O algoritmo de ordenação *Radix* usa do princípio de ordenar uma casa decimal de cada vez. Temos duas variantes do radix sort, a primeira é *Least significant digit* (LSD – Dígito menos significativo) começa ordenando pelas unidades, em seguida ordena somente as dezenas, depois centenas assim por diante até estarem todas ordenadas. A outra variante *Most significant digit* (MSD – Dígito mais significativo) segue-se os mesmos passos, com a única diferença que é de trás para frente, começando do número decimal mais alto, por exemplo, dezena de milhar, milhar, centena, dezena até chegar na unidade.

**Exemplo prático:**

Tendo uma lista de números, embaralhados de 0 à 20, vamos supor a sequência de números {13, 3, 17, 6, 9, 15, 16, 0, 1, 8, 20, 7, 14, 19, 5, 2, 18, 4, 11, 12} e vamos desembaralhar ela.

**1º Passo:** Separar 10 espaços

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

0 1 2 3 4 5 6 7 8 9

**2º Passo**: Colocar o número com a primeira casa (unidade) na casa correspondente de 0 a 10

O primeiro número da lista é o 13, a unidade dele é o 3, então vou colocar o 13 na casa de número 3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 13 |  |  |  |  |  |  |

0 1 2 3 4 5 6 7 8 9

O segundo número é o 3, sua *unidade* também é 3, então vou colocá-lo na casa de número 3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 13, 3 |  |  |  |  |  |  |

0 1 2 3 4 5 6 7 8 9

O terceiro número é 17, vamos colocá-lo na casa 7

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 13, 3 |  |  |  | 17 |  |  |

0 1 2 3 4 5 6 7 8 9

E assim sucessivamente até terminar todos os números

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0, 20 | 1, 11 | 2, 12 | 13, 3 | 14, 4 | 15, 5 | 6, 16 | 17, 7 | 8, 18 | 9, 19 |

**3º Passo**: Repetir o segundo passo, mas ao invés de colocar as unidades, vamos colocar cada dezena na sua casa correspondente:

A lista na ordem atual {0, 20, 1, 11, 2, 12, 13, 3, 14, 4, 15, 5, 6, 16, 17, 7, 8, 18, 9, 19}

O primeiro número da lista é 0, sua decimal também é 0, então vamos colocá-lo na casa 0

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 |  |  |  |  |  |  |  |  |  |

0 1 2 3 4 5 6 7 8 9

O segundo número é 20, sua decimal é 2, então vamos colocá-lo na casa 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 |  | 20 |  |  |  |  |  |  |  |

0 1 2 3 4 5 6 7 8 9

O terceiro número da casa é 1, então sua decimal será 0, sendo como se fosse “01”, será colocada na casa 0

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0, 01 |  | 20 |  |  |  |  |  |  |  |

0 1 2 3 4 5 6 7 8 9

O quarto número da casa é 11, sua decimal é 1, o número será colocado na casa 1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0, 1 | 11 | 20 |  |  |  |  |  |  |  |

0 1 2 3 4 5 6 7 8 9

E assim sucessivamente até termos

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 | 11, 12, 13, 14, 15, 16, 17, 18, 19 | 20 |  |  |  |  |  |  |  |

0 1 2 3 4 5 6 7 8 9

Agora em ordem nessa sequência temos: {0,1,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,20}

Se fosse necessário ordenar uma lista com números tendo mais casas decimais, como números de 100, 1.000 ou maiores, é só repetir o 3º passo com a casa da centena, milhar e assim sucessivamente.

O algoritmo *Quick Sort* se baseia no princípio qual, em uma lista ordenada, se você escolhe um número aleatório (pivô), todos números a sua esquerda são menores e todos a sua direita são maiores que o mesmo. Para iniciar o estamos considerando o pivô como número do meio da lista. Iniciaremos a varredura dos extremos, a esquerda iniciando da posição primeira e o da direita iniciando da última posição, indo em direção ao centro realizando a varredura à procura de um número à direita do pivô que seja menor que o próprio pivô e um número da esquerda que seja maior que o pivô e a partir daí, realizar a troca entre esses dois números fazendo isso sucessivamente até garantir a ordenação da lista.

Para exemplifica consideraremos essa sequência de números: 10, 5, 6, 4, 2, 1 e 0.

Com esses dados seguiria a seguinte lógica:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Chamadas da função** | | | | **Resultado** |
| E: 1º, D: 7º, P: 4  **10**,5,6,4,2,1,**0**,  10 ≥ 4 ≥ 0 → **10**><**0** | E: 2º, D: 6º, P: 4  0,**5**,6,4,2,**1**,10,  5 ≥ 4 ≥ 1 → **5**><**1** | E: 3º, D: 5º, P: 4  0,1,**6**,4,**2**,5,10,  6 ≥ 4 ≥ 2 → **6**><**2** | E: 5º, D: 6º, P: 5  0,1,2,4,**6**,**5**,10,  6 ≥ 5 ≥ 5 → **6**><**5** | 0,1,2,4,6,5,10 |

A função do *Quick Sort* para atender ao seu proposito utiliza o princípio da recursividade chamando a função até que seja concluída. Isso em lógica de programação representado em pseudocódigo ficará da seguinte forma:

quickSort(array, 0, arr.comprimento - 1)

funcao quickSort(array:inteiro[], esquerda:inteiro, direita:inteiro):vazio

início

*... algoritmo de ordenação ...*

fim-funcao

Escolha do pivô, nesse caso foi escolhido o elemento do meio, alocado na variável “p”:

p 🡨 array[(esquerda + direita) / 2]

A partir do momento que foi escolhido o pivô é executado o laço para comparar os dois lados do pivô

Laço será executado enquanto e esquerda e direita não se encontrarem

enquanto e <= d faça

...

fim-enquanto

Dentro desse laço será feito código de varredura e troca para encontrar o número maior que o pivô que esteja à direita e menor que o pivô a esquerda e trocá-los.

Laço fará iteração enquanto esquerda for menor ou igual ao pivô (que é o que queremos que seja) ou menor que a direita.

enquanto array[e] < p && e < direita faça

e++

fim-enquanto

E da mesma forma será feita a iteração enquanto direita for maior ou igual ao pivô (que é o que queremos que seja) ou maior que a esquerda.

enquanto array[d] > p && d > esquerda faça

d--

fim-enquanto

Troca é feita, com a variável auxiliar “t”, apenas se a posição da esquerda é menor ou igual a posição da direita, indicando que os dois lados realizaram sua varredura.

se e <= r faça

t 🡨 array[e]

array[e] 🡨 array[d]

array[d] 🡨 t

e++

d--

fim-se

E ao final, após a efetuação esse troca é executado o código para averiguar a necessidade de rechamada do algoritmo *Quick Sort.*

Se posição da direita for maior que a esquerda, função é chamada com restauro da posição de esquerda (0).

se d > esquerda faça

quickSort(array, esquerda, d)

fim-se

Se posição da esquerda for maior que a direita, funcao é chamada com restauro da posição de esquerda (array.comprimento - 1) .

se e < direita faça

quickSort(array, e, direita)

fim-se

se naoOrdenado faça

quickSort(array, posicaoEsquerda, posicaoDireita)

fim-se