Chapter 1

Introduction to Algorithms, Correctness and Efficiency.

Computer science differs from other scientific disciplines in that it focuses not on solving or making discoveries, but on questioning how good is our current understanding. The fact that one has successfully come up with an idea for a certain problem immediately raises the question of optimality. At the most basic level, we would like to answer what is the 'best' program that exists for a particular problem. To do so, we must have a notation that allows us to determine if an algorithm is indeed solving the task, quantify its performance, and compare it to other algorithms. In this chapter, we introduce this basic notation. The chapter is divided into two main parts: the first is about induction, a mathematical technique for proving claims, and the second presents asymptotic notation, which we use to describe the behavior of algorithms over large inputs.

Note 1: text for right-hand side of pages, it is set justified.

1.1 Peaks-Finding.

Example 1.1.1 (Leading Example.). *Consider an n-length array A such that $A_1, A_2, ..., A_n \in \mathbb{R}$. We will say that A_j is a peak (local minimum) if it is greater than its neighbors. Namely, $A_i \geq A_{i\pm1}$ if $i \pm 1 \in [n]$. Whenever $i \pm 1$ is not in the range $[n]$, we will define the inequality $A_i \geq A_{i\pm1}$ to hold trivially. For example, for $n = 1$, $A_1 = A_n$ is always a peak. Write an algorithm that, given A, returns the position of an arbitrary peak.*

Example 1.1.2. *Warming up. How many peaks do the following arrays contain?*

1. $A[i] = 1 \; \forall i \in [n]$
2. $A[i] = \begin{cases} i & i < n/2 \\ n/2 - i & \textit{else} \end{cases}$
3. $A[i] = i \; \forall i \in [n]$

1.2 Naive solution.

For capturing an understanding of the problem, let's study a simple solution before suggesting a much more interesting one. Consider the algorithm examining each of the items A_i one by one.

Result: returns a peak of $A_1...A_n \in \mathbb{R}^n$
1 **for** $i \in [n]$ **do**
2   **if** A_i *is a peak* **then**
3   **return** i
4   **end**
5 **end**

Algorithm 1: naive peak-find alg.

Correctness. We will say that an algorithm is correct, with respect to a given task, if it computes the task for any input. Let's prove that the above algorithm is doing the job.

Proof. Assume towards contradiction that there exists an n-length array A such that the algorithm peak-find fails to find one of its peaks, in particular, the Alg. returns $j' \in [n]$ such that $A_{j'}$ is not a peak. Denote by j the first position of a peak in A, and note that if the algorithm gets to line (2) in the jth iteration then either it returns j or A_j is not a peak.

Hence it must hold that $j' < j$. But a satisfaction of the condition on line (2) can happen only if $A_{j'}$ is a peak, which contradicts the minimality of j. $\square$

Running Time. Question, How would you compare the performance of two different algorithms? What will be the running time of the naive peak-find algorithm? On the lecture you will see a well-defined way to treat such questions, but for the sake of getting the general picture, let's assume that we pay for any comparison a quanta of processing time, and in overall the checking if an item in a given position is a peak we pay at most $c \in \mathbb{N}$ time, a constant independent on n.

1.3 An attempt for sophisticated solution.

Result: returns a peak of $A_1...A_n \in \mathbb{R}^n$
$i \leftarrow \lceil n/2 \rceil$
if A_i *is a peak* **then**
    return i
end
else
    return find-peak($A_i, A_{i+1}..A_n$)
end

Algorithm 2: fail attempt for more sophisticated alg.

Result: returns a peak of $A_1 ... A_n \in \mathbb{R}^n$
$i \leftarrow \lceil n/2 \rceil$
if A_i *is a peak* **then**
  return i
end
else if $A_{i-1} \leq A_i$ **then**
  return find-peak($A_i, A_{i+1} .. A_n$)
end
else
  return find-peak($A_1, A_2, A_3 .. A_{i-1}$)
end

Algorithm 3: sophisticated alg.

1.4 Sophisticated solution.

1.5 Induction.

Suppose that a teacher, who is standing in front of his class, is willing to prove that he can reach the door at the corner. One obvious way to do so is to actually reach the door; that is, move physically to it and declare success. For small classes containing a small number of students, this protocol might even be efficient, lasting less than several seconds. But what if the class is really big, maybe the length and width of a football stadium? In that case, proving by doing might take time. So the obvious question to ask is, what else can we do? Is there a more efficient way to prove this?

