Honors Discrete Mathematics: Lecture 12 Notes

Gerandy Brito Spring 2022

Sarthak Mohanty

Algorithmic Analysis

In Computer Science, one of the main goals is to find algorithms that are fast, or *efficient*. Usually we can write these algorithms in the form $T(n)$ as the number of operations (worst case!) for an input of size n.

In prior CS courses, you have probably considered the best-case, average-case, and worst-case runtime of algorithms. In this course, however, we will usually consider only the worst-case runtime.

TA Remark. There are a few reasons for this. The following are taken from CLRS.

- The worst-case running time of an algorithm gives us an upper bound on the runtime time for any input. Knowing it provides a guarantee that the algorithm will never take any longer. We need not make some educated guess about the running time and hope that it never gets much worse.
- For some algorithms, the worst case occurs fairly often. For example, in searching a database for a particular piece of information, the searching algorithm's worst case will often occur when the information is not present in the database. In some applications, searches for absent information may be frequent.
- The "average case" is often roughly as bad as the worst case. Suppose that we randomly choose n numbers and apply insertion sort. How long does it take to determine where in subarray $A[1 \dots, j-1]$ to insert element $A[j]$? on average, half the elements in $A[1, \dots, j-1]$ to insert element $A[j]$? On average, half the elements in $A[1 \dots, j-1]$ are less than $A[j]$, and half the elements are greater. On average, therefore, we check half of the subarray $A[1, \dots, j-1]$ and so t_j is about $\frac{j}{2}$. The result average-case running time turns out to be a quadratic function of the input size, just like the worst-case running time.

Recursive Algorithms

Definition. A *recurrence* is an equation or inequality that describes a function in terms of its value on smaller inputs.

Recursive algorithms can come in many forms:

- A recursive algorithm that loops through the input to eliminate one item, such as the example we will show below.
- A recursive algorithm that divides the input into halves or more parts. Such algorithms are commonly known as divide-and-conquer algorithms, and will be explored further in CS 3510/3511.

Our goal in this course will be to 1) derive a recurrence formula from the given recursive algorithm and 2) solve the recurrence formula to find an explicit, or *closed-form*, expression of the recurrence.

To illustrate this, let us return to topological sorting. Recall that for any poset $(S, \subseteq)$, a topological sorting can be found using the following algorithm

1. Find a minimal element of S, s_1.
2. Iterate on $S \setminus \{s_1\}$.

For some poset of size n, step 1 takes $n-1$ comparisons. Hence the runtime is of the form

$$T(n) = T(n-1) + n - 1,\ T(1) = 0.$$

Now we will attempt to solve the recurrence relation. There are many methods for doing so; in this course, we will cover two of them: the *iteration method* and the *substitution method*.

Iteration Method

Note that

$$\begin{aligned} T(n) &= T(n-1) + n - 1 \\ &= (T(n-2) + n - 2) + n - 1 \\ &= T(n-3) + (n-3) + (n-2) + (n-1) \\ &\vdots \\ &= T(1) + 1 + 2 + \cdots + (n-1) \\ &= \textstyle\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}. \end{aligned}$$

Substitution Method

Using repeated substitutions, we have

$$\begin{aligned} T(1) &= 0 = \tfrac{(1-1)\cdot 1}{2}, \\ T(2) &= T(1) + 2 - 1 = 1 = \tfrac{(2-1)\cdot 2}{2}, \\ T(3) &= T(2) + 3 - 1 = 3 = \tfrac{(3-1)\cdot 3}{2}, \\ &\vdots \\ T(n) &= T(n-1) + n - 1 = \tfrac{(n-1)n}{2}. \end{aligned}$$

We prove the n-th term guess by mathematical induction. Let $P(n)$ be the statement

$$T(n) = \tfrac{(n-1)n}{2}.$$

Base Case: $P(1)$ is true, since $T(1) = 0 = \frac{(1-1)\cdot 1}{2}$.
Inductive Step: Now let $n \in \mathbb{N}$ such that $P(n)$ is true. Then $T(n+1) = T(n) + n = \frac{(n-1)n}{2} + n = \frac{(n+1)(n)}{2}$, so $P(n+1)$ is true as well.
Conclusion: Therefore by induction, for all $n \geq 1$, $P(n)$ is true.

Calculator Example

We have a calculator, with two inputs given by : $a \in \mathbb{R} \setminus \{0\}$ and $n \in \mathbb{N}$. The output is given by a^n.
To create this output, the calculator executes the following function

Algorithm 1 Calculator(a, n)

```
if n = 1 then
    output a
else if n > 1 then
    Calculator(a, n - 1) times a
end if
```

TA Remark. There is no coding prerequisite for this course. Any pseudocode given on homeworks/exams should be easily understandable.

The recurrence formula for this algorithm is

$$T(n) = T(n-1) + 1.$$

It is left as an exercise for the students to find that $T(n) = n$.

Algorithm 2 Calculator(a, n)

if $n = 1$ **then**
  return a
else if n is even **then**
  $cal(a, \frac{n}{2}) \times Cal(a, \frac{n}{2}$.
else if n is odd **then**
  $cal(a, \frac{n-1}{2}) \times cal(a, \frac{n-1}{2}) \times a$
end if

Consider another calculator function:
The recurrence formula for this algorithm is $T(n) \leq T(n/2) + 2(2isworstcase)$. Consider $n = 2^k$. Then

$$T(n) = T(\tfrac{n}{2}) + 1, T(1) = 0.$$

We will first solve this recursion using the iteration method. Note that

$$\begin{aligned} T(n) &= T(\tfrac{n}{2}) + 1 \\ &= \big(T(\tfrac{n}{4}) + 1\big) + 1 \\ &= \big(T(\tfrac{n}{8}) + 1\big) + 1 + 1 \\ &\vdots \\ &= T(\tfrac{n}{2^k}) + k. \end{aligned}$$

Let $2^k = n$, then $k = \log_2(n)$, so

$$T(n) = T(1) + \log_2(n) = \log_2(n).$$

We can also use the substitution method to obtain the same answer. Using repeated substitutions, we have

$$\begin{aligned} T(1) &= 0 = \log_2(1), \\ T(2) &= T(1) + 1 = 1 = \log_2(2), \\ T(4) &= T(2) + 1 = 2 = \log_2(4), \\ &\vdots \\ T(n) &= T(\tfrac{n}{2}) + 1 = \log_2(n). \end{aligned}$$

Let $P(k)$ be the statement

$$T(n) = \log_2(n), \text{ where } n = 2^k$$

Base Case: $P(0)$ is true, since $T(0) = 0 = \log_2(1)$
Inductive Step: Now let $k \in \mathbb{N}$ such that $P(k)$ is true. Then $T(2^{k+1}) = T(\frac{2^{k+1}}{2}) + 1 = T(2^k) + 1$. So $P(k+1)$ is true as well.
Conclusion Therefore by induction, for all $k \geq 1$, $P(k)$ is true.