Programação De Computadores II

Métodos de Ordenação Eficiente

Profª Valéria
Profº Jadson

Slides adaptados do Profº Paulo Afonso - UFLA

Introdução

- Algoritmos *Insertion sort* e *Selection sort*
  - Em geral são pouco eficientes

- Utilizar algoritmos que utilizam a estratégia de divisão e conquista
  - *Quick Sort*
  - *Merge Sort*

Quick Sort

- É o algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações.

- A ideia básica é dividir o problema de ordenar um conjunto com *N* itens em problemas menores sucessivas vezes.

Quick Sort

- A parte mais delicada do método é relativa à **partição do vetor**.
- O vetor *v[esq...dir]* é particionado, por meio da escolha arbitrária de um **pivô** *x*, em:
  - Uma parte esquerda com chaves menores ou iguais a *x*: os itens *v[esq], v[esq + 1], ..., v[j]*.
  - Uma parte direita com chaves maiores ou iguais a *x*: os itens *v[i], v[i + 1], ..., v[dir]*.

Quick Sort

- Funcionamento do particionamento:
  1. Escolha arbitrariamente um pivô *x*.
  2. Percorra o vetor a partir da esquerda até que *v[i]* ≥ *x*.
  3. Percorra o vetor a partir da direita até que *v[j]* ≤ *x*.
  4. Troque *v[i]* com *v[j]*.
  5. Incremente o valor de *i* e decremente o valor de *j*.
  6. Continue este processo (passos 2 A 5) até os índices *i* e *j* se cruzarem.

Quick Sort

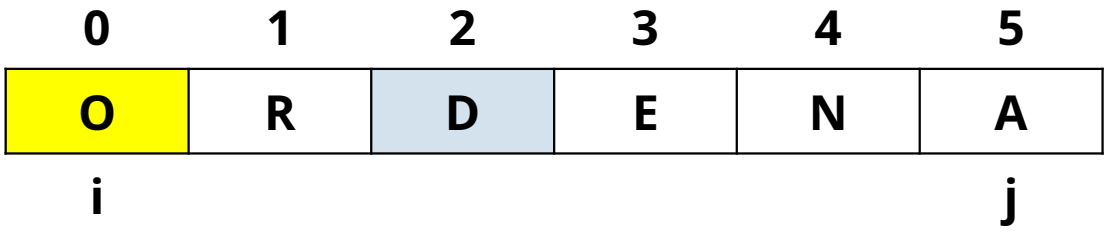
- Ilustração do processo de partição:

0	1	2	3	4	5
O	R	D	E	N	A

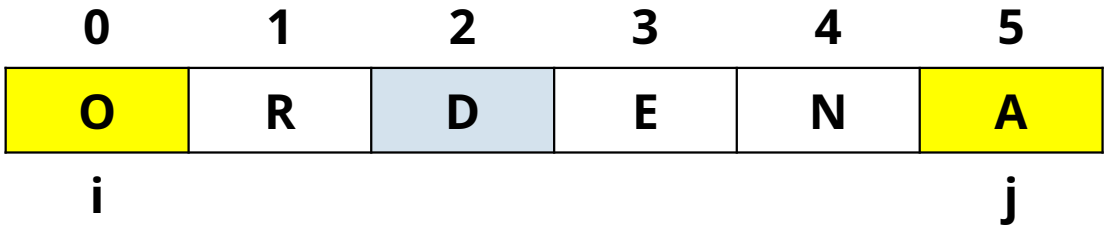
- Passo 1: escolha arbitrariamente um pivô *x*.
  - O pivô é dado por *v[(i + j)/2]*, onde *i* = índice inicial e *j* = índice final do vetor.
  - Se a divisão *(i + j)/2* não for um número inteiro, considera-se o **piso**.
  - Como inicialmente, *i = 0* e *j = 5*, então ***x = v[2] = D***.

Quick Sort

- Passo 2: percorra o vetor a partir da esquerda até que *v[i] ≥ x*.

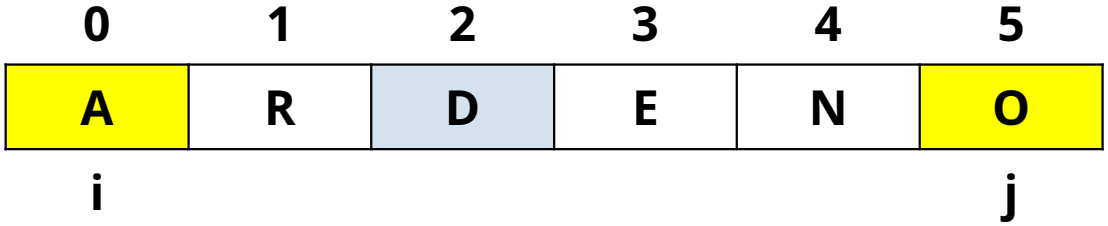


- Passo 3: percorra o vetor a partir da direita até que *v[j] ≤ x*.

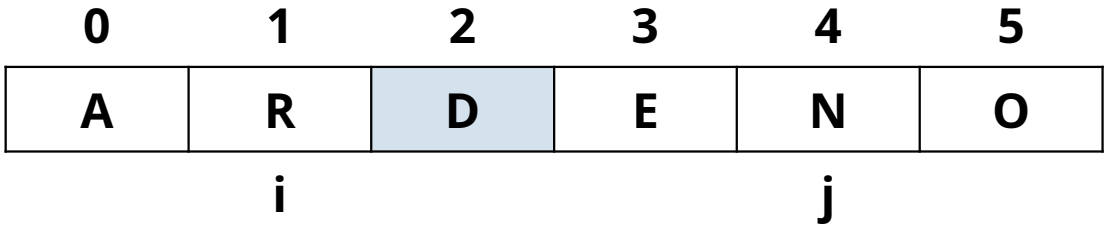


Quick Sort

- Passo 4: troque *v[i]* com *v[j]*.

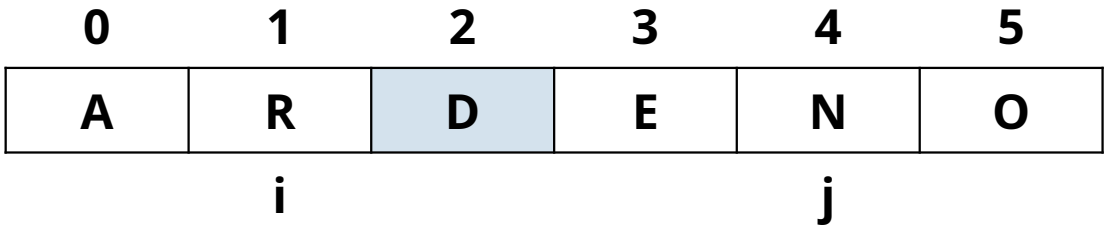


- Passo 5: incremente o valor de *i* e decremente o valor de *j*.

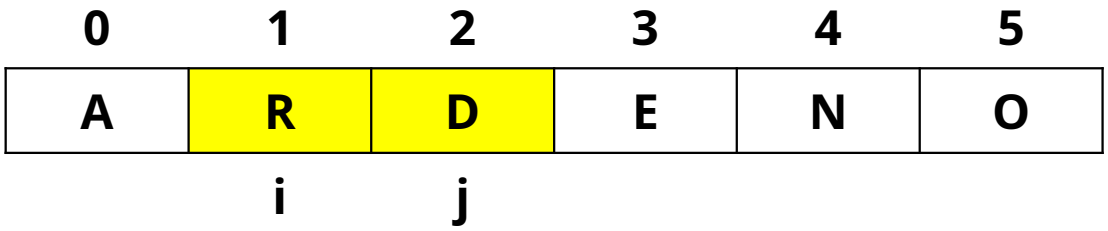


- Continue o processo (passos 2 à 5) até que os índices *i* e *j* se cruzem.

Quick Sort

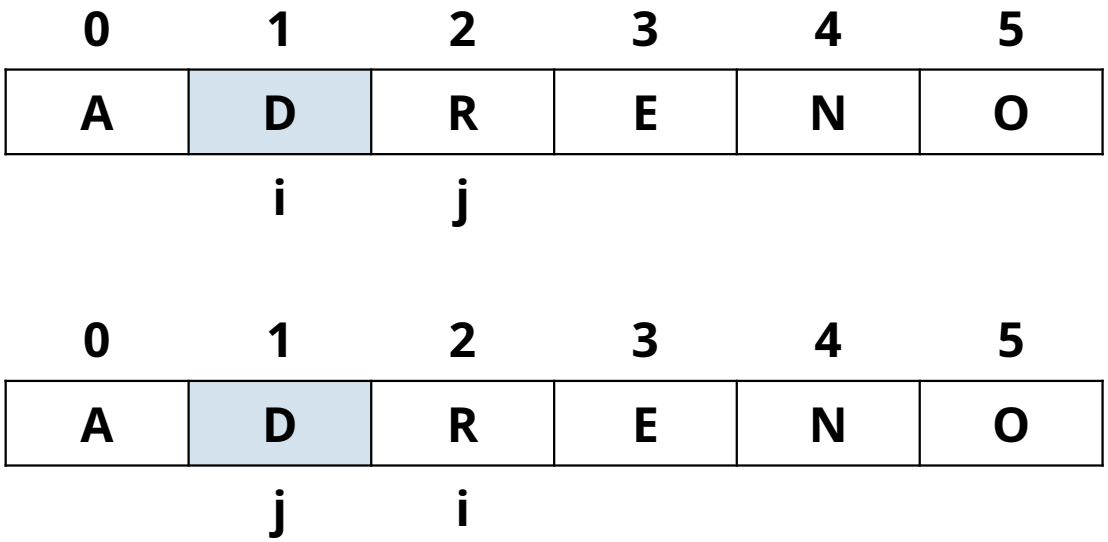


- Passos 2 e 3: percorra o vetor a partir da esquerda até que *v[i] ≥ x*. Percorra o vetor a partir da direita até que *v[j] ≤ x*.

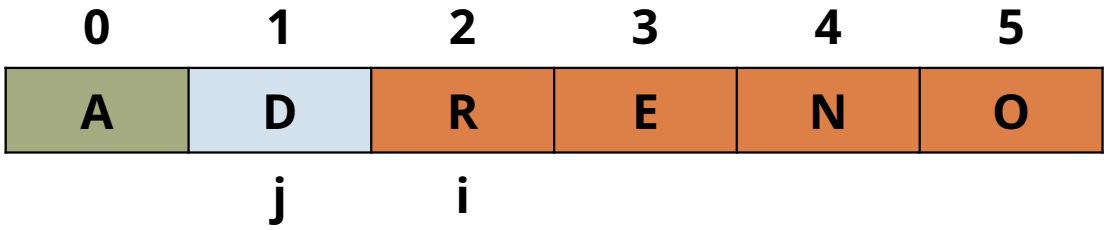


Quick Sort

- Passo 4 e 5: troque *v[i]* com *v[j]*. Incremente o valor de *i* e decremente o valor de *j*.



