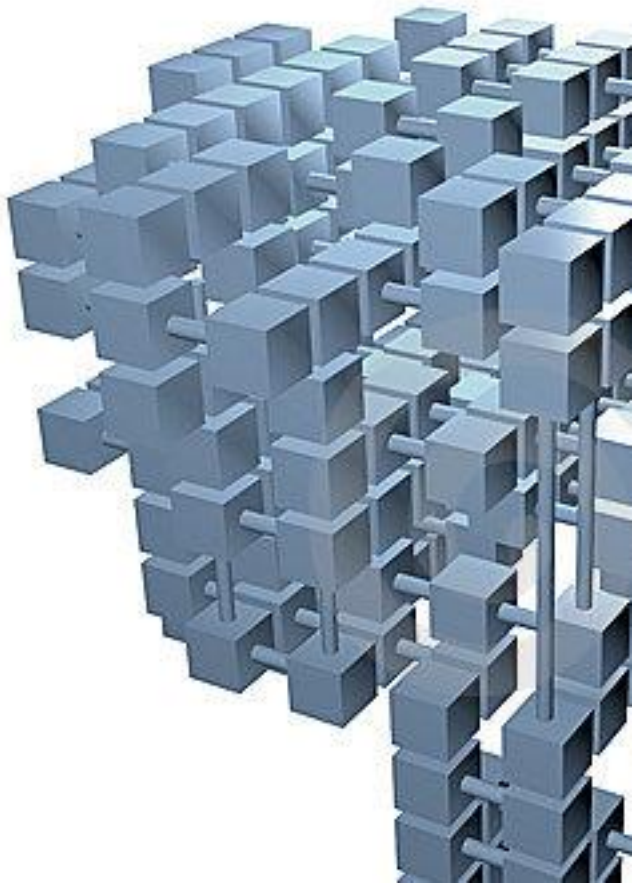
Campus de Frutal

UEMG UNIVERSIDADE DO ESTADO DE MINAS GERAIS

Sistemas de Informação

Estrutura de Dados
II
Merge Sort



Prof. Ivan José dos Reis Filho
ivanfilhoreis@gmail.com

Aulas Anteriores

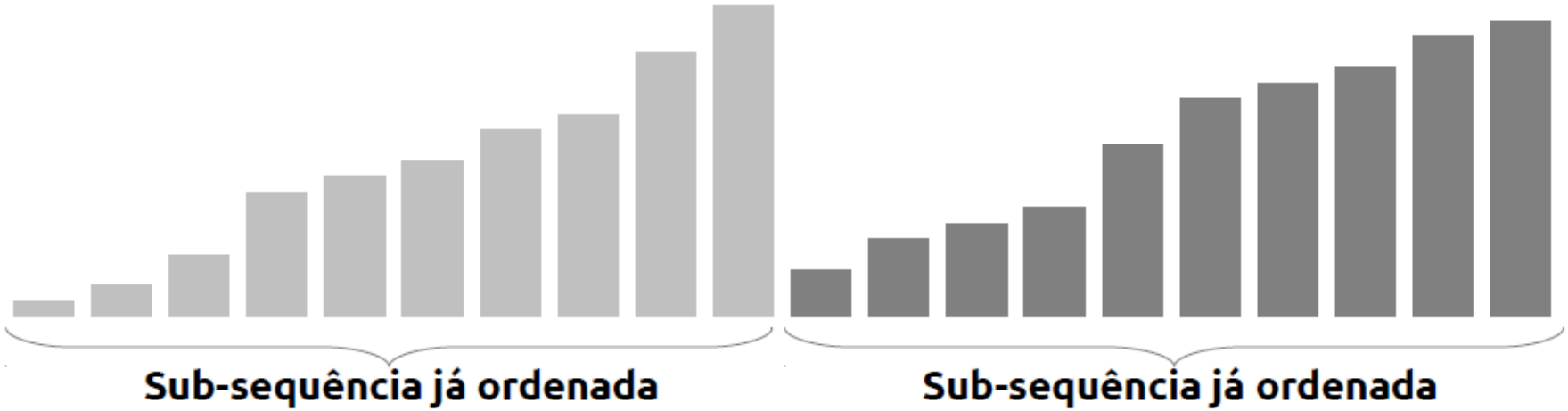
- Ordenação por Seleção
- Ordenação por Inserção
- Ordenação por Troca
- Ordenação por Particionamento
- Método Rápido (Quick Sort)

