[Design and Analysis of Algorithms]

Design and Analysis of Algorithms (CSc 523)

Samujjwal Bhandari
Central Department of Computer Science and Information Technology (CDCSIT)
Tribhuvan University, Kirtipur,
Kathmandu, Nepal.

Objectives of this Course

- Introduce data structures that are fundamental in computer science
- Introduce mathematical foundation for analysis of algorithms
- To know why we need design and analysis knowledge
- Techniques for algorithms analysis and their classifications
- Alternative ways of algorithm design
- Analyze some of the well known algorithms
- Design some kind of algorithm and analyze them

Course Introduction

- Mathematical tools for design and analysis of algorithms
- Data structure overviews
- Complexity analysis and recurrence relations
- Divide and conquer paradigm: quick sort, matrix multiplication, etc. and its application
- Greedy paradigm: minimum spanning trees-prim's, Kruskal's shortest path algorithm, etc. and its application
- Dynamic Programming: Floyd's algorithm, string matching problems, etc.
- Graph algorithms: BFS, DFS, Finding cycle in a graph, etc.
- Geometric algorithms: line segment intersection, point inclusion, etc.
- Introduction to NP-complete problems, cook’s theorem, approximation algorithms.

- **Textbook:** "Introduction To Algorithms", Cormen, Leiserson, Rivest, Stein.

Prerequisites

- Simple mathematical tools like recurrences, set, proof techniques, big _Oh notation, etc.
- Data structures lists, stacks, queues, graphs, trees etc.
- Programming knowledge (any one).

Evaluation Criterion

Assignments.	10%
2 mid term assessments.	10%
1 final exam.	80%

Note: No late submissions of assignments are marked.

Note: The mid term assessments will cover all the completed materials before exam day.

Algorithms

An algorithm is a finite set of computational instructions, each instruction can be executed in finite time, to perform computation or problem solving by giving some value, or set of values as input to produce some value, or set of values as output. Algorithms are not dependent on a particular machine, programming language or compilers i.e. algorithms run in same manner everywhere. So the algorithm is a mathematical object where the algorithms are assumed to be run under machine with unlimited capacity.

Examples of problems:

- You are given two numbers, how do you find the Greatest Common Divisor.
- Given an array of numbers, how do you sort them?

We need algorithms to understand the basic concepts of the Computer Science, programming. Where the computations are done and to understand the input output relation of the problem we must be able to understand the steps involved in getting output(s) from the given input(s).

You need designing concepts of the algorithms because if you only study the algorithms then you are bound to those algorithms and selection among the available algorithms. However if you have knowledge about design then you can attempt to improve the performance using different design principles.

The analysis of the algorithms gives a good insight of the algorithms under study. Analysis of algorithms tries to answer few questions like; is the algorithm correct? i.e. the algorithm generates the required result or not?, does the algorithm terminate for all the inputs under problem domain? (*More on this later*). The other issues of analysis are efficiency, optimality, etc. So knowing the different aspects of different algorithms on the similar problem domain we can choose the better algorithm for our need. This can be done by knowing the resources needed for the algorithm for its execution.

Two most important resources are the time and the space. Both of the resources are measures in terms of complexity for time instead of absolute time we consider growth rate (time complexity), similarly for space instead of absolute value growth rate of memory needed is considered (space complexity).

Algorithms Properties

Input(s)/output(s): There must be some inputs from the standard set of inputs and an algorithm's execution must produce outputs(s).
Definiteness: Each step must be clear and unambiguous.
Finiteness: Algorithms must terminate after finite time or steps.
Correctness: Correct set of output values must be produced from the each set of inputs.
Effectiveness: Each step must be carried out in finite time.

Here we deal with correctness and finiteness.

Expressing Algorithms

There are many ways of expressing algorithms; the order of ease of expression is natural language, pseudo code and real programming language syntax. In this course I intermix the natural language and pseudo code convention.

Random Access Machine Model

This RAM model is the base model for our study of design and analysis of algorithms to have design and analysis in machine independent scenario. In this model each basic operations (+, -) takes 1 step, loops and subroutines are not basic operations. Each memory reference is 1 step. We measure run time of algorithm by counting the steps.

Best, Worst and Average case

Best case complexity gives lower bound on the running time of the algorithm for any instance of input(s). This indicates that the algorithm can never have lower running time than best case for particular class of problems.
Worst case complexity gives upper bound on the running time of the algorithm for all the instances of the input(s). This insures that no input can overcome the running time limit posed by worst case complexity.
Average case complexity gives average number of steps required on any instance(s) of the input(s).

In our study we concentrate on worst case complexity only.

Example 1: Fibonacci Numbers

Input: *n*
Output: nth Fibonacci number.
Algorithm: assume *a* as first(previous) and *b* as second(current) numbers

```
fib(n)
{
        a = 0, b= 1, f=1 ;
        for(i = 2 ; i <=n ; i++)
        {
                f = a+b ;
                a=b ;
                b=f ;
        }
        return f ;
}
```

Correctness

The algorithm adds previous and current values i.e. *a* and *b*, and produces nth Fibonacci number (n>0) after n-2 iterations. So at each step the generated value will be correct. The algorithm terminates after n-2 iterations.

Efficiency

Time Complexity: The algorithm above iterates up to n-2 times, so time complexity is O(n).

Space Complexity: The space complexity is constant i.e. O(1).

Next is the recursive version of fib(n) as rfib(n)

```
rfib(n)
{
        if(n<2)
                return 1 ;
        else
                return( rfib(n-2)+rfib(n-1))
}
```

Correctness:

The above algorithm rfib(n),for n>0 produces correct output for n=1, 2, ….up to n and recursive call do obtain rfib(1) and rfib(2) as terminating condition i.e. every rfib(n) is recursively computed unless n becomes 0 or 1.

Efficiency

Time Complexity: In the above algorithm T(0) = t(1) = 2 and T(n) = T(n-2)+T(n-1) +

3.By induction we can prove $T(n) > T(n-1)$ holds for all $n>=1$, and $T(n) > T(n-2)$. Using this show that $T(n)$ is at least $2^{n/2}$ i.e. $T(n) = \Omega(2^{n/2})$.

Space Complexity: The recursive call needs stack as temporary storage and that call to stack during execution of the above algorithm never exceeds n, so the space complexity is O(n).

Exercises

1. You are given 12 balls of identical shape all of which, but one, are of same weight. You are also given a balance. Find the ball of the irregular weight and its type of irregularity (heavier or lighter), by using balance only 4 times.
2. Consider the ***searching problem***:

   Input: A sequence of n numbers $A = <a_1, a_2, \dots, a_n>$ and a value **v.**

   Output: An index i such that **v**=A[i] or the special value NIL if **v** doesn't appear in A.

   Write pseudo code for ***linear search***, which scans through the sequence, looking for **v**.
3. Explore the word algorithm and its history.