




ATV - ORDENAÇÃO

⚙ Status	Fazendo
➤ Course	 <u>AEDS</u>
📅 Due date	@25 de outubro de 2023
⌵ Type	Teste



Dica do Notion: Utilize esta página para ajudá-lo a se preparar para esta tarefa. Escreva o conteúdo do teste, adicione notas de revisão e compartilhe com amigos, e muito mais! Saiba mais sobre como compartilhar uma página [aqui](#).

Atividade pratica de AEDS

Valor: 30 pontos

Atividade de ordenação que envolva:

☒ Inserção

☒ Seleção

Devera conter:

1. Implementar os algoritmos
2. Analise de complexidade
3. **Simulação:**

- ✓ ~~Bolha~~
- ✓ ~~Merge Sort~~

- Vetores ordenados crescente e decrescente
- Vetores de tamanho 100, 1k, 100k...
- Vetores com tamanhos e ordens aleatórias

A atividade deverá coletar o tempo de início e término de execução para cada rodada e exibir em gráfico. **EXEMPLO DE GRAFICO:** *tempo X tamanho*.



! fixar o seed.



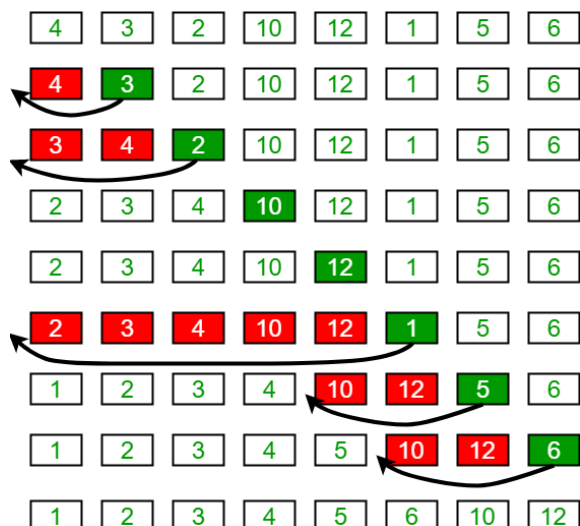
T.C. = Tempo Crescente

T.D. = Tempo Decrescente

Insert Sort:

Tamanho Vetor	T. C. - milissegundos	T.C. - Segundos	T. D. - milissegundos	T. D - Segundos
100	0	0	0	0
1000	2	0,002	2	0,002
10000	11	0,011	11	0,011
100000	709	0,709	702	0,702
500000	18167	18,167	18068	18,068
1000000	76184	76,184	73458	73,458

Insertion Sort Execution Example



Características - Negativas:

- Complexidade em casos médios e piores $O(n^2)$
- Péssimo com valores muito grandes

Insert Sort é um algoritmo simples usado principalmente em ordenações de pequenos valores.

Funcionamento: Consiste em percorrer o vetor da direita para esquerda, começando do segundo elemento e verificar a partir daí se a posição está correta de acordo com a ordenação (crescente ou decrescente)

Características - Boas:

- Ótimo com pequenos valores
- Estável e sem alocar espaço adicional na memória
- Complexidade em casos melhores $O(n)$

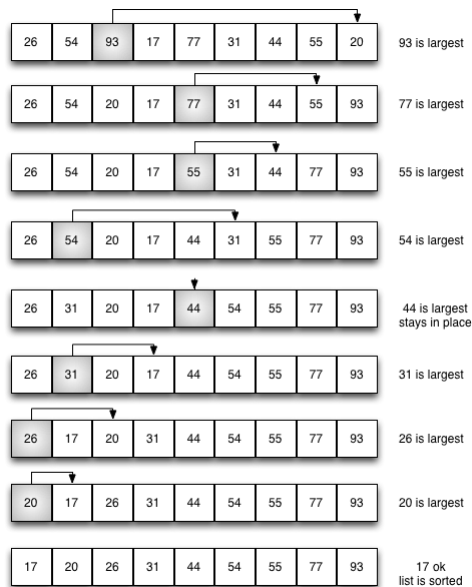
```
for (int i = 1; i < vet.length; i++) {
    int key = vet[i];
    int j = i - 1;

    while (j >= 0 && vet[j] > key) {
        vet[j + 1] = vet[j];
        j = j - 1;
    }
    vet[j + 1] = key;
}
```

Selection Sort:

Tamanho Vetor	T. C. - milissegundos	T.C. - Segundos	T. D. - milissegundos	T. D - Segundos
100	0	0	0	0
1000	2	0,002	2	0,002
10000	38	0,038	41	0,041
100000	3541	3,541	3614	3,614

Tamanho Vetor	T. C. - milissegundos	T.C. - Segundos	T. D. - milissegundos	T. D - Segundos
500000	89640	89,640	87807	87,807
1000000	369010	369,010	352949	352,949



Selection Sort

Funcionamento: Consiste em selecionar o menor elemento do vetor e coloca-lo na primeira posição, selecionar o segundo menor e colocar na segunda posição, continua até terminar a ordenação.