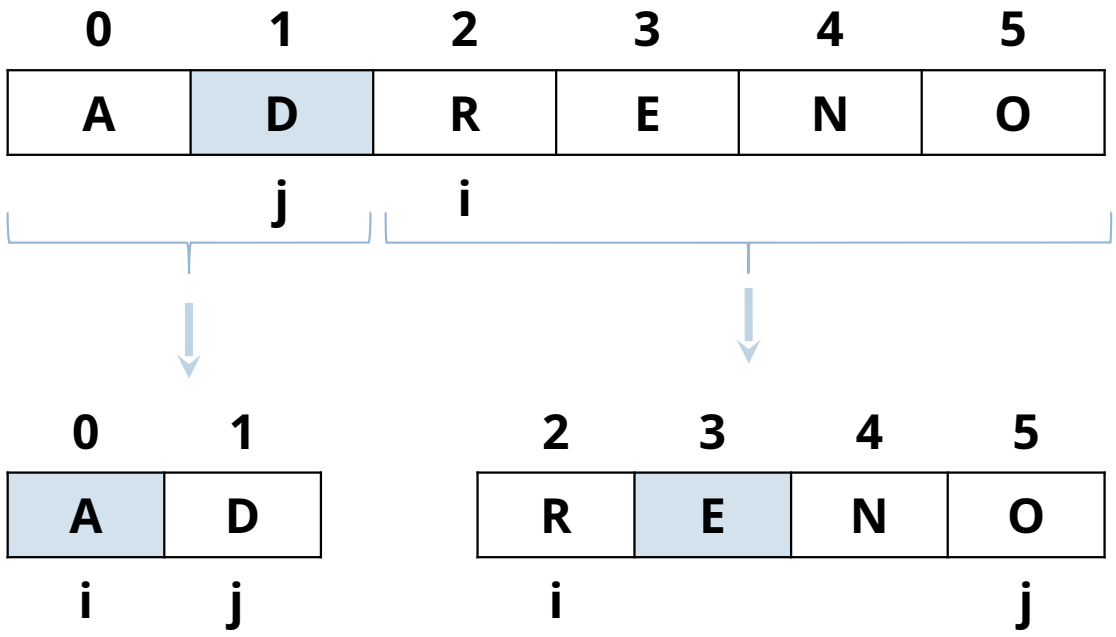
Quick Sort



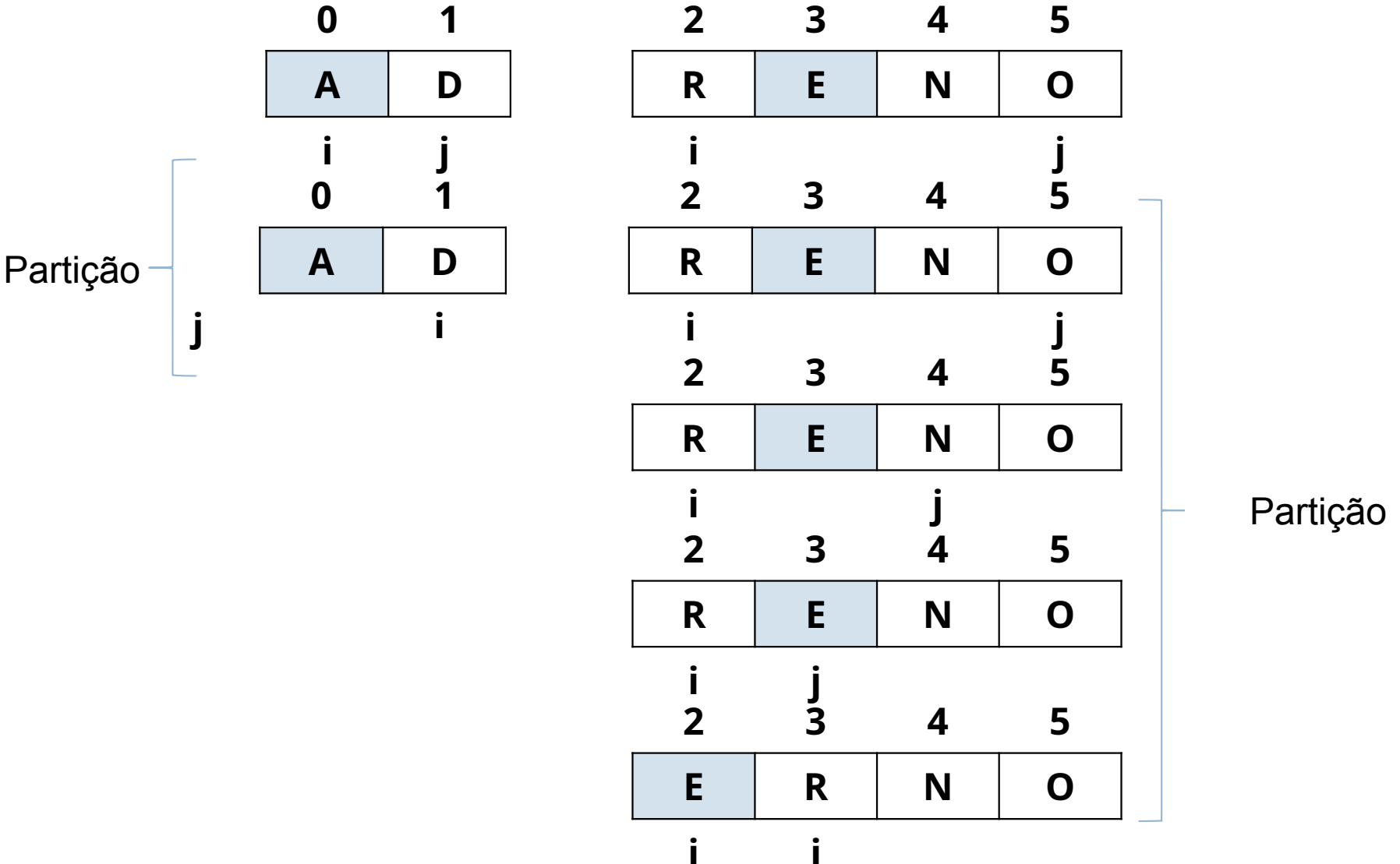
- Como os índices *i* e *j* se cruzaram, o algoritmo de partição termina.
- Observe que:
  - os elementos de *v[0]* até *v[j]* são menores ou iguais ao pivô; e
  - os elementos de *v[i]* até *v[5]* são maiores ou iguais ao pivô.

Quick Sort

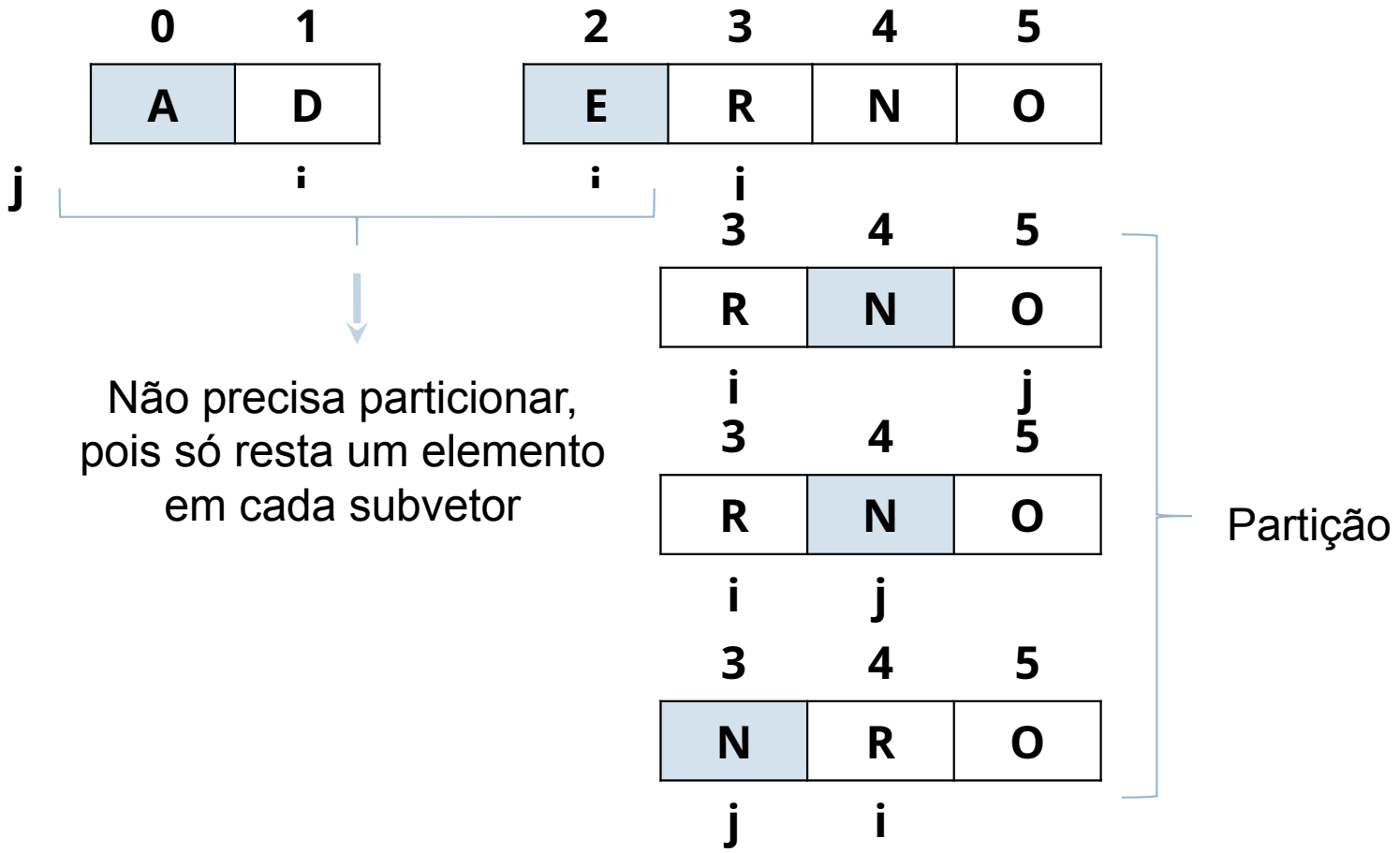
- Após obter as duas “partes” do vetor por meio do particionamento, o processo é repetido para cada parte, recursivamente.



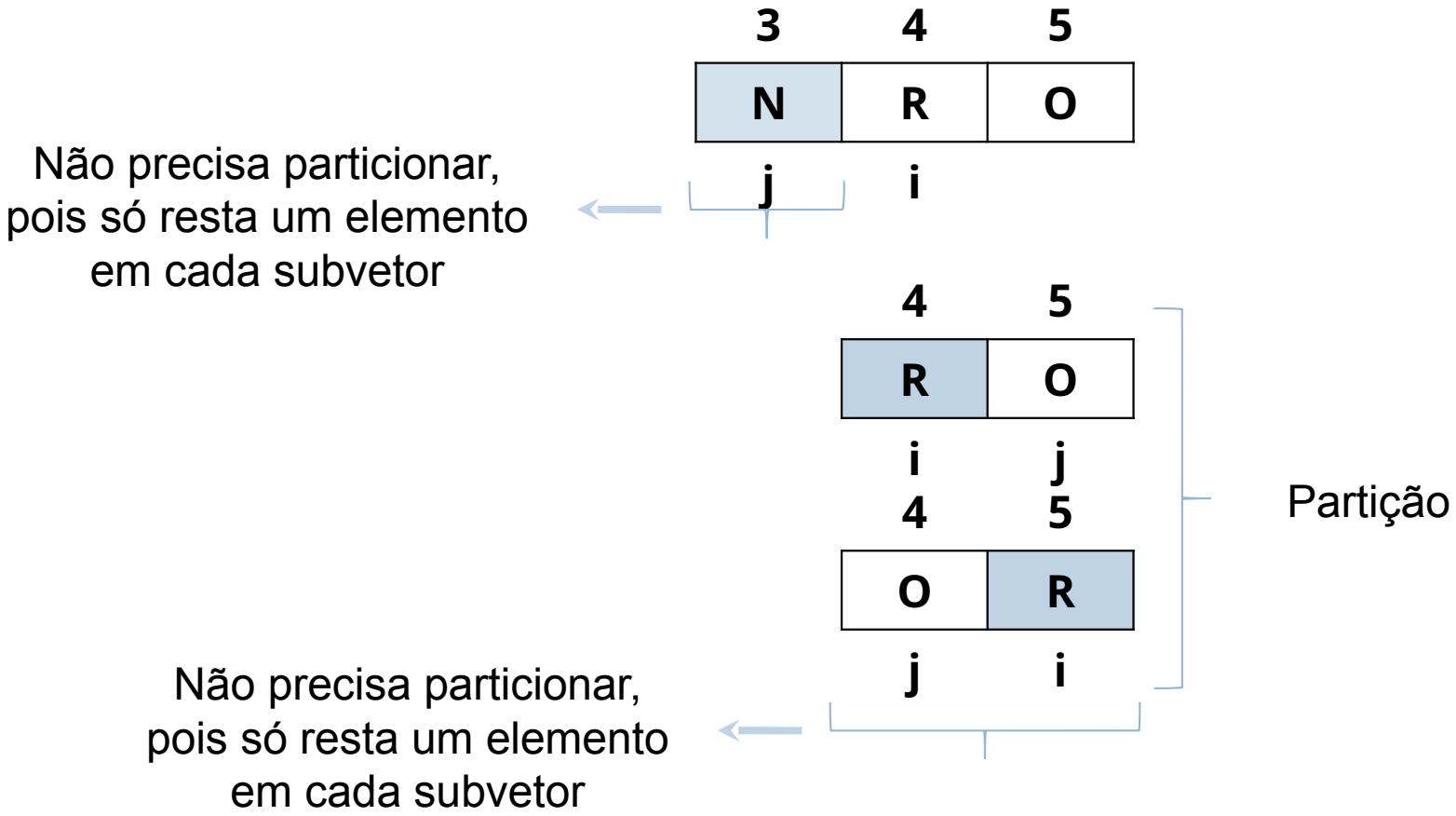
Quick Sort


0 1 2 3 4 5
A D R E N O
i j
Partição
0 1 2 3 4 5
A D R E N O
j i i j
2 3 4 5
R E N O
i j
2 3 4 5
R E N O
i j
2 3 4 5
R E N O
i j
2 3 4 5
E R N O
i i
Partição

Quick Sort


0 1 2 3 4 5
A D E R N O
j i i i
3 4 5
R N O
i j
3 4 5
R N O
i j
3 4 5
N R O
j i
Não precisa particionar, pois só resta um elemento em cada subvetor
Partição

Quick Sort


3
4
5
N
R
O
j
i
Não precisa particionar, pois só resta um elemento em cada subvetor
4
5
R
O
i
j
4
5
O
R
j
i
Partição
Não precisa particionar, pois só resta um elemento em cada subvetor

Quick Sort

- Juntando as partes, o vetor ficou assim:

0	1	2	3	4	5
A	D	E	N	O	R

- Então, a ideia da ordenação é:
  - Dividir o vetor maior em dois vetores menores, um vai de *esq* até *j* e o outro de *i* até *dir*.
  - Se os vetores menores possuírem, pelo menos, dois elementos, chama-se algoritmo de partição recursivamente e o processo se repete até que todo vetor maior esteja ordenado.

