CS 341 — Lecture 4

Bartosz Antczak *Instructor: Bin Ma* *September 19, 2017*

Recall

When we say to **reduce** problem X to problem Y, that means using the algorithm for Y to solve X.

4.1 Divide and Conquer

Example 4.1.1. *Recurrence 1*

Solve $T(n) \leq 2 \cdot T(\frac{n}{2}) + \sqrt{n}$.

We know from analyzing the recurrence relation for merge sort that $T'(n) = 2 \cdot T(\frac{n}{2}) + n \in O(n \log n)$, so we theorize that $T(n) = O(n) \implies T(n) \leq cn$.

Proof

- It is trivial to see that the base case when $n = 1$ is true
- Assume that the statement is correct for $n < m$. Now let's prove the case for $n = m$:

$$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + \sqrt{n} \tag{4.1}$$
$$\leq 2 \cdot \frac{cn}{2} + \sqrt{n} \qquad \text{(By our hypothesis)} \tag{4.2}$$
$$= cn + \sqrt{n} \tag{4.3}$$

  We've arrived at a problem. Notice that dangling $\sqrt{n}$. We cannot finish a proof by arriving at a weaker property than stated. So we need another approach. Let's prove that $T(n) \leq cn - 3\sqrt{n}$:

  - It is trivial to see that the base case when $n = 1$ is true
  - Assume that the statement is true for $n < m$. Now let's prove for $n = m$:

$$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + \sqrt{n} \tag{4.4}$$
$$\leq 2 \cdot \left(c \cdot \frac{n}{2} - 3\sqrt{\frac{n}{2}}\right) + \sqrt{n} \tag{4.5}$$
$$= cn - (3\sqrt{2} - 1)\sqrt{n} \tag{4.6}$$
$$< cn - 3\sqrt{n} \qquad \text{(Because } 3\sqrt{2} - 1 < 4\text{)} \quad \square \tag{4.7}$$

Example 4.1.2. *Recurrence 2*

Solve $T(n) = 2 \cdot T(\sqrt{n}) + \log_2 n$.

Let's use substitution! Let $n = 2^m$. We define

$$S(m) = T(2^m) \tag{4.8}$$
$$= 2 \cdot T(2^{\frac{m}{2}}) + m \tag{4.9}$$
$$= 2 \cdot S\left(\frac{m}{2}\right) + m \tag{4.10}$$

We see that $S(m) = O(m \log m)$. From this, we see that

$$T(n) = S(\log n) \tag{4.11}$$
$$= O(\log n \cdot \log \log n) \quad \square \tag{4.12}$$

4.1.1 Master Theorem

Given $T(n) = a \cdot T(\frac{n}{b}) + n^c$, then

$$T(n) = \begin{cases} \Theta(n^c) & c > \log_b a \\ \Theta(n^c \log n) & c = \log_b a \\ \Theta(n^{\log_b a}) & c < \log_b a \end{cases}$$

Let's prove the case where $c > \log_b a \implies \Theta(n^c)$.

Proof

We want to show that $T(n) = \gamma n^c$ for some γ that will be defined later.

- It is trivial to see that the base case when $n = 1$ is true
- Assume that the statement is correct for $n < m$. Now let's prove the case for $n = m$:

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + n^c \tag{4.13}$$
$$= a \cdot \gamma\left(\frac{n}{b}\right)^c + n^c \qquad \text{(By our inductive hypothesis)} \tag{4.14}$$
$$= (a \cdot \gamma \cdot b^{-c} + 1) \cdot n^c \tag{4.15}$$

Now we only need to determine γ such that $a \cdot \gamma \cdot b^{-c} + 1 = \gamma$. This can be simply solved to arrive at

$$\gamma = \frac{1}{1 - ab^{-c}}$$

Example 4.1.3. *Movie Rankings*

You and a friend are ranking films. You each rank your films in an integer array A where the movie defined by index i has a rank of $A[i]$. How do you compare the movies you like versus your friend? One approach is to count the number of inversions (how many films your friend liked more than you). For instance:

```
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 3, 5, 8, 2, 4, 7, 6]
```

There exist 8 inversions between these two arrays.

Algorithm I

The trivial algorithm: try every pair. Has a run time of $O(n^2)$.

Algorithm II

```
SortAndCount:
(S = array, k = number of inversions)
1. If n < 3, then solve it trivially
2. (S1, k1) = SortAndCount(A[1 .. n/2])
3. (S2, k2) = SortAndCount(A[n/2 + 1 ... n])
4. (S1, k3) = MergeAndSort(S1, S2)
5. return (S, k1 + k2 + k3)
```

The code for `MergeAndCount` is:

- **Input:** `A` (length `m`) and `B` (length `n`) sorted
- **Output:** C (sorted list) and `k` (number of inversions)

```
i = 1, j = 1, k = 0, C = {}
while (i < m && j < n)
    if (A[i] < B[j]) then
        C = C + A[i]
        i++
    else
        C = C + B[j]
        k = k + (m-i+1)
        j++

C = C + A[i, m] + B[j, n]
return (C, k)
```