Características - Boas:

- Sem alocar espaço adicional na memória
- Ótimo com pequenas listas

Características - Negativas:

- Instável
- Perda de desempenho em comparação com insert
- Complexidade em melhor, médio e pior caso de $O(n^2)$

```
for (int fixo = 0; fixo < vet.length - 1; fixo++) {
    menor = fixo;
    for (int i = menor + 1; i < vet.length; i++) {
        if (vet[i] < vet[menor]) {
            menor = i;
        }
    }
    if (menor != fixo) {
        int t = vet[fixo];
        vet[fixo] = vet[menor];
        vet[menor] = t;
    }
}
```

Bubble Sort:

Tamanho Vetor	T. C. - milissegundos	T.C. - Segundos	T. D. - milissegundos	T. D - Segundos
100	632	0,632	640	0,640
1000	637	0,637	643	0,643
10000	1653	1,653	1684	1,684
100000	15090	15,090	15184	15,184
500000	349743	349,743	382148	382,148
1000000	1403686	1403,686	1395196	1395,196

Bubble Sort

Funcionamento: Consiste em comparar pares de elementos adjacentes e ordena-los n vezes, tendo como os valores maiores indo para as ultimas posições.

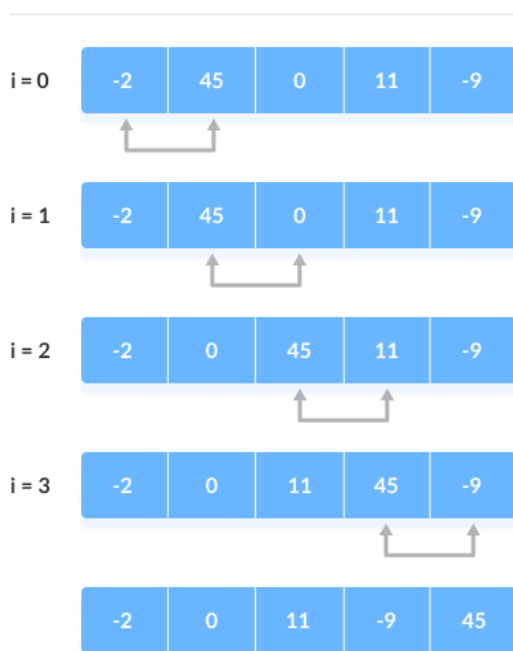
Características - Boas:

- Simplicidade na implementação

Características - Negativas:

- Instável
- Pouco eficiente me listas grandes
- Execução lenta com complexidade de $O(n^2)$ em melhores, médios e piores casos
- Tem um número muito grande de movimentação de elementos

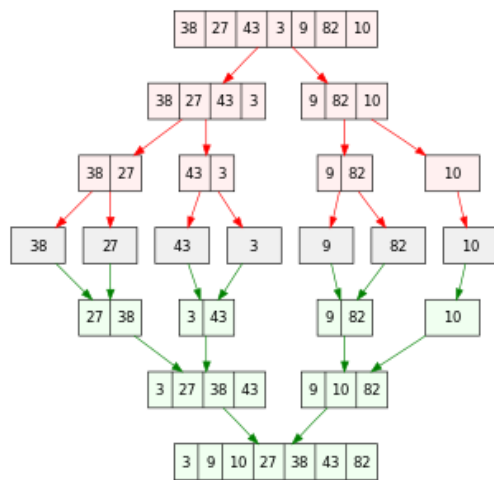
step = 0



```
for (int ord = vet.length - 1; ord < vet.length; ord--) {
    for (int e = 0; e < ord; e++) {
        if (vet[e] > vet[e + 1]) {
            cont = vet[e];
            vet[e] = vet[e + 1];
            vet[e + 1] = cont;
        }
    }
}
```

Merge Sort:

Tamanho Vetor	T. C. - milissegundos	T.C. - Segundos	T. D. - milissegundos	T. D - Segundos
100	0	0	0	0
1000	1	0,001	1	0,001
10000	58	0,058	67	0,067
100000	2477	2,477	2346	2,346
500000	46539	46,539	48461	48,461
1000000	193694	193,694	186084	186,084



Merge Sort - “dividir para conquistar”

Funcionamento: Consiste em dividir o vetor em partes menores de forma recursiva, se repete até a divisão chegar em n pares de elementos. Após isso, os pares são ordenados de acordo com a ordem desejadas. Logo é juntado cada par e ordenado. O processo é repetido até chegar novamente em um único vetor.

Características - Boas:

- Melhor caso $O(n \cdot \log_2 n)$
- Adaptado a memória secundária
- Estável e Eficiente em vetores grandes.