Quick Sort


```
void particiona(int v[], int esq, int dir) {
    int i = esq; int j = dir;
    int pivo = v[((i + j) / 2)];
    do {
        while (v[i] < pivo)
           i++;
        while (v[j] > pivo)
           j--;
        if (i <= j) {
              int aux = v[i];
              v[i] = v[j];
              v[j] = aux;
              i++; j--;
        }
    } while (i <= j);

    if (esq < j) particiona(v, esq, j);
    if (i < dir) particiona(v, i, dir);
}
```

Quick Sort

- Para invocar o *Quick Sort* a partir da função *main*, faz-se:

```
void quickSort(int v[], int n) {
   particiona(v, 0, n - 1);
}

int main() {
   int TAM = 10;
   int v[] = {5, 3, 2, 2, 1, 6, 5, 7, 9, 10};
   quickSort(v, TAM);
   return EXIT_SUCCESS;
}
```

Quick Sort

- Vantagens:
  - É extremamente eficiente para ordenar grandes arquivos de dados.
  - Necessita apenas de uma pequena pilha como memória auxiliar.
  - Requer cerca de *nlgn* comparações em média para ordenar *n* itens.

Quick Sort

- Desvantagens:
  - Tem um pior caso de n^2 comparações.
  - Sua implementação é delicada e difícil se comparada a dos métodos simples.

Merge Sort

- Outro algoritmo eficiente para ordenação é o *Merge Sort*, desenvolvido por John von Neumann.

- É considerado um dos primeiros métodos de ordenação inventados.

Merge Sort

- A propriedade mais atrativa deste método de ordenação é que ele é capaz de ordenar um vetor qualquer de *n* de elementos em um tempo proporcional a *nlgn*.

- Isto é, não importa a maneira como os elementos estão organizados no vetor, sua complexidade será sempre a mesma.

Merge Sort

- Uma das desvantagens deste método, entretanto, é que ele requer espaço extra de memória proporcional a *n*.

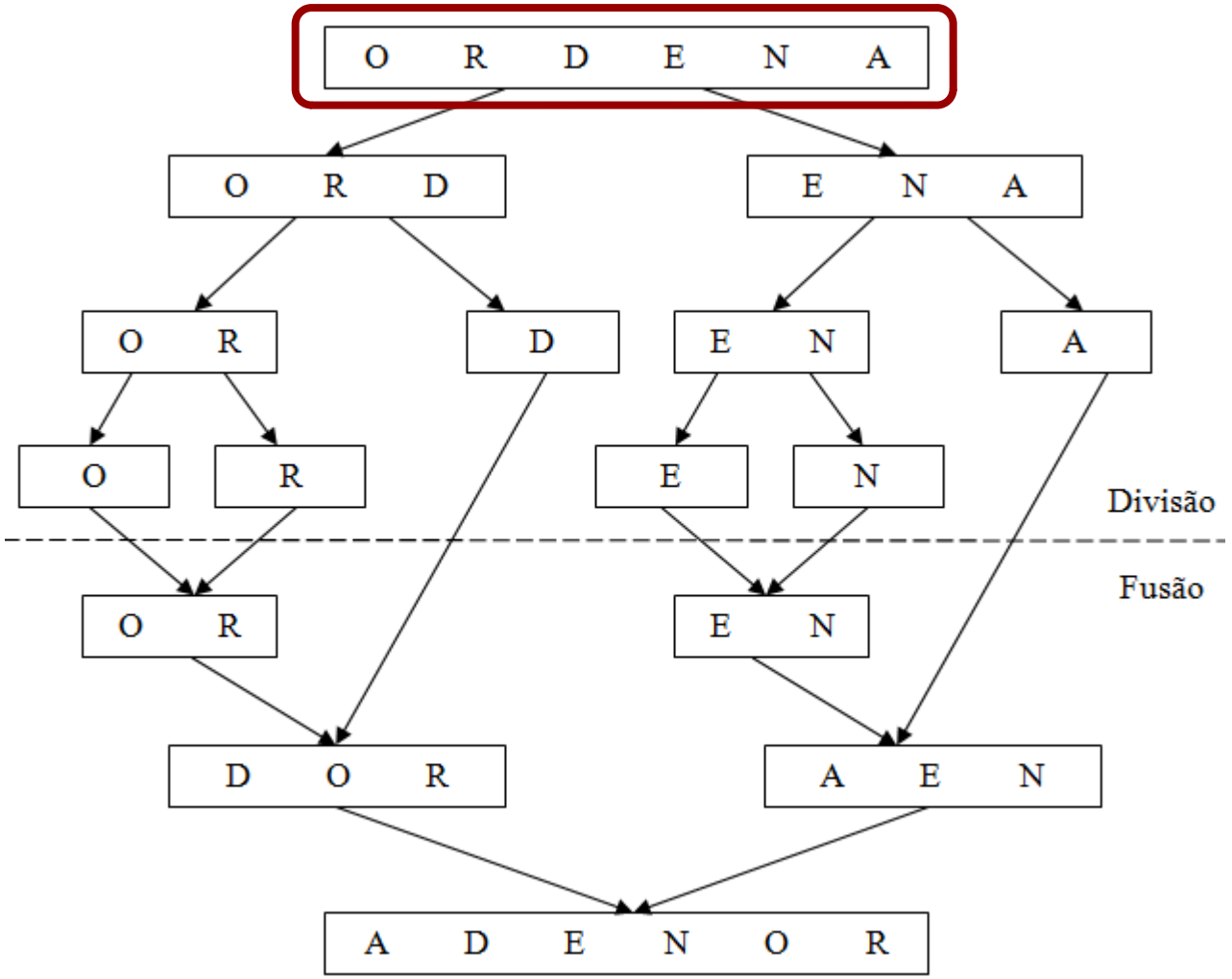
- Assim, *Merge Sort* é ideal para aplicações que precisam de ordenação eficiente, que não toleram desempenho ruim no pior caso e que possuam espaço de memória extra disponível.

Merge Sort

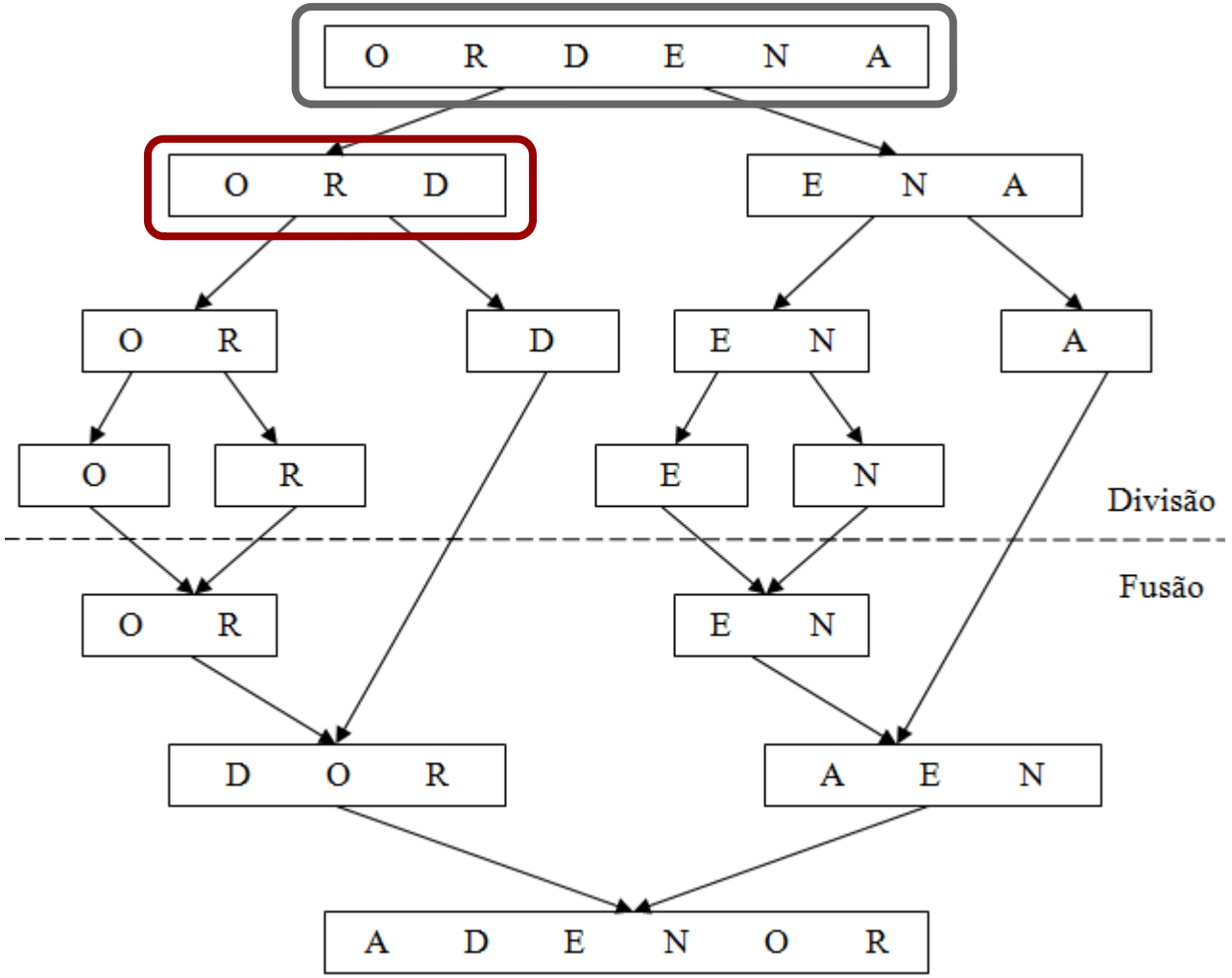
- O processo-chave do *Merge Sort* consiste em dividir o vetor original em subvetores cada vez menores até que se tenha pequenos subvetores **com um elemento apenas.**

- A partir daí, cada par de subvetores é fundido (*merged*) de forma intercalada até se obter um único vetor ordenado com todos os elementos.

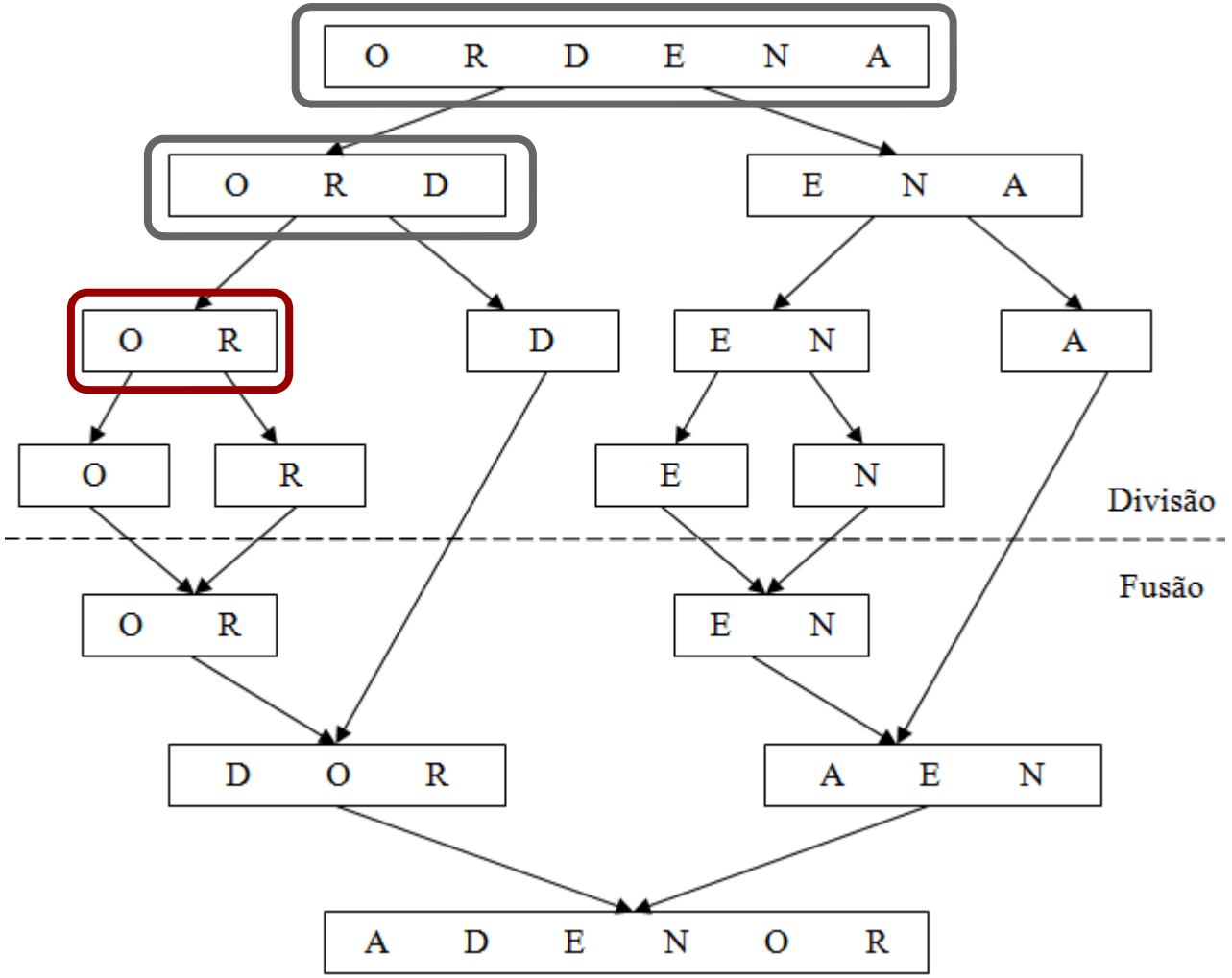
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

