Reading 4 - Recursion Tree Method

4.1 - The maximum-subarray problem

Tasks

- *Know the runtime of the brute-force solution and why it is what it is*
- *Understand the rephrased problem after the "transformation" described in the book*
- *Understand the solution using D&C, specifically the crucial fine_max_crossing_subarray() function*
- *Have an idea how equation 4.7 came to be*

Brute Force

- **A Brute Force Solution**: try every input possible one by one in the hopes you get lucky.
  - **Runtime**: $\theta(n^2)$

A Transformation

maximum subarray: a range of an array that produces the highest sum - *Example*: In the array `[-3, -16, -23, 18, 20, -7, 12, -5, -22]`, `[-23, 18, 20, -7, 12]` adds up to 43, so this range is the **max subarray** - Flaws: - If all array numbers are positive, then the whole array is *ALWAYS* the **max subarray**.

Getting the **maximum subarray** of your array is considered to be a **transformation**.

A solution using divide-and-conquer

Our goal is to split the array into two subarrays. To do that, we need to define:

- `low`
- `medium`
- `high`

For example, we can do:

```
master_array = [-3, -16, -23, 18, 20, -7, 12, -5, -22]
subarr_1 = master_array[0:len(master_array/2)]
subarr_2 = master_array[(len(master_array/2) + 1):len(master_array)]
```

Understanding how equation 4.7 came to be

Equation 4.7:

$$T(n) = \begin{cases} \theta(1) & \text{if } n = 1, \\ 2T(n/2) + \theta(n) & \text{if } n > 1. \end{cases}$$

4.4 - The recursion-tree method for solving recurrences

Tasks

- *Know how to expand the recurrence*
- *Know how to figure the costs per level ("breadth")*
- *Know how many levels there are ("depth")*

Expanding Recurrence

Determining costs per level ("breadth")

Determining number of levels ("depth")