Syllabus - 3 credits

1 - Basic concepts of Algorithms

1.1 - Fundamentals of Algorithmic problem solving
1.2 - Important Problem types
1.3 - Fundamentals of the analysis of algorithm efficiency
1.4 - Analysis Framework
1.5 - Asymptotic Notations
1.6 - Asymptotic Notations & Basic Efficiency Classes
1.7 - Recurrence relations
1.8 - Methods for solving recurrence relations

2 - Mathematical Analysis of Algorithms.

2.1 - Mathematical Analysis of Non-recursive Algorithms
2.2 - Non-recursive Algorithms and examples
2.3 - Mathematical Analysis of Recursive Algorithms
2.4 - Fibonacci numbers
2.5 - Empirical Analysis of Algorithms.

3 - Brute Force & Divide & conquer techniques

3.1 - Selection sort
3.2 - Bubble sort
3.3 - Brute-force string matching
3.4 - Merge sort
3.5 - Multiplication of Two-n-Bit Numbers
3.6 - Quick sort

3.7 - Binary Search
3.8 - Binary tree traversal

4 - Algorithm Design paradigm

4.1 - Decrease and conquer Technique : Insertion Sort
4.2 - Depth first search & Breadth first search
4.3 - Transform & conquer Technique : Presorti
4.4 - Dynamic programming : Computing a Binomial coefficient.
4.5 - Warshall's & Floyd's Algorithm
4.6 - The knapsack problem and Memory funct
4.7 - optimal Binary Search trees
4.8 - Greedy Technique : Huffman trees

5 - NP Hard and NP-complete problems

5.1 - P and NP problems
5.2 - NP complete problems
5.3 - Backtracking : N-Queen's problem
5.4 - Hamiltonian circuit problem
5.5 - Branch and Bound Techniques
5.6 - Traveling salesman problem.

19/2/24

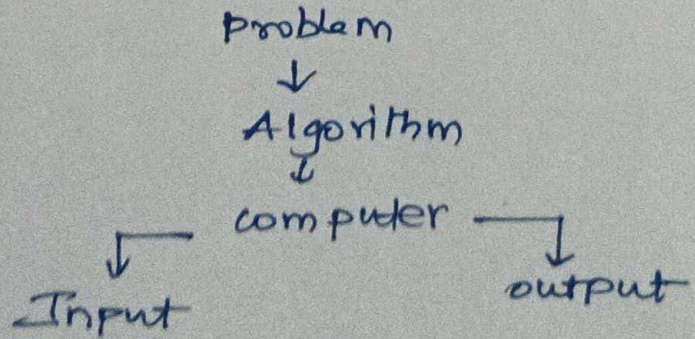
Design & Analysis of Algorithms

Algorithm

An algorithm is a sequence of unambiguous Instructions for solving a problem for obtaining a required output for any legitimate input in a finite amount of time.

* The nonambiguity requirement for each step of an algorithm cannot be compromized.
* The range of inputs for which an algorithm works as to be specified carefully.
* The same algorithm can be represented in several different ways.
* Several algorithms for solving the same problem may exist.
* Algorithm for the same problem can be based on very different ideas and can solve the problem with dramatically different speeds.

