Indeed, $\rho(S) = -i, \rho(T) = e^{2i\pi/12}$ gives a homomorphism (not obvious!) from $SL(2,\mathbf{Z})$ onto the group of all 12-th roots of unity in $\mathbf{C}^*$. Actually, one may explicitly write

$$\rho\begin{pmatrix} a & b \\ c & d \end{pmatrix} = e^{2i\pi((1-c^2)(bd+3(c-1)d+c+3)+c(a+d-3))/12}.$$

It is surprising to realize this is a homomorphism unless we use the theory of modular forms.

- The fact that the alternating groups A_n for $n \geq 9$ are quotients of $PSL(2,\mathbf{Z})$ is easy to prove by observing that the alternating group can be generated by an element of order 2 and an element of order 3. Interestingly, this was first observed by the great group theorist G.A. Miller in 1901 where he used the assumption that there exists a prime between n and $2n$ when $n \geq 6$ (known now as Bertrand's postulate). Of course, now one can prove this without Bertrand's postulate.
- In contrast to the unsurprising fact above, it is an astonishing fact that the infinite groups $GL(n,\mathbf{Z})$ for *all* large enough n are quotients of $SL(2,\mathbf{Z})$!

A popular myth (with no basis whatsoever!) credits Ramanujan with the above-mentioned assertion on the closeness of $e^{\pi\sqrt{163}}$ to an integer. Talking of Ramanujan, we may say:

Ramanujan did mathematics somehow;
we still can't figure out even now.
He left his mark on "p of n",
and wrote pi in series quite often.
The theta functions he called 'mock'
are subject-matter of many a talk.
He died very young – yes, he too!
He was only thirty-two!
His name prefixes the function tau.
Truly, that was his last bow!

Asymptotic Notations and its Applications

V. P. Ramesh and R. Gowtham
Department of Mathematics, Central University of Tamil Nadu, Thiruvarur, India
E-mail:

Abstract. In this expository article we have presented the asymptotic notations and illustrated a few usage of asymptotic notations through examples. We have also given a few plots to give a geometric view of these notation and we believe that it would help the beginners to perceive and understand these notations easily. Usage of such notations in analysing a given algorithm is vague to many students. To address this vagueness we have presented the analysis of bubble sort algorithm and we believe that it will help the readers to apply these notations to analyse their algorithms. We have also illustrated through examples, analysing growth of functions and error approximations.

1. Introduction

Asymptotic notations are used as a tool of analysis in places where we are concerned about asymptotic behaviour of the material of interest. For example while analysing algorithms with running time as our scale of measurement, the widely accepted way is analysing the behaviour of algorithm as a function of input size for arbitrarily large inputs. Asymptotic analysis is used in various branches of science such as in computer science it is used to analyse the complexity of algorithms and thereby to classify the set of all problems [2,5], in mathematics as a tool to gauge numerical algorithms and also to capture the behaviour of highly chaotic functions.

We introduce a set denoted by, $R(\mathbb{N},\mathbb{R})$, the set of all functions from $\mathbb{N}$ to $\mathbb{R}$. This function space is used to analyse the complexity of the algorithms by measuring the response of the algorithm for a given input size. For example the measure could be the time or space (memory) or communication taken by the algorithm to respond for a given input. It is practically obvious that the response time or the space required for an algorithm increases with increase in the input size. Therefore these functions are expected to be an increasing (non decreasing) function. This perception will help the reader to understand the analysis better. Therefore it is appropriate to consider the set $R(\mathbb{N},\mathbb{R}^+)$, however we are considering the set $R(\mathbb{N},\mathbb{R})$ due to the reason that there are many useful and inevitable functions like $\log(\log(n))$ which are eventually positive. For theoretical purpose, there is no harm in considering all the functions from $\mathbb{N}$ to $\mathbb{R}$. For further information about these notations the readers may refer to [2,3].

