Algorithm

[Abu Ja'far Mohammed ibn Musa al Khowarizmi]

☛ **Algorithm is a clearly specified set of simple instructions to be followed to solve a problem.**

Definition

Algorithm is a finite set of instructions that is followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:

1. **INPUT:** Zero or more quantities are exactly supplied.
2. **OUTPUT:** At least one quantity is produced.
3. **DEFINITENESS:** Each instruction is clear and unambiguous.
4. **FINITENESS:** If one trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.
5. **EFFECTIVENESS:** Every instr on must very basic so that an individual using pencil a paper can carry it out, in principle.

NOTES:

☛ An algorithm is composed of a finite set of steps, each of which may require one or more operations.

☛ Algorithms that are definite and effective are also called COMPUTATIONAL PROCEDURE.

Ex: Operating System

☛ A PROGRAM is the expression of an algorithm in a programming language.

[3] Greedy algorithm

- An algorithm which always takes the best immediate, or local, solution while finding an answer.
- Greedy algorithms will always find the overall, or globally, optimal solution for some optimization problems, but may find less-than optimal solutions for some instances of other problems.
- These algorithms are very easy to design for optimization problems.

e.g., determining Huffman codes, minimal spanning tree, integer knapsack, single source shortest path.

[4] Backtrack algorithm

- An algorithm technique to find solutions by trying one of several choices.
- If the choice proves incorrect, computation backtracks or restarts at the point of choice and tries another choice.
- It is often convenient to maintain choice points and alternate choices using recursion.
- The usual way to implement a backtracking algorithm is to write a function or procedure, which traverses the solution space e.g., game tree.

[5] Approximation algorithm

- An algorithm to solve an optimi[illegible] t runs in polynomial time in the length of [illegible] outputs a solution that is guaranteed to be close to the optimal solution.

[3] Greedy algorithm

- An algorithm which always takes the best immediate, or local, solution while finding an answer.
- Greedy algorithms will always find the overall, or globally, optimal solution for some optimization problems, but may find less-than optimal solutions for some instances of other problems.
- These algorithms are very easy to design for optimization problems.

e.g., determining Huffman codes, minimal spanning tree, integer knapsack, single source shortest path.

[4] Backtrack algorithm

- An algorithm technique to find solutions by trying one of several choices.
- If the choice proves incorrect, computation backtracks or restarts at the point of choice and tries another choice.
- It is often convenient to maintain choice points and alternate choices using recursion.
- The usual way to implement a backtracking algorithm is to write a function or procedure, which traverses the solution space e.g., game tree.

[5] Approximation algorithm

- An algorithm to solve an optimization problem that runs in polynomial time in the length of the input and outputs a solution that is guaranteed to be close to the optimal solution.

[6] Recursive algorithm

A recursive algorithm is one which calls itself to solve "smaller" versions of an input problem.

[7] Randomized algorithm

- An algorithm is a randomized algorithm if some of the decisions made in the algorithm depend on the output of a RANDOMIZER.
- A randomized algorithm is called Las Vegas algorithm if it always produces the same correct output for the same input.
- If output differs from run to run for the same input, we call it MONTE CARLO ALGORITHM.
- In case of MONTE CARLO ALGORITHM probability of an incorrect answer is low.

[8] Parallel algorithm

A parallel algorithm is an algorithm that has been specifically written for execution on a computer with two or more processors (i.e., a parallel computer).

[9] Serial Algorithm

A serial algorithm is an algorithm that has -been specifically written for execution on a computer with just one processor (i.e., a serial computer).

[10] Genetic Algorithm

- The continuing price/performance improvements of computational systems have made it attractive.
- It is a effective solution of problems of optimizations and was introduced by J.H. Holland in 1975.
- It manipulates bit strings analogously to DNA evolution.
- It has been developed to general computing model of resolving problems of optimization by simulating evolving process of the nature.
- To use a genetic algorithm, one must represent a solution to the problem as a genome (or chromosome). The GA then creates a population and applies genetic operators such as mutation and cross over to evolve the solutions in order to find the best one(s).