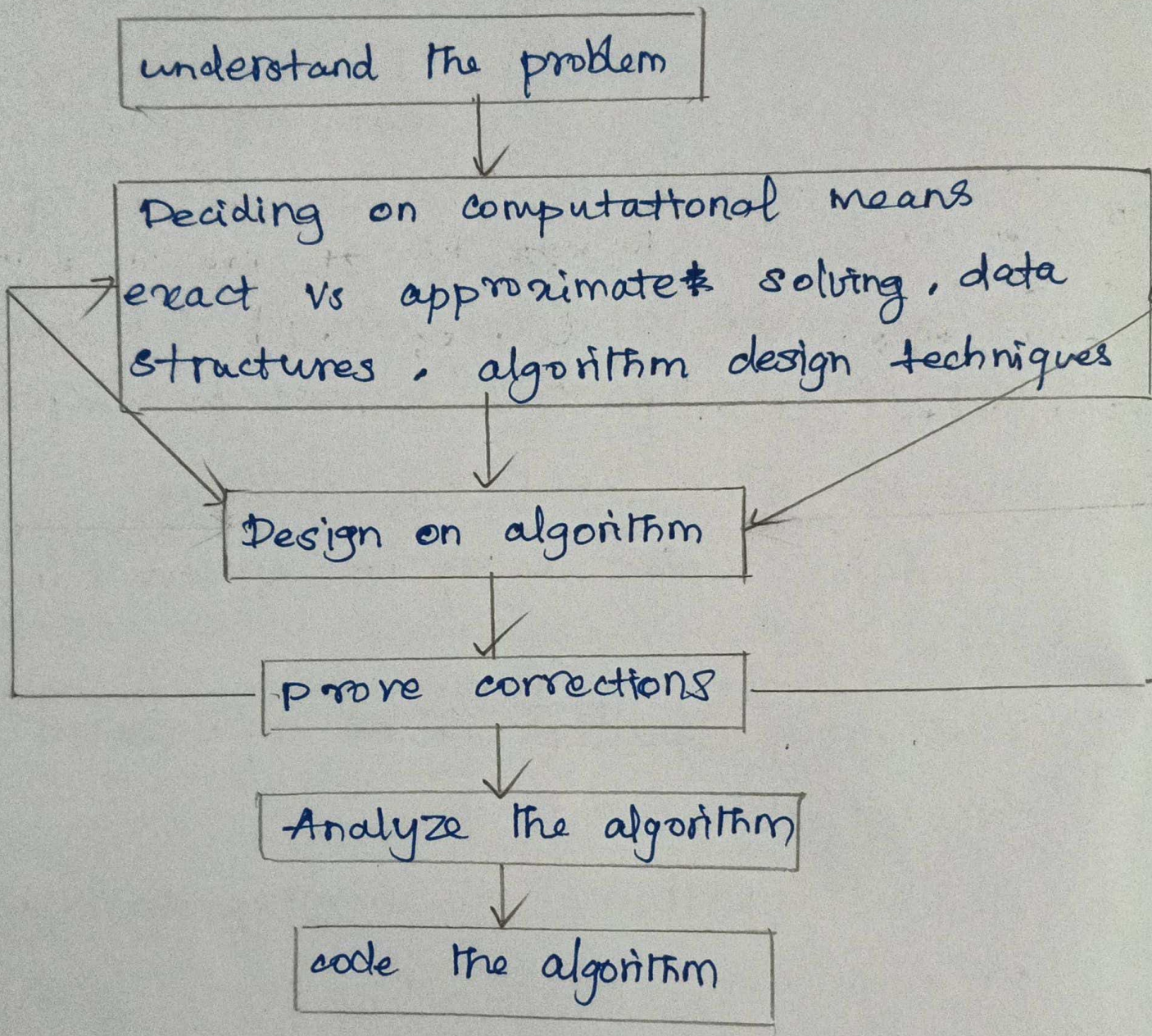
Notation of Algorithm



4/3/24

Fundamentals of Algorithmic problem solving

6m



understanding the problem

An input to an algorithm specifies the instance of the problem the algorithm solves.

Ascertaining the capabilities of computational devices

RAM: * Sequential algorithm
* Parallel algorithm

Exact Vs Approximate Solving

If exact ans comes it is exact algo
eg: $\sqrt{4} = 2$

If approximate ans comes it is AA
eg: $\sqrt{2} = 1.414...$

* square root
* nonlinear equations
* Evaluating definite integrals

Deciding on Appropriate Data Structures

* Sorting, * Searching

Data Structures + Algorithms
↓
programs

In the new world of object oriented programing data structures remains important for both design and analysis of algorithm.

Algorithmic design technique

q) An algorithm design technique or strategy of paradigm is a general approach to solving problems algorithmictyally that is

---

q: 1) how do you design an algorithm to solve a given problem?

a) design an algorithm to compute the area and circumference of an circle
pseudocode, fc, algo, code

applicable to a variety of problems from different areas of computing.)

Methods of specifying an algorithm

2m * pseudocode → It is a mixture of natural language and programming language like constructs. A pseudocode is usually more precise than a natural languge and its usage of an each more succint algorithm description

* flowchart → A method of expressing an algorithm by collection of connected Geometric shapes containing descriptions of the algorithm steps

Providing an algorithm's correction yields the

* To prove that the algorithm is a required result for every legitimate input in a finite amount of time.

* A common technique for proving correctness is to use mathematical induction because an algorithms iteration provide a natural sequence of steps needed for such proof.

Analyzing the algorithm

speed, space

3. What are the important problem types with eg.

Analyze → time efficiency, Space efficiency

Time efficiency → how fast the algorithm runs

Space efficiency → how much exact memory the algorithm needs

simplicity, generality

8/3

Important problem types

* Sorting [key] (Shell, Quick, Bubble, Selection, Insertion, Merge, Heap, Radix)
* Searching [key] (binary, linear, heap)
* String processing (is a sequence of char)
* Graph problems
* combinational problems
* Geometric problems
* Numerical problems

String processing is the applications dealing with non numerical data has intencified the interest of researchers & computing practionaters in String handling algorithm.

