CS 213 Notes

S.I.M.P.L.E

July, 2021

CONTENTS

1 INTRODUCTION

ALGORITHM: Outline, the essence of a computational procedure, step-by-step instructions.

PROGRAM: An implementation of an algorithm in some programming language.

DATA STRUCTURE: Organization of data needed to solve the problem.

An **Algorithmic Problem** has a specification of an input and a given specification of an output. The Algorithm would describe actions on the input instance to get the desired output instance.

A **Good Algorithm** is characterized by its Efficiency which depends on

- Running Time
- Space Used

In this course we shall attempt to use a high level description of the altorithm and take into account all possible inputs for the same. This would allow one toe evaluate the efficiency of any algorithm in a way that is independant of the hardware and the software environment.

1.1 PSEUDO CODE

A mixture of natural language and high-level programing concepts that describes the main ideas behind a generic implementation of a data structure of algorithm is called a **Pseudo Code**.

Pseudo Code is more structured than usual prose but less formal than a programming language.
Following is the Pseudo Code terminology that would be used

- Standard mathematical symbols to describe numeric and boolean expressions.
- Using $\leftarrow$ for assignment and $=$ for equality relationship
- Programming Constructs like the following

  - decision structures: **if ... then ... [else ...]**
  - while-loops: **while ... do**
  - repeat-loops: **repeat ... until ...**
  - for-loop: **for ... do**
  - array indexing: **A[i], A[i,j]**

- Methods like
  - calls: object method(args)
  - returns: **return** value

1.2 ANALYSIS OF ALGORITHMS

Primitive Operation: Low-level operation independant of programming language. Can be identified in Pseudo-Code. For example

- Data Movement (assignment)
- Control (branch, subroutine call, return)
- arithmetic and logical operations (addition, comparison)

1.3 INSERTION SORT

```
Input: $A[1...n]$ is an array of integers
Output: a permutation of $A$ such that $A[1] \leq A[2] \leq \cdots \leq A[n]$
1 for $j \leftarrow 2$ to $n$ do
2     $key \leftarrow A[j]$;
3     $i \leftarrow j - 1$;
4     while $i > 0$ $A[i] > key$ do
5         $A[i+1] \leftarrow A[i]$;
6         $i \leftarrow i - 1$;
7     end
8     $A[i+1] \leftarrow key$;
9 end
```

1.3.1 *Analysis of the Algorithm*

Say that the cost of the i-th line is c_i. Further say that for each j in $2, 3, \cdots, n$; one enters the while loop t_j times. Then, the total cost of the algorithm would be

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Simplification of the above equation gives us

$$T(n) = (c_1 + c_2 + c_3 - c_5 - c_6 + c_8)n + (c_4 + c_5 + c_6)\sum_{j=2}^{n} t_j - (c_2 + c_3 - c_5 - c_6 + c_8)$$

There are now the following possible cases

BEST CASE: $t_j = 1$ for all permissible j. In this case, $T(n)$ is a linear function of n.

WORST CASE: $t_=j$ for all permissible j. In this case, $T(n)$ is a quadratic function of n.

AVERAGE CASE: $t_j = \frac{j}{2}$ for all permissible j. In this case, $T(n)$ is a quadratic function of n.

1.4 ASYMPTOTIC ANALYSIS

This is used when we would like to get rid of the details of execution which may be hardware dependant and would just like to capture the essence of how the running time of the algorithm increases with the increase in input size.

1.4.1 *Asymptotic Notation*

Definition 1.1 (Big-Oh Notation). Let f and g be two well defined functions. Then we write $f(n) = O(g(n))$ if there exist constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

It must be noted that there is no *unique* Big-Oh of a function. In the sense that $f(n) = 6n + 9$ is both $O(n)$ and $O(n^2)$.

Definition 1.2 (Big-Omega Notation). Let f and g be two well defined functions. Then, we write $f(n) = \Omega(g(n))$ if there exist constants c and n_0 such that $f(n) \geq cg(n)$ for all $n \geq n_0$.

Definition 1.3 (Big-Theta Notation). Let f and g be two well defined functions. Then we write $f(n) = \Theta(g(n))$ if there exist constants c_1, c_2 and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

From the above definitions it is now obvious that $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Definition 1.4 (Little-Oh Notation). Let f and g be well defined functions taking non-negative real values. Then, we write $f(n) = o(g(n))$ if for every $\epsilon > 0$, there exists n_0 such that $f(n) \leq \epsilon g(n)$ for all $n \geq n_0$.

Definition 1.5 (Little-Omega Notation). Let f and g be two well defined functions taking non-negative real values. Then, we write $f(n) = \omega(g(n))$ if for every $\epsilon > 0$, there exists n_0 such that $f(n) \geq \epsilon g(n)$ for all $n \geq n_0$.

Frequently the following terminology shall be used to refer to the time-complexities of algorithms.

LOGARITHMIC: $O(\log n)$

LINEAR: $O(n)$

QUADRATIC: $O(n^2)$

POLYNOMIAL: $O(n^k)$ where $k \in \mathbb{Z}$.

EXPONENTIAL: $O(a^n)$ where $a \in \mathbb{R}^+$