☛ DNA Computing

- The first breakthrough in DNA computing came in 1994 via Leonard Adleman, a professor at the University of Southern California.
- He first used DNA to solve the "traveling salesman" problem, which finds a path for a salesman to visit customers in every listed city in the shortest distance possible.

PERFORMANCE ANALYSIS OF ALGORITHM

[The amount of memory and time needed to run a program]

APPROACHES:

- Performance analysis
  (ANALYTICAL METHOD) or THEORETICAL APPROACH

- Performance measurement
  (THROUGH EXPERIMENTS) or EMPIRICAL APPROACH

SPACE COMPLEXITY

The space complexity of an algorithm is the amount of memory it needs to run to completion.

Why space-complexity?

- **If the program is to be run on a multi-user computer system, then it is required to specify the amount of memory to be allocated to the program.**

- **For any computer system a user is interested to know in advance whether or not sufficient memory is available to run the program.**

- **A problem might have several solution with different space requirements, users prefers a smaller compiler generated code (interpreter), that leaves the user with more memory for other task.**

- **The space complexity is used to estimate the size of the largest problem a program can solve.**

PERFORMANCE ANALYSIS OF ALGORITHM

[The amount of memory and time needed to run a program]

APPROACHES:

- **Performance analysis**
  (ANALYTICAL METHOD) or THEORETICAL APPROACH

- **Performance measurement**
  (THROUGH EXPERIMENTS) or EMPIRICAL APPROACH

SPACE COMPLEXITY

The space complexity of an algorithm is the amount of memory it needs to run to completion.

Why space-complexity?

- **If the program is to be run on a multi-user computer system, then it is required to specify the amount of memory to be allocated to the program.**

- **For any computer system a user is interested to know in advance whether or not sufficient memory is available to run the program.**

- **A problem might have several solution with different space requirements, users prefers a smaller compiler generated code (interpreter), that leaves the user with more memory for other task.**

- **The space complexity is used to estimate the size of the largest problem a program can solve.**

Example: Circuit simulation program requires ***280K + 10(c+w)***bytes with *C* components and ***w*** wires. A circuit with $c + w \leq 36K$ can be simulated.

SPACE REQUIREMENTS OF AN ALGORITHM [PROGRAM]

[1] FIXED PART OF THE ALGORITHM

A fixed part of the algorithm is the parts/components that has independent characteristics from of input/output.

Ins

☛ **Instruction space [space for code]**

- **Compiler used to compile the program into machine code.**
- **The compiler options in effect at the time of completion**
- **The target computer.**

☛ **Data Space [space for simple variable]**

☛ **Aggregate [Fixed size component variable]**

☛ **Space for constant and so on.**

[2] A VARIABLE PART OF ALGORITHM

A variable part of an algorithm is consists of the space needed by component variables whose size depend on the particular problem ins[illegible]ng solved.

☛ **Space needed by [illegible] variables [Based on ins[illegible]acteristic]**

☛ **Environment STACK space**

S (P) denotes space requirement of any algorithm P

S (P) = c + S_P ; Where C is a constant [Fixed part] & S_P is a function of instance Characteristics

Example: Circuit simulation program requires ***280K + 10(c+w)***bytes with ***C*** components and ***w*** wires. A circuit with $c + w \leq 36K$ can be simulated.

SPACE REQUIREMENTS OF AN ALGORITHM [PROGRAM]

[1] FIXED PART OF THE ALGORITHM

A fixed part of the algorithm is the parts/components that has independent characteristics from of input/output.

Ins

☛ **Instruction space** [space for code]

- **Compiler used to compile the program into machine code.**
- **The compiler options in effect at the time of completion**
- **The target computer.**

☛**Data Space** [space for simple variable]

☛**Aggregate** [Fixed size component variable]

☛**Space for constant** and

[2] A VARIABLE PART OF ALGORITHM

A variable part of an algorithm is consists of the space needed by component variables whose size depend on the particular problem instance being solved.

