Faculdade de Informática e Administração Paulista

Algoritmos de Ordenação

Carolina Goudromihos Puig- rm95078
Guilherme Costa- rm93642

São Paulo
2022

Configurações da Máquina

PROCESSADOR: 1,8 GHz Intel Core i5 Dual-Core
RAM: 8GB
HD: 128 GB
Placa: Intel HD Graphics 6000 1536 MB

Bubble Sort

Descrição do Algoritmo:
Compara-se a primeira posição do vetor com a segunda, na segunda iteração (repetição), compara-se a segunda posição do vetor com a terceira, e assim sucessivamente.

Código do algoritmo:

```
def bubbleSort(lista):
   for i in range(len(lista)-1,0,-1):
       for j in range(i):
           if lista[j]>lista[j+1]:
               temp = lista[j]
               lista[j] = lista[j+1]
               lista[j+1] = temp
```

Tamanho da Lista:

Lista 1: 1.000 elementos
Lista 2: 5.000 elementos
Lista 3: 10.000 elementos
Lista 4: 20.000 elementos
Lista 5: 50.000 elementos

Tabela com os tempos obtidos:

Rep	Lista 1	Lista 2	Lista 3	Lista 4	Lista 5
1	0.122	3.119	12.127	52.131	386.187
2	0.074	1.759	7.212	31.038	285.898
3	0.072	1.735	7.215	31.585	265.250
4	0.071	1.744	8.570	34.960	259.917
5	0.071	2.075	9.215	31.367	275.990
6	0.070	2.291	8.005	31.222	265.187
7	0.071	1.911	8.063	32.925	267.133
8	0.080	1.876	7.629	30.959	213.532
9	0.097	2.390	7.698	30.470	207.091
10	0.092	3.383	8.317	30.826	208.383
11	0.085	1.809	7.716	33.130	1155.430
12	0.096	1.730	8.166	31.349	209.608

13	0.120	1.880	8.780	31.442	207.096
14	0.102	2.407	7.916	32.208	1844.029
15	0.096	2.246	7.632	31.043	241.517
16	0.084	3.224	7.715	31.288	244.602
17	0.085	1.832	7.671	30.938	221.784
18	0.088	1.770	7.975	29.846	218.328
19	0.089	1.782	9.339	29.927	210.327
20	0.092	1.850	9.028	29.932	212.393

Média de tempo obtido:

	Lista 1	Lista 2	Lista 3	Lista 4	Lista 5
Média(seg)	0.088	2.141	8.300	32.429	369.984

Análise:

O Bubble Sort apesar de ser o menor algoritmo (mais rápido de se programar), quando utilizado para ordenar listas de grande porte, apresenta uma baixa eficácia pelo longo tempo necessário para executar a ordenação.

Selection Sort

Descrição do Algoritmo:
Este algoritmo é baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o segundo menor valor para a segunda posição e assim sucessivamente, até os últimos dois elementos.

Código do algoritmo:

```
def selectionSort(lista):
   for i in range (len(lista)):
       min_index = i
       for j in range(i+1, len(lista)):
           if lista[j] < lista[min_index]:
               min_index = j
       lista[i], lista[min_index] = lista[min_index], lista[i]
```

Tamanho da Lista:

Lista 1: 1.000 elementos
Lista 2: 5.000 elementos
Lista 3: 10.000 elementos
Lista 4: 20.000 elementos
Lista 5: 50.000 elementos

Tabela com os tempos obtidos:

Rep	Lista 1	Lista 2	Lista 3	Lista 4	Lista 5
1	0.058	1.326	6.032	23.197	153.048
2	0.059	1.315	6.225	26.692	178.271
3	0.057	1.318	6.158	29.730	174.279
4	0.056	1.323	6.432	24.159	170.681
5	0.055	1.429	5.755	29.846	169.401
6	0.055	2.103	5.613	23.328	175.149
7	0.055	2.982	5.642	24.437	163.067
8	0.055	1.371	6.885	22.885	160.879
9	0.058	1.386	6.341	24.441	164.442
10	0.61	1.432	7.127	29.841	162.874
11	0.056	1.959	5.656	23.470	165.487

