*Atividade de Computabilidade e Complexidade de Algoritmos*

Docente: M.a Ecilia Alves de Oliveira Migliori

GRUPO TURMA **CCA 99**

Membros:

- Leonardo Couto – RGM 22741241 – SEXTO SEMESTRE – TURMA 6B

- Gustavo Gabriel de Almeida – RGM 19027265 – SEXTO SEMESTRE – TURMA 6B

- Clarice Braga Tolentino – RGM 15040666 – SEXTO SEMESTRE – TURMA 6B

- Lucas Veiga Sgrott – RGM 18897983 – SEXTO SEMESTRE - TURMA 6B

- Pedro Bastos Constantino Junior – RGM 24502995 – SEXTO SEMESTRE - TURMA 6B

**Problema - 01**

**Melhor Caso – Elemento na primeira posição**

**R: O(1)**

**Caso Médio – Elemento se encontra em N / 2**

**R: O(N)**

**Pior Caso – Elemento na ultima posição do Vetor**

**R: O(N)**

**Problema – 02**

**Melhor Caso – Elemento se encontra em N / 2**

**R: O(1)**

**Caso Médio – Elemento na posição inicial ou no final do vetor**

**R: O(log n)**

**Pior Caso – Elemento não encontrado**

**R: O(log n)**

**Problema - 03**

**R1: O(1)**

**R2: O(n)**

**Problema - 04**

**R: O(n3)**

**Problema – 05**

**Problema – 06**

**Equação de Recorrência: T(n) =1 SE (N = 1)**

**Equação de Recorrência: T(n) =2T(n/ 2) + n SE (N > 1)**

**R1: Complexidade O(c) se n = 1**

**R2: Divide-se o vetor V em duas metades recursivamente, cada metade é ordenadas e ambas são intercaladas criando um novo vetor ordenado, paralelamente usa um outro vetor para auxiliar a intercalação.**

**Complexidade: O (n log(n)) – Pior Caso**

**Quantidade de Ordenações : 19999 para n = 10.000**

**Problema – 07**

**R1: O(n) (Melhor Caso)**

**Equação de Recorrência: T(n) = 2T(n / 2) + O(n) (Melhor Caso)**

**Equação de Recorrência : T(n) = T(n – 1) + O(n^2) (Pior caso)**

**Quantidade de Trocas: 20.000 para n = 10.000**

**R2: Na primeira chamada recursiva temos o número inicial como pivô. O processo segue em diante. Verifica-se se 0 é menor que o pivô, então troca-se i e J, Quando I ultrapassa J então troca-se o pivô por Vetor(j)**

**Logo mais, o pivô ocupa a posição corretamente, o que nos resta duas metades para serem ordenadas, então aplica-se o algoritmo mais uma vez recursivamente nas duas metades restantes.**

**R3: O melhor caso ocorre quando as duas metades são balanceadas, ou seja, cada partição com n divido por 2 elementos.**

**----- Alguns algoritmos usados na pesquisa (C#) -----**

**Sites usados para busca de algoritmos: StackOverFlow, ExceptionNotFound,GeeksForGeeks,C-SharpCorner,Microsoft Docs.**

**Créditos vão para os respectivos autores dos algoritmos, e para o grupo desse trabalho que usou o código e modificou para fins de melhor coleta de informações.**

**-- MERGE SORT--**

1. **class** Program
2. {
4. **static** **int** count = 0;
5. **static** **void** Main(**string**[] args)
6. {
8. var rand = **new** Random();
10. **int**[] arr = **new** **int**[10000];
12. **for**(**int** i = 0; i < arr.Length; i++)
13. {
14. arr[i] = rand.Next(1, 10);

17. }
19. MergeSort(arr, 0, arr.Length - 1);
20. Console.WriteLine("Valores Ordenados:");
21. **for** (**int** i = 0; i < arr.Length; i++)
22. Console.WriteLine(arr[i]);
24. Console.WriteLine("Quantidade de ordenações : " + count);
25. }

28. **static** **void** Merge(**int**[] input, **int** left, **int** middle, **int** right)
29. {
30. **int**[] leftArray = **new** **int**[middle - left + 1];
31. **int**[] rightArray = **new** **int**[right - middle];
33. Array.Copy(input, left, leftArray, 0, middle - left + 1);
34. Array.Copy(input, middle + 1, rightArray, 0, right - middle);
36. **int** i = 0;
37. **int** j = 0;
38. **for** (**int** k = left; k < right + 1; k++)
39. {
40. **if** (i == leftArray.Length)
41. {
42. input[k] = rightArray[j];
43. j++;
44. }
45. **else** **if** (j == rightArray.Length)
46. {
47. input[k] = leftArray[i];
48. i++;
49. }
50. **else** **if** (leftArray[i] <= rightArray[j])
51. {
52. input[k] = leftArray[i];
53. i++;
54. }
55. **else**
56. {
57. input[k] = rightArray[j];
58. j++;
59. }
60. }
61. }
63. **static** **void** MergeSort(**int**[] input, **int** left, **int** right)
64. {
65. count++;
67. **if** (left < right)
68. {
69. **int** middle = (left + right) / 2;
71. MergeSort(input, left, middle);
72. MergeSort(input, middle + 1, right);
74. Merge(input, left, middle, right);
75. }
76. }
77. }
78. }