☛ **Space needed by referenced variables**
[Based on instance characteristic]

☛ **Environment STACK space**

S (P) denotes space requirement of any algorithm P

$S(P) = c + S_P$; Where C is a constant [Fixed part] & S_P is a function of instance Characteristics

DATA SPACE

Space allocated to simple variables in BC++(16bit)

Type	Space (bytes)	Range
char	1	-128 to 127
unsigned char	1	0 to 255
short	2	-32,768 to 32,767
int	2	-32,768 to 32,767
unsigned int	2	0 to 65,535
long	4	-2^{31} to $2^{31}-1$
unsigned long	4	0 to $2^{32}-1$
float	4	+- 3.4E+-38
double	8	+- 1.7E+-308
long double	10	3.4E-4932- to 1.1E+4932
pointer	2	(near, _cs, _ds, _es, _ss pointer
pointer	4	(far, huge pointers)

Example:

```
double    a [100]
int   maze[rows]  [cols] ;
```

TIME COMPLEXITY

Time Complexity of an algorithm is the amount of computer time it needs to run to completion

Why time-complexity?

- **Some computer requires the user to provide the upper limit on the amount of time the program will run.**
- **The program might be designed to provide a satisfactory real time result.**
- **Most feasible solution selection is based on the expected performance difference among these solutions.**

T (P): Time taken by a program P
= Sum of the compile time plus execution (run) time of P

- **Compile time does not depend on the instance characteristic**
- **Only run time of an algorithm is considered for calculation of time complexity**

Let t_P(instance characteristics] and defined as t_P = number of operations (micro operations) required to execute the program.

$T_P(n) = C_a\ ADD(n) + C_s\ SUB($ [illegible] $UL(n) + C_d\ DIV(n) + \ldots$

- **ADD, SUB, MUL and DIV .... Are the functions whose value are number of operations that are to be performed for the code of P.**
- **C_a, C_s, C_m, C_d, and so on denote the time required to carryout ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION and so on.**

- **A PROGRAM is the expression of an algorithm in a programming language.**

Active area of Algorithm research

1. **How to devise algorithm:**
   - **The act of creating an algorithm**
   - **Study of Design techniques**
2. **How to express algorithm [programming style]**
   - **Structure programming**
3. **How to validate algorithm**
   - **Algorithm (program) proving**
   - **Algorithm (program) verification**
4. **How to analyze algorithm**
5. **How to test a program [2phase process]**
   - **Debugging: Is the process of executing programs on sample data sets to determine if faulty results occur and, if so, to correct them.**
   - **Profiling: The process of executing a correct program on data set and measuring the time and space it tales to c[illegible]e the results possible inputs**

- **How to estimate the time required for a program?**
- **How to reduce the running time of a program?**

TIME COMPLEXITY

Time Complexity of an algorithm is the amount of computer time it needs to run to completion

Why time-complexity?

- **Some computer requires the user to provide the upper limit on the amount of time the program will run.**
- **The program might be designed to provide a satisfactory real time result.**
- **Most feasible solution selection is based on the expected performance difference among these solutions.**

T (P): Time taken by a program P
= Sum of the compile time plus ex[illegible]on (run) time of P

- Compile time does not depend on the instance characteristic
- Only run time of an algorithm is considered for calculation of time complexity

Let t_P(instance characteristics] and defined as t_P = number of operations (micro operations) required to execute the program.

$$T_P(n) = C_a\ ADD(n) + C_s\ SUB(n) + C_m\ MUL(n) + C_d\ DIV(n) + \ldots$$

- ADD, SUB, MUL and DIV .... Are the functions whose value are number of operations that are to be performed for the code of P.
- C_a, C_s, C_m, C_d, and so on denote the time required to carryout ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION and so on.

