Programação Dinâmica

Quando usar?

- Problema pode ser dividido em subproblemas com sobreposição.
- Utilizado para **otimizar recursões pesadas**, evitando recomputações.

Exemplo de código: Fibonacci recursivo e com **memoization**

```python
def fibonacci(n, memo={}):
    if n in memo:
        return memo[n]
    if n <= 1:
        return n
    memo[n] = fibonacci(n - 1, memo) + fibonacci(n - 2, memo)
    return memo[n]
```

Merge Sort Tree

Quando usar?

- Quando é necessário **responder consultas de contagem ou ordenação em intervalos**, como:
  → Quantos elementos ≤ k em [l, r]?
  → Qual o k-ésimo menor em [l, r]?
  → Quantos elementos estão dentro de um intervalo [a, b] em [l, r]?

Exemplo de código: Merge Sort Tree com Consulta de Elementos Menores ou Iguais a k

```python
import bisect

def buildTree(idx, ss, se, a, sTree):
    if ss == se:
        sTree[idx] = a[ss]
        return

    mid = (ss + se) // 2

    buildTree(2 * idx + 1, ss, mid, a, sTree)
    buildTree(2 * idx + 2, mid + 1, se, a, sTree)
    sTree[idx] = sorted(sTree[2 * idx + 1] + sTree[2 * idx + 2])

def queryRec(node, start, end, ss, se, k, a, sTree):
    if ss > end or start > se:
        return 0

    if ss <= start and se >= end:
        return bisect.bisect_right(sTree[node], k) - start

    mid = (start + end) // 2

    p1 = queryRec(2 * node + 1, start, mid, ss, se, k, a, sTree)
    p2 = queryRec(2 * node + 2, mid + 1, end, ss, se, k, a, sTree)
    if p1 + p2 < 0:
        return 0

    return p1 + p2

def query(start, end, k, a, n, sTree):
    return queryRec(0, 0, n - 1, start, end, k, a, sTree)
```