Analysis and Design of Algorithms

UNIT-1

Recurrence Relations

Content

- Conclusion Analysis of Algorithm
- Sequences and Recurrence Relations
- Recursion and Recurrence
- Recursive Algorithms
- Recurrence Relation
- Forming Recurrence Relation
- Solving Recurrence Relations

Conclusion

Analysis of Algorithm

Algorithm Analysis → It provides background information that gives us a general idea of how long an Algo will take for a given problem set.
suppose A problem have n Inputs. If we get comparison n times for arrange them into Ascending order so n×n operations performed.

- purpose of Algo is "not to find a formula".
- Analysis of Algo without regard to any specific computer type.

Algo → Counts the No. of Different characters in a file.

```
For all 256 char do
    Assign zero to the counter          ] loop 1
end For loop.
While there are more characters in the file do
    Get the next character
    Increment the counter for this character by one   ] loop 2
end while loop.
```

Description → - total 256 passes for the Initialization loop ie loop 1
- If n char in the input file there are n passes for the second loop ie loop 2.

- For each pass of the loop there is a check exist that is for loop variable is ~~within~~ within the bounds.
- Initialization loop (i.e. loop 1) does a set of 257 assignments
  - 256 Increments for the loop variable
  - 257 checks that this variable is within the loop bounds.

For the second loop, we will need to do check of the condition N+1 times (+1 for the last check when the file is empty.) & we will increments N counters.

The total No. of operations is
- Increment N + 256
- Assignments 257
- checks (N + 258) conditions.

So If increase char in file i.e.→ 500, 1000, 10000 etc we have to increase the operation. & checks.

So Algo. Analysis requires a set of Rules to determine how operations are to be counted.

Exact Analysis Rules →

(1) Assume A time unit.
(2) Execution of one of the following operations take time 1 (one):
   (a) Assignment operation (b) I/O operation (single)
   (c) Single Boolean operation, numeric operation
   (d) Single Arithmetic operations
   (e) function Return
   (f) Array index operations, pointer Dereference
(3) Running time of a selection statement (switch) is the time for the condition evaluation + Maximum Running time for the individual clauses in the selection.

④ Loop execution time is the sum of the body loop over the number the is executed + time for the loop check + update operations + time for the loop setup.

⑤ Running time of a function call is 1 for setup + The time for any parameter calculations + Time Required for the execution of the function body.

The analysis of an Algo. is to evaluate the performance of the Algo. based on the given models + metrics.

① Input size.

② Running time → worst case + Average case
generally we finding only the worst case Running ~~time~~.

③ Order of growth → we generally see the growth Rate of the Running time.

- We only consider the leading terms of a time formula.
The leading term is n^2 in expression $n^2 + 100n + 5000$.

So Algo Analysis is a part of Computational Complexity Theory. In Theoretical Analysis of Algo It is Common to estimate Their Complexity in ~~[illegible]~~ "Asymptotic sense" ie → To estimate the complexity function for large length of Input.

→ Usually Asymptotic estimates are used because different implementation of the same Algo. may differ in efficiency.

Kinds of analyses

Worst-case: (usually)

- $T(n)$ = maximum time of algorithm on any input of size n.

Average-case: (sometimes)

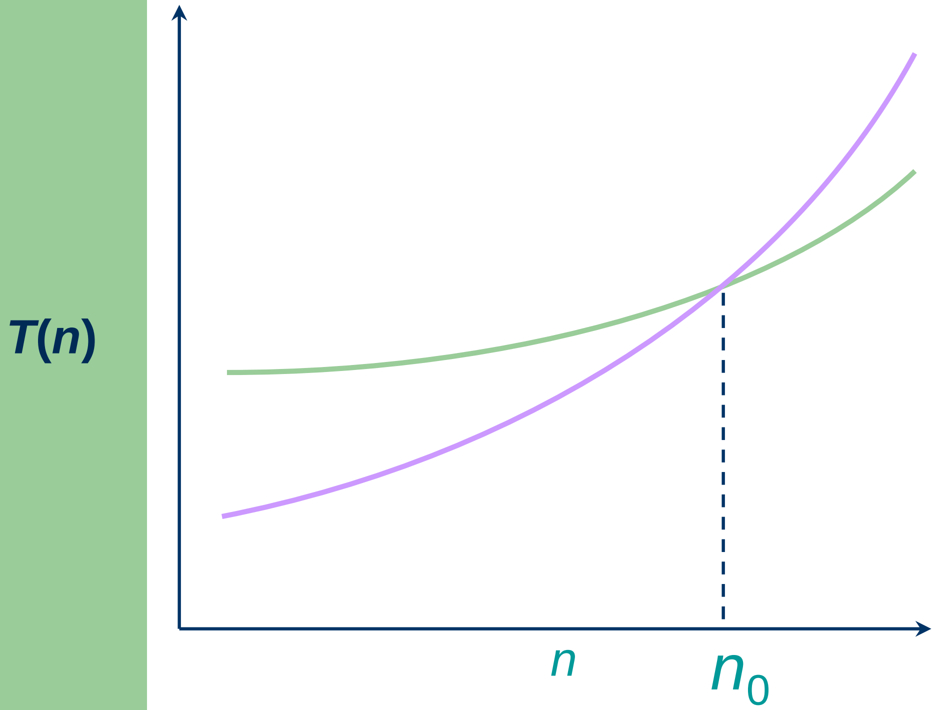
- $T(n)$ = expected time of algorithm over all inputs of size n.
- Need assumption of statistical distribution of inputs.