☛ **A PROGRAM is the expression of an algorithm in a programming language.**

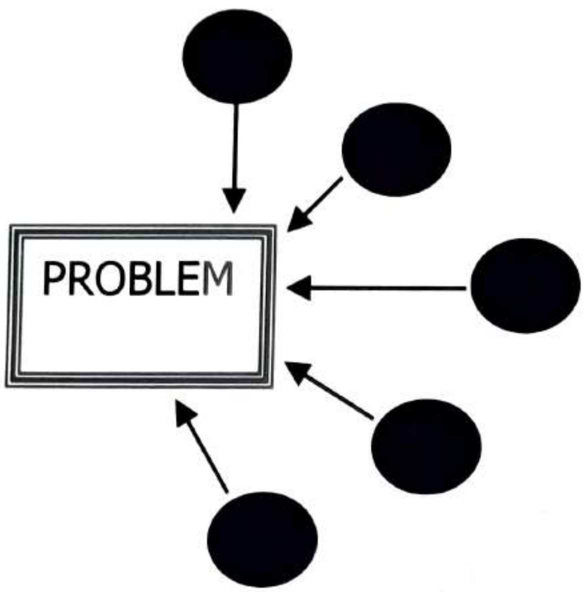
Active area of Algorithm research

1. **How to devise algorithm:**
   - ☛ **The act of creating an algorithm**
   - ☛ **Study of Design techniques**
2. **How to express algorithm [programming style]**
   - ☛ **Structure programming**
3. **How to validate algorithm**
   - ☛ **Algorithm (program) proving**
   - ☛ **Algorithm (program) verification**
4. **How to analyze algorithm**
5. **How to test a program [2, [illegible] process]**
   - ☛ **Debugging: Is the process of executing programs on sample data sets to determine if faulty results occur and, if so, to correct them.**
   - ☛ **Profiling: The process of executing a correct program on data set and measuring the time and space it tales to compute the results possible inputs**

➡ **How to estimate the time required for a program?**

➡ **How to reduce the running time of a program?**

Performance Analysis of Algorithm



Several suitable algorithms

Which of several algorithms is preferable?

1. **A priori estimates [THEORETICAL APPROACH]**
2. **A posteriori testing [EMPIRICAL APPROACH]**

☛ **A PRIORI ESTIMATE determines mathematically the quantity of resources that needed by each algorithm as a function of the size of the instances considered.**

