Projeto de Algoritmo - Divisão e Conquista

Projeto e Análise de Algoritmo — QXD0041

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Prof. Fabio Dias
fabiodias@ufc.br

Universidade Federal do Ceará

2º semestre/2024

Divisão e Conquista

Divisão e Conquista

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

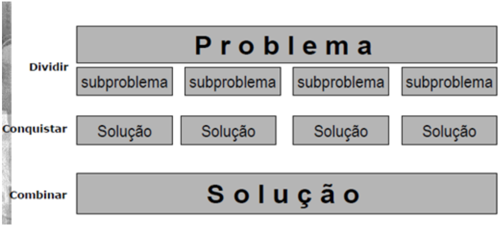
- A recursão parte do princípio que é mais fácil resolver problemas menores.
- Para certos problemas, podemos dividi-los em duas ou mais partes.
- Um algoritmo de divisão e conquista é aquele que resolve o problema desejado, combinando as soluções de um ou mais subproblemas, obtidas **recursivamente**.

Etapas de um Algoritmo de Divisão e Conquista

- **Dividir:** Se o problema não é suficientemente pequeno, divida-o em vários subproblemas que são similares ao problema original, mas de tamanho menor.

- **Conquistar:** Resolva os subproblemas recursivamente.

- **Combinar:** Combine as soluções desses subproblemas de maneira a obter uma solução para o problema original.



Algoritmo Genérico

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Divisao-Conquista(x)
se a instância x é suficientemente pequena **então**
    devolva Solucao(x)
senão
    decomponha x em instâncias menores $x_1, x_2, \cdots, x_k$
    para i **de** 1 **até** k **faça**
        $y_i \leftarrow$ Divisao-Conquista(x_i)
    combine as soluções y_i para obter uma solução y
    devolva y

Encontra o maior elemento em um vetor

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Dado um vetor de tamanho n, retorna o maior no vetor.
Solução usando o projeto de algoritmo incremental:

```c
int maior(int v[], int n) {
  int maior = v[0];
  for (int i = 1; i < n; i++)
    if (v[i] > maior)
      maior = v[i];
  return maior;
}
```

- Qual o tempo de execução desse algoritmo?

Encontra o maior elemento em um vetor

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- **Dividir:** Divide a entrada de tamanho n duas entradas menores, de tamanho aproximadamente $n/2$, se a entrada tiver tamanho $>= 2$.
- **Conquistar:** Encontre o maior elemento das duas entradas menores recursivamente usando o algoritmo.
- **Combinar:** O maior elemento da entrada original é o máximo entre o maior das duas entradas menores.

Encontra o maior elemento em um vetor

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Solução usando o projeto de algoritmo de divisão e conquista:

```
int maiorDC(int v[], int ini, int fim) {
  if(ini == fim) return v[ini];
  else{
    //Fase de Divisao
    int meio = (ini + fim)/2;

    //Fase de Conquista
    int maior1 = maiorDC(v, ini, meio);
    int maior2 = maiorDC(v, meio + 1, fim);

    //Fase de Combinacao
    if(maior1 >= maior2) return maior1;
    else return maior2;
  }
}
```

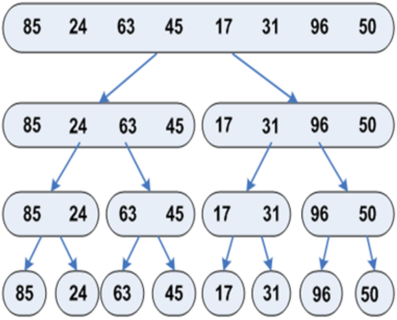
- Qual o tempo de execução desse algoritmo?

Ordenação Merge-sort

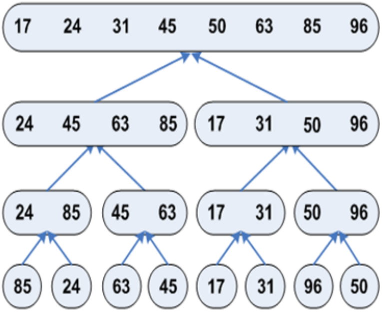
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- **Dividir:** Divide a entrada de tamanho n duas entradas menores, de tamanho aproximadamente $n/2$, se a entrada tiver $tamanho >= 2$.
- **Conquistar:** Ordene as duas entradas menores recursivamente usando o Merge-Sort.
- **Combinar:** Intercale as duas entradas menores ordenadas para produzir a entrada original ordenada.

Ordenação Merge-sort



(a) Fase de Divisão



(b) Fase de Conquista

Merge-Sort

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Dado um vetor v de tamanho n....

