Centro Universitário da FEI

Programa de Mestrado e Doutorado em Engenharia Elétrica

PEL 201 – Algoritmos Computacionais

**Relatório 2**

**Algoritmos de ordenação: BubbleSort e QuickSort**

Claudio Aparecido Borges Junior

Matrícula: 120122-7

**BUBBLESORT**

O BubbleSort é um algoritmo de ordenação *in-place*, isto é, sua complexidade de espaço é constante . O algoritmo percorre o vetor *(n – 1)* vezes e a cada passagem o elemento de maior valor é levado a última posição do vetor através de sucessivas trocas. A complexidade do pior caso é quadrática em relação a quantidade de elementos, isto é,. O melhor caso ocorre quando o vetor já está ordenado, neste caso o algoritmo é linear (

Abaixo o código do algoritmo escrito em Python limitada superiormente e inferiormente por :

1 def bubblesort(A):

2 for i in xrange(len(A) - 1, 0, -1):

3 for j in xrange(i - 1, -1, -1):

4 if (A[j] > A[i]):

5 A[i], A[j] = A[j], A[i]

6 return A

**QUICKSORT**

O QuickSort é um algoritmo de ordenação com estratégia de divisão e conquista. Sua etapa de divisão consiste em particionar um vetor em dos outros vetores nulos ou não, um contendo somente valores menores do que um pivô e outro somente valores maiores; sendo *A[p..r]* o vetor a ser particionado, *A[p, q-1]* contém valores menores do que *A[q]*, que é menor do que qualquer elemento de *A[q+1, r]*, considera-se que *A[q]* o pivô. A etapa de conquista é recursivamente aplicar o algoritmo QuickSort nos dois sub-vetores *A[p, q-1]* e *A[q+1, r]*. Não há nada a ser feito na etapa de combinação já que ambos sub-vetores estão organizados elvando o vetor todo a estar organizado. O particionador é responsável por determinar quem é *A[q].*

Esse algoritmo também ordena *in-place,* e sua complexidade no pior caso é de. O pior caso ocorre quando o particionamento produz um vetor com *(n – 1)* elementos e outros vetor vazio, chamado de particionamento desbalanceado. O melhor caso, que possui complexidade de , ocorre quando o particionamento produz vetores de tamanho *n/2*, chamado de particionamento balanceado*.*

Dado um vetor a ser ordenado, se utilizarmos um pivô aleatório, dificilmente encontraremos um particionamento desbalanceado nos diversos níveis de recursão. O particionamento irá alterar entre o pior caso e o melhor caso. Intuitivamente, o caso médio é igual ao melhor caso, porém com uma constante *c* escondida pela notação big-oh maior.

Abaixo o algoritmo QuickSort em Python:

1 def quicksort(A):

2 return \_quicksort(A, 0, len(A) - 1)

3

4 def \_quicksort(A, p, r):

5 if p < r:

6 med = partition(A, p, r)

7 \_quicksort(A, p, med - 1, fpartition)

8 \_quicksort(A, med + 1, r, fpartition)

9 return A

10

11 def partition(A, p, r):

12 med = p

13 pivot = A[r]

14 for j in xrange(p, r):

15 if A[j] < pivot:

16 A[med], A[j] = A[j], A[med]

17 med = med + 1

18 A[med], A[r] = A[r], A[med]

19 return med

O QuickSort com pivô randomizado pode ser obtido alterando a função chamada na linha 6 pela função apresentada abaixo:

1 def random\_partition(A, p, r):

2 k = randint(p, r)

3 A[k], A[r] = A[r], A[k]

4 return partition(A, p, r)

**COMPARATIVO DE DESEMPENHO DE TEMPO**

Inicialmente foi realizado um comparativo com os dois algoritmos e suas variações. Nessa comparação fica claro que o *Bubblesort* é assintoticamente limitado a e as variações do *Quicksort* não apresentaram o mesmo comportamento. Isso ocorre porque o vetor de ordenação foi criado randomicamente e o pior caso do *Quicksort* sem randomização não foi testado.