Características - Negativas:

- Uso de memória adicional
- Complexo

```
private static void mergeSort(int[] vet, int inicio, int fim) {
    if (inicio < fim - 1) {
        int meio = (inicio + fim) / 2;
        mergeSort(vet, inicio, meio);
        mergeSort(vet, meio, fim);
        juntar(vet, inicio, meio, fim);
    }
}
```

```

    }

    private static void juntar(int[] vet, int inicio, int meio, int fim) {

        int[] vetAuxiliar = new int[vet.length];
        int inicio2 = inicio, meio2 = meio, pos = 0;

        while (inicio2 < meio && meio2 < fim) {
            if (vet[inicio2] <= vet[meio2]) {
                vetAuxiliar[pos] = vet[inicio2];
                pos = pos + 1;
                inicio2 = inicio2 + 1;

            }else {
                vetAuxiliar[pos] = vet[meio2];
                pos = pos + 1;
                meio2 = meio2 + 1;
            }
        }

        while (inicio2 < meio) {
            vetAuxiliar[pos] = vet[inicio2];
            pos = pos + 1;
            inicio2 = inicio2 + 1;
        }

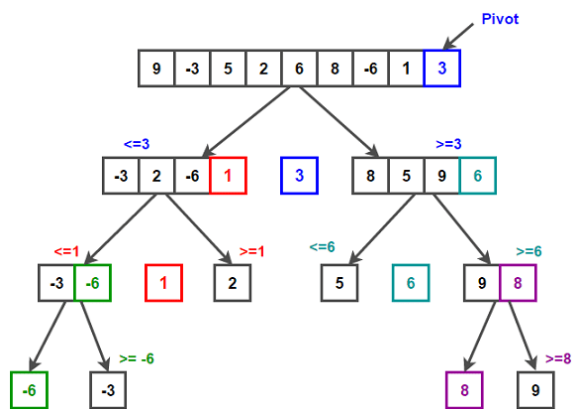
        while (meio2 < fim) {
            vetAuxiliar[pos] = vet[meio2];
            pos = pos + 1;
            meio2 = meio2 + 1;
        }

        for (pos = 0, inicio2 = inicio; inicio2 < fim; inicio2++, pos++) {
            vet[inicio2] = vetAuxiliar[pos];
        }
    }
}

```

Quick Sort

Tamanho Vetor	T. C. - milissegundos	T.C. - Segundos	T. D. - milissegundos	T. D - Segundos
100	0	0	0	0
1000	1	0,001	0	0
10000	2	0,002	1	0,001
100000	10	0,010	13	0,013
500000	26	0,026	27	0,027
1000000	53	0,053	54	0,054



QuickSort - Também se baseia em “divisão e conquista”

Funcionalidade: funciona ordenado qualquer sequência de valores, dividindo-a em subsequências menores, aplicando recursão para ordenar cada subsequência e, finalmente, concatenando-as novamente em um vetor.

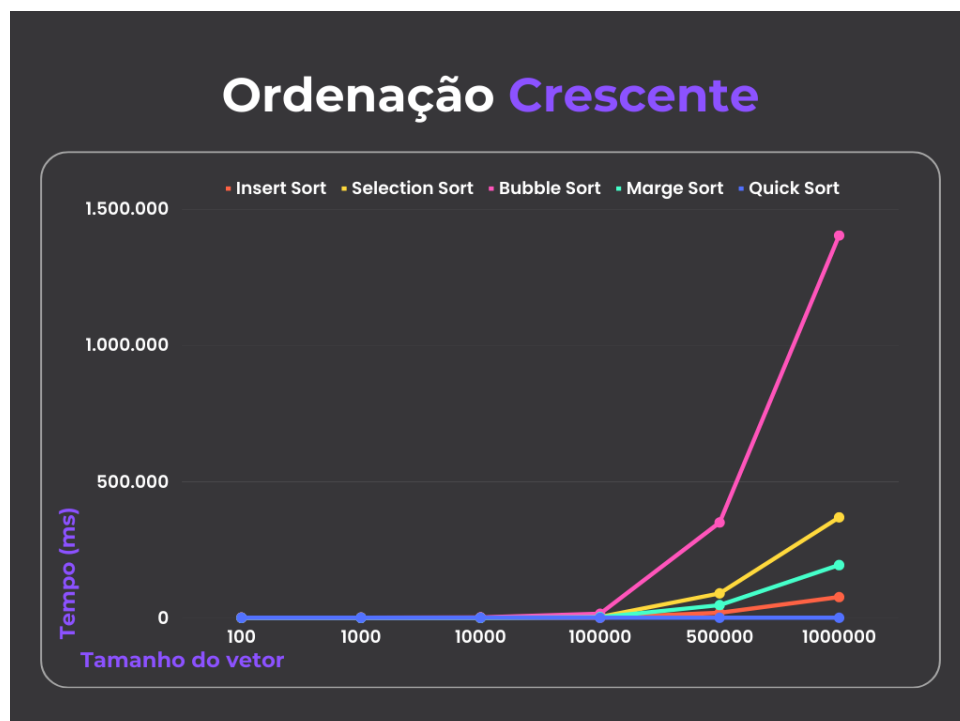
Características - Boas:

- Bastante Eficiente em grandes e pequenos vetores
- Não requer espaço de memória

Características - Negativas:

- Dependência da Escolha do Pivô
- Não é Estável: elementos iguais pode gerar esta instabilidade

Gráficos:



Ordenação Decrescente