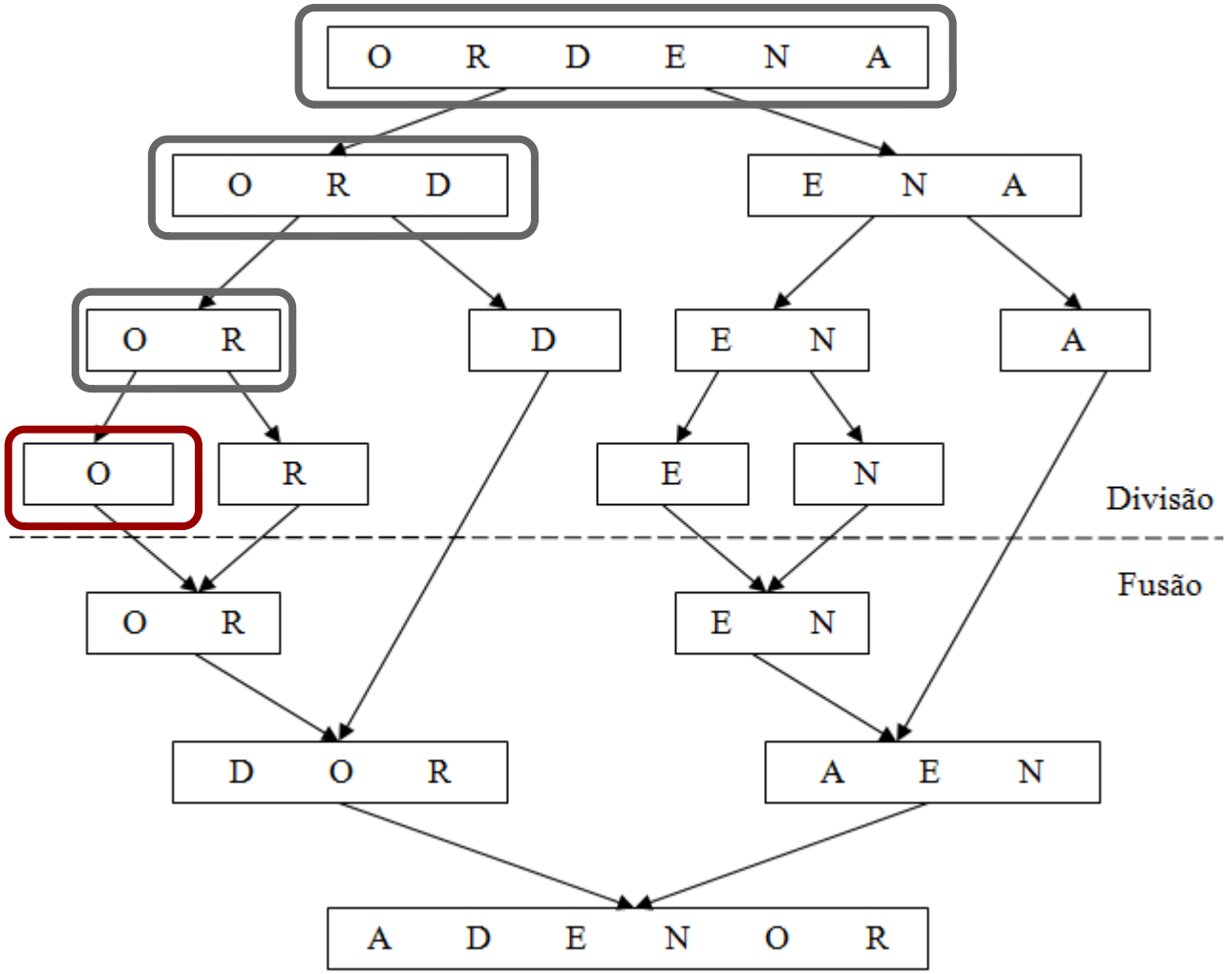
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

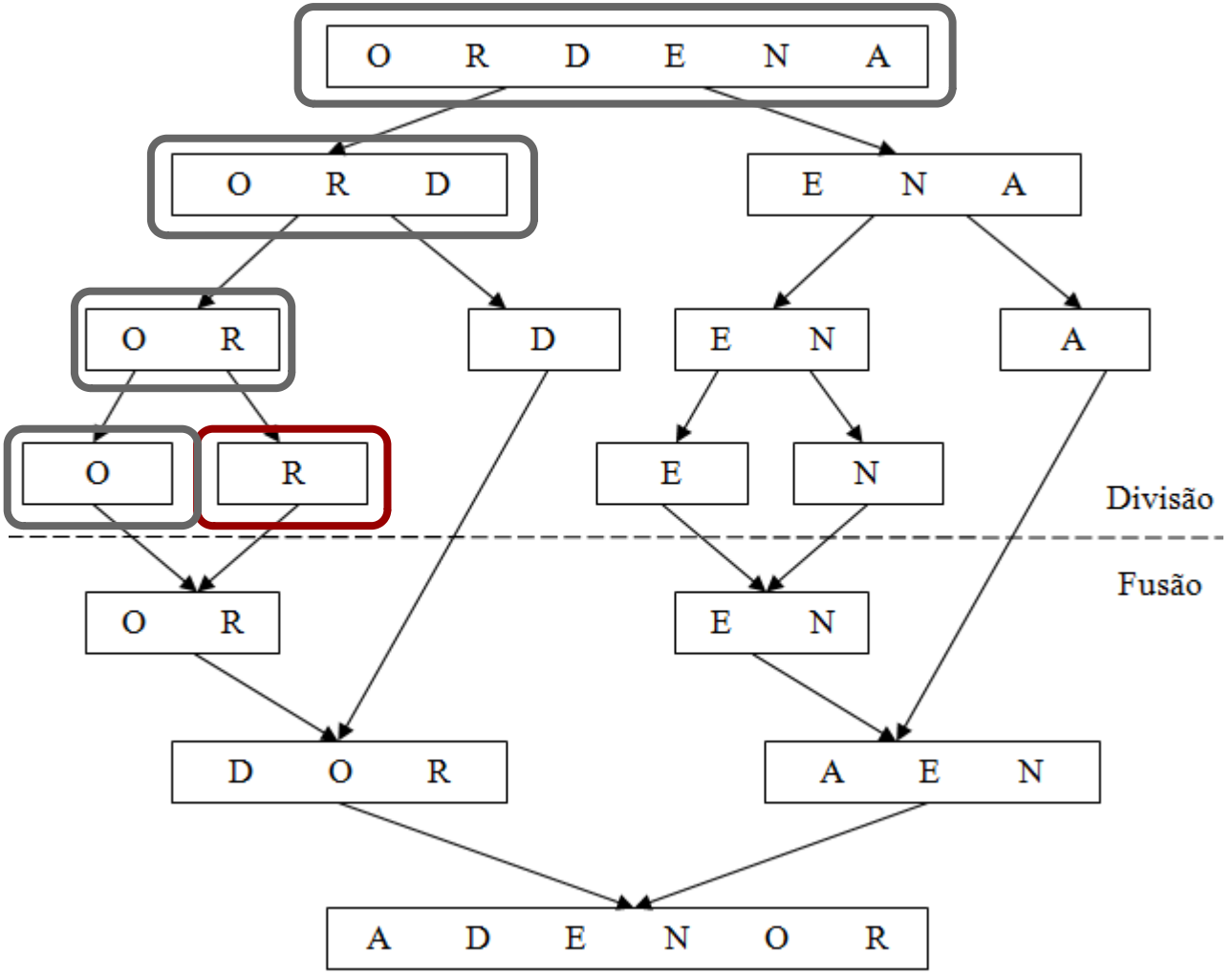
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

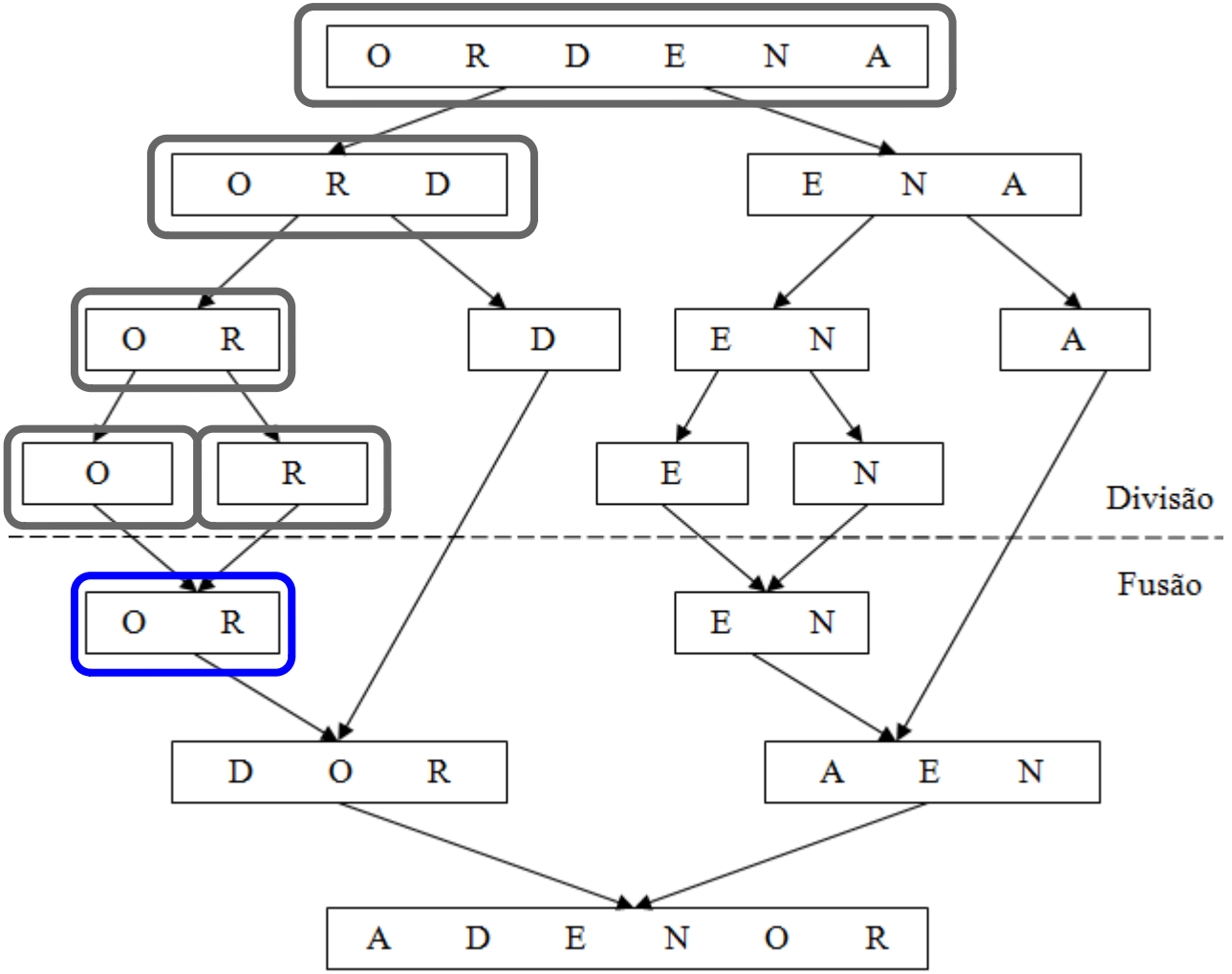
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