Já Vimos

- Merge Sort

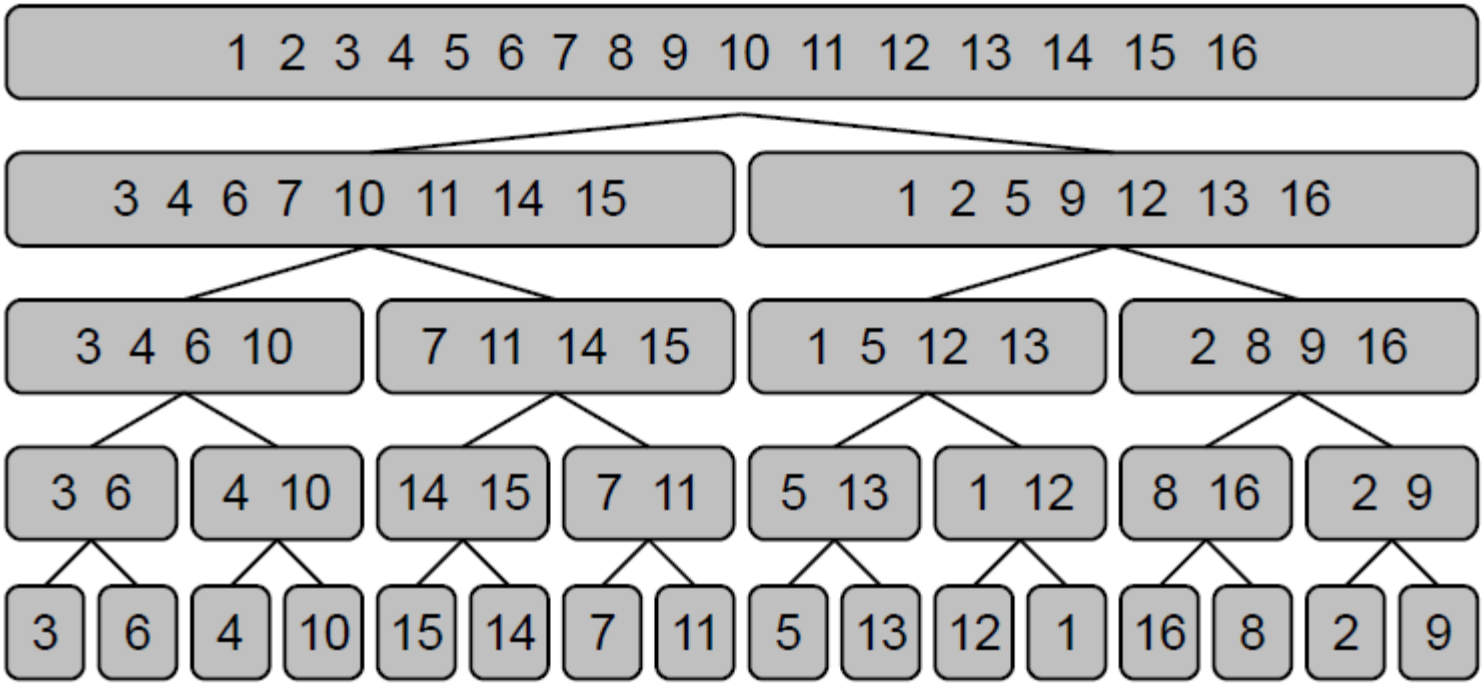
Merge Sort

- Dividir para conquistar
- Divisão da sequência em partes menores para facilitar a ordenação
- União de sequências menores já ordenadas, gerando sequências maiores ordenadas.



