Nome: Gustavo de Castro Nogueira

Especificações da máquina utilizada

Processador: Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz, 3001 Mhz, 4 Core(s), 4 Logical Processor(s)

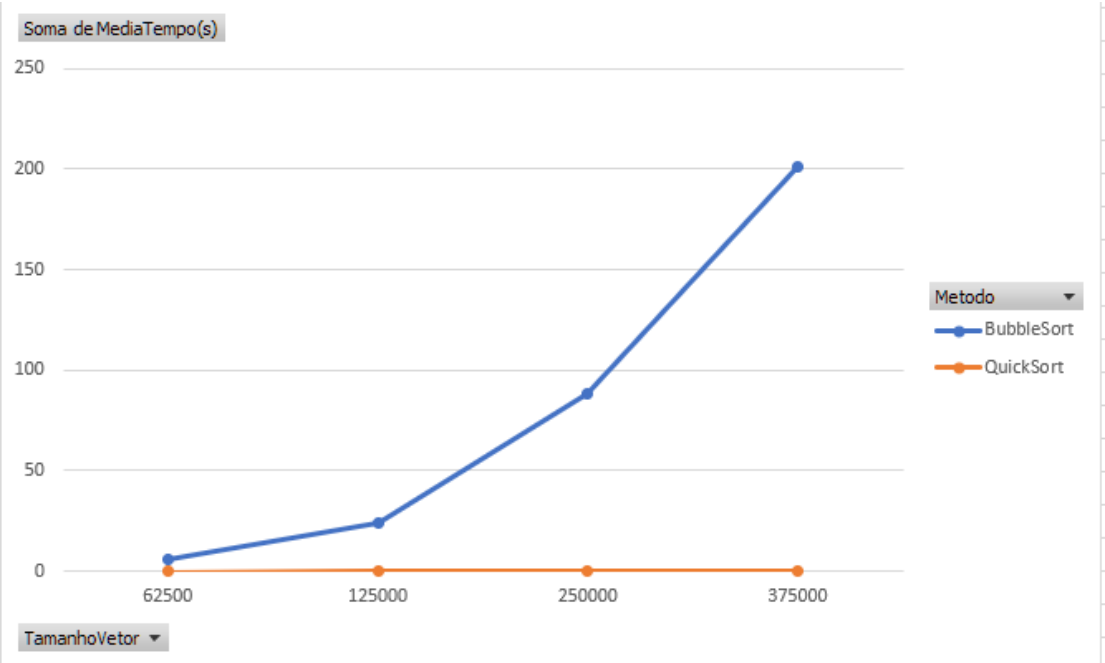
Ram: 8.00 Gb

OS: Microsoft Windows 10 Pro

Parte 1

Resultados obtidos:

Tamanho Vetor	Media Tempo Quicksort(ms)	MediaTempo Bubblesort(ms)
62500	4.889188	5690
125000	11	23900
250000	20.9	88300
375000	36.3	201000



Análise Bubblesort

O caso médio de complexidade do algoritmo BubbleSorte é O(n2) , portanto com o aumento de n (tamanho do vetor) espera-se que o tempo aumente em proporção quadrática. Segue os dados obtidos para o algoritmo de bubble sort:

Tamanho do vetor (n)	MediaTempo Bubblesort(ms)	tempo bubblesort / n
62500	5690.00	0.09104
125000	23900.00	0.1912
250000	88300.00	0.3532
375000	201000.00	0.536

Tamanho do vetor (n)	Proporção aumento de N sobre o menor tamanho	Proporção aumento do tempo medio	Proporção esperada aumento do tempo (n2)
62500	**1**	**0**	**1**
125000	**2**	**4.200351494**	**4**
250000	**4**	**15.51845343**	**16**
375000	**6**	**35.32513181**	**36**

Como esperado, com o aumento do tamanho de N verificamos que o aumento do tempo seguiu bem próximo do caso médio esperado (n2), para todos os tamanhos de vetor testados.

Análise Quicksort

O caso médio de complexidade do algoritmo QuickSort é O(n log n) ,ou seja um aumento seguindo uma proporção logarítmica. É esperado um tempo muito menor e com um menor aumento de tempo em relação ao aumento do tamanho do vetor, se comparado com o bubblesort.

Segue os dados obtidos para o algoritmo QuickSort:

Tamanho do vetor (n)	MediaTempo Quicksort(ms)	Tempo quicksort / n
62500	**5690**	**0.000078227008**
125000	**23900**	**0.000088**
250000	**88300**	**0.0000836**
375000	**201000**	**0.0000968**

Tamanho do vetor(n)	Proporção aumento de n sobre o menor tamanho	Proporção aumento do tempo medio	Proporção esperada aumento do tempo (n log n)

62500	**1**	**1**	**1**
125000	**2**	**2.249862349**	**2**
250000	**4**	**4.274738464**	**8**
375000	**6**	**7.424545753**	**15.509775**

Como esperado, os tempos e as proporções de aumento utilizando o algoritmo Quicksort foram bem menores do que quando utilizando o algoritmo Bubblesort. Inclusive vale ressaltar que para os dois maiores tamanhos de vetor (**250000 e 375000**) o aumento foi aproximadamente a metade do esperado.

Conclusão

Tendo as análises acima e o hardware utilizado em vista, para esse tipo de problema o algoritmo de Quick Sort se mostrou, assim como esperado, mais eficiente que o algoritmo BubbleSort. Podemos ver também que as proporções de aumento tempo/n seguiram muito próximos do esperado teoricamente no algoritmo do bubblesort e para os 2 menores casos do quicksort. Para os casos dos 2 maiores tamanhos de vetor no quicksort o aumento no tempo foi de aproximadamente 50% menor do esperado, o que pode ter sido causado por um menor uso da memória no momento de execução do algoritmo em relação às outras execuções.

Parte 2

media tempo vetor ordenado(ms)
32.997

A execução das ordenações utilizando o quicksort em vetores não ordenados teve uma média de 32.997 milissegundos após as 10 ordenações. Na execução do algoritmo de ordenação QuickSort no vetor de tamanho 375.000 já previamente ordenado, encontramos o erro StackOverflow. Esse erro ocorreu uma vez que o algoritmo de quicksort utilizado divide o vetor em partições utilizando sempre o ultimo elemento como pivot. Assim é gerado um particionamento de vetores de tamanho 0 e n-1 para cada elemento do vetor, o algoritmo tenta criar 375000 partições recursivamente, gerando o erro de StackOverflow.

Uma possível correção é substituir o pivot do particionamento , atualmente fixado como o ultimo elemento, para um valor aleatório de até o tamanho da partição.

Antes:

```java
static public int particao(int[] dados, int inicio, int fim){
    int posicao = inicio-1;
    int pivot = dados[fim];
    for (int i = inicio; i < fim; i++) {
        if(dados[i]<pivot){
            posicao++;
            trocar(dados, posicao, i);
        }
    }
    posicao++;
    trocar(dados, posicao, fim);
    return posicao;
}
```

Depois:

```java
static public int particao(int[] dados, int inicio, int fim){
    int posicao = inicio-1;
    Random r = new Random();
    int pivot = dados[r.nextInt(fim)];
    for (int i = inicio; i < fim; i++) {
        if(dados[i]<pivot){
            posicao++;
            trocar(dados, posicao, i);
        }
    }
    posicao++;
    trocar(dados, posicao, fim);
    return posicao;
}
```