Algoritmos de Ordenação n-Logarítmicos

Pesquisa, Ordenação e Técnicas de Armazenamento

Prof. Me. Orlando da Silva Junior

Agenda

- ❑Conceituação de partição e divisão e conquista
- ❑O Quicksort e o Mergesort
- ❑Análise de performance
- ❑Aplicabilidade dos algoritmos
- ❑Exercícios diversos com ordenação

Divisão e Conquista

Técnica para projeto de algoritmos.

1. **Dividir:** o problema é dividido em subproblemas menores até que a solução atômica seja encontrada;

2. **Conquistar:** os subproblemas suficientemente pequenos são resolvidos;

3. **Combinar:** as soluções dos subproblemas pequenos são combinados até obter a solução do problema maior

Júlio César (100 a.C - 44 a.C)

"divide et impera"

Merge Sort

Ideia:

- ***Dividir:*** *quebra a sequência de n elementos em duas subsequências de $n/2$ elementos*
- ***Conquistar:*** *ordena-se ambas as sequências recursivamente*
- ***Combinar:*** *intercala-se as sequências para formar a solução original do problema*

- **Prós:** ligeiramente melhor que outros algoritmos para entradas suficientemente grandes
- **Contras:** requer, no mínimo, o dobro de memória em comparação com outros algoritmos

Melhor caso: $O(n \lg n)$

Pior caso: $O(n \lg n)$

6 5 3 1 8 7 2 4

Merge Sort

```
public void mergeSort(int[] v, int n) {
    int[] aux = new int[n];
    mergeSort(v, aux, 0, n - 1);
}
```

```
private void mergeSort(int[] v, int[] aux, int e, int d) {
    if (d <= e) {
        return;
    }
    int m = (d + e) >> 1;
    mergeSort(v, aux, e, m);
    mergeSort(v, aux, m + 1, d);
    merge(v, aux, e, m, d);
}
```

Pior caso: $O(n \lg n)$

Merge Sort

```java
public void mergeSort(int[] v, int n) {
    int[] aux = new int[n];
    mergeSort(v, aux, 0, n - 1);
}
```

Aloca um vetor auxiliar e chama o método de ordenação para todo o vetor.

```java
private void mergeSort(int[] v, int[] aux, int e, int d) {
    if (d <= e) {
        return;
    }
    int m = (d + e) >> 1;
    mergeSort(v, aux, e, m);
    mergeSort(v, aux, m + 1, d);
    merge(v, aux, e, m, d);
}
```

Critério de parada: encerra a recursão quando o lado direito for menor que o esquerdo.

Encontra o pivô.

Aplica a ordenação do início ao meio e do meio + 1 ao fim.

Junta tudo ao final.

Merge Sort

```
public void mergeSort(int[] v, int n) {
    int[] aux = new int[n];
    mergeSort(v, aux, 0, n - 1);
}
```

```
private void mergeSort(int[] v, int[] aux, int e, int d) {
    if (d <= e) {
        return;
    }
    int m = (d + e) >> 1;
    mergeSort(v, aux, e, m);
    mergeSort(v, aux, m + 1, d);
    merge(v, aux, e, m, d);
}
```

$\Theta(1)$

$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$

$\Theta(n)$

Merge Sort

```
private void merge(int[] v, int[] aux, int e, int m, int d) {
    int i, j, k;
    i = k = e;
    j = m + 1;

    while ((i <= m) && (j <= d)) {
        if (v[i] < v[j]) {
            aux[k++] = v[i++];
        } else {
            aux[k++] = v[j++];
        }
    }
    while (i <= m) {
        aux[k++] = v[i++];
    }
    while (j <= d) {
        aux[k++] = v[j++];
    }
    while (e <= d) {
        v[e] = aux[e++];
    }
}
```

1º laço: no pior caso, é executado até alcançar algum dos limites (m ou d), restando apenas 1 elemento a inserir.

2º laço: insere o que restou do lado esquerdo no primeiro laço.

3º laço: insere o que restou do lado direito no primeiro laço.

4º laço: transfere os elementos ordenados do vetor auxiliar para o original.

Merge Sort

```
private void merge(int[] v, int[] aux, int e, int m, int d) {
    int i, j, k;
    i = k = e;
    j = m + 1;

    while ((i <= m) && (j <= d)) {
        if (v[i] < v[j]) {
            aux[k++] = v[i++];
        } else {
            aux[k++] = v[j++];
        }
    }
    while (i <= m) {
        aux[k++] = v[i++];
    }
    while (j <= d) {
        aux[k++] = v[j++];
    }
    while (e <= d) {
        v[e] = aux[e++];
    }
}
```

$\Theta(n)$

Processa exatamente todos os elementos: da esquerda ao meio e do meio à direita.

Quick Sort

Ideia: *eleger um pivô e garantir que todos os elementos à esquerda são menores e todos os elementos à direita são maiores. Ao final do processo, o pivô estará em sua posição final. Repita o processo recursivamente.*

- **Prós:** é o método de ordenação mais rápido e não requer memória adicional
- **Contras:** difícil implementação, ineficiente para entradas altamente desbalanceadas

Melhor caso: $O(n \lg n)$

Pior caso: $O(n^2)$

6 5 3 1 8 7 2 4

Quick Sort

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

```
private void quickSort(int[] v, int e, int d) {
    if (d <= e) {
        return;
    }

    // Parte 1
    int i, j;
    …

    i = e;
    j = d;
    // Parte 2
    while (true) {
        …
    }
    v[i] = pivo;
    // Parte 3
    …
}
```

Pior caso: $O(n^2)$

Quick Sort

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

Chama o método de ordenação para o vetor inteiro.

$$\Theta(1)$$

```
    // Parte 1
    int i, j;
    i = ((int) Math.random()) % (d - e) + e + 1;
    int pivo = v[i];
    v[i] = v[e];
    v[e] = pivo;
```

Parte 1: atribuição do pivô.

$$\Theta(1)$$

```
    // Parte 2
        …
```

```
    // Parte 3
    quickSort(v, e, i - 1);
    quickSort(v, i + 1, d);
```

Parte 3: particionamento e chamada à ordenação (da esquerda ao pivô e do pivô + 1 à direita).

$$\Theta(1) + T(n-1)$$

Quick Sort

```
// Parte 2
while (true) {
    while ((j > i) && (v[j] > pivo)) {
        j--;
    }
    if (i == j) {
        break;
    }
    v[i] = v[j];
    i++;
    while ((i < j) && (v[i] < pivo)) {
        i++;
    }
    if (i == j) {
        break;
    }
    v[j] = v[i];
    j--;
}
```

Parte 2: ordenação dos elementos por comparação em apenas metade do vetor, considerando cada particionamento da recursão.

$$O(n)$$

Exercícios

1. Explique em poucas palavras a ideia geral do algoritmo Merge Sort.
2. Explique em poucas palavras a ideia geral do algoritmo Quick Sort.
3. Sabe-se que o pior caso do algoritmo Quick Sort é $O(n^2)$. Em qual situação isso acontece?

Exercícios

1. Modifique a implementação do Quick Sort para permitir a ordenação de cadeias de caracteres (String).
   - `String[] v = {“João”, “Maria”};`
2. Modifique a implementação do Merge Sort para permitir a ordenação de caracteres (char).
   - `char[] v = {‘d’, ‘b’, ’e’};`