Universidade Federal do Ceará - Campus Quixadá
QXD0010 – Estruturas de Dados – 2025.2
Prof. Atílio Gomes

Ordenação por Intercalação — Mergesort

```c
/* A funcao recebe vetores crescentes A[p..q] e A[q+1..r]
 * e rearranja A[p..r] em ordem crescente */
void Intercala (int A[], int p, int q, int r) {
  int W[r-p+1]; // Vetor auxiliar
  int i = p;
  int j = q+1;
  int k = 0;

  // Intercala A[p..q] e A[q+1..r]
  while (i <= q && j <= r) {
    if (A[i] <= A[j])
      W[k++] = A[i++];
    else
      W[k++] = A[j++];
  }
  while (i <= q) W[k++] = A[i++];
  while (j <= r) W[k++] = A[j++];

  // Copia vetor ordenado W para o vetor A
  for (i = p; i <= r; i++)
    A[i] = W[i-p];
}

void mergesort(int A[], int p, int r) {
  if (p < r) {
    int q = (p + r) / 2; // Dividir
    // Conquistar
    mergesort(A, p, q);
    mergesort(A, q + 1, r);
    // Combinar
    Intercala(A, p, q, r);
  }
}
```

Listing 1: Algoritmo de ordenação MergeSort.

1. Versão recursiva. Escreva uma versão recursiva do algoritmo de intercalação. Antes, convém reformular o problema da seguinte maneira: dados vetores crescentes $U[0..m-1]$ e $V[0..n-1]$, produzir um vetor crescente $W[0..m+n-1]$ que contenha o resultado da intercalação dos dois vetores.

2. Estabilidade. Um algoritmo de intercalação é *estável* se não altera a posição relativa de elementos de mesmo valor. A função `Intercala` é estável? Se a comparação "`A[i] <= A[j]`" for trocada por "`A[i] < A[j]`", a função é estável ou não-estável?

3. O que acontece se trocarmos "`(p+r)/2`" por "`(p+r-1)/2`" no código de `mergesort`? O que acontece se trocarmos "`(p+r)/2`" por "`(p+r+1)/2`"?

4. A função `mergesort` é estável?

5. Mostre que o consumo de tempo da função `Intercala` é proporcional a n em cada rodada de `mergesort`. (Obs.: n é o tamanho do vetor dado como entrada.)

6. O algoritmo Mergesort (decrescente) usa uma função auxiliar, chamada intercala, que recebe como entrada dois vetores em ordem decrescente $v[p \ldots q-1]$ e $v[q \ldots r-1]$ e rearranja $v[p \ldots r-1]$ em ordem decrescente. Escreva a função intercala para o Mergesort decrescente.

7. Ordem descrescente. Escreva uma versão do algoritmo `mergesort` que rearranje um vetor $A[p..r]$ em ordem decrescente. (Será preciso reescrever o algoritmo de intercalação.)

8. Invocações repetidas de new e delete. Durante uma execução da função `mergesort`, a função `Intercala` é invocada muitas vezes e cada execução de `Intercala` invoca as funções `new` e `delete`. Para evitar as repetidas execuções de `new` e `delete`, escreva uma versão da função `mergesort` que incorpore o código da função de intercalação e invoque `new` e `delete` uma só vez.

9. O algoritmo `mergesort` (decrescente) usa uma função auxiliar, chamada `intercala`, que recebe como entrada dois vetores em ordem decrescente $v[p \ldots q-1]$ e $v[q \ldots r-1]$ e rearranja $v[p \ldots r-1]$ em ordem decrescente. Escreva a função intercala para o Mergesort decrescente.

Quicksort

```
1  /* Recebe um vetor A[p..r] com p <= r. Rearranja os
2   * elementos do vetor e devolve j em p..r tal que
3   * A[p..j-1] <= A[j] < A[j+1..r]. */
4  int separa (int A[], int p, int r) {
5      int c = A[r];
6      int j = p;
7      for (int k = p; k < r; k++) {
8          if (A[k] <= c) {
9              // troca A[k] com A[j]
10             int aux = A[k];
11             A[k] = A[j];
12             A[j] = aux;
13             j++;
14         }
15     }
16     A[r] = A[j];
17     A[j] = c;
18     return j;
19 }
20
21 /* Esta funcao rearranja o vetor A[p..r], com p <= r+1,
22  * de modo que ele fique em ordem crescente. */
23 void quicksort (int A[], int p, int r) {
24     if (p < r) {
25         int j = separa(A, p, r);
26         quicksort(A, p, j-1);
27         quicksort(A, j+1, r);
28     }
29 }
```

Listing 2: Algoritmo de ordenação QuickSort.

10. Discuta como a escolha do pivô pode influenciar no desempenho do método `quicksort`. Proponha estratégias para a escolha do pivô, visando melhorar seu desempenho.

11. Mostre a execução da função `separa` no vetor $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11 \rangle$.

12. Qual o valor que a função `separa` retorna quando todos os elementos do vetor $A[p..r]$ têm o mesmo valor?

13. Reescreva o procedimento de partição do `quicksort` (`Separa`) tomando como referência (pivô) o primeiro elemento (na implementação mais usual, toma-se o último).

14. Aplique a função `separa` a um vetor cujos elementos são todos iguais. Aplique a função a um vetor cujos elementos só têm dois possíveis valores. Aplique a função a um vetor crescente e a um decrescente.

15. Aplique a função `quicksort` a um vetor crescente com n elementos. Prove que o número de comparações entre elementos do vetor é proporcional a n^2. Repita o exercício com vetor decrescente.

16. Digamos que um vetor $A[p..r]$ está *arrumado* se existe j em $p..r$ que satisfaça $A[p..j-1] \leq A[j] < A[j+1..r]$. Escreva um algoritmo que decida se $A[p..r]$ está arrumado. Em caso afirmativo, o seu algoritmo deve devolver o valor de j.

17. Estabilidade A função `separa` produz um rearranjo estável do vetor?

18. Escreva uma versão recursiva do algoritmo de separação.

19. O que acontece se trocarmos "`if (p < r)`" por "if (p != r)" no código da função `quicksort`?

20. Compare o código da função `quicksort` com o da função `mergesort`. Discuta as semelhanças e diferenças.

21. A função `quicksort` produz uma ordenação estável?

22. Escreva uma versão não recursiva do algoritmo QuickSort.

23. Escreva uma versão do algoritmo Quicksort que rearranje uma lista encadeada de modo que ela fique em ordem crescente. Sua função não deve alocar novas células na memória.

24. Escreva uma implementação do algoritmo Quicksort que evite aplicar a função a vetores com menos que dois elementos.

25. [Tail Recursion] Mostre que a segunda invocação da função `Quicksort` pode ser eliminada se trocarmos o `if` por um `while` apropriado.