Merge Sort

- Árvore de divisão



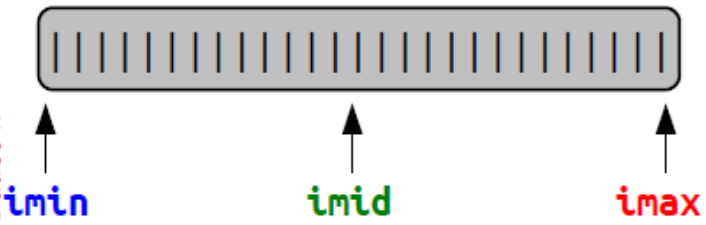
- Abordagens de implementação
  - Top-Down
  - Boton-up

Merge procedure

```
template <class Item>
void merge(Item vetor[], int imin, int imid, int imax)
{
    int i = imin, j = imid+1;
    // aux[Nmax]
    for (int k = imin; k <= imax; k++)
        aux[k] = vetor[k];

    for (int k = imin; k <= imax; k++)
        if (i > imid)
            vetor[k] = aux[j++];
        else if(j > imax)
            vetor[k] = aux[i++];
        else if(aux[j] < aux[i])
            vetor[k] = aux[j++];
        else
            vetor[k] = aux[i++];
}
```

Merge Sort/Recursivo/Top-Down

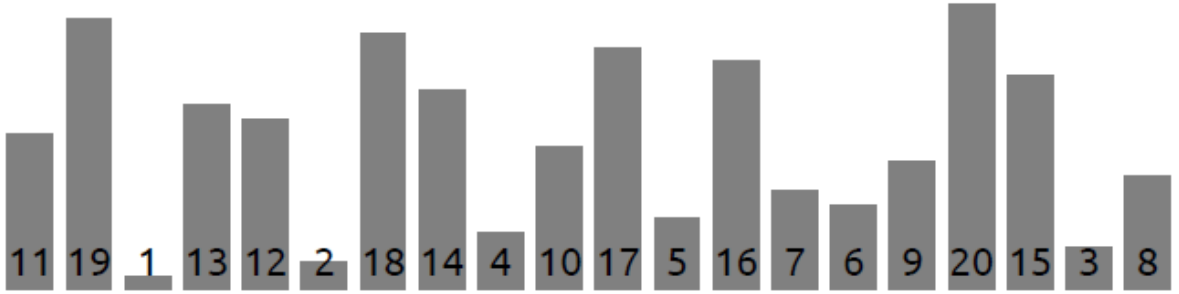
```
template <class Item>
void mergesort(Item vetor[], int imin, int imax)
{
    if (imax <= imin)
        return;

    int imid = imin + ((imax - imin) / 2);

    mergesort(vetor, imin, imid);
    mergesort(vetor, imid+1, imax);

    merge(vetor, imin, imid, imax);
}
```