In this expository article we have not used the traditional notations. In the traditional notations, it is convention that the set theoretic and algebraic symbols are treated in a way which is unfamiliar. A computer scientist, Gilles Brassard in his article [1] argued the need for a different notation, discussing some drawbacks of the convention and talked about the alternatives. The notations we have used in this article are as suggested by Gilles and we expect it to be helpful for the students to understand the basic ideas quickly. So when we say $n \in O(n)$, it is equivalent to saying $n = O(n)$. On the other hand, this notation of Gilles will stress that we are handling sets.

In the forthcoming subsections we have presented a few asymptotic notations and for better understanding we have also presented a few graphs to perceive these notations geometrically.

1.1 The O-notation

Let $g \in R(\mathbb{N}, \mathbb{R})$, we define a set $O(g)$ (read "big-oh of g") as,

$$O(g) := \{f \in R(\mathbb{N},\mathbb{R}) \mid \exists\, C \in \mathbb{R}^+ \times (\exists\, n_0 \in \mathbb{N}(\forall\, n \geq n_0(f(n) \leq Cg(n))))\}$$

For example, we can prove that the function $f(n) = n$ is in $O(n + \log(n))$ because, for any $C \geq 1$, the function $C(n + \log(n))$ eventually dominates n.

The Figure 1 shows the dominance for $C = 2$. It is also to be noted that for any $C < 1$ the dominance would be other way around, for example for $C = \frac{1}{2}$, $\frac{1}{2}(n + \log(n)) = \frac{1}{2}n + \frac{1}{2}\log(n) \leq \frac{1}{2}n + \frac{1}{2}n = n$.

By saying a function $f \in O(g)$, it means, for all except for a finite number of small inputs, f is bounded above by a constant multiple of g, sometimes g is called as an asymptotic upper bound of f. In the context of analysis of algorithm, while analysing the *worst case running time* of an algorithm we are interested in asymptotic upper bound. The *worst case running time* is defined as a function $f : \mathbb{N} \to \mathbb{R}$, where $f(n)$ is the maximum of all possible running time of an algorithm, on any input of size n. For further information on worst case running time the readers may refer to [2,5].

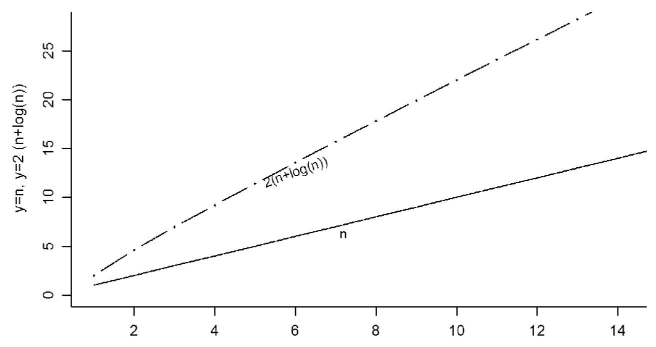

Figure 1. Graphs of functions $y = 2(n + \log(n))$ and $y = n$

1.2 The o-notation

Let $g \in R(\mathbb{N}, \mathbb{R})$, we define a set $o(g)$ (read "small-oh of g") as

$$o(g) := \{f \in R(\mathbb{N},\mathbb{R}) \mid \forall\, C \in \mathbb{R}^+ \times (\exists\, n_0 \in \mathbb{N}(\forall\, n \geq n_0(f(n) < Cg(n))))\}$$

For example, we can prove that the function $f(n) = n + \log(n)$ is in $o(n\log(n))$ because, for any $C > 0$, the function $C(n\log(n))$ eventually dominates the function $n+\log(n)$. The Figure 2 shows the dominance for a few values of C.

It is easy to estimate that for any $C > 0$ the functions $Cn\log(n)$ dominates both n and $\log(n)$, i.e., $n \in o(n\log(n))$ and $\log(n) \in o(n\log(n))$. From this we can estimate that $n + \log(n)$ is eventually less than that of $Cn\log(n)$ for any $C > 0$. For any $C > 0$, $\exists N \in \mathbb{N}$ such that $n + \log(n) < \frac{C}{2}n\log(n) + \frac{C}{2}n\log(n) = Cn\log(n)$, for all $n \geq N$, hence $n + \log(n) \in o(n\log(n))$.