Indeed, there is. Instead of proving that he can reach the door, he can prove that while he do not stand next to the door, nothing can stop him from keeping moving forward. If that is indeed the case, then it's clear that not reaching the door in the end would be a contradiction to being just one step away from it (why?), which, in turn, would also contradict being two steps away from it. Repeating this argument leads to a contradiction for the fact that the teacher was in the classroom at the beginning.

What is induction?

1. A mathematical proof technique. It is essentially used to prove that a property $P(n)$ holds for every natural number n.

2. The method of induction requires two cases to be proved:

   (a) The first case, called the base case, proves that the property holds for the first element.

   (b) The second case, called the induction step, proves that if the property holds for one natural number, then it holds for the next natural number.

3. The domino metaphor.

The two types of induction, their steps, and why it makes sense (Strong vs Weak) - Emphasize the change in the induction step.

Example 1.5.1 (Weak induction). *Prove that* $\forall n \in \mathbb{N}, \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$.

Proof. Base: For $n = 1$, $\sum_{i=0}^{1} 1 = 1 = \frac{(1+1)\cdot 1}{2}$. Assumption: Assume that the claim holds for n.Step:

$$\sum_{i=0}^{n+1} i = \left(\sum_{i=0}^{n} i\right) + n + 1 = \frac{n(n+1)}{2} + n + 1$$
$$= \frac{n(n+1) + 2\cdot(n+1)}{2} = \frac{(n+1)(n+2)}{2}$$

□

Example 1.5.2 (Weak induction.). *Let* $q \in \mathbb{R}/\{1\}$, *consider the geometric series* $1, q, q^2, q^3 \ldots q^k \ldots$ *Prove that the sum of the first* k *elements is*

$$1 + q + q^2 + \ldots + q^{k-1} + q^k = \frac{q^{k+1} - 1}{q - 1}$$

Proof. Base: For $n = 1$, we get $\frac{q^{k+1}-1}{q-1} = \frac{q-1}{q-1} = 1$. Assumption: Assume that the claim holds for k. then: Step:

$$1 + q + q^2 + \ldots + q^{k-1} + q^k + q^{k+1} = \frac{q^k - 1}{q - 1} + q^{k+1} = \frac{q^{k+1} - 1 + q^{k+1}(q-1)}{q-1} =$$
$$\frac{q^{k+1} - 1 + q^{k+2} - q^{k+1}}{q - 1} = \frac{q^{k+2} - 1}{q - 1}$$

□

Example 1.5.3 (Strong induction). *Let there be a chocolate bar that consists of* n *square chocolate blocks. Then it takes exactly* $n - 1$ *snaps to separate it into the* n *squares no matter how we split it.*

Proof. By strong induction. Base: For $n = 1$, it is clear that we need 0 snaps. Assumption: Assume that for **every** $m < n$, this claim holds.

Step: We have in our hand the given chocolate bar with n square chocolate blocks. Then we may snap it anywhere we like, to get two new chocolate bars: one with some $k \in [n]$ chocolate blocks and one with $n - k$ chocolate blocks. From the induction assumption, we know that it takes $k - 1$ snaps to separate the first bar, and $n - k - 1$ snaps for the second one. And to sum them up, we got exactly

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

snaps. □

1.6 Asymptotic Notations.

Definition 1.6.1. *Let $f, g : \mathbb{N} \to \mathbb{R}^+$. We say that $f(n) = O(g(n))$ if $\exists N \in \mathbb{N}, \exists c > 0$ s.t. $\forall n \geq N \ : \ f(n) \leq c \cdot g(n)$*

Example 1.6.1. *For exmaple, if $f(n) = n + 10$ and $g(n) = n^2$, then $f(n) = O(g(n))$ (Draw the graphs) for $n \geq 5$: $f(n) = n+10 \leq n+2n = 3n \leq n \cdot n = n^2$*

Definition 1.6.2. *Let $f, g : \mathbb{N} \to \mathbb{R}$ We say that $f(n) = \Omega(g(n))$ if $g(n) = O(f(n))$, equivalently, $\exists N \in \mathbb{N}, \exists c > 0$ s.t. $\forall n \geq N c_0 g(n) \leq f(n)$*

Example 1.6.2. *Also if $f(n) = 5n$ and $g(n) = n^2$, then $f(n) = O(g(n))$ (Now discuss intuition - no matter how much we "stretch" f, g is still the winner)*

Definition 1.6.3. *Let $f, g : \mathbb{N} \to \mathbb{R}$, We say that $f(n) = \Omega(g(n))$ if: $\exists N \in \mathbb{N}, \exists c > 0$ s.t $\forall n \geq N \ f(n) \geq c \cdot g(n)$.*

Example 1.6.3. *For exmaple, if $f(n) = n + 10$ and $g(n) = n^2$, then $g(n) = \Omega(f(n))$*