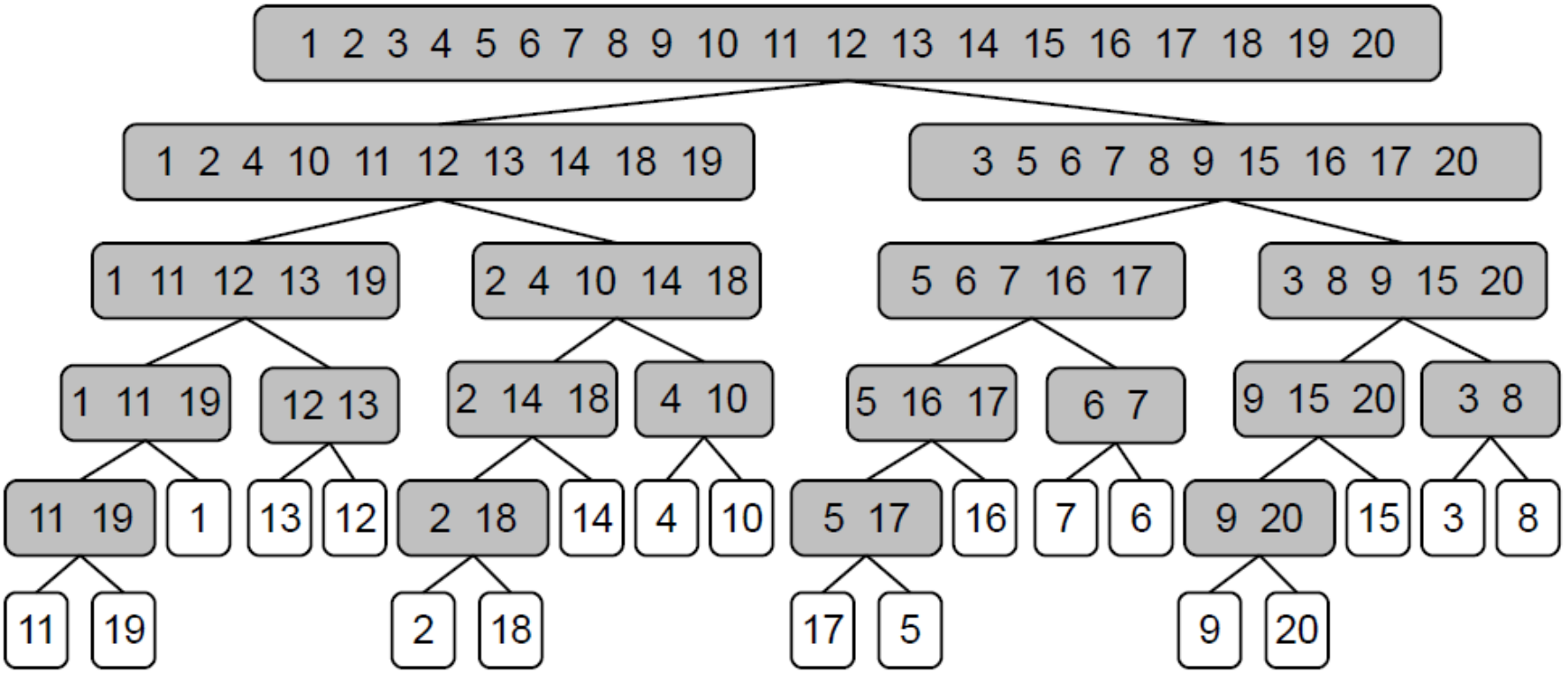



```
template <class Item>
void merge(Item vetor[], int imin, int imid, int imax)
{
      int i = imin, j = imid+1;
      // aux[Nmax]
      for (int k = imin; k <= imax; k++)
            aux[k] = vetor[k];

      for (int k = imin; k <= imax; k++)
            if (i > imid)
                  vetor[k] = aux[j++];
            else if(j > imax)
                  vetor[k] = aux[i++];
            else if(aux[j] < aux[i])
                  vetor[k] = aux[j++];
            else
                  vetor[k] = aux[i++];
}
```

MergeSort Top-Down

Entrada:





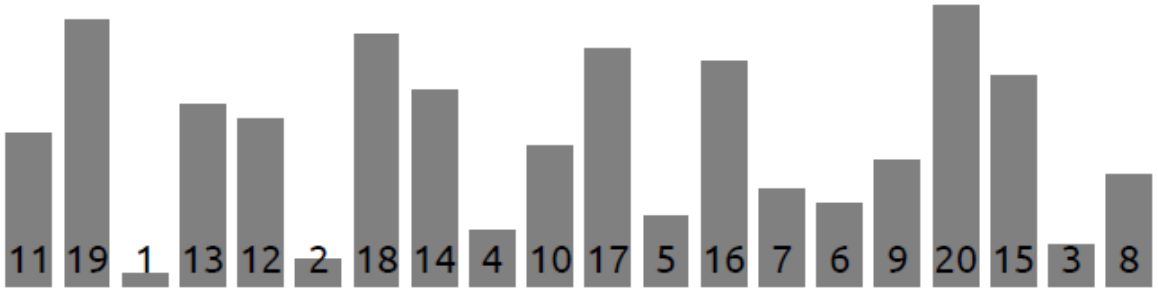
Merge Sort/Iterativo/Botton-up

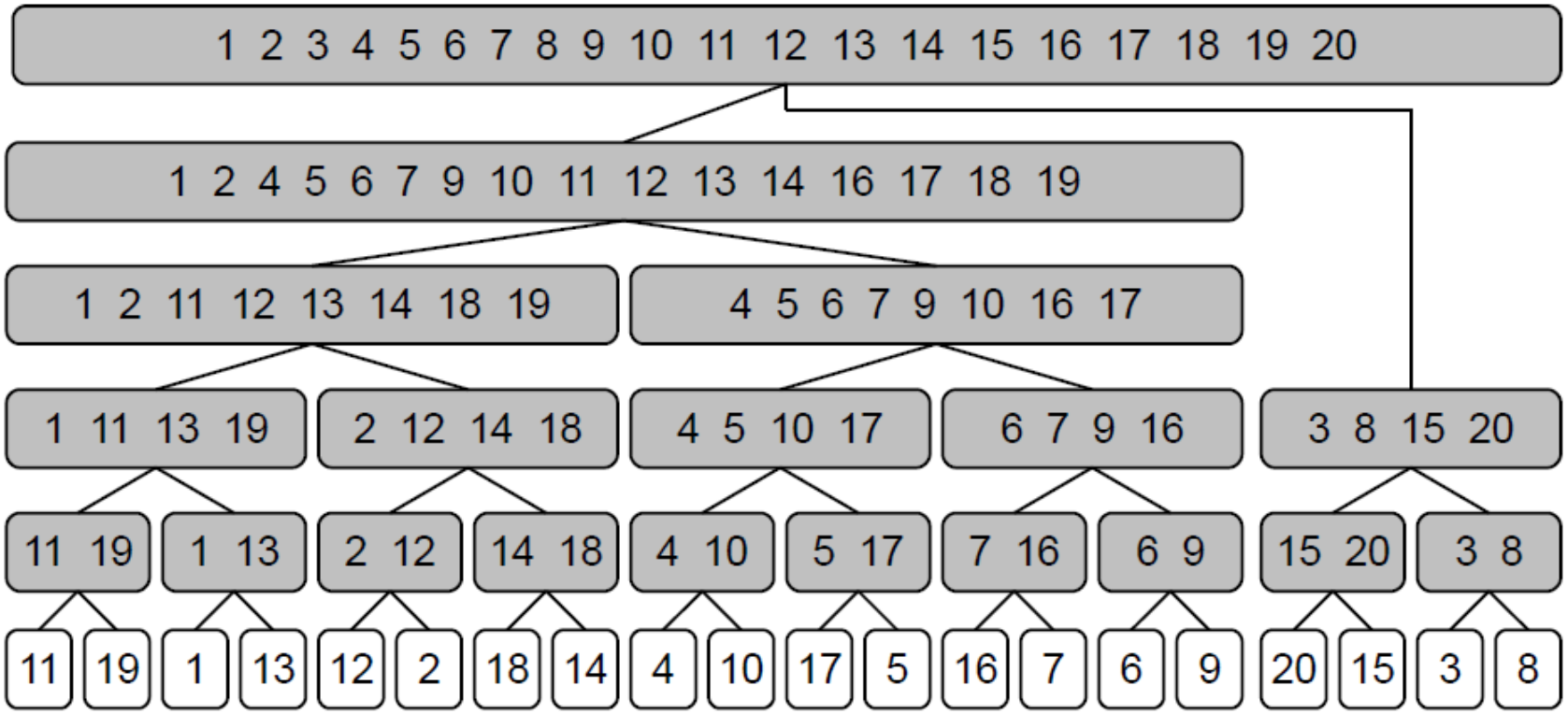
```
inline int min(int A, int B)
{
    return (A < B) ? A : B;
}

template <class Item>
void mergesortBU(Item vetor[], int imin, int imax)
{
    for (int m = 1; m <= imax-imin; m *= 2)
        for (int i = imin; i <= imax-m; i += 2*m)
            merge(vetor, i, i+m-1, min(i+m+m-1, imax));
}
```

```
template <class Item>
void merge(Item vetor[], int imin, int imid, int imax)
{
     int i = imin, j = imid+1;
     // aux[Nmax]
     for (int k = imin; k <= imax; k++)
          aux[k] = vetor[k];

     for (int k = imin; k <= imax; k++)
          if (i > imid)
               vetor[k] = aux[j++];
          else if(j > imax)
               vetor[k] = aux[i++];
          else if(aux[j] < aux[i])
               vetor[k] = aux[j++];
          else
               vetor[k] = aux[i++];
}
```