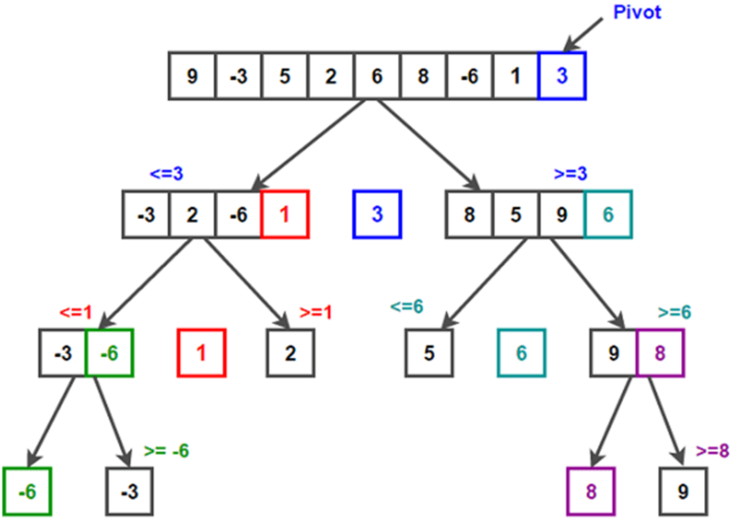
```
void mergeSort(int vet[], int ini, int fim) {
  if (ini < fim){
    //Fase de Divisao
    int meio = (ini + fim)/2;

    //Fase de Conquista
    mergeSort(vet, ini, meio);
    mergeSort(vet, meio + 1, fim);

    //Fase de Combinacao
    merge(vet, ini, meio, fim);
  }
}
```

- Qual o tempo de execução desse algoritmo?

Ordenação Quick-sort

- **Dividir:** Utilizando o particiona, divide a entrada de tamanho n duas entradas menores, possivelmente de tamanho diferentes, se a entrada tiver $tamanho >= 2$.
- **Conquistar:** Ordene as duas entradas menores recursivamente usando o Quick-Sort.
- **Combinar:** A combinação foi feita junto do particiona.

Ordenação Quick-sort



Quick-Sort

Dado um vetor v de tamanho n....

```
void quickSort(int v[], int ini, int fim){
   if(ini < fim){
      //Fase da Divisao
      int q = particiona(v, ini, fim);

      //Fase da Conquista
      quickSort2(v, ini, q - 1);
      quickSort2(v, q + 1, fim);
   }
}
```

- Qual o tempo de execução desse algoritmo?

Quick-Sort

```c
void trocar(int* a, int* b) {
  int aux = *a;
  *a = *b;
  *b = aux;
}
/*Particiona o vetor escolhendo o pivo como sendo o ultimo
    elemento*/
int particiona(int v[], int ini, int fim){
   int pivo = v[fim];
   int i = fim, j;

   for(j = fim-1; j >= ini; j--){
      if(v[j] >= pivo){
         i--;
         trocar(&v[i], &v[j]);
      }
   }
   trocar(&v[i], &v[fim]);
   return i;
}
```

- Qual o tempo de execução desse algoritmo?

Busca Binária

- **Dividir:** Utilizando o particiona, divide a entrada de tamanho n duas entradas menores, possivelmente de tamanho diferentes, se a entrada tiver $tamanho >= 2$.
- **Conquistar:** Descarte uma das parte menores que comprovadamente a chave não se encontra e na outra, verifique se a chave se encontra recursivamente usando a busca binária.
- **Combinar:** Não precisa.

Análise de Complexidade

- Quando um algoritmo apresenta uma chamada a si próprio (algoritmo recursivo), o tempo de execução é descrito por uma equação de recorrência ou recorrência, que descreve o tempo total de execução de uma entrada de tamanho n em termos do tempo de execução em entradas menores.

- Para uma divisão em a subproblemas de tamanho n/b cada um. Temos o tempo D(n) para dividir o problema e C(n) para combinar:

$$T(n) = \begin{cases} O(1) & , n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & , n > c \end{cases}$$

Análise de Complexidade - Quick-Sort

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Pior Caso: Ocorre quando o particiona gera um subproblema de tamanho $n-1$ e outro de tamanho 0. Partições totalmente desbalanceadas:

$$T(n) = \begin{cases} O(1) & , n \leq 1 \\ T(n-1) + T(0) + O(n) & , n > 1 \end{cases}$$

- Melhor Caso: Ocorre quando o particiona gera dois subproblema de tamanho aproximadamente igual $\frac{2}{2}$. Partições totalmente balanceados:

$$T(n) = \begin{cases} O(1) & , n \leq 1 \\ 2T(\frac{n}{2}) + O(n) & , n > 1 \end{cases}$$