMPARISON OF BOYER-MOORE AND KNUTH-MORRIS-PRATT

Theoretical Analysis

KMP's execution time is $O(m+n)$, where m is the length of the search string, and n is the length of the string to be searched. It performs at most $(2n-1)$ text character comparisons during the searching phase.

Boyer-Moore's execution time is $O(m+n)$, where m is the length of the search string, and n is the length of the string to be searched.

Experimental Analysis

String Length (n)	Pattern Length (m)	Brute Force(ms)	KMP(ms)	BM(ms)
210	10	21	2.20	1.73
1000	10	100	10.10	8.91
2000	10	200	20.10	18.00
3000	10	300	30.10	27.09
4000	10	400	40.10	36.18
5000	10	500	50.10	45.27
6000	10	600	60.10	54.36
7000	10	700	70.10	63.45
8000	10	800	80.10	72.55
9000	10	900	90.10	81.64
10000	10	1000	100.10	90.73
11000	10	1100	110.10	99.82
12000	10	1200	120.10	108.91
13000	10	1300	130.10	118.00
14000	10	1400	140.10	127.09

Table 3. Comparison data for string search algorithms.

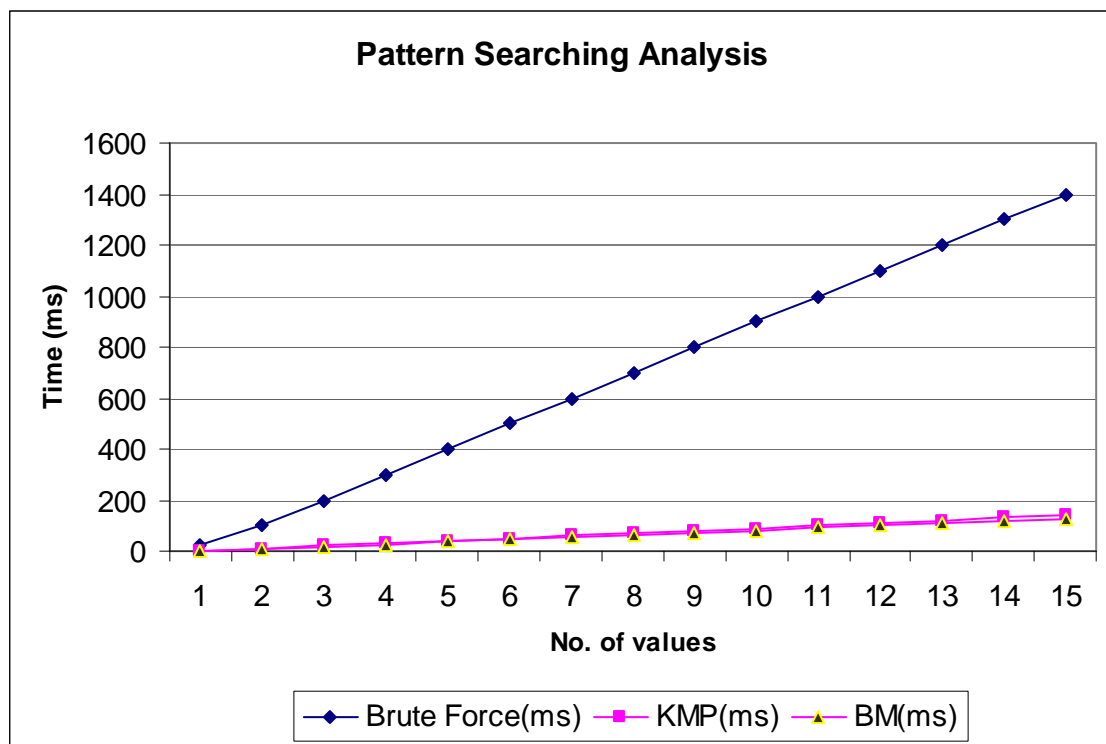


Figure 4. Data taken from figure 3.

In graph, KMP and BM lines are overlapped.

9.6 EXERCISES

1. The following are famous names associated with string searching. Prepare a concise web biography of each in a format similar to that shown for Albert Einstein at <http://www.algana.co.uk/FamousNames/E/einstein.htm>

- (a) Boyer (Boyer-Moore algorithm),
- (b) Moore (Boyer-Moore algorithm),
- (e) Knuth (Knuth-Morris-Pratt algorithm),
- (f) Morris (Knuth-Morris-Pratt algorithm),
- (b) Pratt (Knuth-Morris-Pratt algorithm).

2. Assuming a left-to-right search procedure, explain clearly how the Knuth-Morris-Pratt algorithm can search through the text string:

GCATCGCAGAGAGTACAGTACG

attempting to match the pattern “GCAGAGAG”.

3. Investigate Levenshtein’s algorithm further by comparing and contrasting the implementations in:
 - (a) Java
 - (b) C++
 - (c) Visual Basic

You might wish to visit: <http://www.merriampark.com/ld.htm>.

9.7 BIBLIOGRAPHY & REFERENCES

1. <http://www.algana.co.uk/Algorithms/Strings/stringsdisplay.html>
2. <http://www.cs.duke.edu/~mlittman/courses/Archive/cps13097/lectures/lect14/node10.htm>
3. <http://www.merriampark.com/ld.htm>
4. <http://www.algana.co.uk/Algorithms/Strings/StringsFrameset.html>

10. PAST EXAMINATION QUESTIONS & SOLUTIONS

CSCI 319

FINAL EXAMINATION

FALL 2001

ALGORITHMS

Time Allowed: 2 hours

Answer at least FOUR questions. If you answer more than four, only the best four will be taken into account. All questions carry equal marks. You are advised to read each question completely before writing your answer. You may refer to course notes but you must not communicate with anyone other than the instructor during the examination. Show all working in the booklet so that credit can be given for any planning or method of calculation.

The notation $\lg N$ is used throughout to represent $\log_2(N)$, the logarithm of N to base 2.

1. (a) Write down three mathematical expressions implied by describing three functions f_1 , f_2 and f_3 as “ $O(N)$, $O(N \log N)$ and $O(N^2)$ ”, respectively.
(3 marks)
- (b) The run-times for three different algorithms, labelled ALG1, ALG2 and ALG3 respectively, were recorded in an experiment solving similar-sized problems. Each algorithm was respectively applied to problems of size N (measured in decimal form) and the corresponding run-time values recorded in scientific notation. The results for various values of N are shown in figure Q1.

N (problem size)	RUN-TIME ALG 1 (milliseconds)	RUN-TIME ALG2 (milliseconds)	RUN-TIME ALG3 (milliseconds)
32	6.388083E+03	1.850945E+02	5.839600E-01
64	6.776033E+03	4.367473E+02	1.966360E+00
128	7.551934E+03	1.011954E+03	7.495960E+00
256	9.103736E+03	2.306168E+03	2.961436E+01
512	1.220734E+04	5.182200E+03	1.180880E+02
1024	1.841455E+04	1.150947E+04	4.719824E+02
2048	3.082896E+04	2.531442E+04	1.887560E+03
4096	5.565778E+04	5.522516E+04	7.549870E+03
8192	1.053154E+05	1.196483E+05	3.019911E+04
16384	2.046307E+05	2.576978E+05	1.207961E+05

Figure Q1. Run-times are shown in milliseconds for three algorithms.

Suppose that prior research into the three algorithms collectively had revealed that one is order N , another order $N \log N$ and a third order N squared but the precise identity of each was not known.

- (i) Produce graphical plots to identify ALG1, ALG2 and ALG3 appropriately based on the recorded data.
(16 marks)
- (ii) Which algorithm would you use to solve “small” problems in which the problem size is never likely to exceed 16384?
(3 marks)
- (iii) Which algorithm would you use to solve “large” problems in which the problem size is always likely to exceed 32768?
(3 marks)

2. (a) Explain briefly the essential characteristics of a brute-force algorithm and a greedy algorithm.

(4 marks)

- (b) For the weighted graph in figure Q2, show clearly the steps involved in attempting to find the shortest path from V_1 to V_4 using the following algorithms:

- (i) Exhaustive depth-first search with ordering (V_1, V_2, V_3, V_4) (5 marks)
- (ii) Branch-and-bound expanding the best one node (5 marks)
- (iii) Exhaustive breadth-first search with ordering (V_1, V_2, V_3, V_4) (5 marks)

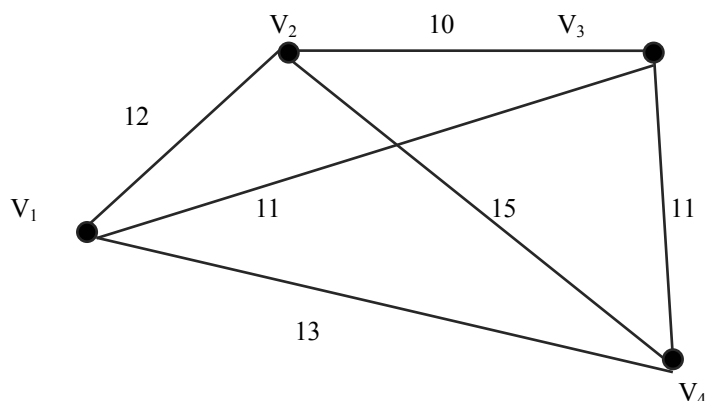


Figure Q2. A weighted graph over vertices V_1, V_2, V_3 and V_4 .

- (c) Classify the three algorithms in part (b) as either brute-force or greedy.

(3 marks)

- (d) Explain clearly whether each of the algorithms in part (b) is guaranteed to lead to a minimal solution for weighted graph problems.

(3 marks)

3. (a) Euclid's algorithm is one of the world's the oldest known algorithms. Show clearly how Euclid's algorithm can be used to find the greatest common divisor of the two decimal numbers 1440 and 575.

(6 marks)

- (b) Assuming a left-to-right search procedure, explain clearly how the Knuth-Morris-Pratt algorithm and the Boyer-Moore algorithm can search through the text string

RICHMONDAMERICANINTERNATIONALUNIVERSITYINLONDON

attempting to match the pattern "ERS". Indicate briefly the relative merits of each algorithm.

(12 marks)

- (c) Explain clearly and concisely how you would build a Huffman tree to compress a large text file containing alphanumeric characters. What would you expect to be the approximate percentage space saving compared to using a standard 8-bit code?

(7 marks)

4. Valid moves in a *Jumping Counters* board game involve moving one space forward or jumping over a counter of a different type. There are two different types of counter, **O** and **X**, respectively, and **O** counters move only to the right while **X** counters move only to the left. The initial position of the board has two vacant spaces between the n **O** counters on the left and the m **X** counters on the right, as shown in the first row of the diagrams in figure Q4. The end game position shows all **O** counters on the right-hand side of the board and all **X** counters on the left. Figure Q4 shows minimal moves for the particular Jumping Counters (n,m) games for which $(n,m) = (1,1)$ and $(n,m) = (1,2)$.

O			X
	O		X
		O	X
	X	O	
	X		O
X			O

O			X	X
	O		X	X
		O	X	X
	X	O		X
	X		O	X
X			O	X
X		X	O	
X	X		O	
X	X			O

Figure Q4. Minimal moves for the Jumping Counters (n,m) board game when $(n,m) = (1,1)$ (figure left) and when $(n,m) = (1,2)$ (figure right). The initial position of the board has two vacant spaces.

- (b) Reason through the various steps in the Jumping Counters (n,m) game for which (n,m) takes values in the list $[(1,1), (1,2), (2,1), (2,2), (2,3), (3,2), (3,3)]$ and create a table of values for the minimal number of moves needed. (12 marks)
- (b) Suggest by conjecture a possible generalised explicit function in terms of n and m for the (n,m) game. Comment briefly on your expectation that this function should be symmetric or otherwise. (6 marks)
- (c) Check whether your solution in part (b) satisfies the $(3,4)$ game. (4 marks)
- (d) Show that the computational complexity of your solution in part (b) is $O(k^2)$ based on the assumption that both n and m are $O(k)$, where k is a very large integer. (3 marks)

5. (a) Given a sequence of pairs of integers representing connections between objects (see Figure Q5, left column), the task of a connectivity algorithm is to output those pairs that provide new connections (see Figure Q5, centre column). For example, the pair 2-9 is not part of the output because the connection 2-3-4-9 is implied by previous connections. This evidence is shown in Figure Q5, (right column). Give the output that a connectivity algorithm should produce when given the input:-

1-2, 1-6, 2-3, 4-1, 0-4, 0-7, 5-3, 2-0, 3-6, 4-5, 6-4, 0-6, 7-6, 0-5.

