Algorithm Design and Analysis

Lecture - 1

Reference books:

i) Introduction to Algorithms (Third Edition) by Thomas H.Cormen
ii) Fundamental of Computer Algorithms(Second Edition) by Sartaj Sahni

Algorithms

- Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output.

- For example, one might need to sort a sequence of numbers into nondecreasing order. This problem arises frequently in practice and provides fertile ground for introducing many standard design techniques and analysis tools. Here is how we formally define the sorting problem:

- Input: A sequence of n numbers $a_1, a_2, \ldots a_n$

- Output: A permutation (reordering) a1, a2,…, a_n of the input sequence such that $a_1 \leq a_2 \leq \cdots \leq a_n$.
- For example, given the input sequence 31, 41, 59, 26, 41, 58, a sorting algorithm returns as output the sequence 26, 31, 41, 41, 58, 59.
- Such an input sequence is called an **instance** of the sorting problem. In general, an instance of a problem consists of the input (satisfying whatever constraints are imposed in the problem statement) needed to compute a solution to the problem.

Characteristics of an Algorithm

- Input
- Output
- Definiteness
- Finiteness
- Effectiveness

What kinds of problems are solved by algorithms?

- The Human Genome Project has the goals of identifying all the 100,000 genes in human DNA, determining the sequences of the 3 billion chemical base pairs that make up human DNA, storing this information in databases, and developing tools for data analysis.
- Each of these steps requires sophisticated algorithms.
- While the solutions to the various problems involved are beyond the scope of us, ideas from many of the chapters in this book are used in the solution of these biological problems, thereby enabling scientists to accomplish tasks while using resources efficiently.
- The savings are in time, both human and machine, and in money, as more information can be extracted from laboratory techniques.

What kinds of problems are solved by algorithms?(cont.)

- The Internet enables people all around the world to quickly access and retrieve large amounts of information.
- In order to do so, clever algorithms are employed to manage and manipulate this large volume of data.
- Examples of problems which must be solved include finding good routes on which the data will travel and using a search engine to quickly find pages on which particular information resides.

What kinds of problems are solved by algorithms?(cont.)

- Electronic commerce enables goods and services to be negotiated and exchanged electronically each of these steps requires sophisticated algorithms.
- The ability to keep information such as credit card numbers, passwords, and bank statements private is essential if electronic commerce is to be used widely.
- Public-key cryptography and digital signatures are among the core technologies used and are based on numerical algorithms and number theory.

Algorithm as a technology

- Algorithms devised to solve the same problem often differ dramatically in their efficiency. These differences can be much more significant than differences due to hardware and software.

Example:

- The first, known as insertion sort, takes time roughly equal to $c_1 n^2$ to sort n items, where c1 is a constant that does not depend on n.
- The second, merge sort, takes time roughly equal to c2nlgn, where lgn stands for log2n and c2 is another constant that also does not depend on n.
- Insertion sort usually has a smaller constant factor than merge sort, so that c1 < c2.
- Where merge sort has a factor of lgn in its running time, insertion sort has a factor of n, which is much larger.
- Although insertion sort is usually faster than merge sort for small input sizes, once the input size n becomes large enough, merge sort's advantage of lgn vs. n will more than compensate for the difference in constant factors.

An Example

For a concrete example, let us pit a faster computer (computer A) running insertion sort against a slower computer (computer B) running merge sort. **They each must sort an array of one million numbers**. Suppose that computer A executes one billion instructions per second and computer B executes only ten million instructions per second, so that computer A is 100 times faster than computer B in raw computing power. To make the difference even more dramatic, suppose that the world's craftiest programmer codes insertion sort in machine language for computer A, and the resulting code requires 2n2 instructions to sort n numbers. (Here, c1 = 2.) Merge sort, on the other hand, is programmed for computer B by an average programmer using a high-level language with an inefficient compiler, with the resulting code taking 50n lgn instructions (so that c2 = 50). To sort one million numbers, computer A takes

$$\frac{2 \cdot (10^6)^2 \text{ instructions}}{10^9 \text{ instructions/second}} = 2000 \text{ seconds},$$

while computer B takes

$$\frac{50 \cdot 10^6 \lg 10^6 \text{ instructions}}{10^7 \text{ instructions/second}} \approx 100 \text{ seconds}.$$

Example cnt.