Graph problem is a problem of finding shortest toure through n cities that visits every city exactly once

↳ salesman problem.

Graph coloring problem

It seeds assign the smallest no. of colors to the vertexes of a graph. So that known to the adjacent vertices are same color. This prblm comes in several such as event scheduling. If the events are represented by vertices that are ~~some~~ connected by the edge if and only if the corresponding events can't be scheduled at the same time. The solu to the graph coloring problem yields an optimal solution.

Combinational problems

these are the problem that as explicitly or implicitly to find a combinational object such as permutation, combinations or a subset that satisfies certain constraints a desire combinational object may also be required to have some additional property such as maximum value or minimum cost.

Fundamentals of Analysis of Algorithm Efficiency

Analysis Framework

Time efficiency, Space efficiency

Measuring the input size

eg: It takes larger to short larger array.
multiply larger matrix therefore it is logical to investicate an algorithm efficiency as a function of some particular parameter 'n' indicate the algorithm input size

eg: i) large array sort
ii) mul of matrix
iii) Evaluating polynomial

units of measuring runtime:
Milliseconds, seconds, microseconds, nano seconds

Drawbacks:

* defedencies on speed of particular computer on the quality of prgm implementing the Algorithm.
* Difficulty in calculating the accurate runtime of an algorithm.
two approaches:
count the no. of time each of the algorithm operation is executed but it is exessively difficult.

18/3

To identify the most important operation of the algorithm called basic operations.

The operations contribute the most of the total running time and compute the no. of times the basic operations is executed.

eg: sorting algorithm.

- Matrix Multiplication
- polynomial evaluation.

established framework for the analysis of an algorithms time efficiency. measuring it by counting the no. of times of the algorithms basic operations is executed of the I/P of size 'n'

COP: execution time of an algorithm basic operation $C(n)$ - no. of times this operation need to be performed.

$T(n)$ → RunTime of the program

$$T(n) \approx cop(n)$$

Assuming that $c(n) = \frac{1}{2} n(n-1)$

$$C(n) = \frac{1}{2} n^2 - \frac{1}{2} n$$

$$\approx \frac{1}{2} n^2$$

$$\frac{T(2n)}{T(n)} = \frac{cops(2n)}{cops(n)} = \frac{\frac{1}{2}(2n)^2}{\frac{1}{2}n^2} = 4$$

The efficiency analysis framework ignores multiplicative constants and concentrate on order of growth.

order of growth.

n	$\log_2 n$	n	$n\log n$	n^2	n^3	2^n	$n!$
10	3.3	10	3.3×10	10^2	10^3	10^3	3.6×10^6
10^2	6.6	10^2	6.6×10^2	10^4	10^6	1.3×10^{30}	9.3×10^{157}
10^3	10	10^3		10^6	10^9		
10^4	13	10^4		10^8	10^{12}		
10^5	17	10^5		10^{10}	10^{14}		
10^6	20	10^6		10^{12}			

① worst case, best case, average case

sequential search (A[0..n-1], k)

I/P : array A[0 ... n-1] and search key-k

O/P : The index of the 1st element of A that matches k or -1 if no match

```
i ← 0
while i < n and A[i] ≠ K do
    i ← i + 1;
if i < n return i;
else return -1
```

Eg : A[] = {7, 8, 9, 2, 4} key = 7, n = 5

time complexity T(C) = 1

① Write algorithm and give example of best, worst, average case.

Average case efficiency :

* standard assumptions

i) Probability of a successful search ($0 \le p$

ii) probability of 1st match occurs in ith position

$$c_{avg}(n) = \left[1 + \frac{p}{n} + 2\frac{p}{n} + \cdots i\frac{p}{n} + \cdots n\frac{p}{n}\right] + n(1-p)$$

$$= \frac{p}{n}[1 + 2 \cdots n] + n(1-p)$$

$$= \frac{p}{n} * \frac{n(n+1)}{2} + n(1-$$

~~If i < n set~~

$$c_{avg}(n) = \frac{p(n+1)}{2} + n(1-p)$$

Best case = $p = 1$, $\frac{n+1}{2} + 0 = \frac{n+1}{2}$ $p = 0$ worst case

Amortized Efficiency

It applies not to a single run of an algorithm but rather to a sequence of operations performed on the same data structure.