(Note: Assume input is read from left to right).

(7 marks)

3-4	3-4	
4-9	4-9	
2-3	2-3	
5-6	5-6	
2-9		2-3-4-9
5-9	5-9	
6-9		6-5-9

Figure Q5. A connectivity example.

- (b) Bresenham's Line Drawing algorithm provides a simple effective means of generating a line in computer graphics applications. With reference to any simple example of your choice, trace through the various steps in the algorithm.

(6 marks)

- (c) Show that the merge-sort algorithm is $O(N \lg N)$ by finding and solving the recurrence formula for $T(N)$, the running time for merge-sorting N data,

$$T(N) = 2 T(N/2) + N, \text{ given that } T(1) = 1,$$

where N is a power of 2.

(6 marks)

- (d) A sequence of Fibonacci numbers can be readily generated using recursion. However, in its simplest form, the cost of implementing the recursive solution is exponential. Describe a modified approach using dynamic programming that provides a linear solution. (Note: You are NOT required to write specific programming code).

(6 marks)

End of examination paper

SKETCH SOLUTIONS – FALL 2001 FINAL EXAM

1. (a)

- (i) $f_1 \leq a_1 * N + b_1$, where a_1 and b_1 are constants,
 (ii) $f_2 \leq a_2 * N * \lg N + b_2$, where a_2 and b_2 are constants,
 (iii) $f_3 \leq a_3 * N * N + b_3$, where a_3 and b_3 are constants.

(3 marks)

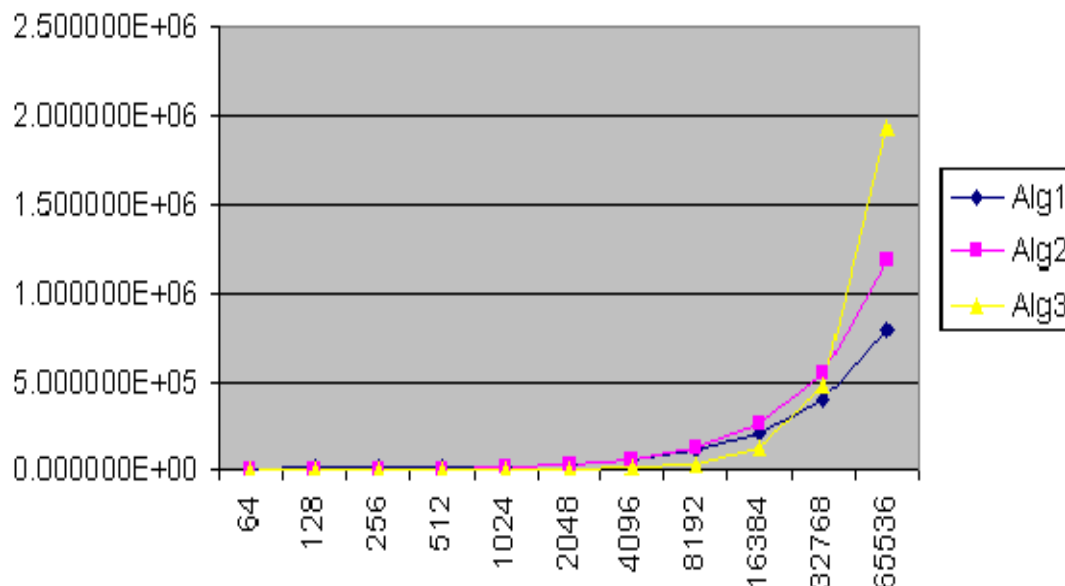
(b)

Choosing any two values in each case gives the constants a_i and b_j to produce each formula.

	N	RUN-TIME ALG 1 (milliseconds)	RUN-TIME ALG2 (milliseconds)	RUN-TIME ALG3 (milliseconds)
3	8	6.097120E+03	3.230531E+01	1.519600E-01
4	16	6.194108E+03	7.724331E+01	2.383600E-01
5	32	6.388083E+03	1.850945E+02	5.839600E-01
6	64	6.776033E+03	4.367473E+02	1.966360E+00
7	128	7.551934E+03	1.011954E+03	7.495960E+00
8	256	9.103736E+03	2.306168E+03	2.961436E+01
9	512	1.220734E+04	5.182200E+03	1.180880E+02
10	1024	1.841455E+04	1.150947E+04	4.719824E+02
11	2048	3.082896E+04	2.531442E+04	1.887560E+03
12	4096	5.565778E+04	5.522516E+04	7.549870E+03
13	8192	1.053154E+05	1.196483E+05	3.019911E+04
14	16384	2.046307E+05	2.576978E+05	1.207961E+05
15	32768	4.032613E+05	5.522035E+05	4.831839E+05
16	65536	8.005226E+05	1.178028E+06	1.932735E+06

(i)

Alg1 Formula = SUM(PRODUCT(12.12345,N),6000.13231) = $O(N)$
 Alg2 Formula = SUM(PRODUCT(1.12345,N,LOG(N,2)),5.34251) = $O(N \lg N)$
 Alg3 Formula = SUM(PRODUCT(0.00045,N,N),0.12316) = $O(N^2)$



(16 marks)

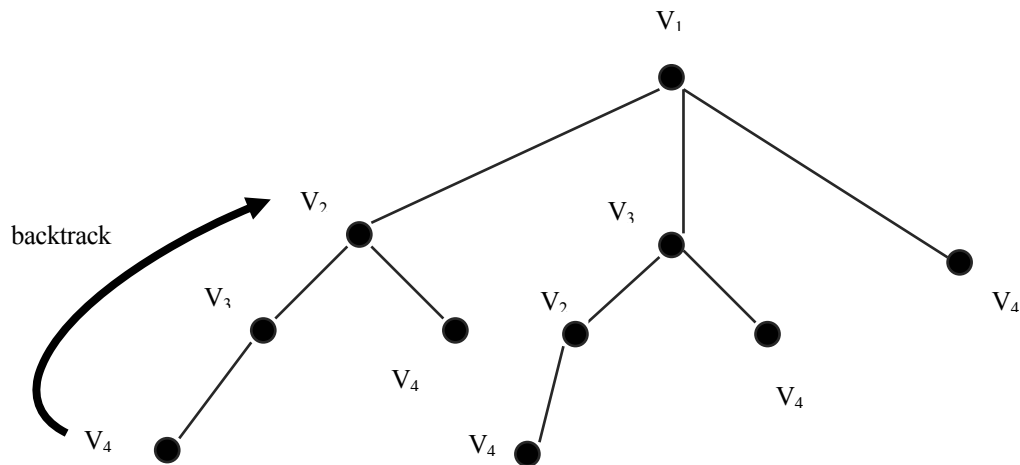
(iii) Alg3 has smaller run-times for values of N not exceeding 18384

(3 marks)

(iii) Alg1 has smaller run-times for values of N exceeding 32768

(3 marks)

2. (a) brute-force algorithm: calculate all possibilities then choose a 'cheap' solution
 greedy algorithm: calculate only 'cheap' solutions as the algorithm proceeds (not all possibilities found).
 (4 marks)
- (b) Exhaustive depth-first search with ordering (V_1, V_2, V_3, V_4)



Five solutions in order: (V_1, V_2, V_3, V_4) , (V_1, V_2, V_4) , (V_1, V_3, V_2, V_4) , (V_1, V_3, V_4) , (V_1, V_4) ; cheapest of the five (V_1, V_4) .

Branch-and-bound:

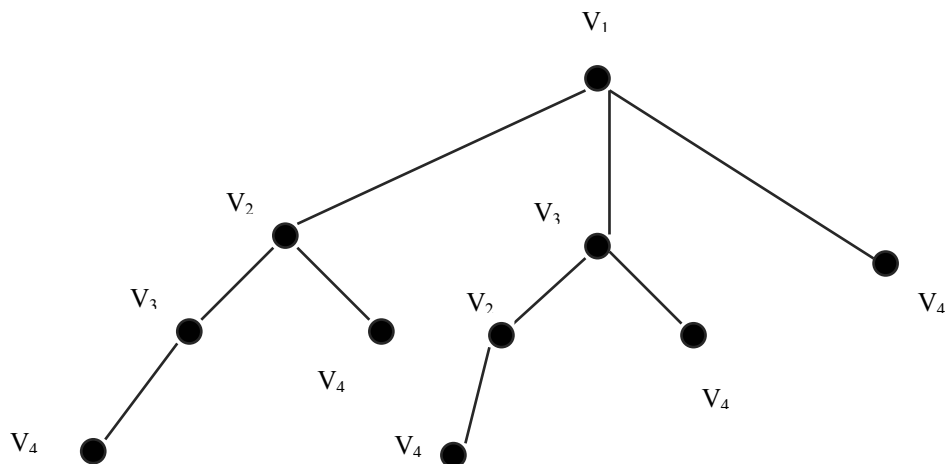
expand V_1 to V_3 (cheapest weight of 11), note other weights;

expand V_3 to V_2 (total weight of 21), disregard since V_1 to V_2 option cheaper, note other weights;

expand V_2 to V_4 (total weight of 27), disregard since V_1 to V_4 option cheaper, note other weights;

cheapest solution (V_1, V_4) .

Exhaustive breadth-first search with ordering (V_1, V_2, V_3, V_4)



Five solutions in order: (V_1, V_4) , (V_1, V_2, V_4) , (V_1, V_3, V_4) , (V_1, V_3, V_2, V_4) , (V_1, V_2, V_3, V_4) ; cheapest of the five (V_1, V_4)
 (5 marks each)

(c) dfs, bfs brute-force; b&b greedy (3 marks)

(d) dfs, bfs guaranteed minimal, b&b not guaranteed minimal. (3 marks)

$$\begin{aligned}
 3. (a) \quad \gcd(1440, 575) &= \gcd(575, 290) \\
 &= \gcd(290, 285) \\
 &= \gcd(285, 5) \\
 &= \gcd(5, 0)
 \end{aligned}$$

$$\gcd = 15$$

(6 marks)

- (b) Assuming a left-to-right search procedure, explain clearly how the Knuth-Morris-Pratt algorithm and the Boyer-Moore algorithm can search through the text string

RICHMONDAMERICANINTERNATIONALUNIVERSITYINLONDON

ERS

ERS

ERS

ERS

ERS

ERS

ERS

Indicate briefly the relative merits of each algorithm.

(12 marks)

- (c) Explain clearly and concisely how you would build a Huffman tree to compress a large text file containing alphanumeric characters.

Read each text character

Perform a frequency count (e.g. “e” – 13%, “t” – 11%, etc.)

Allocate small length code to most frequently occurring characters

Allocate large length code to least frequently occurring characters

Ensure no two characters are allocated the same code

Approximate percentage space saving compared to using say ASCII, 50%.

(7 marks)

5. Various

- (a) Given a sequence of pairs of integers representing connections between objects (see Figure Q4, left), the task of a **connectivity algorithm** is to output those pairs that provide new connections (see Figure Q4, centre). For example, the pair 2-9 is not part of the output because the connection 2-3-4-9 is implied by previous connections. This evidence is shown in Figure Q5, (right). Give the output that a connectivity algorithm should produce when given the input:-

1-2, 1-6, 2-3, 4-1, 0-4, 0-7, 5-3, 2-0, 3-6, 4-5, 6-4, 0-6, 7-6, 0-5.

1-2	1-2	
1-6	1-6	
2-3	2-3	
4-1	4-1	
0-4	0-4	
0-7	0-7	
5-3	5-3	
2-0	2-0	
3-6		3-2-0-4-1-6
4-5		4-1-6-3-5
6-4		6-3-5-4
0-6		0-2-3-6
7-6		7-0-2-3-6
0-5		0-4-5

(7 marks)

- (b) Bresenham's Line Drawing algorithm: Use students chosen line to develop increments.