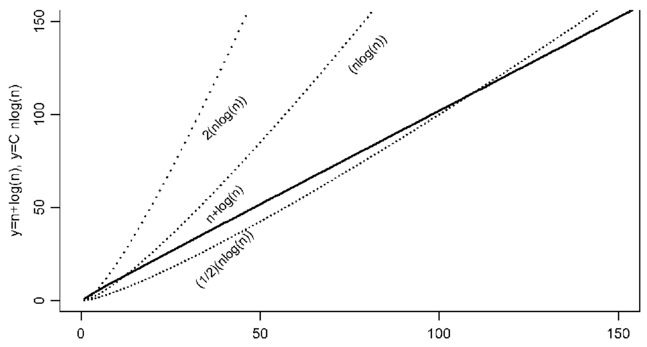

Figure 2. Graph of functions $y = n + \log(n)$ and $y = C(n\log(n))$ for various values of C

Remarks. The statement $f \in O(g)$ does not tell anything about how tight the asymptotic upper bound g is. For example, $n \in O(n^2)$ is not sufficient to conclude that n^2 is a tight asymptotic upper bound for n, since $n \in O(n)$, however the later one is a tight bound. So we can use the o-notation whenever g is not a tight asymptotic upper bound. It is obvious that $o(g) \subsetneq O(g)$, the equality is not true due to the following example, $n \in O(n)$ and $n \notin o(n)$.

1.3 The Ω-notation

Let $g \in R(\mathbb{N}, \mathbb{R})$, we define a set $\Omega(g)$ (read "big-omega of g") as

$$\Omega(g) := \{f \in R(\mathbb{N}, \mathbb{R}) \mid \exists\, C \in \mathbb{R}^+ \times (\exists\, n_0 \in \mathbb{N}(\forall\, n \geq n_0(f(n) \geq Cg(n))))t)\}$$

For example, Figure 3 shows the function $g(n) = n + \log(n)$ will be an asymptotic lower bound for the function $f(n) = n$. Ω-notation is the counterpart of O-notation, as g can be thought as an asymptotic lower bound for f. In the context of analysing algorithms, while analysing *best case running time* of an algorithm (minimum of all possible running time on a input of size n) we are interested in asymptotic lower bound. For further information readers may refer to [2,5]. Similar to O-notation, Ω-notation need not give any information on how tight the bound is.

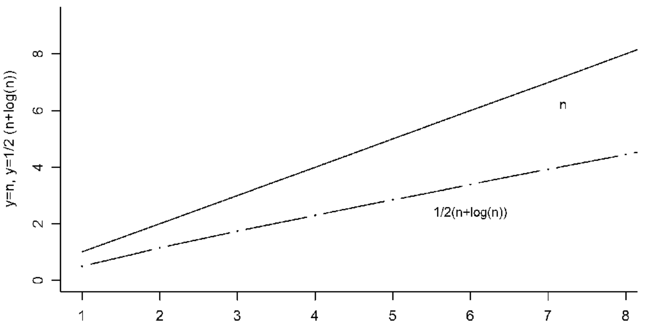

Figure 3. Graphs of functions $y = n$ and $y = \frac{1}{2}(n + \log(n))$

1.4 The ω-notation

Let $g \in R(\mathbb{N}, \mathbb{R})$, we define a set $\omega(g)$ (read "small-omega of g") as

$$\omega(g) := \{f \in R(\mathbb{N}, \mathbb{R}) \mid \forall\, C \in \mathbb{R}^+ \times (\exists\, n_0 \in \mathbb{N}(\forall\, n \geq n_0(f(n) > Cg(n))))\}$$

ω-notation is the counterpart of o-notation. As we discussed for o-notation, ω-notation is useful when we have a asymptotic lower bound which is not tight. For example, the Figure 4 shows that the function $f(n) = n\log(n) \in \omega(n + \log(n))$. It is easy to estimate that for any given $C > 0$, $\exists N_1$ and $N_2 \in \mathbb{N}$, such that $n\log(n) \geq 2Cn$ for all $n \geq N_1$and $n\log(n) \geq 2C\log(n)$ for all $n \geq N_2$. Now, let $N = \max\{N_1, N_2\}$ then $2n\log(n) \geq 2Cn + 2C\log(n)$ for all $n \geq N$. Hence $n + \log(n)$ is an asymptotic lower bound of $n\log(n)$.

