Projeto e Análise de Algoritmos
Divisão e Conquista

Antonio Luiz Basile

Faculdade de Computação e Informática
Universidade Presbiteriana Mackenzie

March 5, 2018

Paradigma Divisão e Conquista

Muitos algoritmos importantes são estruturalmente recursivos. Tais algoritmos seguem tipicamente o paradigma da divisão e conquista (CLR):

- quebram o problema original em pedaços menores,
- resolvem os subproblemas recursivamente, e
- combinam estas soluções para criar uma solução para o problema original.

Paradigma Divisão e Conquista

O paradigma da divisão e conquista envolve 3 passos a cada nível recursivo (CLR):

- **Dividir** o problema em um número de subproblemas que são instâncias menores do mesmo problema.
- **Conquistar** os subproblemas resolvendo-os recursivamente. Se os tamanhos dos subproblemas forem suficientemente pequenos, no entanto, resolva-os de modo direto.
- **Combinar** as soluções dos subproblemas na solução para o problema original.

Mergesort

O algoritmo **mergesort** é um ótimo exemplo para o paradigma da divisão e conquista. Intuitivamente opera como segue:

- **Dividir:** Divida a sequência de n elementos em 2 subsequências de $n/2$ elementos cada.
- **Conquistar:** Ordene as 2 subsequências recursivamente usando o mergesort.
- **Combinar:** Intercale (merge) as 2 subsequências para produzir a resposta ordenada.

A recursão para quando a subsequência a ser ordenada tem tamanho 1. Neste caso não há trabalho a ser realizado, dado que uma sequência de tamanho 1 já está ordenada.

Intercala (2-way)

```
mergeAB(Item c[], Item a[], int N, Item b[], int M )
  { int i, j, k;
    for (i = 0, j = 0, k = 0; k < N+M; k++)
      {
        if (i == N) { c[k] = b[j++]; continue; }
        if (j == M) { c[k] = a[i++]; continue; }
        c[k] = (less(a[i], b[j])) ? a[i++] : b[j++];
      }
  }
```

Figure: merge 2-way (Sedgewick)

Intercala localmente

```
Item aux[maxN];
merge(Item a[], int l, int m, int r)
  { int i, j, k;
    for (i = m+1; i > l; i--) aux[i-1] = a[i-1];
    for (j = m; j < r; j++) aux[r+m-j] = a[j+1];
    for (k = l; k <= r; k++)
       if (less(aux[j], aux[i]))
          a[k] = aux[j--]; else a[k] = aux[i++];
  }
```

Figure: merge on place (Sedgewick)

Mergesort

```
void mergesort(Item a[], int l, int r)
  { int m = (r+l)/2;
    if (r <= l) return;
    mergesort(a, l, m);
    mergesort(a, m+1, r);
    merge(a, l, m, r);
  }
```

Figure: Mergesort (Sedgewick)

Análise do Mergesort

- Vamos supor, por simplicidade, um vetor $A[n]$, onde $n = 2^i$.
- Quanto tempo leva o algoritmo mergesort para executar sobre A ?

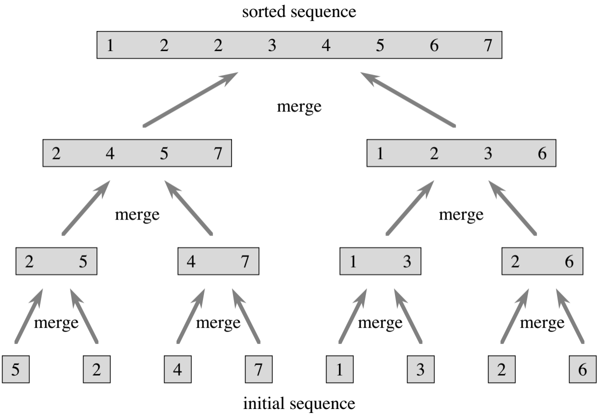

Figure: Simulação do Mergesort (CLR)

Análise do Mergesort

- **Dividir:** O passo da divisão apenas calcula o meio do subvetor, o que leva tempo constante, ou seja, tempo $\Theta(1)$.
- **Conquistar:** Recursivamente resolvemos 2 subproblemas, cada um com tamanho $n/2$, o que contribui $2T(n/2)$ para o tempo de execução.
- **Combinar:** Já observamos que a função Merge para um vetor de tamanho n leva tempo $\Theta(n)$.

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

Figure: Mergesort (CLR)

Recorrência do Mergesort

A recorrência (ou fórmula aberta) do Mergesort pode ser obtida a partir do programa abaixo.

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

$$T(n) = \begin{cases} \Theta(1), & \text{se } n = 1. \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n), & \text{se } n > 1. \end{cases} \tag{1}$$

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