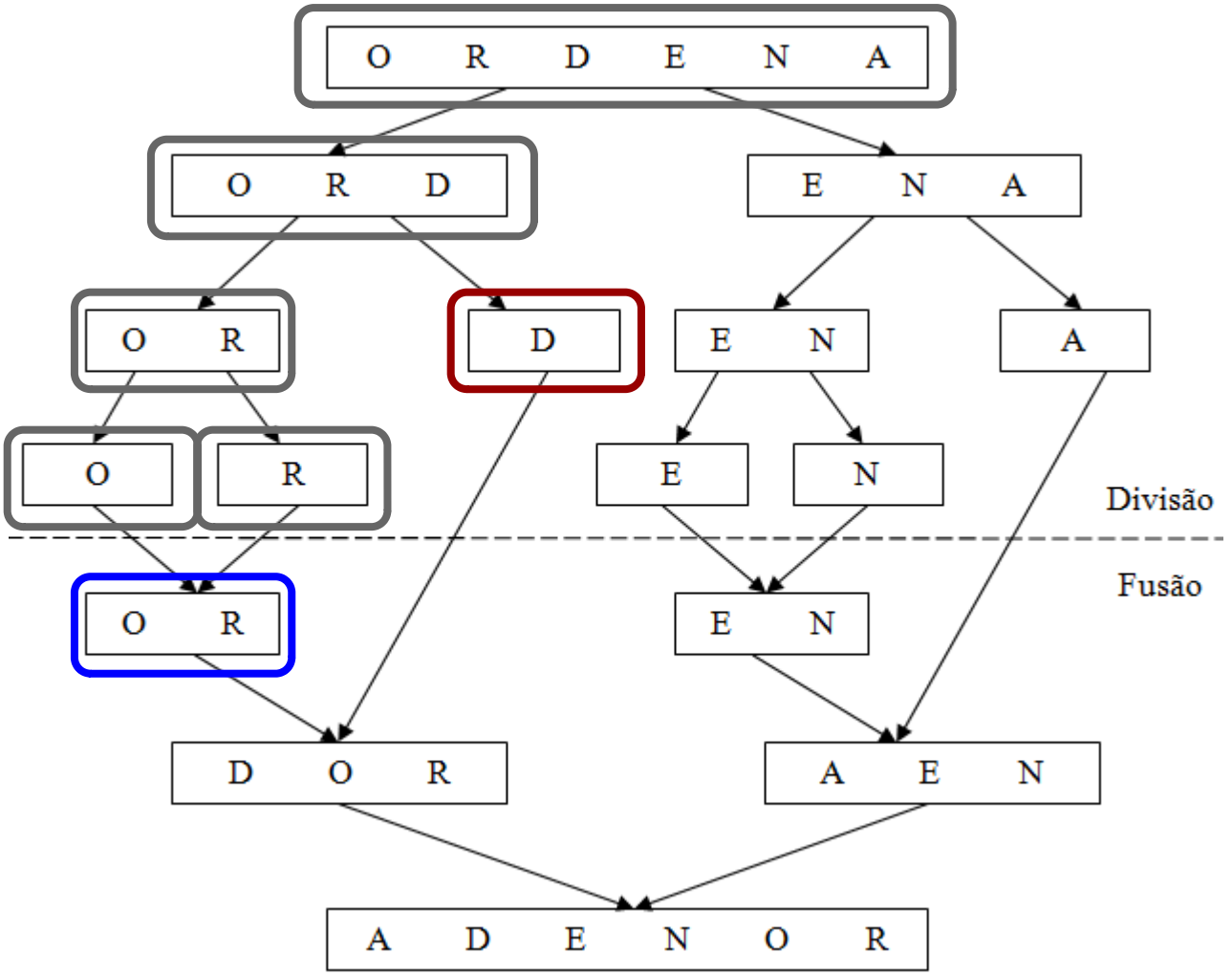
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

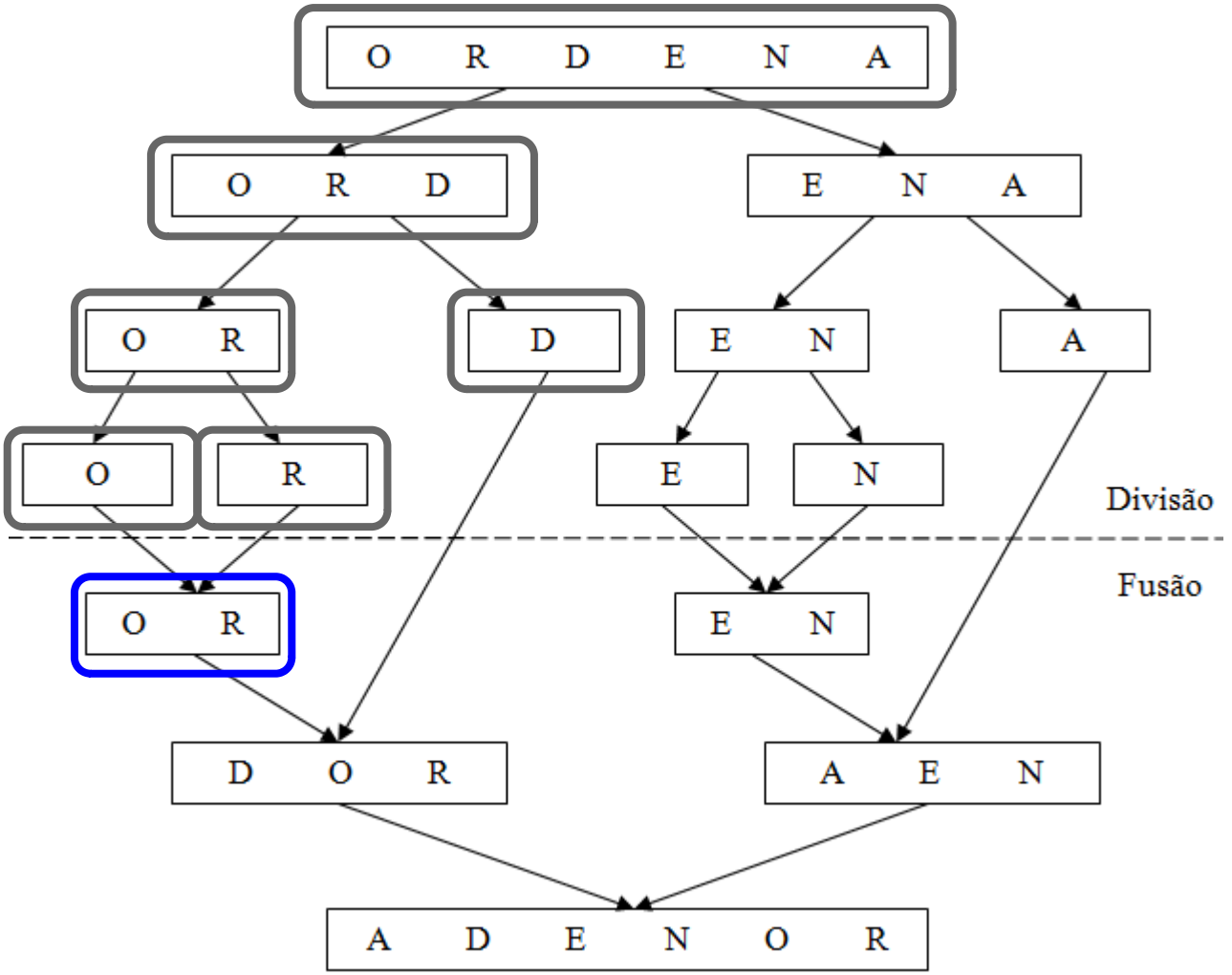
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

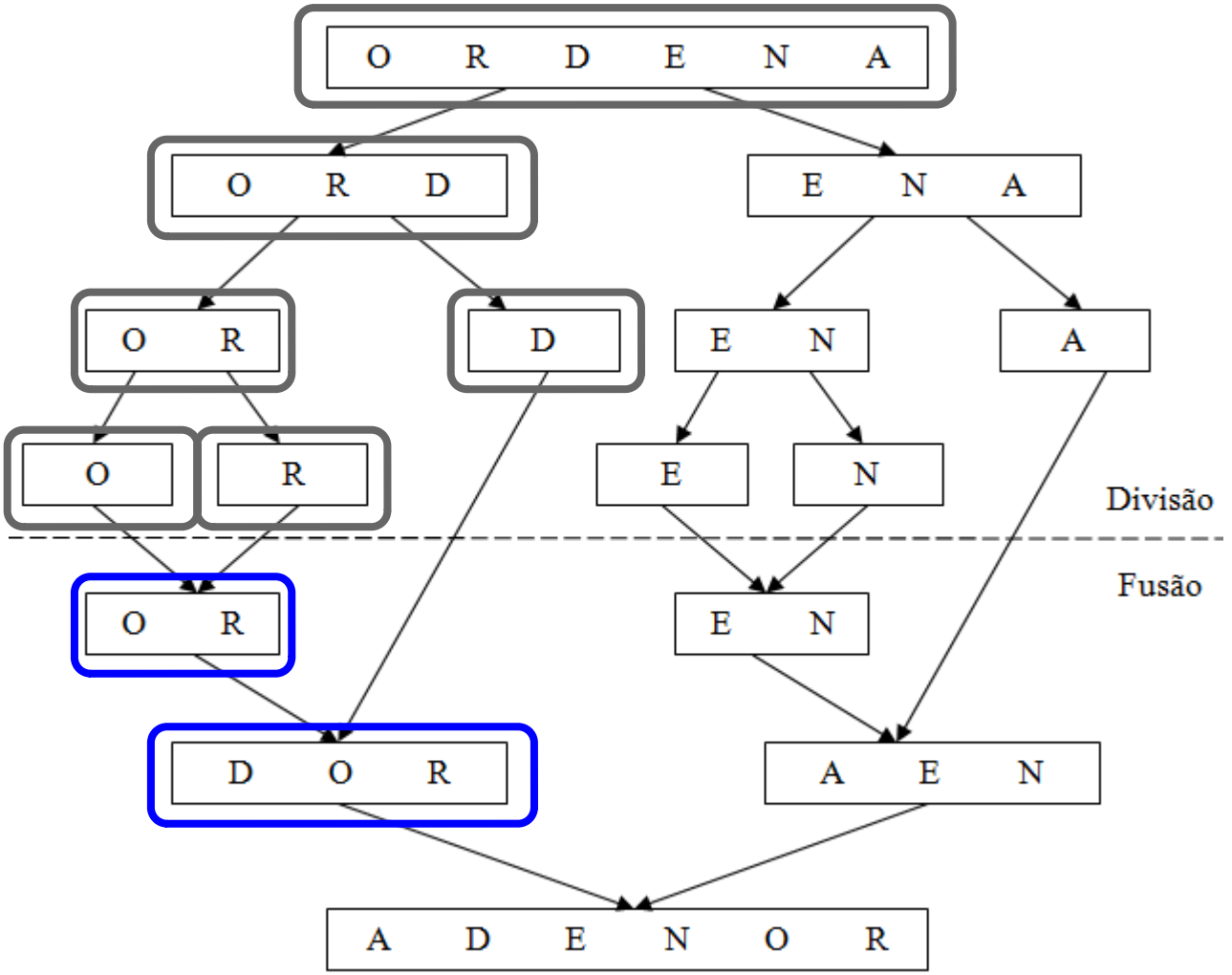
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