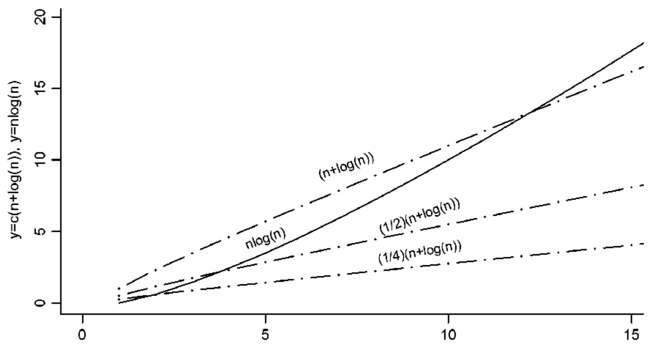

Figure 4. Graphs of functions $y = n\log(n)$ and $y = C(n + \log(n)$ for various values of C

1.5 The Θ-notation

Let $g \in R(\mathbb{N}, \mathbb{R})$, we define a set $\Theta(g)$ (read "theta of g") as,

$$\Theta(g) := \{f \in R(\mathbb{N}, \mathbb{R}) \mid \exists\, C_1, C_2 \in \mathbb{R}^+ \times (\exists\, n_0 \in \mathbb{N}(\forall\, n \geq n_0(C_1 g(n) \leq f(n) \leq C_2 g(n))))\}$$

For example, the Figure 5 shows the geometry of $n \in \Theta(n + \log(n))$. It is easy to estimate that $\frac{1}{2}(n + \log(n)) < n$, for all $n \in \mathbb{N}$, since $n \leq n$ and $\log(n) < n$ for all n. Also $n + \log(n) \geq n$.

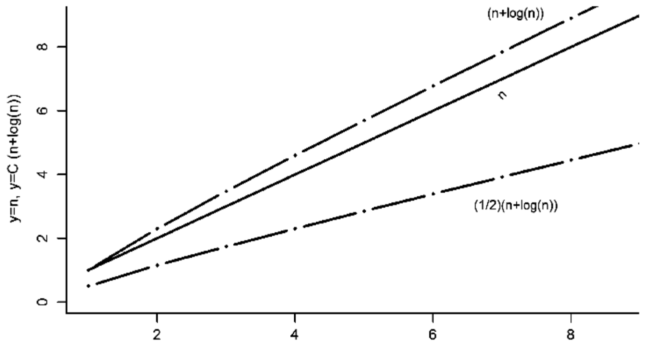

Figure 5. Graphs of functions $y = n, y = n + \log(n)$ and $y = \frac{1}{2}(n + \log(n))$

Remarks. This is one of the very useful asymptotic notations while analysing the algorithms. We claim the running time of an algorithm in $\Theta(g)$ if and only if the running time of the algorithm is sandwiched by constant multiples of the function g. It is also easy to verify that $\Theta(g) = O(g) \cap \Omega(g)$. Practically when we the analyse the algorithms the computational time is always a function from $\mathbb{N}$ to $\mathbb{R}^+$ or in the worst case an eventually positive function. So if we consider the set of all eventually positive functions, this asymptotic notation can be used to partition the set of interest by the following equivalence relation, $fRg \iff f \in \Theta(g)$, which helps us to partition the set based on complexity.

Practically, functions used to analyse the algorithms are many a times complicated to express and compute, so algebraic manipulation of such functions might be tedious, but when we are interested in the asymptotic behaviour it might be easy to work with corresponding asymptotic notations [4]. For detailed introduction about these notations the reader may refer to [3].

In the forthcoming chapters we have demonstrated the applications of these notations.