☛ **The resources are :**

- **computing Time**
- **Storage space**

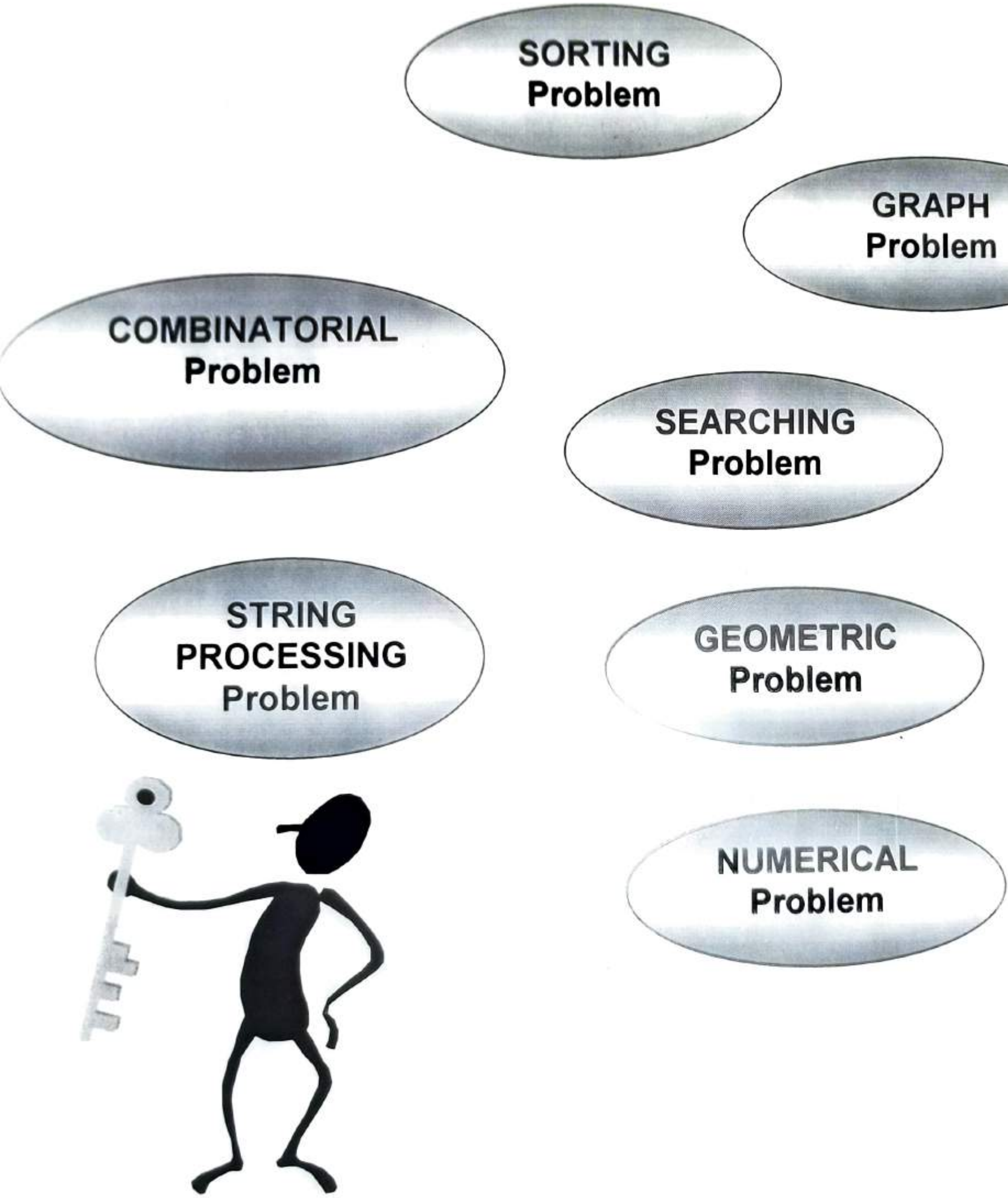
ALGORITHM Design Methods

Divide and Conquer	Binary Search
	Finding the Maximum and Minimum
	Merge Sort
	Quicksort
	Selection Sort
	Strassen's Matrix Multiplication
	Closest pair of points
	Convex Hull
Greedy Method	Knapsack Problem
	Tree Vertex Splitting
	Job Sequencing with Deadlines
	Minimum-cost spanning trees (Kruskal's, Prime's and Sollin's Algorithm)
	Topological sorting
	Optimal storage on Tape
Dynamic programming	Multistage graph
	All pair shortest paths
	Single source shortest paths
	Image compression
	0/1 knapsack problem
	Noncrossing subset of Nets
	Optimal binary search trees
	The traveling salesperson problem
	Component folding
Search and Traversal	Breadth First search and Traversal
	Depth First search and Traversal
	DFS and BFS spanning
	Connected components and spanning trees
Backtracking	0/1 knapsack problem
	The traveling salesperson problem
	8-queens problem
	Graph coloring

Branch and Bound	Least cost search FIFO branch and bound LC branch and bound 0/1 knapsack problem The traveling salesperson problem
NP- Complete and NP- Hard problems	Basic concepts Nondeterministic algorithm The classes NP-hard and NP-complete Simplified NP-hard problems

PROBLEM TO BE SOLVED

IMPORTANT PROBLEM TYPE



Algorithm Design Technique

☛ **In a POSTERIORI APPROACH, an algorithm is subjected for execution on different instances after implemented using programming language with help of computer.**

☛ **SIZE OF INSTANCE Corresponds to the number of bits needed to represent the instance on a computer using some coding scheme.**

ALGORITHMIC TECHNIQUES

[1] Divide and conquer algorithm

The divide and conquer technique:

- Divides a given problem into smaller instances of the same problem
- Solves the smaller problems and then combine their solutions to solve the given problem.
- When the smaller problems are not independent (i.e., when two or more of them are identical or share identical sub-problems), the divide-and-conquer technique may become INEFFICIENT for it does more work than necessary by having to solve the same sub-problems repeatedly

[2] Dynamic programming algorithm

- A dynamic programming algorithm solves every smaller problems (sub-problems) just once and save the solution in a TABLE.
- The solution will be retrieved when the same sub-problem is encountered later on, thereby avoiding the redundant work of recomputing the same solution. e.g., all pairs shortest path.