Best-case: (NEVER)

- Cheat with a slow algorithm that works fast on *some* input.

Asymptotic performance

When *n* gets large enough, a Θ(*n*²) algorithm *always* beats a Θ(*n*³) algorithm.



- Asymptotic analysis is a useful tool to help to structure our thinking toward better algorithm
- We shouldn't ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing

Recurrence Relation

Sequences and Recurrence Relations

Consider the following two sequences:

$$S_1 : \; 3, 5, 7, 9, \ldots$$
$$S_2 : \; 3, 9, 27, 81, \ldots$$

We can find a formula for the nth term of sequences S_1 and S_2 by observing the pattern of the sequences.

$$S_1 : \; 2 \cdot 1 + 1, 2 \cdot 2 + 1, 2 \cdot 3 + 1, 2 \cdot 4 + 1, \ldots$$
$$S_2 : \; 3^1, 3^2, 3^3, 3^4, \ldots$$

For S_1, $a_n = 2n + 1$ for $n \geq 1$, and for S_2, $a_n = 3^n$ for $n \geq 1$. This type of formula is called an **explicit formula** for the sequence, because using this formula we can directly find any term of the sequence without using other terms of the sequence. For example, $a_3 = 2 \cdot 3 + 1 = 7$.

Sequences and Recurrence Relations

A **recurrence relation** for a sequence $a_0, a_1, a_2, \ldots, a_n, \ldots$ is an equation that relates a_n to some of the terms $a_0, a_1, a_2, \ldots, a_{n-2}, a_{n-1}$ for all integers n with $n \geq k$, where k is a nonnegative integer. The **initial conditions** for the recurrence relation are a set of values that explicitly define some of the members of $a_0, a_1, a_2, \ldots, a_{k-1}$.

The equation

$$a_n = 2a_{n-1} + a_{n-2} \quad \text{for all } n \geq 2,$$

as defined above, relates a_n to a_{n-1} and a_{n-2}. Here $k = 2$. So this is a recurrence relation with initial conditions $a_0 = 5$ and $a_1 = 7$.

Recursion and Recurrences

- Recursion is a particularly powerful kind of reduction, which can be described loosely as follows:
- If the given instance of the problem is small or simple enough, just solve it.
- Otherwise, reduce the problem to one or more simpler instances of the same problem.
- Recursion is generally expressed in terms of recurrences.
- In other words, when an algorithm calls to itself, we can often describe its running time by a **recurrence equation.**
- **recurrence equation** describes the overall running time of a problem of size n in terms of the running time on smaller inputs.

Recursive Algorithms

- A recursive algorithm is one in which objects are defined <u>in terms</u> of other objects of the same type.
- Advantages:
  - Simplicity of code
  - Easy to understand
- Disadvantages
  - Memory
  - Speed
  - Possibly redundant work
- Tail recursion offers a solution to the memory problem, but really, do we <u>need</u> recursion?

Recursive Algorithms: Analysis

- We have already discussed how to analyze the running time of (iterative) algorithms
- To analyze recursive algorithms, we require more sophisticated techniques
- Specifically, we study how to defined & solve recurrence relations

Motivating Examples: Factorial

- Recall the factorial function:

$$n! = \begin{cases} 1 & \text{if } n = 1 \\ n.(n-1) & \text{if } n > 1 \end{cases}$$

- Consider the following (recursive) algorithm for computing n!

FACTORIAL

Input: $n \in \mathbb{N}$

Output: n!

1. **If** (n=1) or (n=0)
2. **Then Return** 1
3. **Else Return** n × FACTORIAL(n-1)
4. **Endif**
5. **End**

Factorial: Analysis

How many multiplications M(x) does factorial perform?

- When n=1 we don't perform any
- Otherwise, we perform one…
- … plus how ever many multiplications we perform in the recursive call FACTORIAL(n-1)
- The number of multiplications can be expressed as a formula (similar to the definition of n!

$$M(0) = 0$$

$$M(n) = 1 + M(n-1)$$

- This relation is known as a recurrence relation

Recurrence Relation

- A **recurrence relation** for the sequence, a_0, a_1, ...a_n, is an equation that relates a_n to certain of its predecessors a_0, a_1, ... , a_{n-1}.

- Initial conditions for the sequence a_0, a_1, ... are explicitly given values for a finite number of the terms of the sequence.

Recurrence Relations

- **Definition**: A recurrence relation for a sequence $\{a_n\}$ is an equation that expresses a_n in terms of one or more of the previous terms in the sequence:

$$a_0, a_1, a_2, \ldots, a_{n-1}$$

  for all integers $n \geq n_0$ where n_0 is a non-negative integer.
- A sequence is called a solution of a recurrence if its terms satisfy the recurrence relation

Recurrence Relations: Solutions

- Consider the recurrence relation -

$$a_n = 2 * a_{n-1} - a_{n-2}$$

- It has the following sequences a_n as solutions
  - $a_n = 3n$
  - $a_n = n+1$
  - $a_n = 5$
- The initial conditions + recurrence relation <u>uniquely</u> determine the sequence.