FAGAMMON
FACULDADE PRESBITERIANA GAMMON
30 ANOS

Algoritmos e Estrutura de Dados II

Aula 8

Ordenação Eficiente - MergeSort

Claudiane Maria Oliveira
claudianeo@gmail.com

Introdução

- Ordenação Eficiente
  - Quick Sort
  - **Merge Sort**
  - Shell Sort

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.

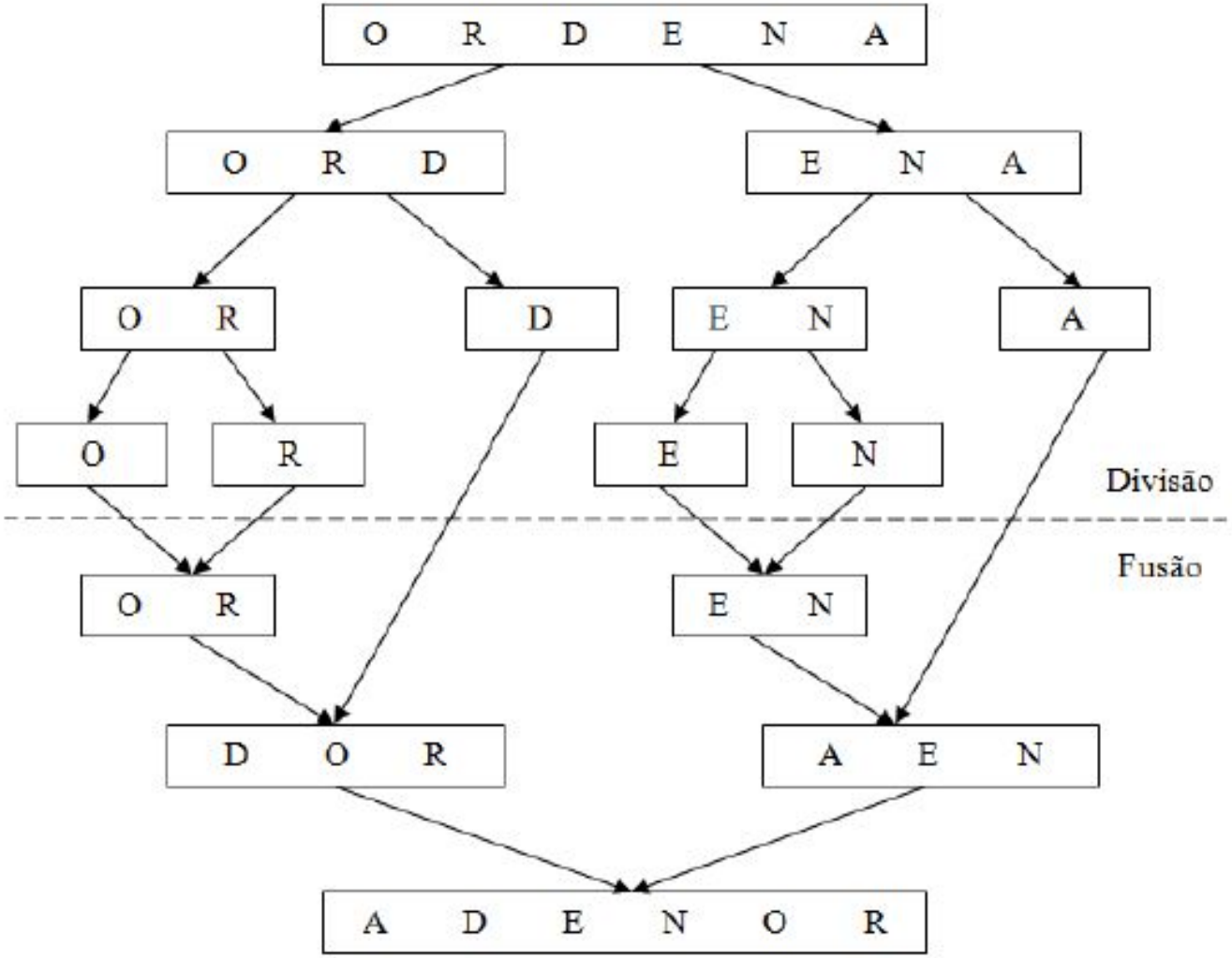
- Assim, Merge Sort é ideal para aplicações que precisam de ordenação eficiente, que não toleram performances ruins no pior caso e que possuam espaço de memória extra disponível.

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


```
void merge(int v[], int esq, int meio, int dir) {
    int i, j;
    for(i = meio + 1; i > esq; i--) aux[i - 1] = v[i - 1];
    for(j = meio; j < dir; j++) aux[dir + meio - j] = v[j + 1];
    for(int k = esq; k <= dir; k++) {
        if (aux[j] < aux[i]) {
                v[k] = aux[j]; j--;
        } else {
                v[k] = aux[i]; i++;
        }
    }
}
void mergesortR(int v[], int esq, int dir) {
        if (esq < dir) {
                int meio = (esq + dir) / 2;
                mergesortR(v, esq, meio);
                mergesortR(v, meio + 1, dir);
                merge(v, esq, meio, dir);
        }
}
```

Merge Sort


```
int *aux;
void mergeSort(int v[], int n) {
     aux = new int[n];
     mergesortR(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

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

- Assim, para o pior caso, melhor caso e caso médio, leva ao seguinte número de comparações para um vetor com n elementos: $C(n) = n\lg n$.

Merge Sort

- Como visto anteriormente, essa equação de recorrência leva ao seguinte número de comparações para um vetor com n elementos: $C(n) = n\lg n$.

Otimizações para o Merge Sort

- A mesma estratégia proposta para o Quick Sort, de se utilizar um método de ordenação mais simples para ordenar subvetores menores, pode ser aplicada ao Merge Sort.

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 e estável.

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

Merge Sort

- 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.

FAGAMMON
FACULDADE PRESBITERIANA GAMMON
30 ANOS

Algoritmos e Estrutura de Dados II

Aula 8

Ordenação Eficiente - MergeSort

Claudiane Maria Oliveira
claudianeo@gmail.com