DA4006
Tutorial 3

Patricia Ebert & Anna Lindeberg

Spring term 2026

Today's Agenda

- How can we solve recurrence relations?
- Substitution Method
- Master Theorem
- Determine runtime of an algorithm
- General questions and suggestions
- A break somewhere!

How can we solve recurrence relations?

1. Substitution Method
2. Master Theorem
3. Recurrence Tree $\leftarrow$ possibly next time

Substitution Method

In essence you need to 1. guess correctly and 2. show with induction that your guess is correct.

Substitution Method

In essence you need to 1. guess correctly and 2. show with induction that your guess is correct.

Prove that $T(n) \in O(n \log n)$, for

$$T(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n, & \text{if } n > 1. \end{cases}$$

Prove that $T(n) \in O\left(n^2\right)$, for

$$T(n) = \begin{cases} 1, & \text{if } n = 1 \\ T(n-1) + n, & \text{if } n > 1. \end{cases}$$

Bonus: What about $T(n) = 2T(\sqrt{n}) + \log n$ with $T(k) = 1$ for $k \leq 2$?

Master Theorem

Master Theorem [simplified version]

Let $a \geq 1$, $b > 1$ and $d \geq 0$ be constants and $n \in \mathbb{N}_{\geq 1}$. If $T(n) = aT(n/b) + \Theta(n^d)$, then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log_2 n) & \text{if } a = b^d \\ \Theta(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

Intuition

(1) fewer branches than by how much we decrease the effort per recursion, i.e., the next step of the recursion tree is less expensive than the previous one

Master Theorem

Master Theorem [simplified version]

Let $a \geq 1$, $b > 1$ and $d \geq 0$ be constants and $n \in \mathbb{N}_{\geq 1}$. If $T(n) = aT(n/b) + \Theta(n^d)$, then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log_2 n) & \text{if } a = b^d \\ \Theta(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

Intuition

(1) fewer branches than by how much we decrease the effort per recursion, i.e., the next step of the recursion tree is less expensive than the previous one

(2) the next level of the recursion tree is as expensive than the previous one and we have $\log_b(n)$ levels

Master Theorem

Master Theorem [simplified version]

Let $a \geq 1$, $b > 1$ and $d \geq 0$ be constants and $n \in \mathbb{N}_{\geq 1}$. If $T(n) = aT(n/b) + \Theta(n^d)$, then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log_2 n) & \text{if } a = b^d \\ \Theta(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

Intuition

(1) fewer branches than by how much we decrease the effort per recursion, i.e., the next step of the recursion tree is less expensive than the previous one

(2) the next level of the recursion tree is as expensive than the previous one and we have $\log_b(n)$ levels

(3) the next level of the recursion tree is more expensive than the previous one and we have $n^{\log_b(a)}$ base cases

Master Theorem

Master Theorem [simplified version]

Let $a \geq 1$, $b > 1$ and $d \geq 0$ be constants and $n \in \mathbb{N}_{\geq 1}$. If $T(n) = aT(n/b) + \Theta(n^d)$, then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log_2 n) & \text{if } a = b^d \\ \Theta(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

Determine $f(n)$ such that $T(n) \in \Theta(f(n))$ using the Master Theorem for all four T below with $T(1) = 1$ and

1. $T(n) = 2T(n/2) + n$
2. $T(n) = 3T(n/3) + \sqrt{n}$
3. $T(n) = 2^n T(n/2) + n$
4. $T(n) = 2T(n/2) + n^3 + 12n^2 - 7$

Determine the Runtime

```
fun(n)
1: j ← 1
2: while j^2 ≤ n do
3:     k ← j
4:     while k > 1 do
5:         k ← k/100
6:     j ← j + 1
7: return j
```

Show that the runtime of `fun`(n) is $O(\sqrt{n}\log(n))$.

Adaption of `Insertion-Sort`

Adapt `Insertion-Sort` so that it sorts non-increasing instead of non-decreasing (i.e. sort large to small).

Adaption of Insertion-Sort

Adapt Insertion-Sort so that it sorts non-increasing instead of non-decreasing (i.e. sort large to small).

Normal Insertion-Sort

1: **for** $j = 1$ to $A.length$-1 **do**
2: $\quad key \leftarrow A[j]$
3: $\quad i \leftarrow j - 1$
4: $\quad$ **while** $i \geq 0$ and $A[i] > key$ **do**
5: $\quad\quad A[i+1] \leftarrow A[i]$
6: $\quad\quad i \leftarrow i - 1$
7: $\quad A[i+1] \leftarrow key$

Adaption of Insertion-Sort

Adapt Insertion-Sort so that it sorts non-increasing instead of non-decreasing (i.e. sort large to small).

```
1: for j = 1 to A.length-1 do
2:     key ← A[j]
3:     i ← j − 1
4:     while i ≥ 0 and A[i] < key do
5:         A[i + 1] ← A[i]
6:         i ← i − 1
7:     A[i + 1] ← key
```