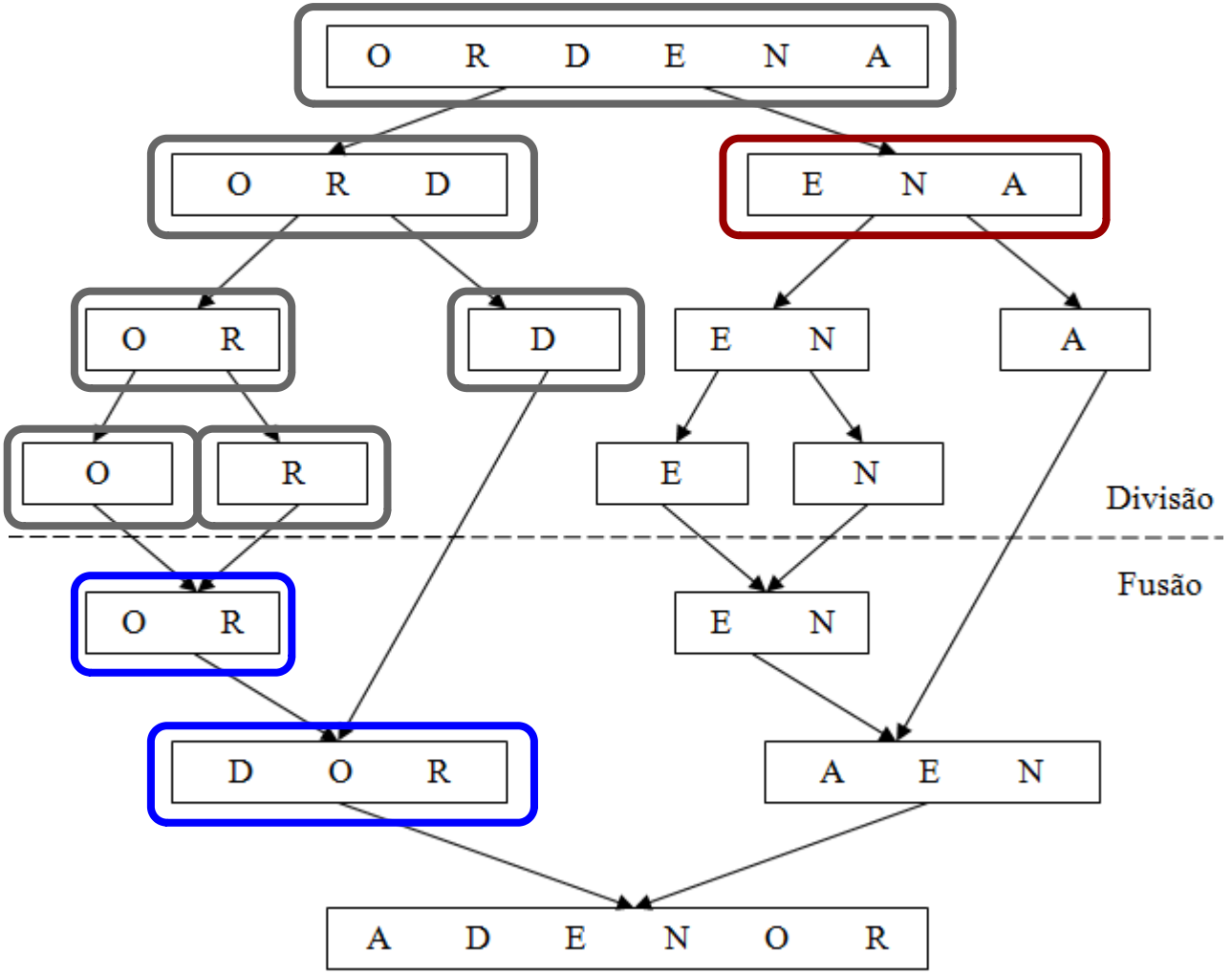
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

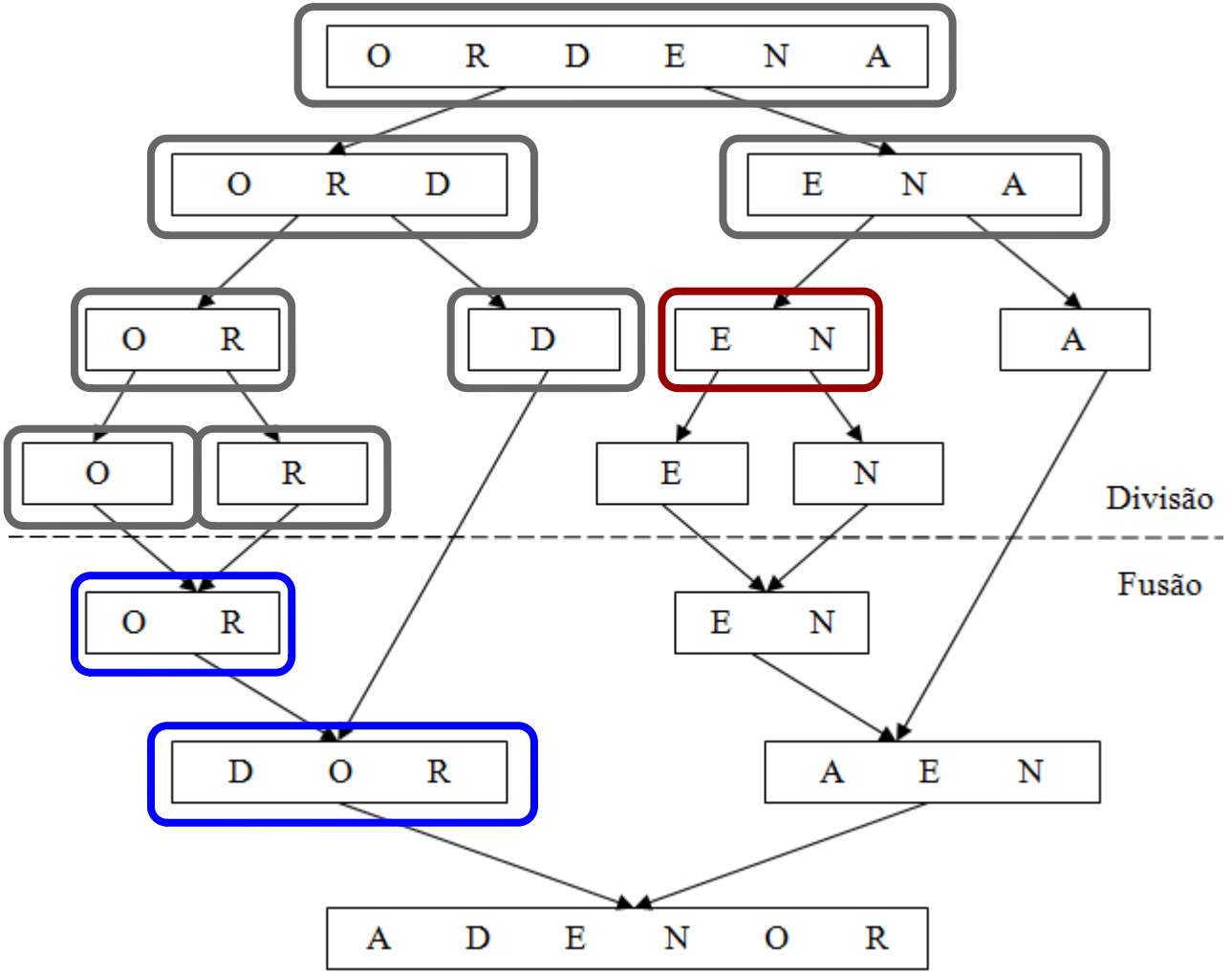
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

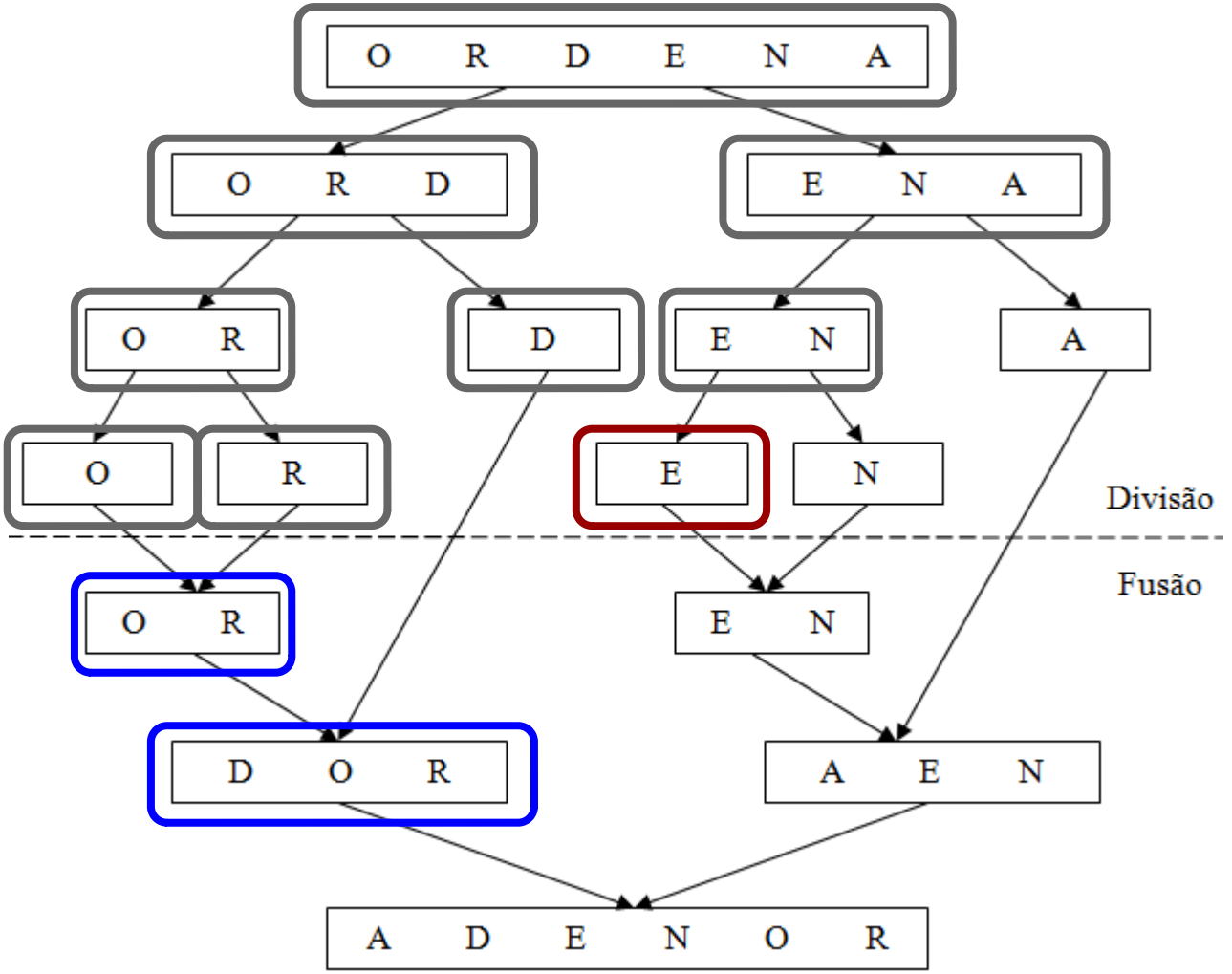
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

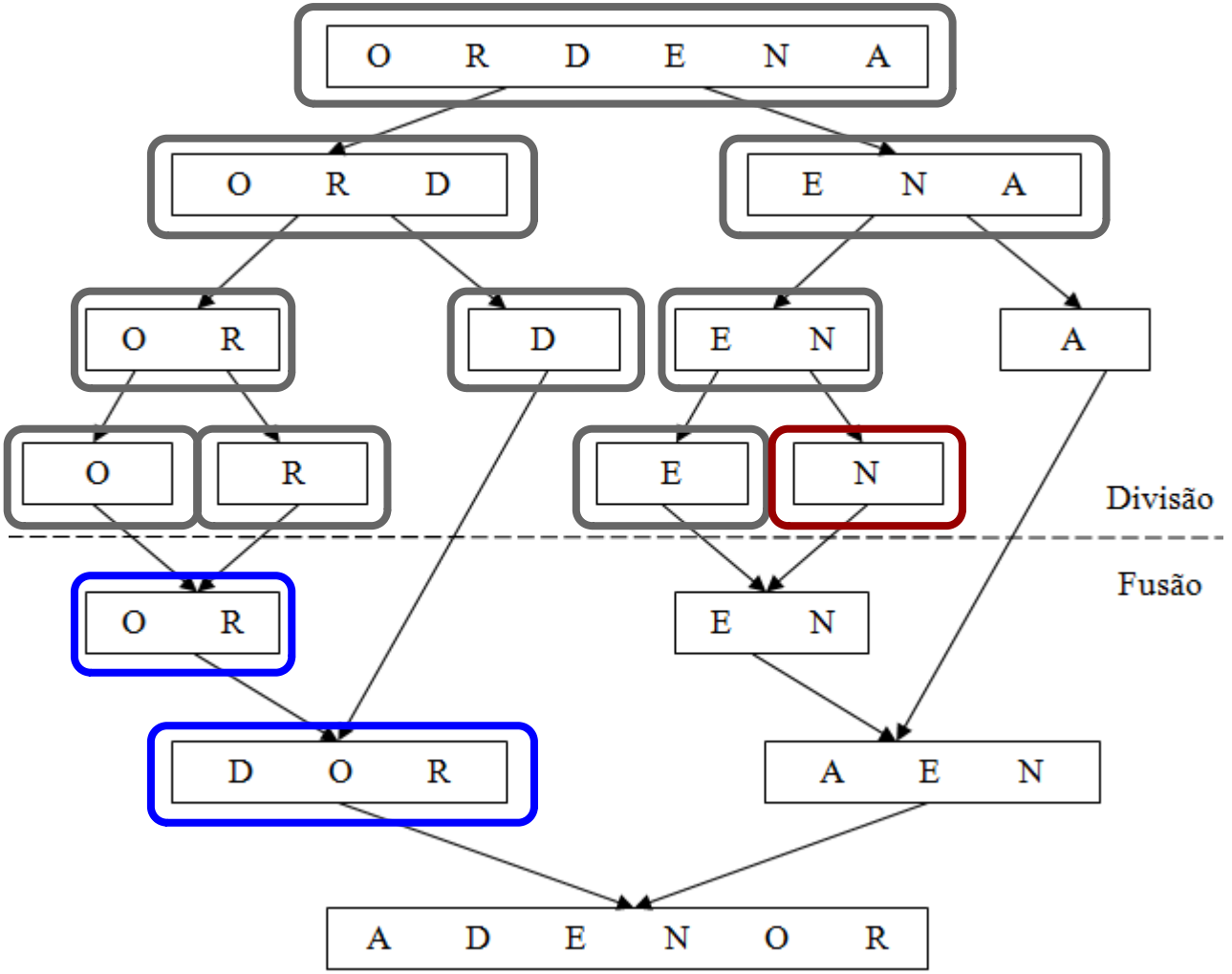
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