A single operations can be expensive but the total time for a entire sequence of n search operations is always significantly better than worst case efficiency

Asymptotic Notations.

big oh (O)
big omega (Ω)
big theta (Θ)

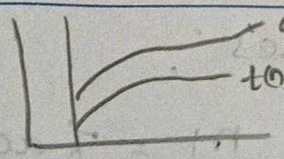
o(g(n)) is a set of all functions with a smaller or same order of growth.

big oh(O) notation or O-notation

A function t(n) is said to be order of g(n) denoted $t(n) \in O(g(n))$

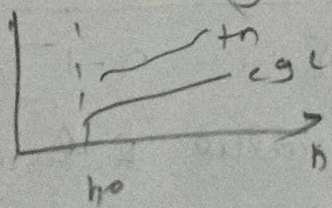
If t(n) is bounded above by some constant multiple of g(n) for large n.

i.e if there exist some positive constant c and some not negative n_0 such that $t(n) \leq cg(n)$ for all $n \geq n_0$



A function t(n) is said to be $\Omega g(n)$ denotes $t(n) \in \Omega(g(n))$ if t(n) is bounded below by some positive constant multiple of g(n) for all large n (ie) if there exist some the constant and some non-re Int such that $t(n) \geq cg(n)$ (for all $n \geq n_0$)

$\Omega(g(n))$ $t(n) \geq cg(n)$

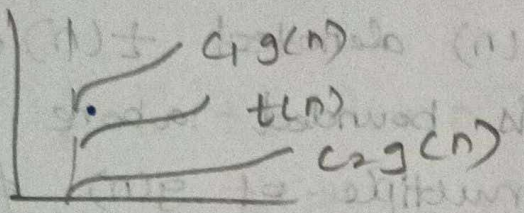


A function $t(n)$ is said to be in Theta g

$t(n) \in \Theta(g(n))$

if $t(n)$ is bounded both above & below by some +ve constant multiples of $g(n)$ for all large n that is if there exist some positive constant c_1 & c_2 and some non-ve Int n_0 such that

$c_2 g(n) \leq t(n) \leq c_1 g(n)$, for all $n \geq$



- O notation : $t(n) \leq c\,g(n)$
- Ω " : $t(n) \geq c\,g(n)$
- Θ " : $c_2 g(n) \leq t(n) \leq c_1 (g(n))$

Basic efficiency classes :

1 - constant
log n - logarithm
n - linear
n log n - n log n
n^2 - quadratic
n^3 - cubic
2^n - exponential
$n!$ = factorial

Mathematical Analysis of non-recursive Algorithms

Algo MaxElement (A[0...n-1])
// Determines the value & largest
I/p. An array A[0...n-1] of real nos
O/p : The value of largest element in A

for i ← i to n-1 do
if A[i] > maxvel
maxval ← A[i]
return maxvalue

$$C(n) = \sum_{i=1}^{n-1} 1 = n-1 \in O(n)$$

non regursive steps

* Desined on parameters indicating the i/p
* Identify the algorithm basic operation,
* & check whether the no. of times the basic operation is executed depends only on the size of the I/P If it also depends on additional properties worst case average case & best case to the investigator.
* Setup a sum expressing no. of times the algorithms basic operation is executed.
* Using standard formula & rules of some manipulation either find a closed form formula of the count or establise its order of growth.

regursive

Ist 3 points are same:

iv) Setup a regrals relation with the appropriate intial condition for the no. of time based the basic operation is executed.

v) Solve the recurance order of growth of its solution,

eg: Towes of hanoi,

unique elements (A[0 . . . n-1])
for i ← 0 to n-2 do
  for j ← i+1 to n-1 do
    if A[i] = A[j] return false
return false.

Rules : I/p size
basic operation
c(n)
notation,

$$C(n) = \sum_{i=0}^{n-2} \left[\sum_{j=i+1}^{n-1} (1) \right]$$

$$= \sum_{i=0}^{n-2} [n-1-(i+1)+1]$$

$$= [n-1-i]$$

$$= \sum_{i=0}^{n-2} (n-1-1)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i$$

$$= (n-1) \sum_{i=0}^{n-2} - [n-2+0+1]$$

$$= (n-1)(n-2-0+1) - (n-2+1)$$

$$= (n-1)(n-1) - (n-1)$$

$$= (n^2-1) - (n-1)$$

$$\approx \frac{1}{2} n^2 \in O(n)$$

Empirical Analysis of Algorithm

1. understand the experimental purpose
2. decide on the efficiency metric to be measured and measurement unit.
3. desire the carecteristics of the I/P sample
4. Prepare a program Implementing the algorithm for the experiment.
5. generating the sample of I/P
6. run the algorithm on the sample I/P and record the data observed
7. Analyze the data o/p