(6 marks)

- (d)

$$T(N) = 2 T(N/2) + N, \text{ given that } T(1) = 1,$$

where N is a power of 2.

$$\begin{aligned} T(1) &= 1 \\ T(2) &= 4 \\ T(4) &= 12 \\ T(8) &= 32 \\ T(16) &= 80 \\ T(n) &= N((\lg N) + 1) \\ T(n) &= O(N \lg N) \end{aligned}$$

(6 marks)

- (d) Fibonacci dynamic programming: maintain values in an array up to, say, F(N) when N=k, then when computing terms higher terms N>k, F(k) does not have to be recalculated. Algorithm O(N).

(6 marks)

CSCI 319

FINAL EXAMINATION

FALL 2002

ALGORITHMS

Time Allowed: 2 hours

Answer at least FOUR questions. If you answer more than four, only the best four will be taken into account. All questions carry equal marks. You are advised to read each question completely before writing your answer. Show all working in the booklet so that credit can be given for any planning or method of calculation.

The notation $\lg N$ is used throughout to represent $\log_2(N)$, the logarithm of N to base 2.

1. Various

- (a) Show clearly how Euclid's algorithm can be used to find the greatest common divisor of the two integers 1440 and 575.

(6 marks)

- (b) Assuming a left-to-right search procedure, explain clearly how the Knuth-Morris-Pratt algorithm can search through the text string

GCATCGCAGAGAGTACAGTACG

attempting to match the pattern "GCAGAGAG".

(6 marks)

- (c) Explain clearly and concisely how you would build a Huffman tree to compress a large text file containing alphanumeric characters. What would you expect to be the approximate percentage space saving compared to using a standard 8-bit code?

(7 marks)

- (d) For the pairs of time complexities of two algorithms A and B shown in table Q1, find the least positive integer N for which algorithm B is better than algorithm A.

(6 marks)

A	B
$N^3 + 16$	$N^2 + 64$
$2N - 1$	$\lg(64N)$

Table Q1.

2. Sorting

(a) The following data are stored in an array and need to be sorted in ascending order from left to right:

14 23 12 24 33 18 16 20

- (i) Explain concisely the steps involved in using the bubblesort algorithm, starting with the last item on each pass. (7 marks)
- (ii) Explain concisely the steps involved in using the quicksort algorithm, where the pivot is chosen to be the last item. (7 marks)
- (iii) Explain the statement “bubblesort is typically $O(n^2)$ whereas quicksort is typically $O(n \lg n)$.” (4 marks)

(b) Show that Mergesort is $O(n \lg n)$ by finding and solving the recurrence formula for $T(n)$, the running time for mergesorting n data,

$$T(n) = 2 T(n/2) + n, \text{ given that } T(1) = 1,$$

where n is a power of 2.

(7 marks)

3. Graph Searching

Explain the terms brute-force and greedy as used to describe certain algorithms.

(4 marks)

For the weighted graph in figure Q3, show clearly the steps involved in finding the shortest path from B to E using the following algorithms:

- (a) Exhaustive depth-first search with ordering (B,R,U,T,E) (6 marks)
- (b) Exhaustive breadth-first search with ordering (B,R,U,T,E) (6 marks)
- (c) Best-first search (expand best node according to weight) (4 marks)

Classify each of the algorithms in parts (a), (b) and (c) according to either brute-force or greedy and explain which of these (if any) are guaranteed to lead to a minimal solution for weighted graphs in general.

(5 marks)

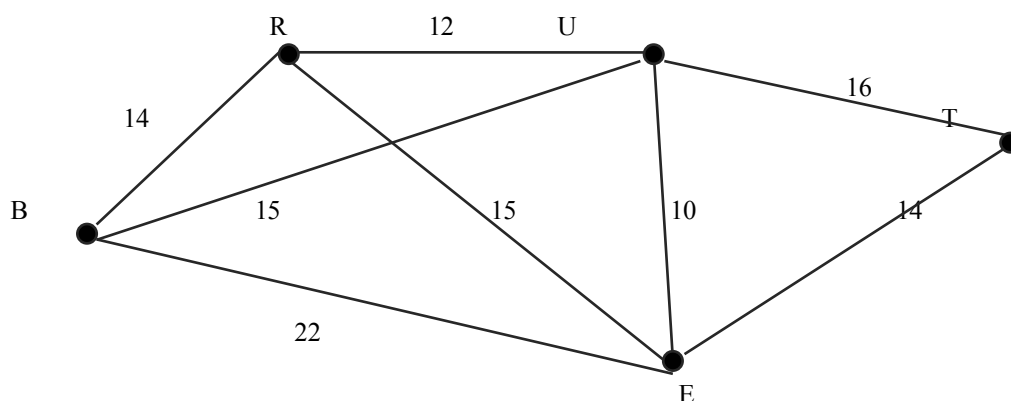


Figure Q3. A weighted graph over vertices B, R, U, T and E.

4. Various

- (b) Given a sequence of pairs of integers representing connections between objects (see Figure Q4, left), the task of a connectivity algorithm is to output those pairs that provide new connections (see Figure Q4, centre). For example, the pair 2-9 is not part of the output because the connection 2-3-4-9 is implied by previous connections. This evidence is shown in Figure Q4, (right). Give the output that a connectivity algorithm should produce when given the input:-

1-0, 1-6, 2-3, 4-0, 1-4, 0-7, 5-3, 2-0, 3-6, 4-5, 6-4, 0-6, 7-6, 1-5, 2-5, 2-1, 4-3, 5-7, 0-8, 7-8.

3-4	3-4	
4-9	4-9	
2-3	2-3	
5-6	5-6	
2-9		2-3-4-9
5-9	5-9	
6-9		6-5-9

Figure Q4. A connectivity example.

(10 marks)

- (b) Write down the first TEN terms of the sequence defined recursively as:

$$F_{k+1} = F_k + F_{k-1}, \text{ where } F_1 = 1, F_2 = 1$$

(5 marks)

and verify algebraically that the nth term can be written explicitly as:

$$F_n = \frac{1}{\sqrt{5}} [a^n] - \frac{1}{\sqrt{5}} [b^n].$$

in which $a = (1 + \sqrt{5}) / 2$ and $b = (1 - \sqrt{5}) / 2$. (Hint: You might wish to try simplifying $F_{n+1} - F_{n-1}$).

(10 marks)

5. Computer Graphics

- (a) Describe with the aid of a suitable example one of the following line drawing algorithms:

- (i) DDA algorithm
- (ii) Bresenham's algorithm

(8 marks)

- (b) Describe with the aid of a suitable example one of the following curve drawing algorithms:

- (i) Midpoint Circle algorithm
- (ii) Midpoint Ellipse algorithm

(8 marks)

- (c) Describe with the aid of a suitable example one of the following line clipping algorithms:

- (i) Cohen-Sutherland algorithm
- (ii) Liang-Barsky algorithm

(9 marks)

End of examination paper

SKETCH SOLUTIONS – FALL 2002 FINAL EXAM

1.

- (a) Euclid's algorithm to find the greatest common divisor of the two numbers 1440 and 575.

$$\begin{aligned}
 \gcd(1440, 575) &= \gcd(575, 290) \\
 &= \gcd(290, 285) \\
 &= \gcd(285, 5) \\
 &= \gcd(5, 0) \\
 \gcd &= 5.
 \end{aligned}$$

(6 marks)

(b)

attempt 1
 GCATCGCAGAGAGTACAGTACG
 GCAGAGAG shift 4
 attempt 2
 GCATCGCAGAGAGTACAGTACG
 GCAGAGAG shift 1
 Attempt 3 .. 8
 GCATCGCAGAGAGTACAGTACG
 GCAGAGAG

String 24
 Pattern 8
 Attempts 8
 Comparisons 18

(6 marks)

- (c) Build a Huffman tree by counting characters as they stream in and placing higher frequency characters further up the tree. Assign 0 to left branch and 1 to right branch to derive the code for each character. Most frequently occurring characters will have a shorter code length. The approximate percentage space saving compared to using a standard 8-bit code is 50%

(7 marks)

(d)

N	$N^3 + 16$	$N^2 + 64$
2	24	68
3	43	73
4	82	80

Least integer value of N = 4

N	$2N - 1$	$\lg(64N)$
2	4	7
4	8	8
8	16	9

Least integer value of N = 8

(6 marks)

2.

(a)

(iv) bubblesort

14	23	12	24	33	18	16	20
12	14	23	16	24	33	18	20
12	14	16	23	18	24	33	20
12	14	16	18	23	20	24	33
12	14	16	18	20	23	24	33
No more swaps							

(7 marks)

(v) quicksort

14	23	12	24	33	18	16	20
14	16	12	18	20	24	23	33
14	16	12	18	20	23	24	33
12	16	14	18	20	23	24	33
12	14	16	18	20	23	24	33
No more swaps							

(7 marks)

(iii) bubblesort uses about $n(n-1)/2$ steps, which is $O(n^2)$ whereas quicksort repeatedly divides $\lg n$ times then conquers, which is $O(n \lg n)$. For large n , quicksort is the better choice.

(c)

$$\begin{aligned}
 T(1) &= 1 \\
 T(2) &= 4 \\
 T(4) &= 12 \\
 T(8) &= 32 \\
 T(16) &= 80 \\
 T(n) &= N((\lg N) + 1) \\
 T(n) &= O(N \lg N)
 \end{aligned}$$

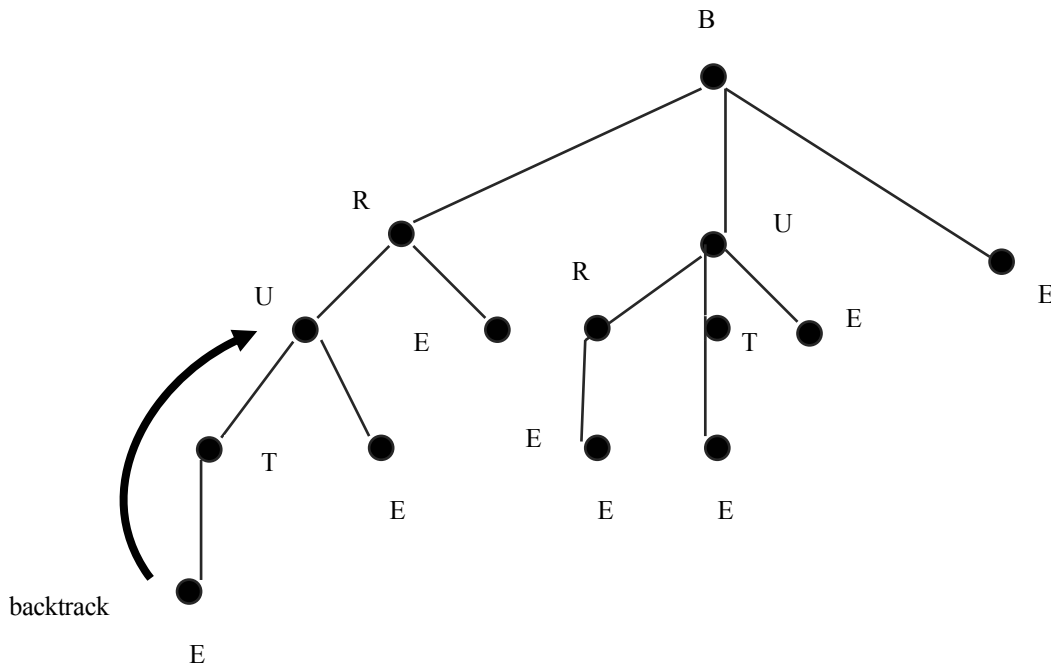
(7 marks)

3.

brute-force algorithm: calculate all possibilities then choose a 'cheap' solution

greedy algorithm: calculate only 'cheap' solutions as the algorithm proceeds (not all possibilities found).

(4 marks)



(a) Exhaustive depth-first search with ordering (B,R,U,T,E) (6 marks)

DFS solutions in order: (B,R,U,T,E), (B,R,U,E), (B,R,E), (B,U,R,E), (B,U,T,E), (B,U,E), (B,E); cheapest (B,E).