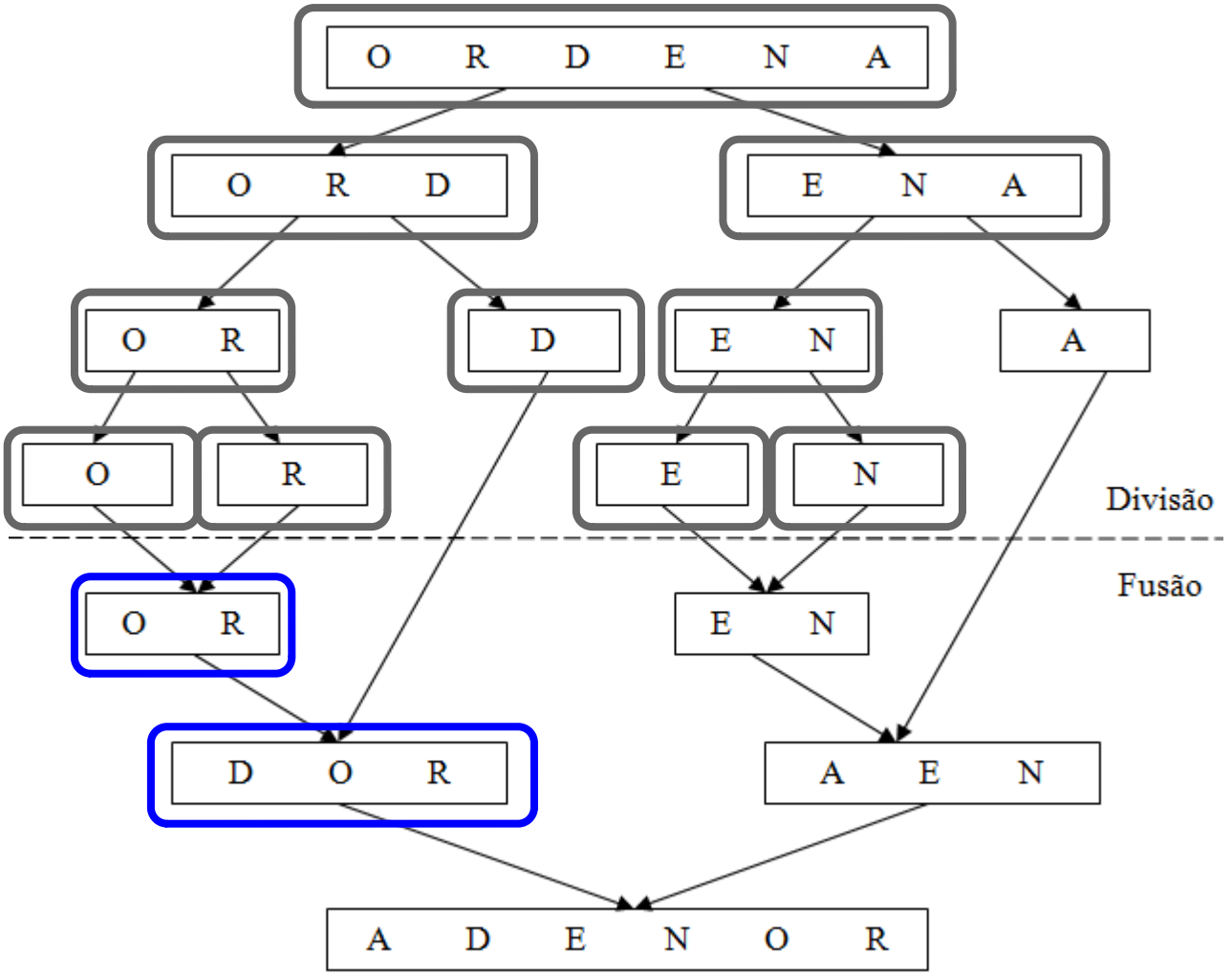
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

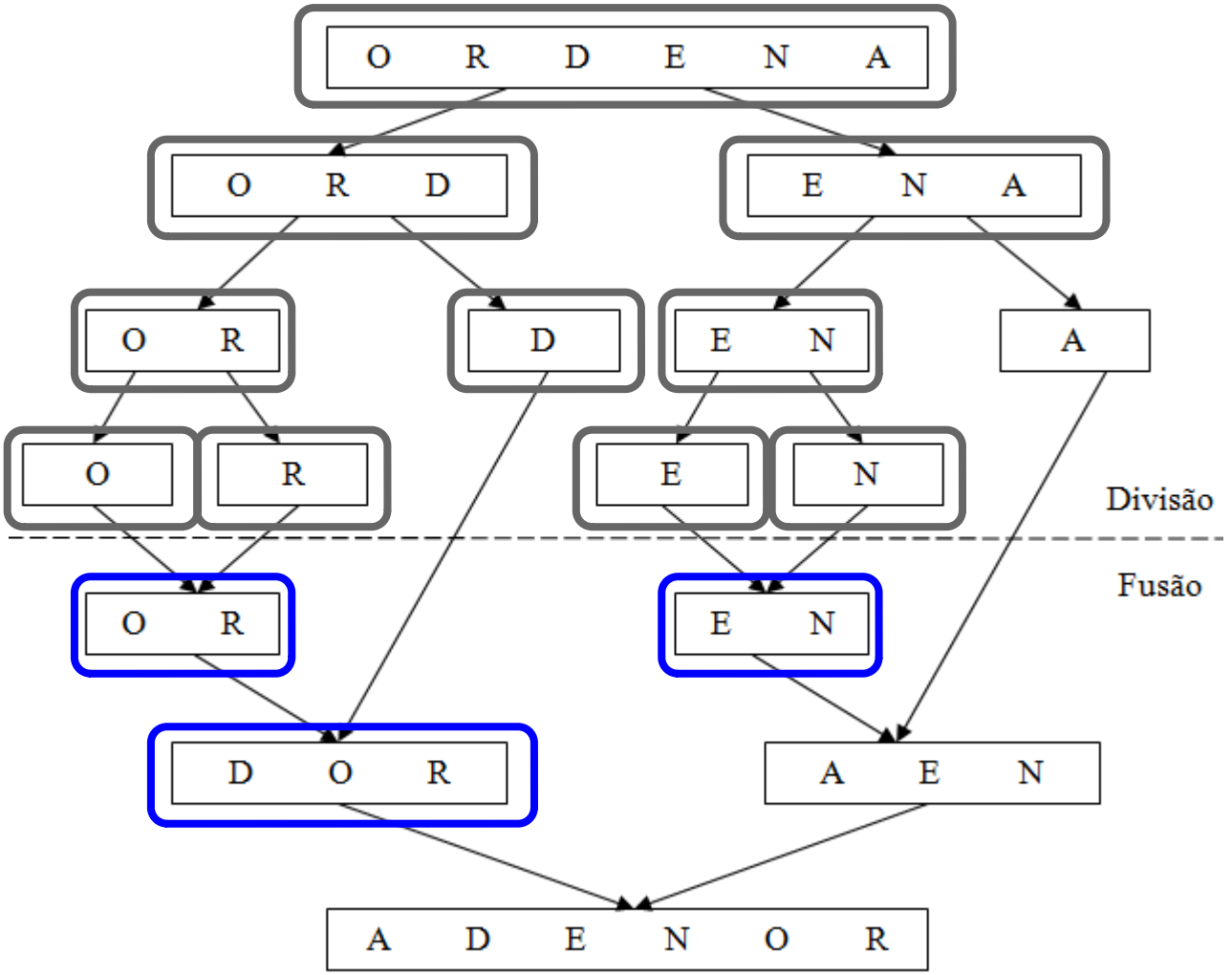
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

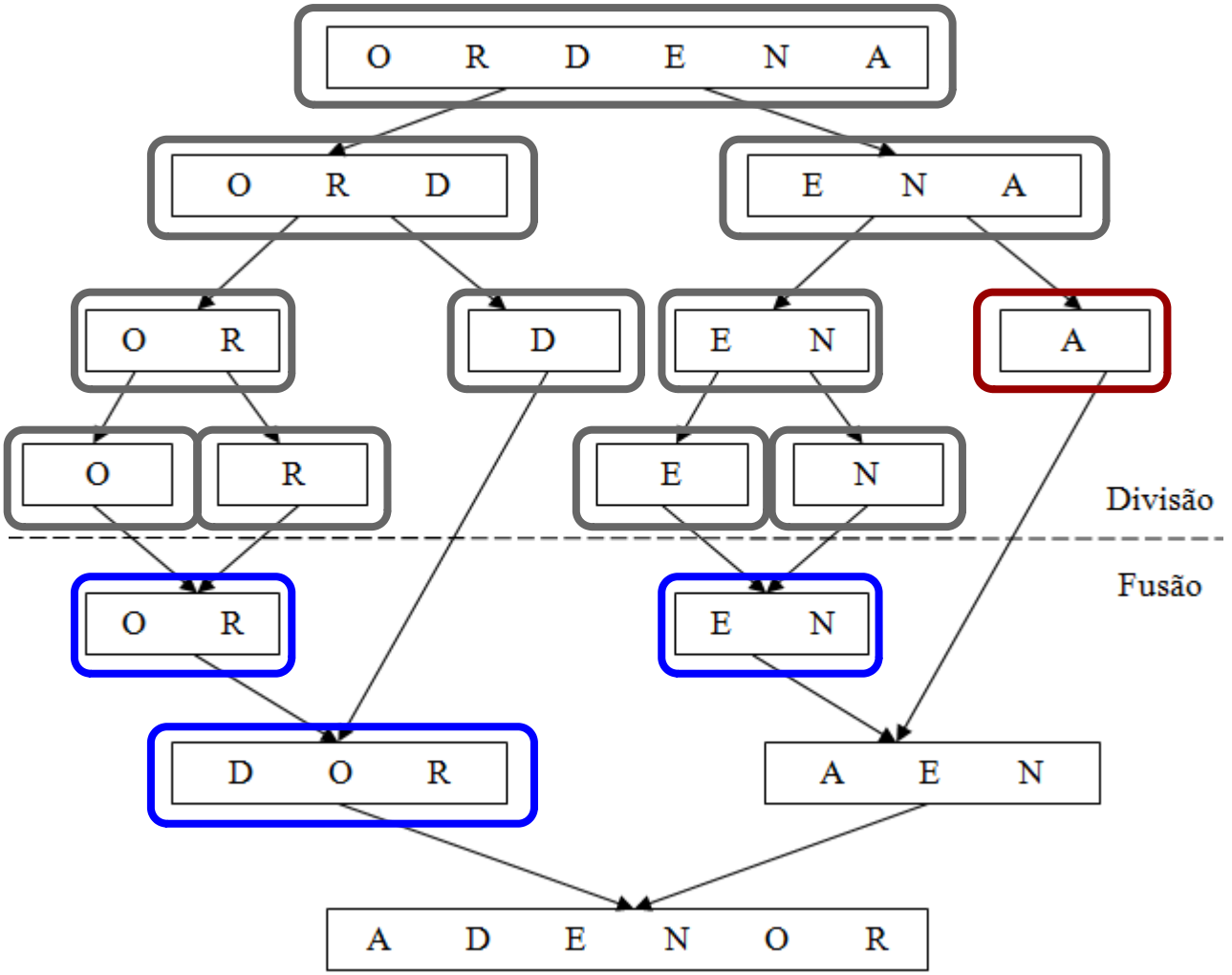
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