Definition 1.6.4. *Let $f, g : \mathbb{N} \to \mathbb{R}$, We say that $f(n) = \Theta(g(n))$ if: $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ That is, we say that $f(n) = \Theta(g(n))$ if: $\exists N \in \mathbb{N}, \exists c_1, c_2 > 0$ s.t. $\forall n \geq N \ c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$*

Example 1.6.4. *For every $f : \mathbb{N} \to \mathbb{R}, f(n) = \Theta(f(n))$*

Example 1.6.5. *If $p(n) = n^5$ and $q(n) = 0.5n^5 + n$, then $p(n) = \Theta(q(n))$*

But why is this example true? This next Lemma helps for intuition:

Lemma 1.6.1. $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) = O(g(n))$

Proof. Assume that $l = \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$. Then for some $N \in \mathbb{N}$ we have that for all $n \geq N$: $\frac{f(n)}{g(n)} < l + 1 \Rightarrow f(n) < (l + 1)g(n)$ Which is exactly what we wanted. □

1.7 Examples with proofs.

Claim 1.7.1. $n = O(2^n)$

(This must seem very silly, but even though we have a strong feeling it's true, we still need to learn how to PROVE it)

Proof. We will prove by induction that $\forall n \geq 1$, $2^n \geq n$, and that will suffice. Basis: $n = 1$, so it is clear that: $n = 1 < 2 = 2^n$ Assumption: Assume that $n < 2^n$ for some n. Step: We will prove for $n + 1$. It holds that:

$$n + 1 < 2^n + 1 < 2^n + 2^n = 2^{n+1}$$

□

Claim 1.7.2. *Let $p(n)$ be a polynomial of degree d and let $q(n)$ be a polynomial of degree k. Then:*

1. $d \leq k \Rightarrow p(n) = O(q(n))$ *(set upper bound over the quotient)*
2. $d \geq k \Rightarrow p(n) = \Omega(q(n))$ *(an exercise)*
3. $d = k \Rightarrow p(n) = \Theta(g(n))$ *(an exercise)*

Proof. Proof (Of 1) First, let's write down $p(n), g(n)$ explicitly:

$$p(n) = \sum_{i=0}^{d} \alpha_i n^i,\ g(n) = \sum_{j=0}^{k} \beta_j n^j$$

Now let's manipulate their quotient:

$$\frac{p(n)}{q(n)} = \frac{\sum_{i=0}^{d} \alpha_i n^i}{\sum_{j=0}^{k} \beta_j n^j} = \frac{\sum_{i=0}^{d} \alpha_i n^i}{\sum_{j=0}^{k} \beta_j n^j} \cdot \frac{n^{k-1}}{n^{k-1}} = \frac{\sum_{i=0}^{d} \alpha_i n^{i-k+1}}{\sum_{j=0}^{k} \beta_j n^{j-k+1}} \leq$$

$$\leq \frac{\sum_{i=0}^{d} \alpha_i}{\beta_k} < \infty$$

And now we can use the lemma that we have proved earlier. $\square$

1.8 Logarithmic Rules.

Just a quick reminder of logarithmic rules:

1. $log_a x \cdot y = log_a x + log_a y$
2. $log_a \frac{x}{y} = log_a x - log_a y$
3. $log_a x^m = m \cdot log_a x$
4. Change of basis: $\frac{log_a x}{log_a y} = log_y x$

And so we get that:

Remark 1.8.1. *For every $x, a, b \in \mathbb{R}$, we have that $log_a x = \Theta(log_b x)$*

Example 1.8.1. *Let $f(n)$ be defined as:*

$$f(n) = \begin{cases} f\left(\lfloor \frac{n}{2} \rfloor\right) + 1 & \textit{for } n > 1 \\ 5 & \textit{else} \end{cases}$$

Let's find an asymptotic upper bound for $f(n)$. let's guess $f(n) = O(\log(n))$.

Proof. We'll prove by strong induction that : $f(n) < c\log(n) - 1$ for $c = 8$ And that will be enough (why? This implies $f(n) = O(\log(n))$). Base: $n = 2$. Clearly, $f(2) = 6 < 8$ Assumption: Assume that for every m ¡ n, this claim holds. Step: Then we get:

$$\begin{aligned} f(n) = f\left(\lfloor \frac{n}{2} \rfloor\right) + 1 \leq c\log\left(\lfloor \frac{n}{2} \rfloor\right) + 1 \\ \leq c\log(n) - c\log(2) + 1 \leq c\log(n) \quad \text{for } c = 8 \end{aligned}$$

$\square$