(b) Exhaustive breadth-first search with ordering (B,R,U,T,E) (6 marks)

BFS solutions in order: (B,E), (B,R,E), (B,U,E), (B,U,R,E), (B,R,U,E), (B,U,T,E), (B,R,U,T,E); cheapest (B,E).

(c) Best-first search (expand best node according to weight) (4 marks)

Best-first solution: (B,R,U,E); cost 36.

dfs, bfs brute-force; best-first greedy (3 marks)

dfs, bfs guaranteed minimal, best-first not guaranteed minimal. (2 marks)

4.

(c) input:-

1-0, 1-6, 2-3, 4-0, 1-4, 0-7, 5-3, 2-0, 3-6, 4-5, 6-4, 0-6, 7-6, 1-5, 2-5, 2-1, 4-3, 5-7, 0-8, 7-8.

,,,,,,,,,,,,,,,,,,,,,

1-0	1-0	
1-6	1-6	
2-3	2-3	
4-0	4-0	
1-4	1-4	
0-7	0-7	
5-3	5-3	
2-0	2-0	
3-6		3-2-0-1-6
4-5		4-0-2-3-5
6-4		6-3-5-4
0-6		0-1-6
7-6		7-0-6
1-5		1-4-5
2-5		2-3-5
2-1		2-0-1
4-3		4-5-3
5-7		5-1-6-7
0-8	0-8	
7-8		7-0-8

(10 marks)

(b)

$$F_{k+1} = F_k + F_{k-1}, \text{ where } F_1 = 1, F_2 = 1$$

first TEN terms = 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.

Verify that $F_{n+1} - F_{n-1} = F_n$

$$\begin{aligned}
 \text{LHS} &= F_{n+1} - F_{n-1} \\
 &= \frac{1}{\sqrt{5}} [a^{n+1}] - \frac{1}{\sqrt{5}} [b^{n+1}] - \frac{1}{\sqrt{5}} [a^{n-1}] + \frac{1}{\sqrt{5}} [b^{n-1}] \\
 &= \frac{1}{\sqrt{5}} (a^2 - 1) [a^{n-1}] - \frac{1}{\sqrt{5}} (b^2 - 1) [b^{n-1}] \\
 &= \frac{1}{\sqrt{5}} a [a^{n-1}] - \frac{1}{\sqrt{5}} b [b^{n-1}] \text{ since } (a^2 - 1) = a \text{ and } (b^2 - 1) = b \\
 &= \frac{1}{\sqrt{5}} [a^n] - \frac{1}{\sqrt{5}} [b^n] \\
 &= F_n \\
 &= \text{RHS}
 \end{aligned}$$

(12 marks)

5.

<p>The Digital Differential Analyzer algorithm (DDA) is a scan conversion line algorithm based on calculating either dx or dy using $dx = m \cdot dy$</p> <p>We sample the line at unit intervals in one coordinate and determine corresponding integer values nearest the line path for the other coordinate.</p> <p>The DDA algorithm is a faster method for calculating pixel positions than the direct use of the equation:</p> $y = m \cdot x + b$ <p>It eliminates the multiplication by taking advantage of raster characteristics, so that appropriate increments are applied in the x and y direction to step to pixel positions along the line path. The accumulation of round-off error in successive addition of the floating point increment, however can cause the calculated pixel positions to drift away from the true line path for long line segments. Furthermore the rounding operations and floating-point arithmetic are still time consuming. We can improve the performance of the DDA algorithm by separating the increments of m and 1/m into integer and fractional parts so that all calculations are reduced to integer operations.</p>	<p>An accurate and efficient raster line-generating algorithm, developed by Bresenham, scan converts lines using only incremental integer calculations that can be adapted to display circles and other curves.</p> <p>Sampling at unit x intervals we need to decide which of two possible pixel positions is closer to the line path at each step.</p> <p>To illustrate Bresenham's approach, we first consider the scan conversion process for lines with positive slope less than 1. Pixel positions along a line path are then determined by sampling at x unit intervals. Starting from the left end-point of a given line (x0,y0), we step to each successive column (x position) and plot the pixel whose scan-line y is closest to the line path. Assuming that at the k-th step we have decided that the pixel at (xk,yk) is to be displayed, we need to determine which pixel to plot in column xk+1. Our choices are the pixels at positions (xk+1,y) and (xk+1, yk+1). Using a parameter p which is updated every frame we can make that decision. The formula for parameter p is very well shown in the example code.</p>
<p>As in the raster line algorithm, we sample at unit intervals and determine the closest pixel position to the specified circle path at each step. For a given radius r and a screen center position (xc,yc), we can first set up our algorithm to calculate pixel positions around a circle path centered at the coordinate origin (0,0). Then each calculated position (x,y) is moved to its proper screen position by adding xc to x and yc to y. Along the circle section from x=0 to x=y in the first quadrant the slope of the curve varies from 0 to -1. Therefore we can take unit steps in the positive x direction over this octant and use the decision parameter to decide which of the two possible y positions is closer to the circle path at each step. Positions in other octants are then obtained by symmetry. To apply the midpoint method we define a circle function: $f(x,y) = x^2 + y^2 - r^2$. This equation has the property that for any given pair (x,y):</p> <ul style="list-style-type: none"> $f(x,y) < 0$ if (x,y) inside the circle boundaries $f(x,y) = 0$ if (x,y) is on the circle boundaries $f(x,y) > 0$ if (x,y) is outside the circle boundaries 	<p>The approach in drawing the ellipse is similar to that used in displaying a raster circle. Given parameters rx and ry, and (xc,yc) we determine points (x,y) for an ellipse in the standard position centered on the origin, and then we shift the points so the ellipse is centered at (xc, yc).</p> <p>The midpoint ellipse method is applied throughout the first quadrant in two parts. We process this quadrant by taking unit steps in the x direction where the slope of the curve has a magnitude less than 1, and taking unit steps in the y direction where the slope has a magnitude greater than 1. Therefore we obtain 2 regions that can be processed in various ways. We can start at position (0,ry) and step clockwise along the elliptical path in the first quadrant, shifting from unit steps in x to unit steps in y when the slopes becomes less than -1.</p> <p>With parallel processors we can calculate pixel positions in the two regions simultaneously.</p>
<p>This is one of the oldest and most popular line-clipping procedures. Generally, the method speeds up the processing of line segments by performing initial tests that reduce the number of intersections that must be calculated. Every line endpoint is assigned a 4 digit binary code, called a region code, that identifies the location of the point relative to the boundaries of the clipping region. By numbering the bit positions in the region code as 1 through 4 from right to left, the coordinate regions can be correlated with the bit positions as:</p> <ul style="list-style-type: none"> bit 1: left bit 2: right bit 3: below bit 4: above <p>Once we have established region codes for all line endpoints, we can quickly determine which lines are completely inside the clip window and which are clearly outside. Any lines that are completely contained within the window boundaries have a region code of 0000 at both endpoints, and we trivially accept those lines. Any lines that have a 1 in the same bit position in the region codes for each end point are completely outside the clipping rectangle, and we trivially reject these lines. Lines that cannot be identified as completely inside or completely outside a clip window by these tests are checked for intersection with the window boundaries.</p>	<p>Faster line clippers have been developed that are based on analysis of the parametric equation of a line segment, which we can write it in the form:</p> $x = x_1 + u \cdot dx \quad 0 \leq u \leq 1$ $y = y_1 + u \cdot dy \quad 0 \leq u \leq 1$ <p>where $dx = x_2 - x_1$ and $dy = y_2 - y_1$. Using these parametric equations, Cyrus and Beck developed an algorithm that is generally more efficient than the Cohen-Sutherland algorithm. Later, Liang and Barsky independently devised an even faster parametric line-clipping algorithm. Following the Liang Barsky approach we first write the point-clipping conditions in the parametric form:</p> $x_{wmin} \leq x_1 + u \cdot dx \leq x_{wmax}$ $y_{wmin} \leq y_1 + u \cdot dy \leq y_{wmax}$ <p>Each of these 4 inequalities can be expressed as: $up_k \leq q_k$ where parameters p and q are defined as:</p> $p_1 = -dx, \quad q_1 = x_1 - x_{wmin}$ $p_2 = dx, \quad q_2 = x_{wmax} - x_1$ $p_3 = -dy, \quad q_3 = y_1 - y_{wmin}$ $p_4 = dy, \quad q_4 = y_{wmax} - y_1$ <p>Any line that is parallel to one of the clipping boundaries has $p_k = 0$ for the value of k corresponding to that boundary. If for that value of k we also found that $p_k < 0$ then the line is completely outside the boundary. For each line we calculate values for parameters u1 and u2, that define that part of the line that lies within the clipping rectangle</p> $u = q_k / p_k$

CSCI 319

FINAL EXAMINATION

FALL 2003

ALGORITHMS

Time Allowed: 2 hours

Answer at least FOUR questions. If you answer more than four, only the best four will be taken into account. All questions carry equal marks. You are advised to read each question completely before writing your answer. Show all working in the booklet so that credit can be given for any planning or method of calculation.

The notation $\lg N$ is used throughout to represent the logarithm of N to base 2.

The symbols “+” and “*” represent arithmetical addition and multiplication, respectively.

1. Puzzles

(a) Write down the first FOUR terms of the sequences generated by:

(i) $F(N) = 3 * N * (N + 1)$ with $N = 1, 2, 3$ and 4 . (4 marks)

(ii) $F(N + 1) = \frac{(N + 2)}{N} * F(N)$ with $F(1) = k$, where k is a constant (4 marks)

(b) The “N-Sliding Blocks” puzzle consists of N blocks marked with “O” and N blocks marked with “X” all positioned in a row. The objective of the puzzle is to transpose the O-blocks and the X-blocks by sliding the blocks along the row using a central column of N spaces to enable blocks to slide past. The “1-Sliding Blocks” puzzle and its solution is shown in figure Q1(a) where it can be seen that there are 6 minimum moves required. The starting positions for the “N-Sliding Blocks” puzzles when $N = 2, 3$ and 4 are shown in figure Q1(b).

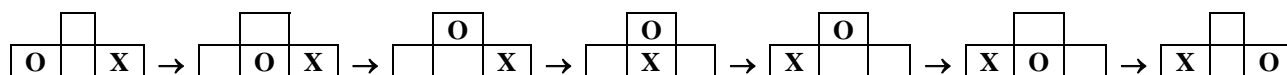


Figure Q1(a). The “1-Sliding Blocks” puzzle solved in six moves.

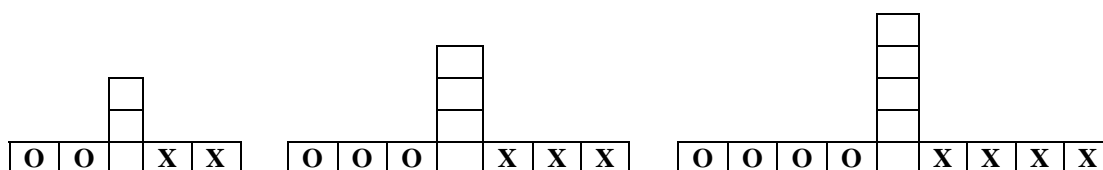


Figure Q1(b). The starting positions for $N = 2, 3$ and 4 in the “N-Sliding Blocks” puzzle.

Question Q1(b) continued over the page ...

Question Q1(b) continued ...

- (i) Reason through the various steps in the “N-Sliding Blocks” puzzle for $N = 1, 2, 3$ and 4 and create a table of values for the minimal number of moves needed. (8 marks)
- (ii) Suggest by conjecture a possible generalised closed-form (explicit) function in terms of N for the minimal number of moves. (Hint: You might wish to recall your answer to part (a)(i)). (2 marks)
- (iii) Suggest by conjecture a possible recurrence relation solution for the minimal number of moves. (Hint: You might wish to recall your answer to part (a)(ii), choosing a suitable value for k). (4 marks)
- (iv) Without reasoning through the specific steps, evaluate the minimal number of moves for the “100-Sliding Blocks” puzzle. (3 marks)

2. Graph Algorithms

Explain the terms brute-force and greedy as used to describe certain algorithms.