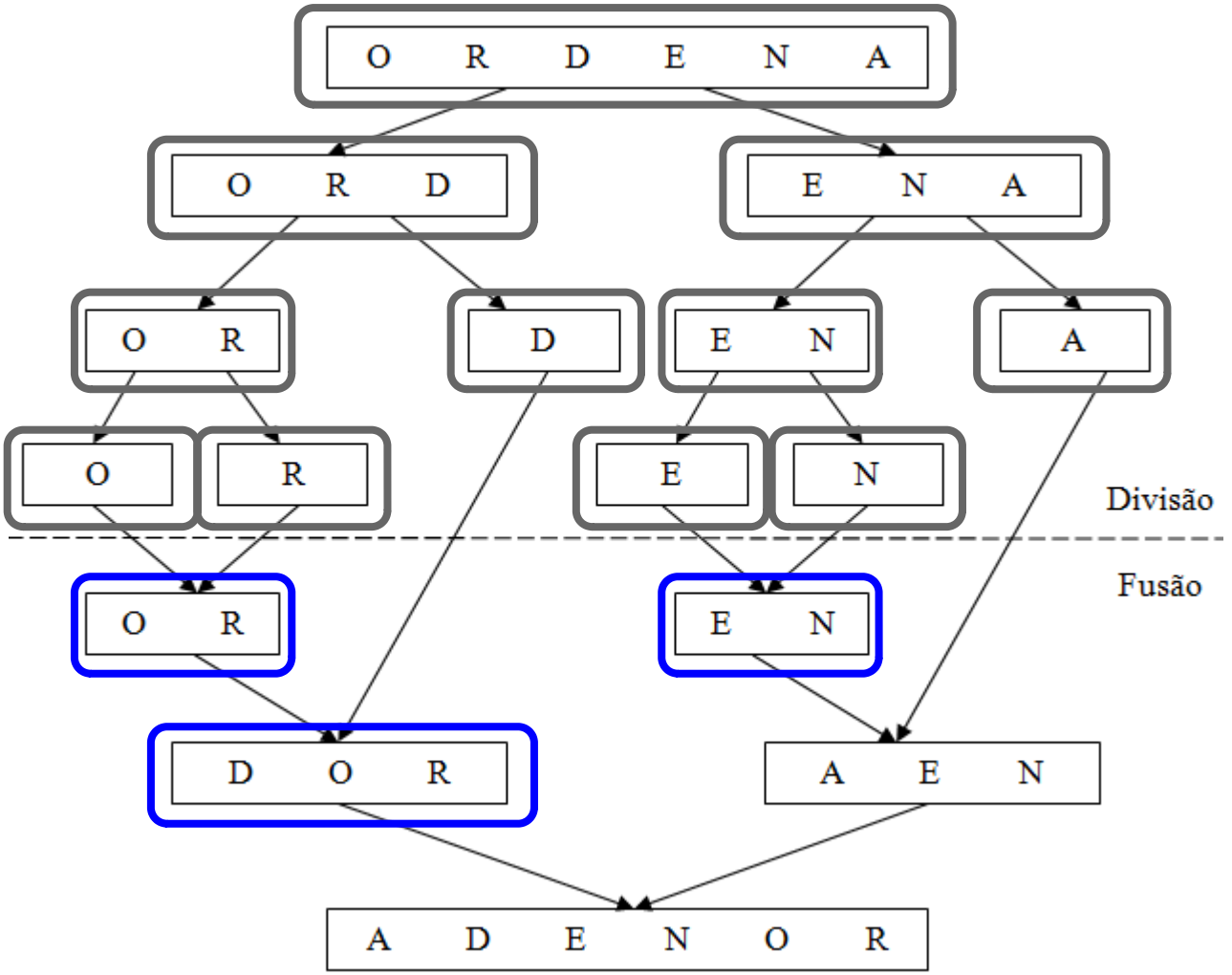
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

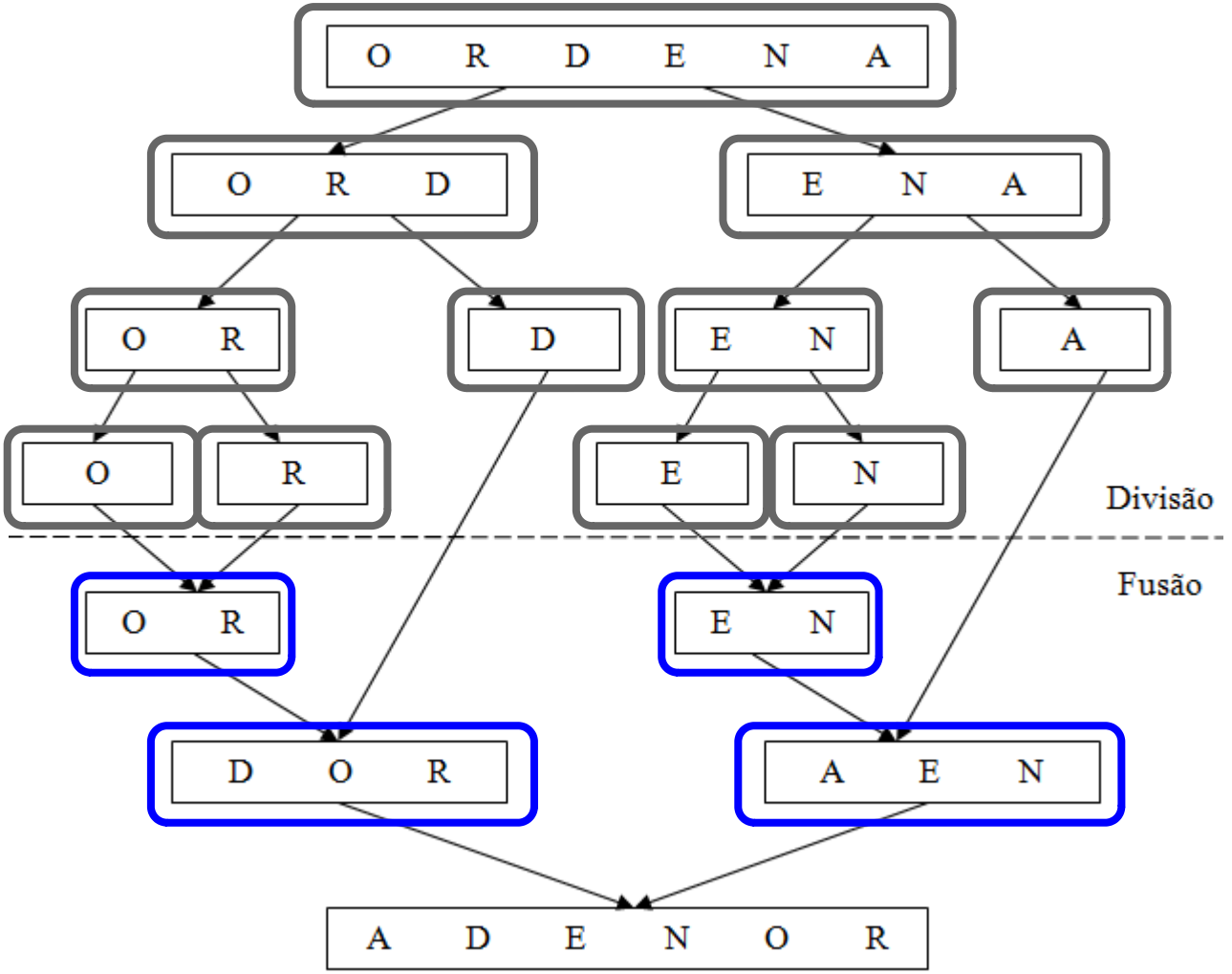
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

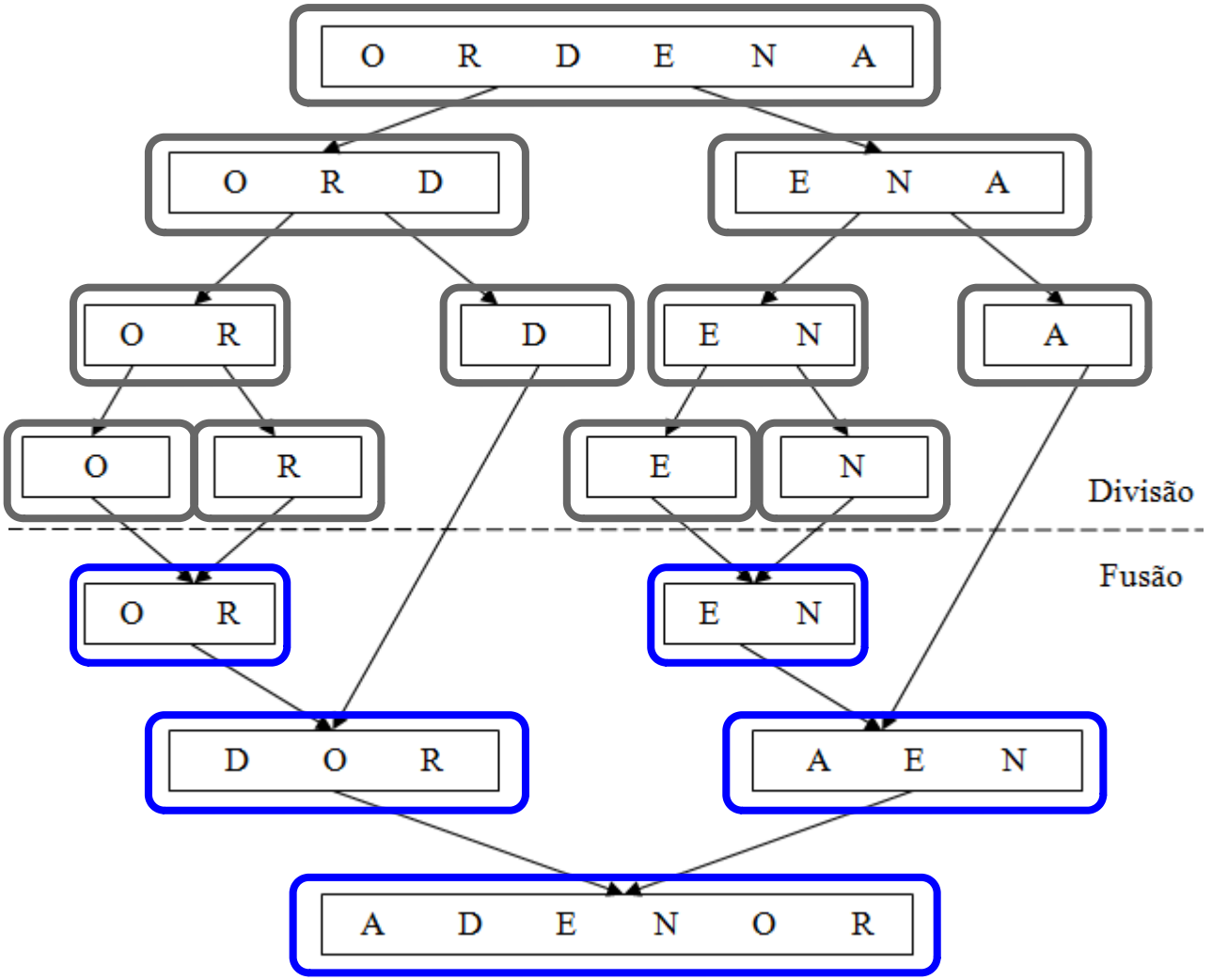
Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

Merge Sort


O R D E N A
O R D
E N A
O R
D
E N
A
O
R
E
N
Divisão
Fusão
O R
E N
D O R
A E N
A D E N O R

Merge Sort


```
int *aux; // Variável global
void mergeSort(int v[], int n) {
     aux = new int[n];
     mergesort(v, 0, n - 1);
     delete[] aux;
}

int main() {
     int TAM = 10;
     int v[] = {5, 3, 2, 2, 1, 6, 5, 7, 9, 10};
     mergeSort(v, TAM);
     return EXIT_SUCCESS;
}
```

Merge Sort


```
void mergesort(int v[], int esq, int dir) {
    if (esq < dir) {
        int meio = (esq + dir) / 2;
        mergesort(v, esq, meio);
        mergesort(v, meio + 1, dir);
        intercala(v, esq, meio, dir);
     }
}
```

Merge Sort

```
void intercala(int v[], int esq, int meio, int dir) {
    int i, j;
    i=meio+1;
    while(i>esq){
        aux[i - 1] = v[i - 1];
        i--;
    }
    j=meio;
    while(j<dir){
        aux[dir + meio - j] = v[j + 1];
        j++;
    }
    for(int k = esq; k <= dir; k++) {
        if (aux[j] < aux[i]) {
            v[k] = aux[j];
            j--;
        }
        else {
            v[k] = aux[i];
            i++;
        }
    }
}
```

Merge Sort

- O número de comparações a serem realizadas pelo método de ordenação *Merge Sort* não depende de como os elementos estão organizados no vetor.

- Assim, para o pior caso, melhor caso e caso médio, a complexidade do Merge Sort é proporcional a ***nlgn***.

Merge Sort

- As vantagens do *Merge Sort* são que ele é um método eficiente e não sensível à ordem em que os elementos aparecem no vetor.

- A principal desvantagem deste método é que ele requer uma quantidade de espaço extra proporcional ao tamanho do vetor.

Considerações Finais

- É responsabilidade do desenvolvedor saber decidir qual método de ordenação melhor atende às suas necessidades.

- Para isso, ele precisa conhecer bem os detalhes do seu problema, bem com as características dos principais métodos de ordenação.

Considerações Finais

- Cabe ressaltar que há outros métodos de ordenação interessantes que não foram abordados nesta disciplina.

- Por exemplo, o método de ordenação *Heap Sort* executa *nlgn* comparações no pior caso e também não é sensível à ordem inicial dos elementos do vetor de entrada.