- By using an algorithm whose running time grows more slowly, even with a poor compiler, computer B runs 20 times faster than computer A
- The advantage of merge sort is even more pronounced when we sort ten million numbers: where insertion sort takes approximately 2.3 days, merge sort takes under 20 minutes.
- In general, as the problem size increases, so does the relative advantage of merge sort.

Time & Space Complexity

What is Time complexity?

The time complexity of an algorithm is the total amount of time required by an algorithm to complete its execution.

OR

Every algorithm requires some amount of computer time to execute its instruction to perform the task. This computer time required is called time complexity.

Generally, running time of an algorithm depends upon the following...

- ✓Whether it is running on Single processor machine or Multi processor machine.
- ✓Whether it is a 32 bit machine or 64 bit machine
- ✓Read and Write speed of the machine.
- ✓The time it takes to perform Arithmetic operations, logical operations, return value and assignment operations etc.,
- ✓Input data

- When we calculate time complexity of an algorithm, we consider **only input data** and ignore the remaining things, as they are machine dependent. We check only, how our program is behaving for the different input values to perform all the operations like Arithmetic, Logical, Return value and Assignment etc.

- Calculating Time Complexity of an algorithm based on the system configuration is a very difficult task because, the configuration changes from one system to another system. To solve this problem, we must assume a model machine with **specific configuration**. So that, we can able to calculate generalized time complexity according to that model machine.

To calculate time complexity of an algorithm, we need to define a model machine. Let us assume a machine with following configuration...

- Single processor machine
- 32 bit Operating System machine
- It performs sequential execution
- It requires 1 unit of time for Arithmetic and Logical operations
- It requires 1 unit of time for Assignment and Return value
- It requires 1 unit of time for Read and Write operations

Example 1

Consider the following piece of code...

```
int sum(int a, int b)
{
    return a+b;
}
```

In above sample code, it requires 1 unit of time to calculate a+b and 1 unit of time to return the value. That means, totally it takes 2 units of time to complete its execution. And it does not change based on the input values of a and b. That means **for all input values**, it requires same amount of time i.e. 2 units.

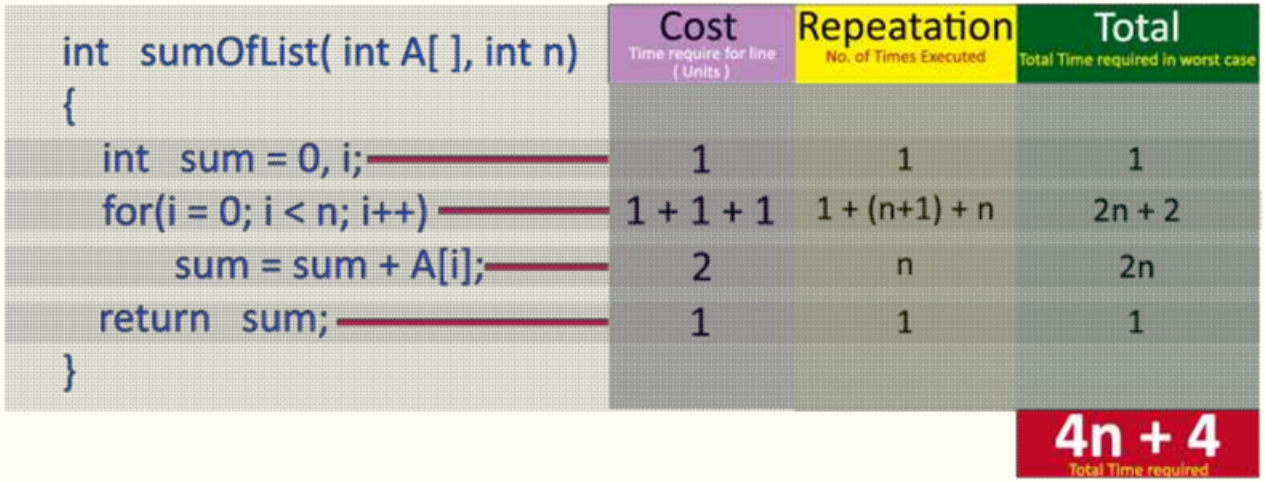
If any program requires fixed amount of time for all input values then its time complexity is said to be **constant time** complexity.