(4 marks)

For the weighted graph in figure Q2, show clearly the steps involved in finding the shortest path from E to S using the following algorithms:

- (a) Exhaustive depth-first search with ordering (D, E, O, R, S) (7 marks)
- (b) Exhaustive breadth-first search with ordering (D, E, O, R, S) (7 marks)
- (c) Best-first search (expand best node according to weight) (4 marks)

Classify each of the algorithms in parts (a), (b) and (c) as either brute-force or greedy.

(3 marks)

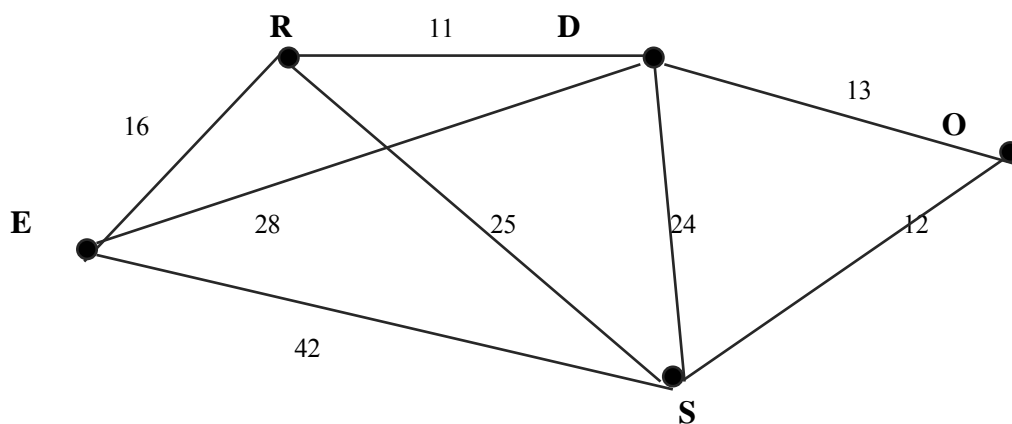


Figure Q2. A weighted graph over vertices E, R, D, O and S.

3. Numerical Algorithms

The sequence of Fibonacci numbers

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, ...

can be generated by the following three distinct procedures, labelled PROC1, PROC2 and PROC3, respectively:

- PROC1: Recursive - using a suitable recurrence relation
 PROC2: Iterative – using a suitable closed-form expression
 PROC3: Modified Recursive – using a suitably modified form of PROC1

Summarise these three procedures, including in your summary an analysis of the corresponding computational complexity treating integer additions and evaluations as distinct from real additions and evaluations. Also include a summary of the relative merits of the procedures when the number of terms in the sequence to be generated is large.

(25 marks)

4. Analysis of Algorithms

- (a) Given that k_1, k_2, k_3 are constants, find the orders of the following three functions for large N :

- (iii) $F_1 = (k_1 * (k_2 + N)) + (k_3 * \lg N)$
 (iv) $F_1 = (k_1 * \lg N * (k_2 + N)) + (k_3 * N)$
 (v) $F_1 = (k_1 * N * (k_2 + N)) + (k_3 * \lg N)$ (6 marks)

- (b) In an experimental analysis, the run-times for three different algorithms, labelled ALG1, ALG2 and ALG3 respectively, were recorded in an experiment solving similar-sized problems. Each algorithm was respectively applied to problems of size N (measured in decimal form) and the corresponding run-time values recorded in scientific notation. The results for various values of N are shown in figure Q4.

N	RUN-TIME ALG 1	RUN-TIME ALG2	RUN-TIME ALG3
(problem size)	(milliseconds)	(milliseconds)	(milliseconds)
16	5.839600E-01	1.850945E+02	6.388083E+03
32	1.966360E+00	4.367473E+02	6.776033E+03
64	7.495960E+00	1.011954E+03	7.551934E+03
128	2.961436E+01	2.306168E+03	9.103736E+03
256	1.180880E+02	5.182200E+03	1.220734E+04
512	4.719824E+02	1.150947E+04	1.841455E+04
1024	1.887560E+03	2.531442E+04	3.082896E+04
2048	7.549870E+03	5.522516E+04	5.565778E+04
4096	3.019911E+04	1.196483E+05	1.053154E+05
8192	1.207961E+05	2.576978E+05	2.046307E+05

Figure Q4. Run-times are shown in milliseconds for three algorithms.

Suppose that prior research into the three algorithms collectively had revealed that one is order N , another order $N \log N$ and a third order N squared but the precise identity of each was not known.

- (iv) Produce graphical plots to identify ALG1, ALG2 and ALG3 appropriately based on the recorded data. (13 marks)
- (v) Which algorithm would you use to solve “small” problems in which the problem size is never likely to exceed 8192? (3 marks)
- (vi) Which algorithm would you use to solve “large” problems in which the problem size is always likely to exceed 16384? (3 marks)

5. String Algorithms

Question parts (a) and (b) both refer to the string shown in figure Q5.

ABU JA'FAR MUHAMMAD IBN MUSA AL-KHWARIZMI

Figure Q5. A string of 41 characters which, unless otherwise stated, should be read from left to right. The spaces and punctuations count as characters.

- (a) Explain clearly and concisely how you would analyse the cost in terms of space (bits) involved in encoding the string shown in figure Q5 using the following two coding techniques:
- (i) The standard 8-bit ASCII code (2 marks)
 - (ii) A suitable Huffman code. (9 marks)
- (b) Explain clearly and concisely how you analyse the cost in terms of time (steps) involved in searching for the pattern **AMM** in the string shown in figure Q5, using the following two well-known algorithms:
- (i) The Boyer-Moore algorithm (7 marks)
 - (ii) The Knuth-Morris-Pratt algorithm. (7 marks)

End of examination paper

SKETCH SOLUTIONS – FALL 2003 FINAL EXAM

1.

(b) The first FOUR terms:

(iii) 6, 18, 36, 60. (4 marks)

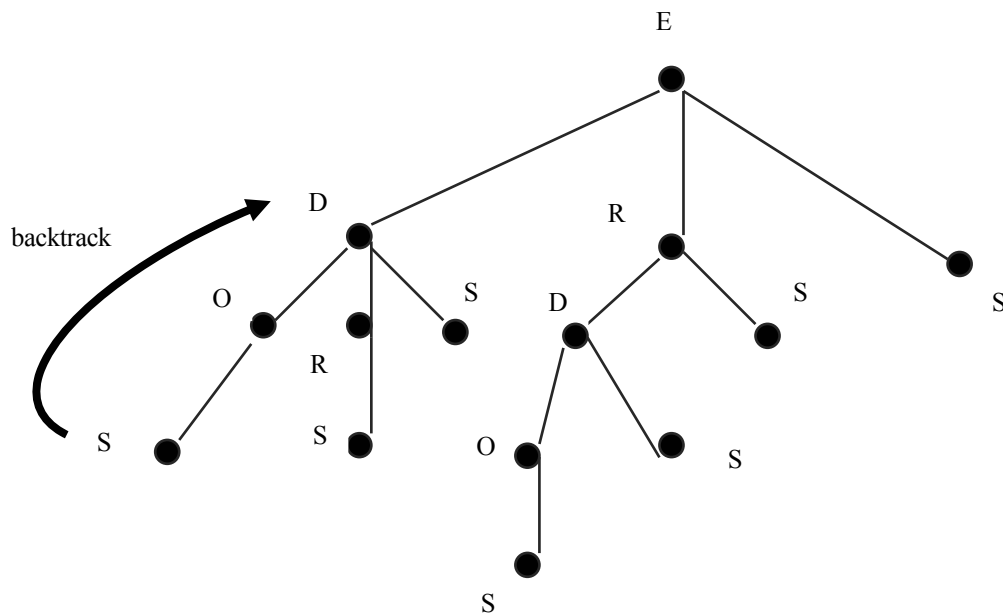
(iv) k, 3k, 6k, 10k (4 marks)

(b)

(v) 6, 18, 36, 60. (8 marks)

(vi) $F(N) = 3 * N * (N+1)$ (2 marks)(v) $F(N + 1) = \frac{(N + 2)}{N} * F(N)$ with $F(1) = 6$ (4 marks)(vii) Minimal number of moves for the “100-Sliding Blocks” puzzle = $3.100.101 = 30300$. (3 marks)

2. (a) brute-force algorithm: calculate all possibilities then choose a 'cheap' solution
 greedy algorithm: calculate only 'cheap' solutions as the algorithm proceeds (not all possibilities found).
 (4 marks)
- (e) Search tree with ordering (D,E,O,R,S)



Exhaustive DFS gives seven solutions in order: (E,D,O,S), (E,D,R,S), (E,D,S), (E,R,D,O,S), (E,R,D,S), (E,R,S), (E,S); cheapest of the five (E,R,S) costs 41 units.

(7 marks)

Exhaustive BFS gives seven solutions in order: (E,S), (E,D,S), (E,R,S), (E,D,O,S), (E,D,R,S), (E,R,D,S), (E,R,D,O,S); cheapest of the five (E,R,S) costs 41 units

(7 marks)

Best-first (branch-and-bound on best one node):
 expand E to R (cheapest weight of 11), note other weights;
 expand R to D (total weight of 27), note other weights;
 expand D to O (total weight of 40), note other weights;
 expand O to S (total weight of 52),
 solution (E,R,D,O,S), cost 52.

(4 marks)

dfs, bfs brute-force; b&b greedy (3 marks)

3.

The sequence of Fibonacci numbers

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, ...

can be generated by the following three distinct procedures, labelled PROC1, PROC2 and PROC3, respectively:

PROC4: Recursive - using a suitable recurrence relation

$\text{Fib}(N+2) = \text{Fib}(N+1) + \text{Fib}(N)$ with $\text{Fib}(1) = \text{Fib}(2) = 1$

$\text{Fib}(3)$ requires 1 integer addition and 2 integer evaluations

$\text{Fib}(4)$ requires 2 integer additions and 3 integer evaluations

$\text{Fib}(5)$ requires 3 integer additions and 4 integer evaluations

...

$\text{Fib}(N)$ requires $(N-1)$ integer additions and $(N-1)$ integer evaluations

Calculating all N terms requires $(N-2)(N-3)/2$ integer additions and $(N-1)(N-2)/2$ integer evaluations. Therefore, PROC1 is $O(N^2)$.

PROC5: Iterative – using a suitable closed-form expression

Algebraically that the n th term can be written explicitly as:

$$F_n = \frac{1}{\sqrt{5}} [a^n] - \frac{1}{\sqrt{5}} [b^n] .$$

in which $a = (1 + \sqrt{5}) / 2$ and $b = (1 - \sqrt{5}) / 2$. Repeatedly inserting a value of n into this expression gives values of F_n .

F_1 requires 2 real evaluations and 1 real addition.

F_2 requires 2 real evaluations and 1 real addition.

F_3 requires 2 real evaluations and 1 real addition.

...

F_n requires 2 real evaluations and 1 real additions.

Calculating all n terms requires $2n$ real evaluations and n real additions. Therefore, PROC2 is $O(n)$.

PROC6: Modified Recursive – using a suitably modified form of PROC1

Use PROC1 but store values up to $\text{Fib}(k)$ into an array. $\text{Fib}(k+1)$ then requires 2 look-up evaluations (viz. $\text{Fib}(k-1)$ and $\text{Fib}(k)$), one addition and one store. This approach is referred to as dynamic programming. Therefore PROC3 is $O(n)$.

PROC1 is $O(N^2)$ and both PROC2 and PROC3 are $O(N)$. When the number of terms in the sequence is large choose either PROC2 or PROC3.

