Divide-and-Conquer

Shusen Wang

Revisit Merge Sort

Merge Sort via Recursion

```
void mergesort(int A[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergesort(A, left, mid);
        mergesort(A, mid+1, right);
        merge(A, left, mid, right);
    }
}
```

Merge Sort via Recursion

```
void mergesort(int A[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergesort(A, left, mid);          // T(n/2) time complexity.
        mergesort(A, mid+1, right);       // T(n/2) time complexity.
        merge(A, left, mid, right);       // c · n time complexity.
    }
}
```

Recurrence: $T(n) = 2 \cdot T(n/2) + c \cdot n.$

Merge Sort via Recursion

Recurrence: $T(n) = 2 \cdot T(n/2) + c \cdot n.$

Time Complexity of Merge Sort

Recurrence: $T(n) = 2 \cdot T(n/2) + c \cdot n.$

- ➔ $T(n) = n \cdot T(1) + c \cdot n \cdot \log n.$
- We know that $T(1) = O(1).$
- Thus, $T(n) = O(n \cdot \log n).$

More recurrence relations…

Algorithm	Recurrence relation:	Time complexity:
Quick Select	$T(n) = T(n/2) + cn$	$T(n) = O(n)$

Algorithm	Recurrence relation:	Time complexity:
Quick Select	$T(n) = T(n/2) + cn$	$T(n) = O(n)$
Merge Sort	$T(n) = 2\,T(n/2) + cn$	$T(n) = O(n \log n)$

Algorithm	Recurrence relation:	Time complexity:
Quick Select	$T(n) = T(n/2) + cn$	$T(n) = O(n)$
Merge Sort	$T(n) = 2\,T(n/2) + cn$	$T(n) = O(n \log n)$
What if	$T(n) = 4\,T(n/2) + cn$	?

Solution

Recurrence relation: $T(n) = 4 \cdot T(n/2) + c \cdot n.$

- ➔ $\dfrac{T(n)}{n} = 2 \cdot \dfrac{T(n/2)}{n/2} + c,\qquad \dfrac{T(n/2)}{n/2} = 2 \cdot \dfrac{T(n/4)}{n/4} + c,$
  $\dfrac{T(n/4)}{n/4} = 2 \cdot \dfrac{T(n/8)}{n/8} + c,\qquad \dfrac{T(n/8)}{n/8} = 2 \cdot \dfrac{T(n/16)}{n/16} + c,\ \cdots$

- ➔ $\dfrac{T(n)}{n} = 2 \cdot \dfrac{T(n/2)}{n/2} + c = 4 \cdot \dfrac{T(n/4)}{n/4} + c \cdot (1 + 2)$
  $= 8 \cdot \dfrac{T(n/8)}{n/8} + c \cdot (1 + 2 + 4) = 16 \cdot \dfrac{T(n/16)}{n/16} + c \cdot (1 + 2 + 4 + 8)$
  $= \cdots = n \cdot \dfrac{T(n/n)}{n/n} + c \cdot \left(1 + 2 + 4 + 8 + \dfrac{n}{2}\right)$
  $= n \cdot T(1) + c \cdot (n - 1).$

Algorithm	Recurrence relation:	Time complexity:
Quick Select	$T(n) = T(n/2) + cn$	$T(n) = O(n)$
Merge Sort	$T(n) = 2\,T(n/2) + cn$	$T(n) = O(n \log n)$
What if	$T(n) = 4\,T(n/2) + cn$	$T(n) = O(n^2)$

The Master Theorem

Algorithm	Recurrence relation:	Time complexity:
Quick Select	$T(n) = T(n/2) + cn$	$T(n) = O(n)$
Merge Sort	$T(n) = 2\,T(n/2) + cn$	$T(n) = O(n \log n)$
	$T(n) = 4\,T(n/2) + cn$	$T(n) = O(n^2)$
What if	$T(n) = 3\,T(n/2) + cn$	?