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **bubblesort [s]** | **quicksort [s]** | **randomized quicksort [s]** |
| 1 | 9,54E-07 | 4,77E-07 | 9,54E-07 |
| 2 | 1,55E-06 | 2,15E-06 | 5,01E-06 |
| 4 | 2,15E-06 | 2,86E-06 | 5,48E-06 |
| 8 | 5,01E-06 | 5,01E-06 | 1,11E-05 |
| 16 | 1,20E-05 | 1,10E-05 | 2,31E-05 |
| 32 | 3,90E-05 | 2,65E-05 | 5,30E-05 |
| 64 | 2,22E-04 | 7,90E-05 | 1,80E-04 |
| 128 | 5,98E-04 | 1,73E-04 | 3,01E-04 |
| 256 | 2,52E-03 | 3,42E-04 | 6,22E-04 |
| 512 | 8,06E-03 | 5,81E-04 | 9,82E-04 |
| 1024 | 3,15E-02 | 1,28E-03 | 2,03E-03 |
| 2048 | 1,23E-01 | 2,68E-03 | 4,25E-03 |
| 4096 | 4,87E-01 | 5,73E-03 | 8,89E-03 |
| 8192 | 1,99E+00 | 1,22E-02 | 1,85E-02 |
| 16384 | 8,07E+00 | 2,61E-02 | 4,08E-02 |

Um outro comparativo foi realizado, porem tendendo para o pior caso para o *Quicksort.* Foi utilizado um vetor reversamente ordenado.

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **bubblesort [s]** | **quicksort [s]** | **randomized quicksort [s]** |
| 1 | 9,54E-07 | 9,54E-07 | 9,54E-07 |
| 2 | 1,91E-06 | 2,15E-06 | 4,41E-06 |
| 4 | 3,46E-06 | 5,01E-06 | 1,00E-05 |
| 8 | 5,48E-06 | 7,15E-06 | 1,16E-05 |
| 16 | 1,69E-05 | 1,90E-05 | 2,38E-05 |
| 32 | 5,75E-05 | 5,56E-05 | 5,51E-05 |
| 64 | 2,94E-04 | 2,44E-04 | 1,20E-04 |
| 128 | 9,02E-04 | 7,72E-04 | 2,80E-04 |
| 256 | 3,37E-03 | 2,38E-03 | 4,53E-04 |
| 512 | 1,32E-02 | 9,41E-03 | 9,96E-04 |
| 1024 | 5,18E-02 | 3,64E-02 | 1,93E-03 |
| 2048 | 1,99E-01 | 1,45E-01 | 3,89E-03 |
| 4096 | 8,19E-01 | 6,08E-01 | 8,71E-03 |
| 8192 | 3,32E+00 | 2,33E+00 | 1,79E-02 |
| 16384 | 1,34E+01 | 9,50E+00 | 3,82E-02 |

Um novo comparativo entre ambas as variações do *QuickSort* foi realizado. Nesse caso ambos limitaram-se a já que o vetor a ser ordenado foi criado randomicamente. É interessante notar que a constante *c* da variação com randomização é superior, visto que há uma chamada de função a mais, e sua implementação, para cada vez que um particionamento ocorre.

|  |  |  |
| --- | --- | --- |
| **n** | **quicksort [s]** | **randomized quicksort [s]** |
| 1 | 1,60E-05 | 2,15E-06 |
| 2 | 5,01E-06 | 5,01E-06 |
| 4 | 5,01E-06 | 8,11E-06 |
| 8 | 9,06E-06 | 1,69E-05 |
| 16 | 2,10E-05 | 7,10E-05 |
| 32 | 4,91E-05 | 1,42E-04 |
| 64 | 1,32E-04 | 3,59E-04 |
| 128 | 4,02E-04 | 7,54E-04 |
| 256 | 7,19E-04 | 1,60E-03 |
| 512 | 8,41E-04 | 1,70E-03 |
| 1024 | 4,06E-03 | 3,95E-03 |
| 2048 | 5,76E-03 | 6,23E-03 |
| 4096 | 8,31E-03 | 1,72E-02 |
| 8192 | 1,81E-02 | 2,28E-02 |
| 16384 | 4,66E-02 | 4,14E-02 |
| 32768 | 5,79E-02 | 1,07E-01 |
| 65536 | 1,35E-01 | 1,82E-01 |
| 131072 | 2,68E-01 | 3,61E-01 |
| 262144 | 5,63E-01 | 7,38E-01 |
| 524288 | 1,30E+00 | 1,62E+00 |
| 1048576 | 2,77E+00 | 3,72E+00 |
| 2097152 | 6,43E+00 | 1,03E+01 |
| 4194304 | 1,63E+01 | 2,13E+01 |
| 8388608 | 3,82E+01 | 4,33E+01 |