(25 marks)

4. (a)

(vi) $F_1 = (k_1 * (k_2 + N)) + (k_3 * \lg N) = O(N)$

(vii) $F_1 = (k_1 * \lg N * (k_2 + N)) + (k_3 * N) = O(N \lg N)$

(viii) $F_1 = (k_1 * N * (k_2 + N)) + (k_3 * \lg N) = O(N^2)$

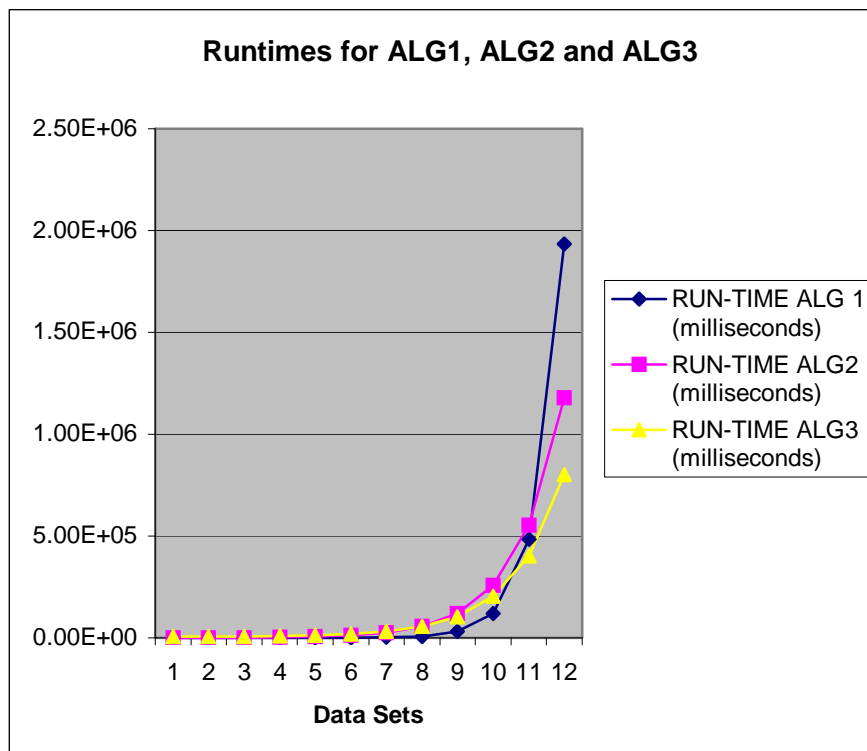
(6 marks)

(b)

(i) Choosing any two values in each case gives the constants a_i and b_j to produce each formula.

N	RUN-TIME ALG 1 (milliseconds)	RUN-TIME ALG2 (milliseconds)	RUN-TIME ALG3 (milliseconds)
16	5.839600E-01	1.850945E+02	6.388083E+03
32	1.966360E+00	4.367473E+02	6.776033E+03
64	7.495960E+00	1.011954E+03	7.551934E+03
128	2.961436E+01	2.306168E+03	9.103736E+03
256	1.180880E+02	5.182200E+03	1.220734E+04
512	4.719824E+02	1.150947E+04	1.841455E+04
1024	1.887560E+03	2.531442E+04	3.082896E+04
2048	7.549870E+03	5.522516E+04	5.565778E+04
4096	3.019911E+04	1.196483E+05	1.053154E+05
8192	1.207961E+05	2.576978E+05	2.046307E+05
16384	4.831839E+05	5.522035E+05	4.032613E+05
32768	1.932735E+06	1.178028E+06	8.005226E+05

Alg1 Formula = SUM(PRODUCT(0.00045,N,N),0.12316) = $O(N^2)$
 Alg2 Formula = SUM(PRODUCT(1.12345,N,LOG(N,2)),5.34251) = $O(N \lg N)$
 Alg3 Formula = SUM(PRODUCT(12.12345,N,),6000.13231) = $O(N)$



(13 marks)

(ii) Alg1 has smaller run-times for values of N not exceeding 8192

(3 marks)

(iii) Alg3 has smaller run-times for values of N exceeding 16384

(3 marks)

5.

ABU JA'FAR MUHAMMAD IBN MUSA AL-KHWARIZMI

- (a) Explain clearly and concisely how you would build a Huffman tree to compress a large text file containing alphanumeric characters.

Read each text character

Perform a frequency count (e.g. “e” – 13%, “t” – 11%, etc.)

Allocate small length code to most frequently occurring characters

Allocate large length code to least frequently occurring characters

Ensure no two characters are allocated the same code

Char	Freq	ASCII Code	Huffman Code
A	8	8 bits	3 bits
M	5	8 bits	4 bits
space	4	8 bits	4 bits
etc
	$\Sigma = 41$	$\Sigma = 328$	$\Sigma = ?$

Approximate percentage space saving of Huffman compared to ASCII is approx 50%.

(11 marks)

(b)

Boyer-Moore algorithm:

ABU JA'FAR MUHAMMAD IBN MUSA AL-KHWARIZMI

AMM
 AMM
 AMM
 AMM
 AMM
 AMM

Check for M then skip. Pattern found at position 14.

(7 marks)

Knuth-Morris-Pratt algorithm:

ABU JA'FAR MUHAMMAD IBN MUSA AL-KHWARIZMI

AMM
 AMM
 AMM
 AMM

Look for A then check for M. Pattern found at position 14.

(7 marks)

CSC 319

FINAL EXAMINATION

FALL 2004

ALGORITHMS

Time Allowed: 2 hours

Answer at least FOUR questions. If you answer more than four, only the best four will be taken into account. All questions carry equal marks. You are advised to read each question completely before writing your answer. Show all working in the booklet so that credit can be given for any planning or method of calculation.

The following notation is used throughout:

- \mathbf{N} represents the set of natural numbers,
- $\lg n$ represents $\log_2(n)$, the logarithm of n to base 2, where $n \in \mathbf{N}$.

1. Compression & Graphics

- (a) For the small alphabet $\{a, f, h, m, n, u\}$, consider the two binary codes shown in figure Q1.

Symbol	Code 1	Code 2
a	001	001
f	000	11
h	010	10
m	100	011
n	011	010
u	101	000

Figure Q1: Code 1 is a fixed length code and Code 2 has the prefix property.

- (i) Encode the characters of the word “huffman” using each code. (4 marks)
- (ii) Using the answer in part (a), calculate the percentage saving for Code 2 compared to Code1. (3 marks)
- (iii) Assuming a left-to-right reading and no errors, explain at least two reasons why the string:
10000011001010
must have used Code 2. (2 marks)
- (iv) Decode the string in part (iii). (2 marks)
- (b) Using a simple, though non-trivial example, explain clearly and concisely the steps involved in any TWO of the following graphics algorithms:
- (vi) Bresenham’s algorithm,
 - (vii) The Cohen-Sutherland algorithm,
 - (viii) The Liang-Barsky algorithm. (7 marks each)

2. Geometric

A “n-omino” is a generalization of the well-known domino to a collection of n squares of equal size arranged with coincident sides. Free n-ominoes can be flipped, so mirror image pieces are considered identical. There is one free 2-omino (the domino), two distinct free 3-ominoes and five distinct free 4-ominoes. These are represented in figure Q2.

- (i) Sketch clearly the twelve free pentominoes. (6 marks)
- (ii) Consider a modified 8X8 chessboard with the two diagonally opposite corners removed. Is it possible to cover the board with free dominoes and, if so, sketch one such covering? If not, explain why not? (4 marks)
- (iii) Consider a modified 8X8 chessboard with one corner square removed. Is it possible to cover the modified board with free triominoes and, if so, sketch one such covering? If not, explain why not? (4 marks)
- (iv) Consider a modified 8X8 chessboard with the four corner squares removed. Is it possible to cover the modified board with free pentominoes and, if so, sketch one such covering? If not, explain why not? (4 marks)
- (v) Give a concise description of an algorithm for enumerating all n-ominoes (polyominoes of n squares) using inductive exhaustive search (7 marks)

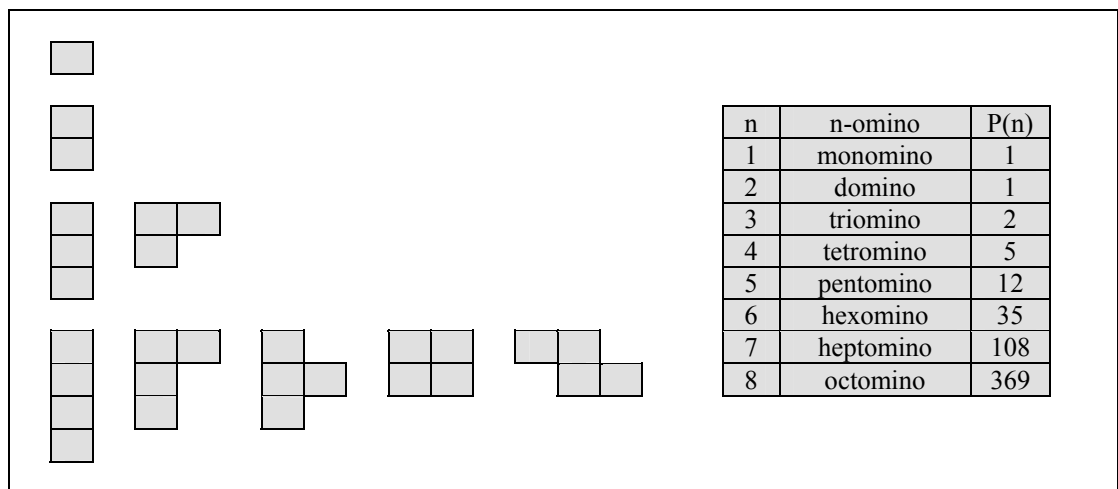


Figure Q2: Free n-omino sketches for $n = 1, 2, 3, 4$ (figure left) and a table of $P(n)$ values up to $n = 8$ (figure right). $P(n)$ is the number of free n-ominoes.

3. Graph Searching & Numerical

- (a) (i) Explain the term brute-force used to describe certain algorithms. (2 marks)
- (ii) For the weighted graph in figure Q3, show clearly the steps involved in finding the shortest path from K to H using the following searching algorithms:
1. Exhaustive depth-first search with ordering (K,N,U,T,H) (8 marks)
 2. Exhaustive breadth-first search with ordering (K,N,U,T,H) (8 marks)

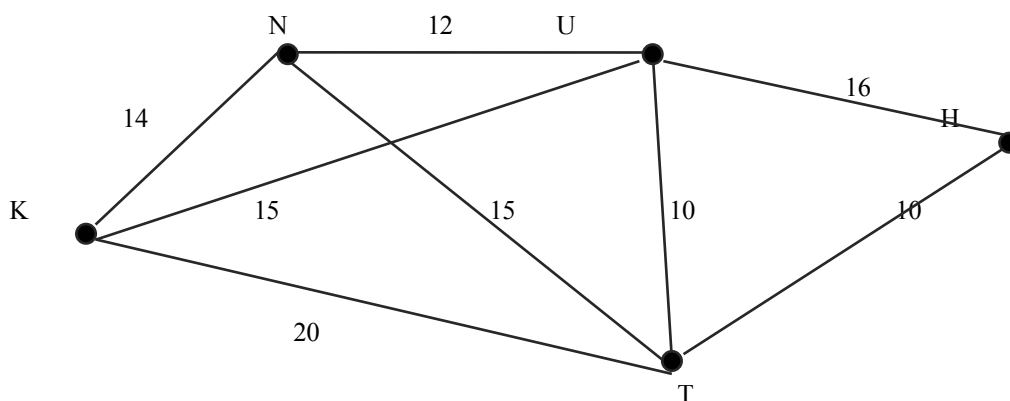


Figure Q3. A weighted graph over vertices K, N, U, T and H.