2. Algorithm Analysis

As we have mentioned earlier we can analyse an algorithm with the aim of knowing how efficient the algorithm is on solving a problem. Usually efficiency is measured based on how much time or space (memory) or communication it needs to solve the problem. Here we are interested in the running time of an algorithm, i.e., time it takes to solve the problem. While theoretically analysing the algorithm based on the running time (as a function of input size) we should capture the behaviour of the algorithm when implemented on a computer. So we make a few assumptions(to reflect a real computer) on our abstract machine [2] where we are going to implement our algorithm and the assumptions are

1. Each step in the algorithm is implemented one after the other (no parallel computing)
2. Each step in our algorithm takes constant amount of time (that is independent of size of input with a few exception such as function call etc.)
3. Each step can take different amount of time to compute.

Each step in the algorithm consist of one or more instructions when implemented on a real computer, each of these instruction takes a constant amount of time to compute on most of the real computers, this justifies our 2nd and 3rd assumptions. Our abstract model may contains instruction for basic arithmetic operations, load, store instructions, branching instruction etc., and it does not contain any unrealistic instruction like one instruction to find maximum, or one that sort etc., these are highly unrealistic.

2.1 Analysis of Bubble Sorting Algorithm

Sorting is the process of rearranging given set of items in some specified order for example, arranging numbers in increasing or decreasing order. Sorting is one of the important operation in computers as well as it has applications in various places. In fact, in any general purpose computer sorting is frequently done to prioritize the task and hence sorting has a role to play when it comes to the efficiency of a operating system. In a sorted list (especially in the case of numbers) many operations like finding the largest element or the smallest element or searching for a particular number are relatively easy. Since sorting is highly useful there are many algorithm devised and a few of the popular algorithms are namely insertion sort, selection sort, bubble sort, quick sort, merge sort, etc. To demonstrate these notations we would like to analyse the bubble sort algorithm.

Bubble Sort Algorithm:
Input: An array of numbers(A).
Output: A sorted array, that is an array of elements of A in increasing order.

Array is indexed from 1 & length(A) returns the number of elements in A

On input of array A,

```
 1: i=1
 2: while i≤length(A)-1 do
 3:   j=length(A)
 4:   while j≥i+1 do
 5:     if A[j]< A[j-1] then
 6:       swap {A[j], A[j-1]}
 7:     end if
 8:     j=j-1
 9:   end while
10:   i=i+1
11: end while
```

Bubble sort algorithm sort the given array of numbers by swapping the adjacent elements of the array, if they are in the wrong order (that is if $A[j] < A[j-1]$) to get them ordered. By swapping each pair of elements $(A[j], A[j-1])$ for the values j from n to 2 we will get to move the largest value to A[1], again by doing the same process for j from n to 3 we will get to move the second largest value to $A[2]$, repeating the same we will get the array sorted.

Now, we will analyse the running time of the algorithm. Let c_i be the time required to execute line i of the algorithm, first line will be executed for one time, line 2 will be executed for n times ($n =$ length (A)), lines 3, 10 and 11 will be executed for $n-1$ times. For each of the iteration of outer while loop, line 4 will run for at most n times, lines 5, 7, 8 and 9 will runs for a maximum of $n-1$ times, on worst case line 6 runs for $n-1$ times. So the running time $T_1(n)$ can be estimated as,

$$\begin{aligned}T_1(n) &\le c_1 + n(c_2 + nc_4 + (n-1)(c_5 + c_6 + c_7 + c_8 + c_9)) \\ &\quad + (n-1)(c_3 + c_{10} + c_{11}) \\ &= c_1 + n(c_2 + c_3 + c_{10} + c_{11}) \\ &\quad + n^2(c_4 + c_5 + c_6 + c_7 + c_8 + c_9) \\ &\quad - n(c_5 + c_6 + c_7 + c_8 + c_9) - (c_3 + c_{10} + c_{11}) \\ &\le c_1 n^2 + n^2(c_2 + c_3 + c_{10} + c_{11}) \\ &\quad + n^2(c_4 + c_5 + c_6 + c_7 + c_8 + c_9) \\ &\le (c_1 + c_2 + c_3 + c_{10} + c_{11} + c_4 \\ &\quad + c_5 + c_6 + c_7 + c_8 + c_9)n^2\end{aligned}$$

Therefore, $T_1 \in O(n^2)$.