12	0.054	1.551	5.604	23.131	159.987
13	0.055	1.433	5.725	22.989	161.772
14	0.055	1.824	5.582	23.137	163.575
15	0.055	1.793	5.628	22.893	161.391
16	0.055	1.555	5.570	23.112	164.700
17	0.055	1.554	5.663	23.241	171.353
18	0.056	1.430	5.879	23.761	159.867
19	0.063	1.552	6.060	24.726	158.536
20	0.054	1.394	6.119	24.653	162.464

Média de tempo obtido:

	Lista 1	Lista 2	Lista 3	Lista 4	Lista 5
Média(seg)	0.056	1.602	5.985	24.683	165.061

Análise:

O Selection Sort pode ser considerado um algoritmo mediano de ordenação, sendo melhor que o Bubble Sort e mais demorado que o resto por passar sempre o menor valor do vetor para a primeira posição.

Insertion Sort

Descrição do Algoritmo:
O Insertion Sort inicia a ordenação dividindo o vetor em duas partições: uma ordenada à esquerda e outra não ordenada à direita. A inserção dos elementos na partição ordenada é realizada em duas etapas. Na primeira etapa, procura-se a posição de inserção. Na segunda etapa, desloca-se os elementos da esquerda para a direita para que a posição de inserção fique livre para que o elemento seja inserido.

Código do algoritmo:

```
def insertionSort(lista):
   for i in range(1, len(lista)):
       key = lista[i]
       j = i-1
       while j >= 0 and key < lista[j] :
               lista[j + 1] = lista[j]
               j -= 1
       lista[j + 1] = key
```

Tamanho da Lista:
Lista 1: 1.000 elementos
Lista 2: 5.000 elementos
Lista 3: 10.000 elementos
Lista 4: 20.000 elementos
Lista 5: 50.000 elementos

Tabela com os tempos obtidos:

Rep	Lista 1	Lista 2	Lista 3	Lista 4	Lista 5
1	0.025	0.728	3.245	11.103	78.808
2	0.026	0.726	2.820	11.479	75.371
3	0.026	0.858	3.112	11.434	76.972
4	0.024	0.727	3.265	11.124	75.479
5	0.024	0.801	2,669	12.000	74.448
6	0.027	0.789	2.821	11.996	75.446
7	0.026	0.887	3.048	12.400	76.588
8	0.028	0.699	2.868	11.579	73.612
9	0.028	0.681	3.241	12.151	76.105
10	0.024	0.696	3.351	12.120	71.363

11	0.028	0.646	3.071	12.120	70.784
12	0.024	0.652	3.169	12.784	77.310
13	0.028	0.682	2.942	12.385	72.389
14	0.029	0.666	2.744	12.137	77.894
15	0.028	0.689	2.977	11.114	75.692
16	0.025	0.664	3.159	11.218	76.401
17	0.027	0.654	2.874	13.468	74.329
18	0.031	0.671	2.606	12.023	68.909
19	0.027	0.676	2.679	11.666	109.166
20	0.030	0.694	2.702	11.448	100.976

Média de tempo obtido:

	Lista 1	Lista 2	Lista 3	Lista 4	Lista 5
Média(seg)	0.027	0.714	2.968	11.826	77.902

Análise:

Outro algoritmo mediano-bom é o Insertion Sort, sendo melhor que o Selection, ele tem o poder de ordenar muito mais rápido uma lista grande, sendo seu tempo de ordenação menor do que a metade do tempo gasto em algoritmos anteriores.

