BRAC University

Algorithms

Lecture 1.1

Rubayat Khan, Lecturer
9/7/2015

What is an algorithm?

The finite set of instructions used by the computer to solve a problem. An algorithm must satisfy the following criteria.

Input: must be able to take 0 or more input
Output: at least one correct output
Definiteness: the instructions are clear and unambiguous
Finiteness: it will halt after a finite number of steps
Effectiveness: each step must be as simple as possible so that everyone can carry out using a pen and a paper

What is a program?

The translation of an algorithm into a programming language. The study of algorithms can be divided into 4 parts:

1. Design an algorithm: Designing/ planning an algorithm based on output.
2. Validate the algorithm: check whether the algorithm developed produces the correct output for a set of valid input. The checking is done by a method of proof.
3. Analyze the algorithm: How much time and space it takes to produce the output.
4. Test the program: Testing is done is 2 ways. Test it with a set of input data and check the corresponding output. Make several programmers program on the algorithm and match the output.

How do we write algorithms?

They are written in a standard form called the pseudocode.

```
print(A, n)
// print is the name of the function, A is the array and n is the size
for i=1 to n-1
        if i%2 == 0
                then print A[i]
```

Data types of variables are not specified. Pseudocodes are kept very general.

Validation (Proof of correctness):

Loop invariant - It is a condition that is true before the start of the loop **(initialization)**, true before the start of each iteration **(maintenance)** and also true after the loop terminates **(termination)**. Now this is very crucial as it is not easy to find the variant just by looking at the algorithm.
Let us look at a sorting algorithm and find the loop variant.

```
sort(A, n)
for i=1 to n
        for j = i+1 to n
                if A[j]<A[i]
                swap A[i] with A[j]
```

Initialization: If i = 1, we know that the array is sorted before the start of the loop for the index i.
Maintenance: After the loop starts we know that before each iteration the array is sorted from position 1 to i-1.
Termination: When the loop terminates we know that the entire array is sorted.

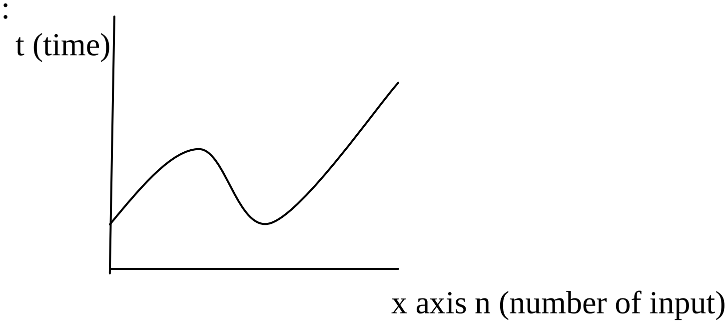
Analyzing Algorithms

Measure the computation time and amount of space required. Determining the resources (I/O, hardware, etc) to allocate also falls under this category but we will focus with former two factors in this course. We refer the computation time as the running time. It depends on the size of the input and therefore we define the running time of any algorithms as a function of the size of the input.

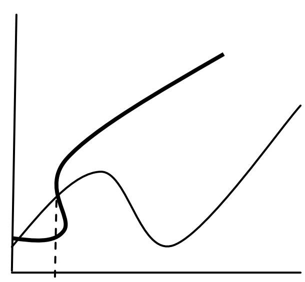
Before we start analyzing the algorithms we need to get familiar with a few notations called asymptotic notations.

Asymptotic notations:

1. Big O (O):



Let there be an algorithm x that behaves like the above graph for n number of input. Let us call it f1(n) where n is the number of input. What will be the upper bound of this function? Upper bound of a function f1(n), is any function f2(n), such that f1(n)<=f2(n) for all input n greater than n_o. n_o is the point where both the functions meet.



The thin curve is f1(n) and the thick one is f2(n). Let us see the correctness of the phrase f1(n)<=f2(n) for all input n greater that n_o.

let f1(n) = 2n+1 and f2(n) = 6n+3
as the curves meet we can write 2n+3 = 6n+1 and we can deduce n to be $^1/_2$ which is n_o. We said f1(n)<=f2(n) for all input n greater that n_o. Put any number greater than .5 in both the equations and see the result. The function f2(n) is the upper bound or the worst time complexity of an algorithm, f1(n) = O(f2(n)). In plain English worst case time complexity is the longest time required for an algorithm to produce its output and it depends on the type of the input. We will look into the types of input as we go along the course.

If 6n+3 is an upper bound then every function with a higher polynomial degree is also the upper bound of the function 2n+1. Therefore we can say that a function might have more than one upper bound. Which one to choose if there are multiple? Choose the one nearest to the original function. For this example although n^2, n^3,.... n^k are upper bounds of the function f1(n), f2(n) was the nearest upper bound and therefore we will choose this one. F2(n) = 6n+3, ignoring the constants and non polynomial terms we get just n. We can also say f1(n) = O(n), as when determining the worst case we are concerned about the HIGHEST polynomial degree.

f1(n) = $2n^3 + 5n^2 + 4$. What will be its upper bound?

Self Assessment:
(i) let f1(n) = 2n+1 and f2(n) = n^2. Is O(f1(n)) = f2(n) ?
(ii) let f2(n) = 2n+1 and f1(n) = n^2. Is O(f1(n)) = f2(n) ?
(iii) let f1(n) = f2(n) = 2n+1. Is O(f1(n)) = f2(n) ?