In the above analysis we considered the worst case that line 6 executed for maximum number of times. Now assume the input is already sorted, on any input of size n ($n =$ length (A)) number of times each line of input executed will be same as before for 1, 2, 3, 10 and 11. Lines 4 to 9 executed for a minimum of $\frac{(n-1)(n-2)}{2}$ times except the line 6, line 6 will never be executed, so the running time $T_2(n)$,

$$\begin{aligned}T_2(n) &\ge c_1 + nc_2 + (n-1)(c_3 + c_{10} + c_{11}) \\ &\quad + \frac{(n-1)(n-2)(c_4 + c_5 + c_7 + c_8 + c_9)}{2} \\ &\ge \frac{(n-1)(n-2)(c_4 + c_5 + c_7 + c_8 + c_9)}{2} \\ &\ge \frac{n^2(c_4 + c_5 + c_7 + c_8 + c_9)}{4} \\ &\text{(since } (n-1)(n-2) \ge \frac{n^2}{2}\ \forall n \ge 10\end{aligned}$$

Therefore, $T_2 \in \Omega(n^2)$.

Since $T_2 \in \Omega(n^2)$, T_1 can't be in $o(n^2)$, because, by the best case (T_2) it follows that $\exists\ C > 0$ ($\exists\ n_0 \in \mathbb{N}$ ($\forall\ n \ge n_0$ ($T_2(n) \ge Cn^2$))) which contradicts the statement $\forall\ C > 0$ ($\exists\ n_0 \in \mathbb{N}$ ($\forall\ n \ge n_0(T_1(n) < Cn^2)$)) (Since $T_1(n) \ge T_2(n)$). Similarly T_2 can't be in $\omega(n^2)$, since $T_1 \in O(n^2)$. So the actual running time of the algorithm (let's say T), bounded above and below by T_1 and T_2 respectively. Hence we can conclude that $T \in \Theta(n^2)$, which implies the computation time of the algorithm will behave like the polynomial n^2.

3. Growth of functions

Consider the function $f : \mathbb{N} \to \mathbb{R}$ where $f(n)$ is defined as the sum of first n natural numbers, and we are interested in analysing how fast the sum will grow. Therefore,

$$\begin{aligned}f(n) &= 1 + 2 + 3 + \cdots + n \\ &= \frac{n(n+1)}{2} \\ &\le n^2 + n \\ &\le 2n^2\end{aligned}$$

Therefore, $f \in O(n^2)$.

Also,

$$\begin{aligned}\frac{n(n+1)}{2} &= \frac{(n^2+n)}{2} \\ &\ge \frac{n^2}{2}\end{aligned}$$

Hence f is also in $\Omega(n^2)$, therefore f is in $\Theta(n^2)$. As we discussed while analysing bubble sort algorithm we can estimate that $f \notin o(n^2)$ and $f \notin \omega(n^2)$.

Now lets look another function, namely $g_r(n)$=nC_r where $r \le n$, which is nothing but the number of ways of choosing r objects out of n distinct objects. We analyse this function in two cases namely when $r \le n - r$ and $r \ge n - r$.

When $r \le n - r$,

$$\begin{aligned}g_r(n) &= nC_r \\ &= \frac{n(n-1)(n-2)\cdots(n-r+1)}{r!}\end{aligned}$$

$$\leq n(n-1)(n-2)\cdots(n-r+1)$$
$$\leq n^r$$

Similarly, when $r \geq n-r$,

$$\begin{aligned} g_r(n) &= nC_r \\ &= nC_{n-r} \\ &\leq n^{n-r} \end{aligned}$$

Hence $g_r \in O(n^r)$ when $r \leq n-r$ and $g_r \in O(n^{n-r})$ when $r \geq n-r$.