Merge Sort

Descrição do Algoritmo:
A ideia do Merge Sort é dividir o vetor em dois subvetores, cada um com metade dos elementos do vetor original. Esse procedimento é reaplicado aos dois subvetores recursivamente. Quando os subvetores têm apenas um elemento (caso base), a recursão para. Então, os subvetores ordenados são fundidos (ou intercalados) num único vetor ordenado.

Código do algoritmo:

```
def mergeSort(lista):
   if len(lista) > 1:
       meio = len(lista)//2
       L = lista[:meio]
       R = lista[meio:]
       mergeSort(L)
       mergeSort(R)
       i=j=k=0
       while i<len(L) and j<len(R):
           if L[i] <= R[j]:
               lista[k] = L[i]
               i+=1
           else:
               lista[k] = R[j]
               j+=1
           k+=1
       while i<len(L):
           lista[k]=L[i]
           i+=1
           k+=1
       while j < len(R):
           lista[k] = R[j]
           j+=1
           k+=1
```

Tamanho da Lista:
Lista 1: 1.000 elementos
Lista 2: 5.000 elementos
Lista 3: 10.000 elementos
Lista 4: 20.000 elementos
Lista 5: 50.000 elementos

Tabela com os tempos obtidos:

Rep	Lista 1	Lista 2	Lista 3	Lista 4	Lista 5

1	0.005	0.045	0.074	0.123	4.049
2	0.004	0.026	0.057	0.104	3.990
3	0.006	0.031	0.057	0.103	4.923
4	0.006	0.031	0.055	0.111	5.210
5	0.007	0.029	0.060	0.107	5.731
6	0.005	0.026	0.059	0.103	4.921
7	0.006	0.031	0.058	0.105	4.102
8	0.007	0.031	0.055	0.104	4.140
9	0.005	0.030	0.055	0.104	4.015
10	0.007	0.026	0.056	0.103	3.866
11	0.006	0.031	0.056	0.103	3.964
12	0.007	0.032	0.055	0.105	4.009
13	0.009	0.030	0.053	0.107	4.008
14	0.005	0.030	0.054	0.110	4.055
15	0.005	0.029	0.053	0.111	3.869
16	0.015	0.027	0.052	0.108	3.907
17	0.010	0.031	0.054	0.109	3.808
18	0.005	0.030	0.054	0.114	3.834
19	0.004	0.029	0.054	0.114	4.213
20	0.10	0.027	0.053	0.107	3.752

Média de tempo obtido:

	Lista 1	Lista 2	Lista 3	Lista 4	Lista 5
Média(seg)	0.007	0.030	0.056	0.108	4.218

Análise:

O merge é um ótimo algoritmo, reduzindo o tempo exponencialmente em relação ao último algoritmo visto, uma boa opção para listas grandes.

Quick Sort

Descrição do Algoritmo:
O algoritmo baseia a ordenação em sucessivas execuções de particionamento, uma rotina que escolhe um pivot e o posiciona no array de uma maneira em que os elementos menores ou iguais ao pivot estão à sua esquerda e os maiores estão à sua direita.

Código do algoritmo:

```
def partition(lista, comeco, fim):
   pivot = lista[comeco]
   low = comeco + 1
   high = fim
   while True:
       while low <= high and lista[high] >= pivot:
           high = high - 1
       while low <= high and lista[low] <= pivot:
           low = low + 1
       if low <= high:
           lista[low], lista[high] = lista[high], lista[low]
       else:
           break
   lista[comeco], lista[high] = lista[high], lista[comeco]
   return high

def quickSort(lista, comeco, fim):
   if comeco >= fim:
       return
   p = partition(lista, comeco, fim)
   quickSort(lista, comeco, p-1)
   quickSort(lista, p+1, fim)
```

Tamanho da Lista:
Lista 1: 1.000 elementos
Lista 2: 5.000 elementos
Lista 3: 10.000 elementos
Lista 4: 20.000 elementos
Lista 5: 50.000 elementos

Tabela com os tempos obtidos:

Rep	Lista 1	Lista 2	Lista 3	Lista 4	Lista 5
1	0.004	0.024	0.045	0.075	0.226
2	0.003	0.022	0.041	0.076	0.198
3	0.004	0.020	0.038	0.077	0.199
4	0.005	0.024	0.038	0.075	0.210
5	0.003	0.024	0.042	0.076	0.262
6	0.004	0.023	0.048	0.075	0.228
7	0.004	0.021	0.040	0.077	0.214
8	0.002	0.020	0.036	0.075	0.216
9	0.003	0.023	0.039	0.076	0.203
10	0.005	0.025	0.038	0.078	0.660
11	0.004	0.019	0.038	0.076	0.313
12	0.003	0.021	0.040	0.078	0.329
13	0.005	0.019	0.037	0.074	0.221
14	0.003	0.024	0.036	0.077	0.267
15	0.002	0.020	0.039	0.076	0.235
16	0.005	0.023	0.038	0.078	0.553
17	0.004	0.021	0.039	0.079	0.232
18	0.003	0.021	0.039	0.080	0.263
19	0.004	0.026	0.038	0.078	0.238
20	0.004	0.025	0.037	0.076	0.251

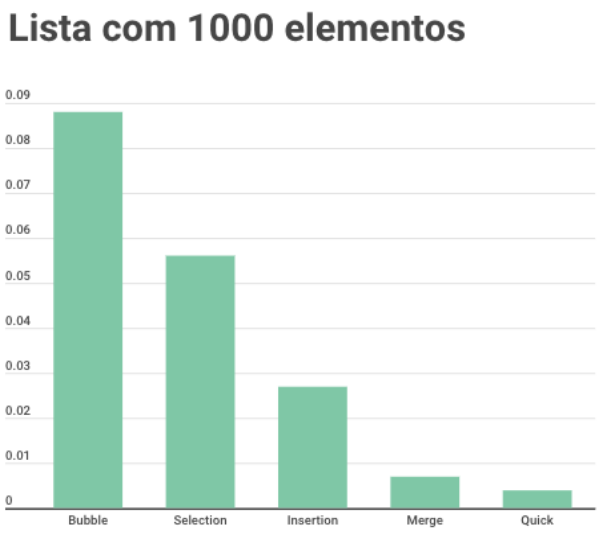
Média de tempo obtido:

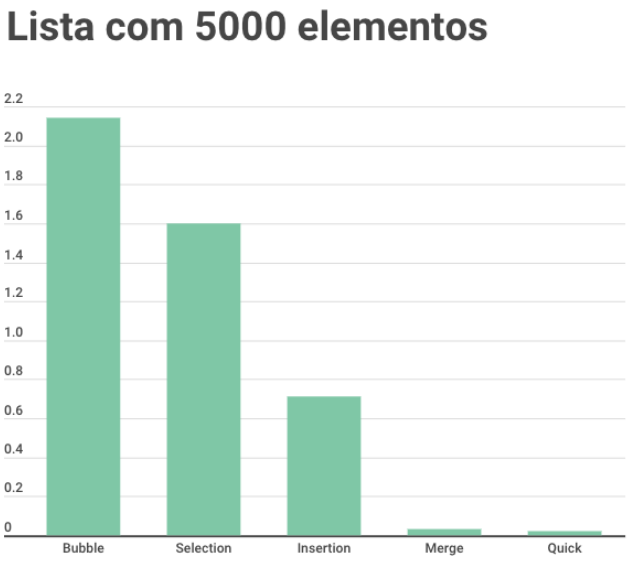
	Lista 1	Lista 2	Lista 3	Lista 4	Lista 5
Média(seg)	0.004	0.022	0.039	0.077	0.276

Análise:

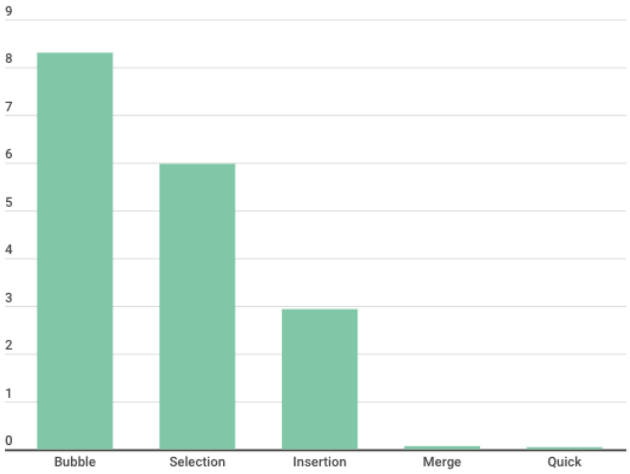
O Quick Sort é sem dúvidas o algoritmo mais rápido de todos os vistos, seu poder é tão grande que listas com milhões de números são organizadas em segundos, na dúvida o quick sort sempre será o mais recomendado, mesmo sendo um algoritmo mais complexo para programar

Gráficos:

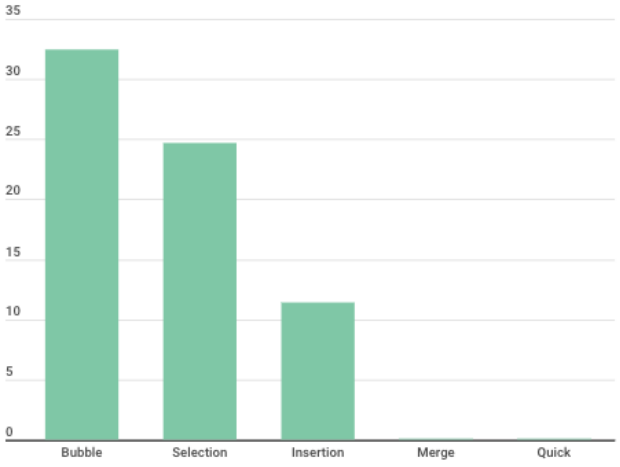


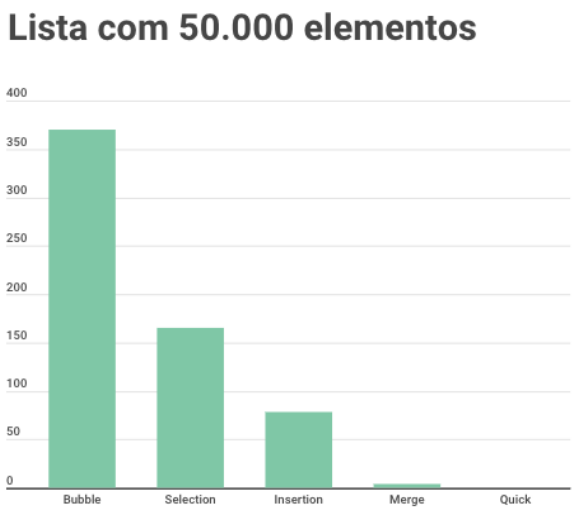


Lista com 10.000 elementos



Lista com 20.000 elementos





Conclusão:

Após rodar os cinco algoritmos diversas vezes com listas de 5 tamanhos diferentes, podemos notar que alguns apresentaram maior eficácia quando falamos em agilidade. O algoritmo que foi considerado mais demorado foi o Bubble Sort, portanto, apesar de ser um algoritmo curto, não é a melhor opção quando precisamos ordenar listas extensas.

Ademais, ao analisar os gráficos,nota-se uma discrepância entre Bubble, Insertion e Selection Sort (algoritmos que tem como princípio rodar o vetor diversas vezes),em relação ao Merge e ao Quick Sort (onde a ordenação é a partir da partição do vetor), ou seja, algoritmos que utilizam esse método de partição, apresentam uma eficácia maior quando é necessário ordenar listas de grande porte em razão de serem mais rápidos.