Example 2

Consider the following piece of code...

```
int sum(int A[], int n)
{
int sum = 0, i;
for(i = 0; i < n; i++)
{
      sum = sum + A[i];
}
return sum;
}
```

For the above code, time complexity can be calculated as follows..



In above calculation

- Cost is the amount of computer time required for a single operation in each line.
- Repetition is the amount of computer time required by each operation for all its repetitions.
- Total is the amount of computer time required by each operation to execute.
- So above code requires '4n+4' Units of computer time to complete the task. Here the exact time is not fixed. And it changes based on the n value. **If we increase the n value then the time required also increases linearly**.

Totally it takes '4n+4' units of time to complete its execution and it is Linear Time Complexity.

If the amount of time required by an algorithm is increased with the increase of input value then that time complexity is said to be **Linear Time** Complexity

What is Space complexity?

Total amount of computer memory required by an algorithm to complete its execution is called as space complexity of that algorithm.

When we design an algorithm to solve a problem, it needs some computer memory to complete its execution. For any algorithm, memory is required for the following purposes...

- Memory required to store program instructions
- Memory required to store constant values
- Memory required to store variable values
- And for few other things

Generally, when a program is under execution it uses the computer memory for THREE reasons. They are as follows...

✓Instruction Space: It is the amount of memory used to store compiled version of instructions.
✓Environmental Stack: It is the amount of memory used to store information of partially executed functions at the time of function call.
✓Data Space: It is the amount of memory used to store all the variables and constants.

When we want to perform analysis of an algorithm based on its Space complexity, we consider only Data Space and ignore Instruction Space as well as Environmental Stack.
That means we calculate only the memory required to store Variables, Constants, Structures, etc.,

To calculate the space complexity, we must know the memory required to store different datatype values (according to the compiler). For example, the C Programming Language compiler requires the following...

- ✓2 bytes to store Integer value,
- ✓4 bytes to store Floating Point value,
- ✓1 byte to store Character value,
- ✓6 (OR) 8 bytes to store double value

Example 1

Consider the following piece of code...

```
int square(int a)
{
return a*a;
}
```

✓In above piece of code, it requires 2 bytes of memory to store variable 'a' and another 2 bytes of memory is used for return value.

✓That means, totally it requires 4 bytes of memory to complete its execution. And this 4 bytes of memory is fixed for any input value of 'a'. This space complexity is said to be Constant Space Complexity.

If any algorithm requires a fixed amount of space for all input values then that space complexity is said to be **Constant Space Complexity**

Example 2

Consider the following piece of code...

```
int sum(int A[], int n)
{
int sum = 0, i;
for(i = 0; i < n; i++)
sum = sum + A[i];  return sum;
}
```

In above piece of code it requires

- 'n*2' bytes of memory to store array variable ‘a[ ]'
- 2 bytes of memory for integer parameter 'n'
- 4 bytes of memory for local integer variables 'sum' and 'i' (2 bytes each)
- 2 bytes of memory for return value.

That means, totally it requires '2n+8' bytes of memory to complete its execution. Here, the amount of memory depends on the input value of 'n'. This space complexity is said to be Linear Space Complexity.

If the amount of space required by an algorithm is increased with the increase of input value, then that space complexity is said to be **Linear Space Complexity**