Now, Lets look at the function from a different point of view and lets assume that $r \leq n-r$, then

$$\begin{aligned} g_r(n) &= nC_r \\ &= n(n-1)(n-2)\cdots(n-r+1)/r! \\ &\geq \frac{(n-r+1)^r}{r!} \quad \because n-k > n-r+1 \, \forall \, k \, < r-1 \\ &\geq \frac{(n-n/2)^r}{r!} \text{ for all } n \geq \, 2(r-1) \\ &= \frac{n^r}{(r!2^r)} \end{aligned}$$

Therefore, $g_r \in \Omega(n^r)$.

When $r \geq n-r$, by similar argument we can estimate that $g_r \in \Omega(n^{n-r})$, hence $g_r \notin o(n^r)$ when $r \leq n-r$ and $g_r \notin o(n^{n-r})$ when $r \geq n-r$. Similarly we can also argue for ω notation. As a conclusion, $g_r \in \Omega(n^r)$ and $g_r \in O(n^r)$ when $r \leq n-r$ and hence $g_r \in \Theta(n^r)$ when $r \leq n-r$. Similarly, $g_r \in \Omega(n^{n-r})$ and $g_r \in O(n^{n-r})$ when $r \geq n-r$ and hence $g_r \in \Theta(n^{n-r})$ when $r \geq n-r$.

4. Error approximation

Let's look at another application of asymptotic notations to analyse the error. As a comparison study we want to analyse the efficiency of forward and the central difference methods in approximating the first order derivative of a function. Let f be a twice differentiable function then, by Taylor's theorem, we have

$$f(x)=f(a)+f'(a)(x-a)+\frac{f''(a)}{2!}(x-a)^2+(x-a)^2R(x)$$

where, the function $R(x) \to 0$ when $x \to a$. Now, let $\frac{f(a+h)-f(a)}{h}$ be the forward difference approximation to $f'(a)$. Now to analyse the error as h tend to 0, let's consider the sequence $\frac{1}{n}$, now

$$\begin{aligned} &\left| \frac{f\left(a+\frac{1}{n}\right)-f(a)}{\frac{1}{n}} - f'(a) \right| \\ &= \left| \frac{f''(a)}{2!}\left(\frac{1}{n}\right) + R\left(a+\frac{1}{n}\right)\left(\frac{1}{n}\right) \right| \\ &= \frac{1}{n}\left| \frac{f''(a)}{2} + R\left(a+\frac{1}{n}\right) \right| \\ &\leq \frac{1}{n}|C| \quad \forall n \gg 1, \end{aligned}$$

for some $C > 0$, since the function R is bounded in some neighbourhood of a. Therefore, the error $\left| \frac{f\left(a+\frac{1}{n}\right)-f(a)}{\frac{1}{n}} - f'(a) \right| \in O\left(\frac{1}{n}\right)$.

Now, Let's analyse the central difference method. Let f be any thrice differentiable function, then by Taylor's theorem,

$$\begin{aligned} f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 \\ + \frac{f'''(a)}{3!}(x-a)^3 + (x-a)^3R(x) \end{aligned}$$

where, the function $R(x) \to 0$ when $x \to 0$. Now let $\frac{f(a+h)-f(a-h)}{2h}$ be the central difference approximation to $f'(a)$. Now to analyse the error as h tends to 0, let's consider the sequence $\frac{1}{n}$,

$$\begin{aligned} f\left(a+\frac{1}{n}\right) &= f(a) + f'(a)\left(\frac{1}{n}\right) + \frac{f''(a)}{2!}\left(\frac{1}{n^2}\right) \\ &\quad + \frac{f'''(a)}{3!}\left(\frac{1}{n^3}\right) + \left(\frac{1}{n^3}\right)R\left(a+\frac{1}{n}\right) \\ f\left(a-\frac{1}{n}\right) &= f(a) - f'(a)\left(\frac{1}{n}\right) + \frac{f''(a)}{2!}\left(\frac{1}{n^2}\right) \\ &\quad - \frac{f'''(a)}{3!}\left(\frac{1}{n^3}\right) - \left(\frac{1}{n^3}\right)R\left(a-\frac{1}{n}\right) \end{aligned}$$