MergeSort Bottom-Up

Entrada:





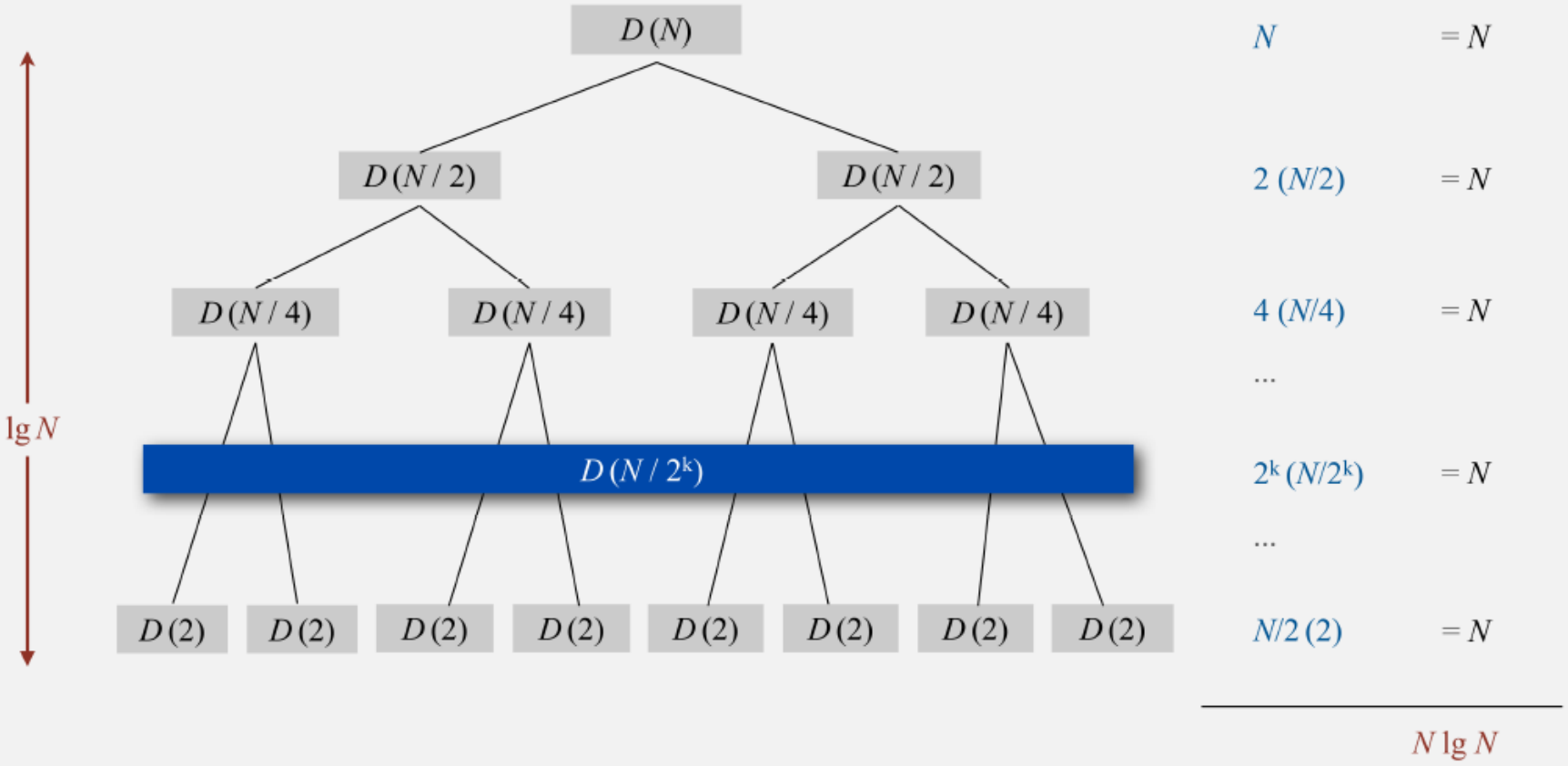
Merge Sort

- Quantas comparações são executadas?
- Quantas trocas são executadas?
- Quantidade de memória?

Divide-and-conquer recurrence: proof by picture

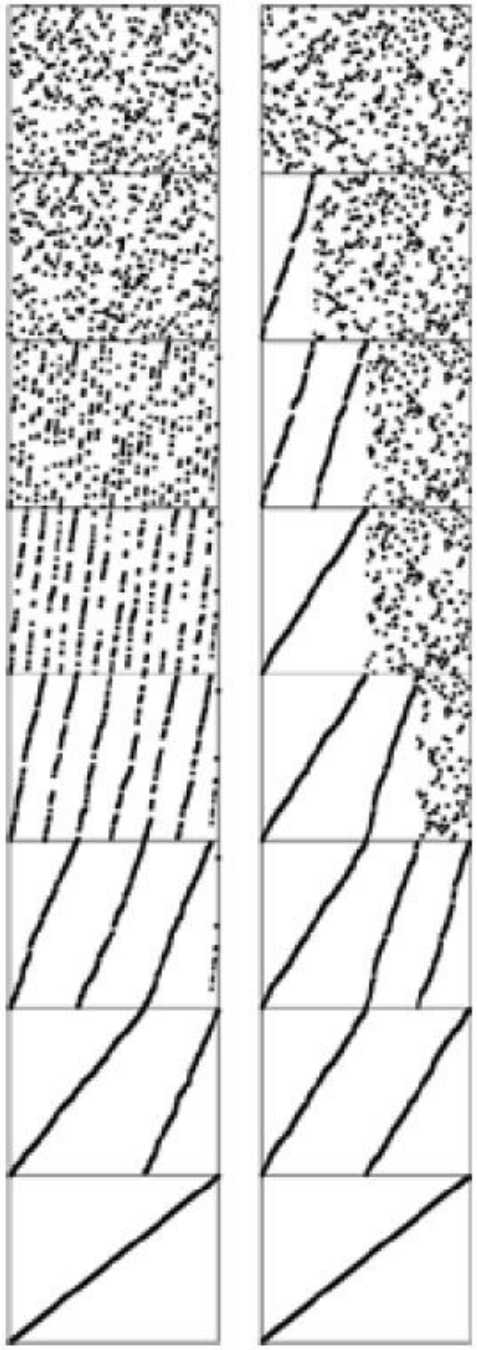
Proposition. If $D(N)$ satisfies $D(N) = 2\,D(N/2) + N$ for $N > 1$, with $D(1) = 0$, then $D(N) = N \lg N$.

Pf 1. [assuming N is a power of 2]



Merge Sort

- Os valores dos dados não interferem na execução do algoritmo
- Crescimento do número de comparações em relação ao tamanho de entrada:
  - **linear logarítmico**
- Crescimento do número de trocas em relação ao tamanho de entrada
  - **linear logarítmico**
- Crescimento do uso de memória em relação ao tamanho da entrada
  - **linear**



DÚVIDAS?

Prof. Ivan José dos Reis Filho