Algorithm	Recurrence relation:	Time complexity:
Quick Select	$T(n) = T(n/2) + cn$	$T(n) = O(n)$
Merge Sort	$T(n) = 2\,T(n/2) + cn$	$T(n) = O(n \log n)$
	$T(n) = 4\,T(n/2) + cn$	$T(n) = O(n^2)$
What if	$T(n) = 3\,T(n/2) + cn$	$T(n) = O\left(n^{\log_2 3}\right) \approx O(n^{1.6})$

What is the pattern?

Recurrence relation: $T(n) = a \cdot T(n/2) + c \cdot n.$

What is the pattern?

Recurrence relation: $T(n) = a \cdot T(n/2) + c \cdot n.$

Case 1: $a = 1$

- $T(n) = T(n/2) + cn$ ➔ $T(n) = O(n).$

Case 2: $a = 2$

- $T(n) = 2 \cdot T(n/2) + cn$ ➔ $T(n) = O(n \log n).$

What is the pattern?

Recurrence relation: $T(n) = a \cdot T(n/2) + c \cdot n.$

Case 1: $a = 1$

- $T(n) = T(n/2) + cn \quad \rightarrow \quad T(n) = O(n).$

Case 2: $a = 2$

- $T(n) = 2 \cdot T(n/2) + cn \quad \rightarrow \quad T(n) = O(n \log n).$

Case 3: $a > 2$

- $T(n) = 3 \cdot T(n/2) + cn \quad \rightarrow \quad T(n) = O\left(n^{\log_2 3}\right).$
- $T(n) = 4 \cdot T(n/2) + cn \quad \rightarrow \quad T(n) = O\left(n^{\log_2 4}\right).$
- $T(n) = 5 \cdot T(n/2) + cn \quad \rightarrow \quad T(n) = O\left(n^{\log_2 5}\right).$

The Master Theorem

Recurrence relation: $T(n) = a \cdot T(n/b) + c \cdot n.$

a: number of sub-problems.

b: factor by which input size shrinks.

cn: cost of creating the sub-problems and combining their solutions.

The Master Theorem

Recurrence relation: $T(n) = a \cdot T(n/b) + c \cdot n.$

- Suppose that $a \geq 1$, $b > 1$, and $c > 0$ are constants independent of n.
- The master theorem:

$$T(n) = \begin{cases} O(n), & \text{if } a < b; \\ O(n \log n), & \text{if } a = b; \\ O\left(n^{\log_b a}\right), & \text{if } a > b. \end{cases}$$

a: number of sub-problems.
b: factor by which input size shrinks.
cn: cost of creating the sub-problems and combining their solutions.

The Master Theorem

Recurrence relation: $T(n) = a \cdot T(n/b) + c \cdot n^d.$

a: number of sub-problems.

b: factor by which input size shrinks.

cn^d: cost of creating the sub-problems and combining their solutions.

The Master Theorem

Recurrence relation: $T(n) = a \cdot T(n/b) + c \cdot n^d$.

- Suppose that $a \geq 1$, $b > 1$, $c > 0$, and $d > 0$ are constants independent of n.

- The master theorem (more general):

$$T(n) = \begin{cases} O(n^d), & \text{if } a < b^d; \\ O(n^d \log n), & \text{if } a = b^d; \\ O(n^{\log_b a}), & \text{if } a > b^d. \end{cases}$$

a: number of sub-problems.

b: factor by which input size shrinks.

cn^d: cost of creating the sub-problems and combining their solutions.

Questions

Use master theorem to find $T(n)$

- Binary search: $T(n) = T\left(\frac{n}{2}\right) + c$, where c is constant.
- Block matrix multiplication: $T(n) = 8 \cdot T\left(\frac{n}{2}\right) + n^2$.
- Strassen algorithm: $T(n) = 7 \cdot T\left(\frac{n}{2}\right) + 4.5n^2$.
- Karatsuba algorithm: $T(n) = 3 \cdot T\left(\frac{n}{2}\right) + cn$, where c is constant.

Thank You!