$\implies$

$$\begin{aligned} &\frac{f\left(a+\frac{1}{n}\right)-f\left(a-\frac{1}{n}\right)}{\frac{2}{n}} \\ &= f'(a) + \frac{f'''(a)}{3!}\left(\frac{1}{n^2}\right) + \left(\frac{1}{n^2}\right)R\left(a+\frac{1}{n}\right)\left(\frac{1}{2}\right) \\ &\quad + \left(\frac{1}{n^2}\right)R\left(a-\frac{1}{n}\right)\left(\frac{1}{2}\right) \end{aligned}$$

Hence the error term will be,

$$\left|\frac{f\left(a+\frac{1}{n}\right)-f\left(a-\frac{1}{n}\right)}{\frac{2}{n}}-f'(a)\right|$$

$$=\left|\frac{f'''(a)}{3!}+R\left(a+\frac{1}{n}\right)\left(\frac{1}{2}\right)+R\left(a-\frac{1}{n}\right)\left(\frac{1}{2}\right)\right|\left(\frac{1}{n^2}\right)$$

$\Longrightarrow$

$$\left|\frac{f\left(a+\frac{1}{n}\right)-f\left(a-\frac{1}{n}\right)}{\frac{2}{n}}-f'(a)\right|\in O\left(\frac{1}{n^2}\right)$$

This idea may be taken as a intuitive process of understanding an algorithm, since we have not presented the analysis for an abstract sequence converging to a. We strongly believe that this would help the beginners to perceive the asymptotic notations broadly. This analysis would certainly implies that central difference approximation might be a better one when compared with forward difference approximation.

Conclusion

In this expository article we have presented a few the asymptotic notations targeted to analyse the algorithms. in several complex variables, the theory of Riemann surfaces, deformation theory and analytic geometry – a diverse spread by any standards. This diversity, as I was to begin learning a decade later, was neither forced nor born of
We have illustrated the use of these asymptotic notations through examples. A few graphs were presented to give a geometric view to the readers.

References

[1] G. Brassard, Crusade for a better notation, ACM SIGACT News **17(1)** (1985) 60–64.

[2] T. H. Cormen, C. Stein, R. L. Rivest and C. E. Leiserson, introduction to algorithms, Third edition, The MIT press (2009).

[3] R. Gowtham, Computability and complexity analysis of decision problems, Master's thesis, Central University of Taminadu (2016).

[4] R. L. Graham, D. E. Knuth and O. Patashnik, Concrete mathematics: a foundation for computer science hardcover, Second edition, Addison Wesley (1994).

[5] M. Sipser, Introduction to the theory of computation, International Thomson Publishing (1996).

Remembering R. R. Simha

Kaushal Verma

Department of Mathematics, Indian Institute of Science, Bangalore 560 012, India

E-mail: kverma@iisc.ac.in

My first encounter with him was an indirect one. It came about in the summer of 1995 when I was attempting to work through his short and elegant paper on the Carathéodory metric of an annulus in the complex plane. It was a paper that I had come upon purely by accident – a fortuitous one as I now realize. His name had been unfamiliar to me then, but the address for correspondence was not – it read 'Tata Institute of Fundamental Research, Colaba, Bombay 400005, India'. The style of writing was direct but demanded a focused engagement. Intrigued both by the beautiful explicit formula for the Carathéodory metric and the unfamiliar author, who was based in a city that was so familiar, I decided to look around for his other papers. With a little bit of effort, there was enough to think about. His work touched upon many foundational topics any compulsion. It was bound by a common thread of eloquent brevity and was vitally alive in his ever cheerful personality – a personality marked with easy humor, warmth and care even towards those he was meeting for the first time. Our first meeting was a brief one; so brief that I remember very little of what transpired. However, I'm sure of some things – that it was towards the end of 2005, that it was in a corridor of the Dept. of Mathematics at IISc and more importantly, his gentle request of addressing him either as 'Raya Simha' or just 'Simha'. I chose the latter, perhaps being mindful of the number of years that separated us. He agreed and from that moment onward till we met in early September 2017, a little before his untimely demise, he was 'Simha' to me on all platforms of engagement, personal or otherwise. It did not take very long for the years