**-- QUICK SORT—**

1. **namespace** QuickSort
2. {
3. **class** Program
4. {
6. **private** **static** **int** count = 0;
8. **static** **void** Main(**string**[] args)
9. {
10. var rand = **new** Random();
12. // int[] array = { 72, 12, 6, 33, 81, 97, 37, 59, 52, 1, 20 };
14. **int**[] testArray = **new** **int**[10000];
15. **for**(**int** i = 0; i < testArray.Length; i++)
16. {
17. testArray[i] = rand.Next(1, 10);
18. }
20. **int** testLength = testArray.Length;
22. // int length = array.Length;

25. Console.WriteLine("QuickSort");
26. **for**(**int** i = 0; i < testArray.Length; i++)
27. {
28. Console.Write(testArray[i] + " ");
29. }
31. Sort(testArray, 0, testLength - 1);
32. Console.WriteLine("Vetor Ordenado");
33. printArray(testArray, testArray.Length);
35. Console.ReadKey();
36. }


40. **static** **void** printArray(**int**[] arr, **int** n)
41. {
42. **for** (**int** i = 0; i < n; ++i)
43. Console.Write(arr[i] + " ");
45. Console.WriteLine();
46. }



51. **static** **int** Partition(**int**[] array, **int** low,
52. **int** high)
53. {
54. //1. Select a pivot point.
55. **int** pivot = array[high];
57. **int** lowIndex = (low - 1);
59. //2. Reorder the collection.
60. **for** (**int** j = low; j < high; j++)
61. {
62. **if** (array[j] <= pivot)
63. {
64. lowIndex++;
66. **int** temp = array[lowIndex];
67. array[lowIndex] = array[j];
68. array[j] = temp;
69. }
70. }
72. **int** temp1 = array[lowIndex + 1];
73. array[lowIndex + 1] = array[high];
74. array[high] = temp1;
76. **return** lowIndex + 1;
77. }
79. **static** **void** Sort(**int**[] array, **int** low, **int** high)
80. {

83. count++;
85. Console.WriteLine("Contador = " + count.ToString());
87. **if** (low < high)
88. {
89. **int** partitionIndex = Partition(array, low, high);
91. //3. Recursively continue sorting the array
92. Sort(array, low, partitionIndex - 1);
93. Sort(array, partitionIndex + 1, high);
94. }
95. }

98. }
99. }

**-- OUTROS ALGORITMOS—**

1. **namespace** EciliaGrupo
2. {
3. **class** Program
4. {

7. **static** **void** Main(**string**[] args)
8. {
9. var rand = **new** Random();
11. Console.WriteLine("Hello World!");
13. BuscaLinear(**new** **int**[10], 10, 5,5);


17. }

20. **static** **int** BuscaLinear(**int**[] V, **int** N, **int** elem, **int** posicao)
21. {
22. **for**(**int** j = 0; j < V.Length; j++)
23. {
24. V[j] = 10;
25. }
27. V[posicao] = elem;
29. **int** i;
30. **for**(i = 0; i < N; i++)
31. {
32. **if** (elem == V[i])
33. {
34. Console.WriteLine("O elemento encontrado é = " + elem + " Na posição " + posicao + " do Vetor");
35. **return** i;
36. }
37. }
39. **return** -1;
40. }
42. **static** **int** Problema04(**int** tamanho)
43. {
44. **int** localK = 0;
46. **int**[,] vetor1 = **new** **int**[tamanho,tamanho];
47. **int**[,] vetor2 = **new** **int**[tamanho, tamanho];
48. **int**[,] vetor3 = **new** **int**[tamanho, tamanho];
49. **for** (**int** i = 0; i < tamanho; i ++)
50. {
51. **for**(**int** j = 0; j < tamanho; j++)
52. {
53. vetor1[i, j] = 0;
54. vetor2[i, j] = 0;
55. vetor3[i, j] = 0;
56. **while** (localK <  tamanho)
57. {
58. vetor1[i, j] = vetor1[i, j] + (vetor2[i \* localK, localK \* j]);
59. localK++;
60. }
62. }
63. }
65. **return** 0;
66. }






74. }
76. }