- (b) Show clearly how Euclid's algorithm can be used to find the greatest common divisor of two binary integers 10101010 and 101010. (7 marks)

4. Matrices & Combinatorics

- (a) (i) For $n \in \mathbf{N}$, explain clearly why the usual number of scalar operations (i.e. the total number of additions and multiplications) required to perform $n \times n$ matrix multiplication, $M(n)$, is given by $M(n) = 2n^3 - n^2$. (4 marks)
- (ii) Strassen (1969) discovered how to multiply two matrices using $S(n)$ scalar operations given by $S(n) = 7(7^{\lg n}) - 6(4^{\lg n})$

In the particular case when n is a power of 2, show that

$$S(n) = 7(n^{\lg 7}) - 6n^2$$

Hence show that $S(n) = O(n^{\lg 7})$. (9 marks)

- (b) We never fully see some of the faces of the smaller blocks in a standard 3X3 Rubik's cube. For instance, corner blocks show only three of their faces while blocks in the middle only show one.
- (i) In the 3X3, 4X4 and 5X5 versions, respectively, how many faces in total are never fully shown? (6 marks)
 - (ii) For $n \in \mathbf{N}$, how many faces in total are never fully shown in the $n \times n$ version? (3 marks)
 - (iii) Show that your answer in part (ii) is $O(n^2)$. (3 marks)

6. Sorting & Cryptographic

(a) For the sorting algorithm shown in figure Q5, explain clearly and concisely:

- (i) the number of swaps involved in sorting the particular array shown (7 marks)
- (ii) the minimum number of swaps involved in sorting a general array of size n (1 mark)
- (iii) the maximum number of swaps involved in sorting a general array of size n (3 marks)

(b) Using a simple though non-trivial example in each case, explain clearly and concisely the steps involved in the following two cryptographic algorithms:

- (i) Public Key Algorithms
- (ii) Secret Key Algorithms (7 marks each)

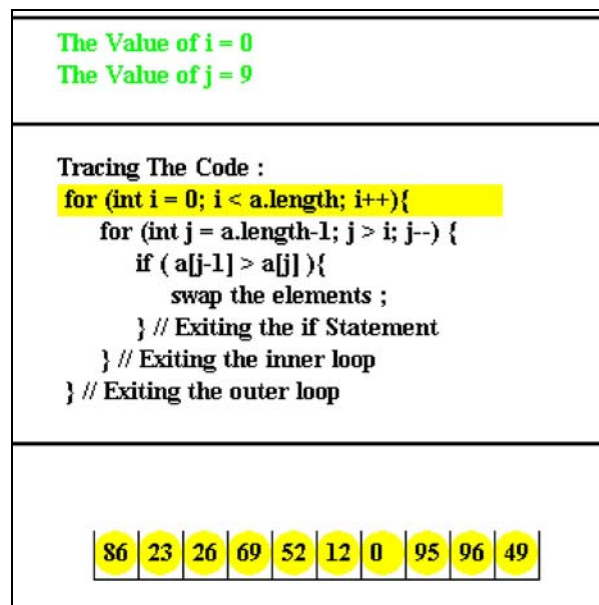


Figure Q5. A sorting algorithm for an array of integers.

End of examination paper

SKETCH SOLUTIONS – FALL 2004 FINAL EXAM

1.

(b)

(i) “huffman” encoding:-

Code 1: 010101000000100001011

Code 2: 100001111011001010

(4 marks)

(ii) 18 bits vs. 21 bits: 14% saving.

(3 marks)

(iii) 1. 14 bits not a multiple of 3 bits

2. reading left-to-right 100 (m) 000(f) 110(no match)

(2 marks)

(iv) “human”

(2 marks)

(b)

Bresenham: An accurate and efficient raster line-generating algorithm, developed by Bresenham, scan converts lines using only incremental integer calculations that can be adapted to display circles and other curves. Sampling at unit x intervals we need to decide which of two possible pixel positions is closer to the line path at each step.

To illustrate Bresenham's approach, we first consider the scan conversion process for lines with positive slope less than 1. Pixel positions along a line path are then determined by sampling at x unit intervals. Starting from the left end-point of a given line (x0,y0), we step to each successive column (x position) and plot the pixel whose scan-line y is closest to the line path. Assuming that at the k-th step we have decided that the pixel at (xk,yk) is to be displayed, we need to determine which pixel to plot in column xk+1. Our choices are the pixels at positions (xk+1,y) and (xk+1, yk+1). Using a parameter p that is updated every frame we can make that decision. The formula for parameter p is very well shown in the example code.

Cohen-Sutherland: This is one of the oldest and most popular line-clipping procedures. Generally, the method speeds up the processing of line segments by performing initial tests that reduce the number of intersections that must be calculated. Every line endpoint is assigned a 4 digit binary code, called a region code, that identifies the location of the point relative to the boundaries of the clipping region. By numbering the bit positions in the region code as 1 through 4 from right to left, the coordinate regions can be correlated with the bit positions as:

bit 1: left

bit 2: right

bit 3: below

bit 4: above

Once we have established region codes for all line endpoints, we can quickly determine which lines are completely inside the clip window and which are clearly outside. Any lines that are completely contained within the window boundaries have a region code of 0000 at both endpoints, and we trivially accept those lines. Any lines that have a 1 in the same bit position in the region codes for each end point are completely outside the clipping rectangle, and we trivially reject these lines. Lines that cannot be identified as completely inside or completely outside a clip window by these tests are checked for intersection with the window boundaries.

Liang-Barsky: Faster line clippers have been developed that are based on analysis of the parametric equation of a line segment, which we can write it in the form:

$$x = x_1 + udx$$

$$y = y_1 + udy \quad 0 \leq u \leq 1$$

where $dx = x_2 - x_1$ and $dy = y_2 - y_1$. Using these parametric equations, Cyrus and Beck developed an algorithm that is generally more efficient than the Cohen-Sutherland algorithm. Later, Liang and Barsky independently devised an even faster parametric line-clipping algorithm. Following the Liang Barsky approach we first write the point-clipping conditions in the parametric form:

$$x_{wmin} \leq x_1 + udx \leq x_{wmax}$$

$$y_{wmin} \leq y_1 + udy \leq y_{wmax}$$

Each of these 4 inequalities can be expressed as: $upk \leq qk$ where parameters p and q are defined as:

$$p_1 = -dx, \quad q_1 = x_1 - x_{wmin}$$

$$p_2 = dx, \quad q_2 = x_{wmax} - x_1$$

$$p_3 = -dy, \quad q_3 = y_1 - y_{wmin}$$

$$p_4 = dy, \quad q_4 = y_{wmax} - y_1$$

Any line that is parallel to one of the clipping boundaries has $pk=0$ for the value of k corresponding to that boundary. If for that value of k we also found that $pk < 0$ then the line is completely outside the boundary. For each line we calculate values for parameters u_1 and u_2 , that define that part of the line that lies within the clipping rectangle

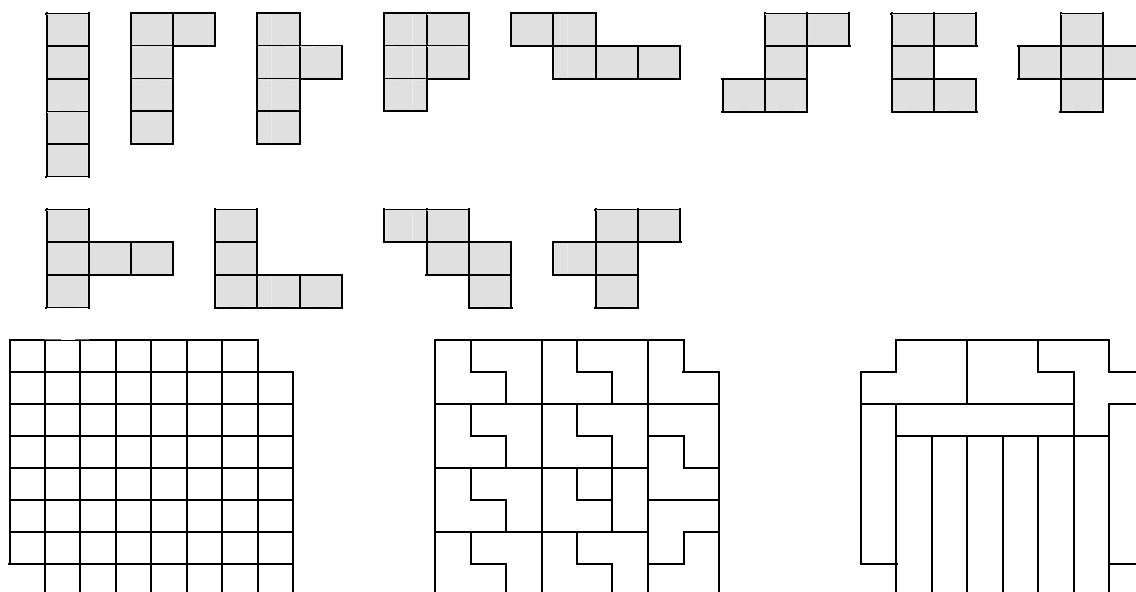
$$u = qk/pk$$

(7 marks each)

2.

(i) Sketch clearly the twelve free pentominoes.

(6 marks)



Not possible (any domino will cover one black, one white square but two black squares are missing)
(4 marks)

Possible
(4 marks)

Possible
(4 marks)

Algorithm for enumerating all n -ominoes (polyominoes of n squares) The polyominoes of area n can be found by inductive exhaustive search. Given a list of polyominoes of area n , take each polyomino in turn, embed it in an $n \times n$ square, surround that square with a collar of cells to create an $(n+2) \times (n+2)$ square. For each vacant cell in that square that is adjacent to at least one occupied cell, fill the cell and strike out a bounding row of vacant cells and a bounding column of vacant cells. The resulting $(n+1) \times (n+1)$ square contains a candidate polyomino of area $n+1$. If this configuration has not been encountered before, it is added to the list of polyominoes of area $n+1$. Comparison with the polyominoes of area $n+1$ already seen must take account of position and symmetry. Position can be accounted for by translating the candidate polyomino to the top left corner of the $(n+1) \times (n+1)$ square. Symmetry can be accounted for by noting that the group of symmetries of a square has eight elements and is generated by alternating reflections about the x -axis and about a diagonal. This procedure can be applied repeatedly starting from the monomino to reach any desired area of polyomino, but this becomes computationally expensive for large areas. For example finding all the dodecominoes using this algorithm consumes nearly 90 seconds of Central_processing_unit time on a 1GHz pentium. The polyominoes of area n can be found by inductive exhaustive search. Given a list of polyominoes of area n , take each polyomino in turn, embed it in an $n \times n$ square, surround that square with a collar of cells to create an $(n+2) \times (n+2)$ square. For each vacant cell in that square that is adjacent to at least one occupied cell, fill the cell and strike out a bounding row of vacant cells and a bounding column of vacant cells. The resulting $(n+1) \times (n+1)$ square contains a candidate polyomino of area $n+1$. If this configuration has not been encountered before, it is added to the list of polyominoes of area $n+1$. Comparison with the polyominoes of area $n+1$ already seen must take account of position and symmetry. Position can be accounted for by translating the candidate polyomino to the top left corner of the $(n+1) \times (n+1)$ square. Symmetry can be accounted for by noting that the group of symmetries of a square has eight elements and is generated by alternating reflections about the x -axis and about a diagonal. This procedure can be applied repeatedly starting from the monomino to reach any desired area of polyomino, but this becomes computationally expensive for large areas. For example finding all the dodecominoes using this algorithm consumes nearly 90 seconds of Central processing unit time on a 1GHz pentium.

(7 marks)

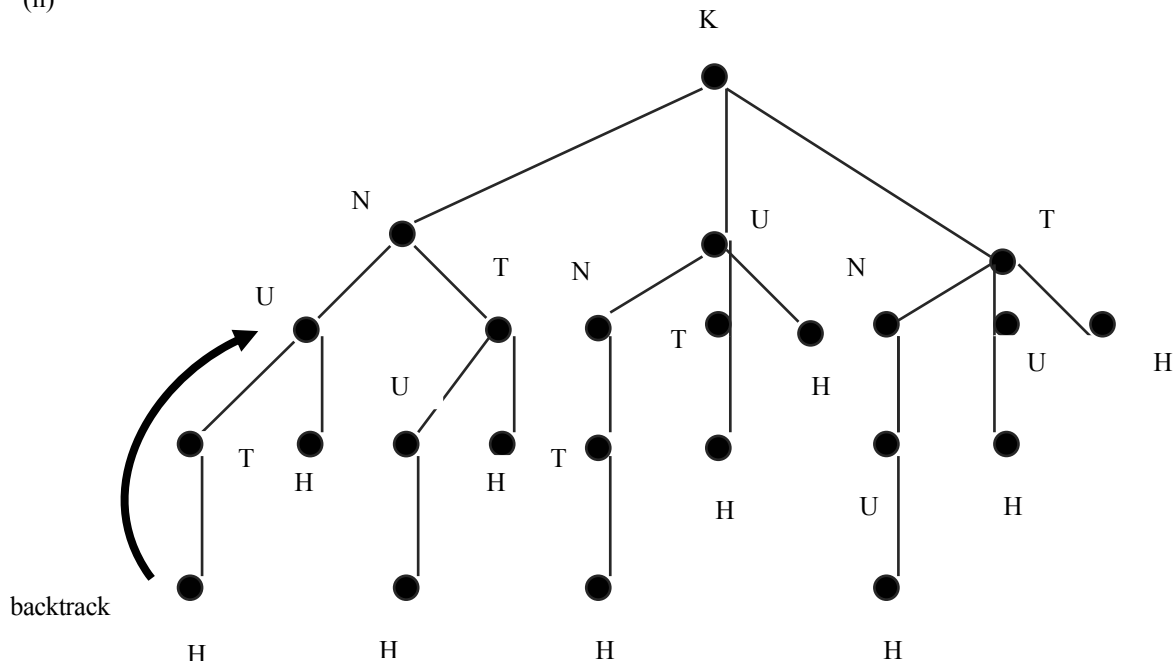
3.

(i)

brute-force algorithm: calculate all possibilities then choose a 'cheap' solution

(2 marks)

(ii)



1. Exhaustive depth-first search with ordering (K,N,U,T,H)

(8 marks)

DFS solutions in order:

(K,N,U,T,H), (K,N,U,H), (K,N,T,U,H), (K,N,T,H), (K,U,N,T,H), (K,U,T,H), (K,U,H); (K,T,N,U,H), (K,T,U,H), (K,T,H).

cheapest (K,T,H): cost 30.

2. Exhaustive breadth-first search with ordering (K,N,U,T,H)

(8 marks)

BFS solutions in order:

(K,U,H), (K,T,H), (K,N,U,H), (K,T,U,H), (K,N,T,H), (K,U,T,H), (K,N,T,U,H), (K,T,N,U,H), (K,N,U,T,H), (K,U,N,T,H).

cheapest (K,T,H): cost 30.

(c) Euclid's algorithm to find the greatest common divisor of the two numbers 10101010 and 101010. Convert to decimal.

$$\begin{aligned} \gcd(170,42) &= \gcd(42,2) \\ &= \gcd(2,0) \end{aligned}$$

$$\gcd = 2.$$

Convert to binary 10.

(7 marks)

4. Matrices & Combinatorics

- (a) (i) Explain clearly why the usual number of scalar operations (i.e. the total number of additions and multiplications) required to perform $n \times n$ matrix multiplication, $M(n)$, is given by

$$M(n) = \text{Mults} + \text{Addns.}$$

$$M(n) = n^3 + n^2(n-1).$$

$$M(n) = 2n^3 - n^2$$

(4 marks)

- (ii) Strassen (1969) discovered how to multiply two matrices using $S(n)$ scalar operations given by

$$S(n) = 7(7^{\lg n}) - 6(4^{\lg n})$$

When n is a power of 2, i.e. $n = 2^k$, we know that $\lg n = \lg 2^k = k \lg 2 = k$

$$\therefore S(n) = 7(7^k) - 6(4^k)$$

$$= 7(2^{\lg 7})^k - 6(2^{\lg 4})^k$$

$$= 7(2^{k \lg 7}) - 6(2^{k \lg 4})$$

$$= 7(2^k)^{\lg 7} - 6(2^k)^2$$

$$= 7(n^{\lg 7}) - 6n^2$$

$$S(n) = 7(n^{\lg 7}) - 6n^2$$

Since $S(n) = 7(n^{\lg 7}) - 6n^2 \leq 7(n^{\lg 7})$ for $n \geq 1$, we see that $S(n) \leq c \cdot n^{\lg 7}$ for $n \geq n_0$ when $c = 7$, $n_0 = 1$.

Hence, by definition, $S(n) = O(n^{\lg 7})$.

(9 marks)

- (b) We never fully see some of the faces of the smaller blocks in a standard 3X3 Rubik's cube. For instance, corner blocks show only three of their faces while blocks in the middle only show one.

- (ii) In the 3X3, 4X4 and 5X5 versions, respectively, how many faces in total are never fully shown?

$$3X3: \text{faces not shown} = (27-1).6 - 9.6 = 102$$

$$4X4: \text{faces not shown} = (64-8).6 - 16.6 = 240$$

$$5X5: \text{faces not shown} = (125-27).6 - 25.6 = 438$$

(6 marks)

$$\begin{aligned} \text{(iii) } NXN: \text{faces not shown} &= (N^3 - (N-2)^3).6 - N^2.6 \\ &= (N^3 - (N-2)^3 - 25).6 \\ &= (5N^2 - 12N + 8).6 \end{aligned}$$

(3 marks)

$$\begin{aligned} \text{(ii) } (5N^2 - 12N + 8).6 &\leq (5N^2 + 8).6 \\ &\leq 6N^2 \text{ for } n \geq 7 \\ &\leq c \cdot N^2 \text{ for } n \geq n_0 \text{ when } c = 6, n_0 = 7. \end{aligned}$$

(3 marks)

5. Sorting & Cryptographic

(a) For the sorting algorithm shown in figure Q5, explain clearly and concisely:

(i) 21 is the number of swaps involved in sorting the particular array shown (7 marks)

(ii) 0 is the minimum number of swaps involved in sorting a general array of size n (1 marks)

(ii) $n(n-1)/2$ is the maximum number of swaps involved in sorting a general array of size n (3 marks)

(b) Using a simple though non-trivial example in each case, explain clearly and concisely the steps involved in the following two cryptographic algorithms:

(i) Public Key Algorithms

Public-key cryptography is a form of modern cryptography which allows users to communicate securely without previously agreeing on a shared secret key. For most of the history of cryptography, a key had to be kept absolutely secret and would be agreed upon beforehand using a secure, but non-cryptographic, method; for example, a face-to-face meeting or a trusted courier. There are a number of significant practical difficulties in this approach to distributing keys. Public-key cryptography was invented to address these drawbacks — with public-key cryptography, users can communicate securely over an insecure channel without having to agree upon a key beforehand. Public-key algorithms typically use a pair of two related keys — one key is private and must be kept secret, while the other is made public and can be widely distributed; it should not be possible to deduce one key of a pair given the other. The terminology of "public-key cryptography" derives from the idea of making part of the key public information. The term asymmetric-key cryptography is also used because not all parties hold the same information. Some public-key algorithms operate a little differently, and use other methods to enable parties to agree on secret keys without having previously exchanged key material.

(ii) Secret Key Algorithms (7 marks)

A secret key algorithm (sometimes called a symmetric algorithm) is a cryptographic algorithm that uses the same key to encrypt and decrypt data. The best known algorithm is the U.S. Department of Defense's Data Encryption Standard (DES). DES, which was developed at IBM in 1977, was thought to be so difficult to break that the U.S. government restricted its exportation. A very simple example of how a secret key algorithm might work might be substituting the letter in the alphabet prior to the target letter for each one in a message. The resulting text - "gdkkn," for example - would make no sense to someone who didn't know the algorithm used (x-1), but would be easily understood by the parties involved in the exchange as "hello." The problem with secret or symmetric keys is how to securely get the secret keys to each end of the exchange and keep them secure after that. For this reason, an asymmetric key system is now often used that is known as the public key infrastructure (PKI).

11. GENERAL BOOKS ON ALGORITHMS

1. Introduction to Algorithms, Second Edition
by Thomas H. Cormen, et al (Hardcover - September 1, 2001)
2. Algorithm Design
by Jon Kleinberg, Éva Tardos (Hardcover - March 16, 2005)
3. Data Structures and Algorithm Analysis in C++ (2nd Edition)
by Mark Allen Weiss
4. Introduction to the Design and Analysis of Algorithms
by Anany V. Levitin (Hardcover - October 30, 2002)
5. An Introduction to Bioinformatics Algorithms (Computational Molecular Biology)
by Neil C. Jones, Pavel A. Pevzner (Hardcover - August 1, 2004)
6. Data Structures and Algorithms in Java (2nd Edition)
by Robert Lafore (Hardcover - November 6, 2002)
10. Algorithms in C++
by Robert Sedgewick (2003)

12. LIST OF WELL-KNOWN ALGORITHMS

3D Rotation	Dda	Intersection	Prim's
(a,b)-tree	De Boor's algorithm	Ip routing	Priority Scheduling
Ac-3 algorithm	Deflate	Iterative closest point	Producer Consumer
Adam7 algorithm	Dekker's algorithm	Iterative reconstruction	Prosthaphaeresis
Aho-Corasick	Delaunay triangulation	Itoh-Tsujii inversion	Pseudo-lru
Aks primality test	Depth-first search	Johnson's	Quick sort
Alias method	Deutsch-Jozsa	JPeg	Rabin-Karp
Amortized analysis	Dijkstra's	Julia set	Radix sort
Appel's	Dijkstra-Scholten algorithm	Knuth-Morris-Pratt	Rainflow-counting
Astronomical algorithm	Doomsday	Knuth-Bendix completion	Ransac
Automatic distillation	Double dabble	Knuth's	Rasterization
Autonomy-oriented	Dpll algorithm	Kruskal's	Reconstruction
Bach's algorithm	E-net	Kuwaiti algorithm	RedBlack Tree
Backpropagation	Election	Lamport's Bakery	Resource Request
Baeza-Yates-Gonnet	Elevator	Lanczos algorithm	Rete
Banker's	Elser difference-map	Las Vegas algorithm	Ricart-Agrawala
Baum-Welch algorithm	Exponential backoff	Leaky bucket	Richardson-Lucy
Belief propagation	Euclid	Lempel-ziv	Risch
Bellman-Ford	Fast folding	Lenstra-Lenstra-Lovász	Roam
Bézier curve	Fast Fourier Transform	Lattice basis reduction	Robinson-Schensted
Binary search	Fastica	Linear search	Round Robin Scheduling
Binary search	Featherstone's	Lloyd's algorithm	Rsync
Binary Tree	Felicitic calculus	Lock-free	Schreier vector
Bitap	Ferguson-Forcade	Maekawa's	Schreier-Sims
Blowfish encryption	Fibonacci coding	Magic Number	Schwartz-Zippel
Bogosort	First fit algorithm	Mamf	Scoreboarding
Borůvka's	Floodfill	Marzullo's	Selection sort
Boundary fill	Flowchart	Maze generation	Semidefinite embedding
Breadth-first search	Floyd-Warshall	Memory bound function	Sieve's
Brent-Salamin algorithm	Fnn algorithm	Mental poker	Shell sort
Bresenham's	Ford-Fulkerson	Merge sort	Shor's
British Museum algorithm	Fractal	Midpoint circle	Simplex
Bubble sort	Franklin	Midpoint Line	Slow sort
Bucket sort	Freivald's algorithm	Minimum degree	Snapshot
Bully	Gauss elimination	Miser	Soft output Viterbi
Butcher's	Gauss-Jordan	Monte carlo	algorithm
Cayley-Purser	Generic cell rate	Mu-law	Spigot algorithm
Christofides	Genetic	Multivariate division	Stable algorithm
Chung Kwei	Gerchberg-Saxton	Nagle's	Strassen's
Circular reference	God's	Nearest neighbour	Sutherland-Hodgman
Circular shift	Graph exploration	Newton's Fractal	Tcp congestion control
Closed hashing	Graphplan	Newton's root finding	Tea
Cocktailshaker sort	Great big lock	Non-blocking	Todd-Coxeter
Cohen-Sutherland	Hash join	Nonlinear dimensionality	Token bucket
Collatz conjecture	Hashing	Null-move heuristic	Tomasulo
Color quantization	Heap sort	Open hashing	Trabb-Pardo-Knuth
Competitive analysis	Hits	Painter's	Tree sort
Control variate	Hshieh-hammond	Parallel	Trim
Cordic	Huang's	Particle-in-cell	Ub-tree
Cuthill-Mckee	Huffman	Paxos	Virtual Memory Clock
Dancing links	Hungarian	Peterson's	Wait-free
Data bases	Hyphenation	Pohlig-Hellman	Warnock's
Data encryption	In-place	Pollard's rho	Warshall's
Davis-Putnam algorithm	Insertion sort	Polly	Z-buffer