Relatório

Nessa tabela irei mostrar o resultado de tempo de execução dos algoritmos de ordenação(Bubble, Selection, Merge e Quick).

Com os valores de 100, 1000, 10000, 100000, 500000 e 1 milhão.

Não sei porque mas quando coloco valor de maior ou igual a 500000 ele não compila.

Mas com o resultado que eu fiz no meus codigos, o quick sort é o mais efieciente entre os outros.

E o bubble é o menor eficiente de todos eles.

Valor	Bubble_Sort	Selection_Sort	Merge_Sort	Quick_Sort
100	0.000000(s)	0.000000(s)	0.000000(s)	0.000000(s)
1000	0.000000(s)	0.000000(s)	0.000000(s)	0.000000(s)
10000	0.175000(s)	0.086000(s)	0.002000(s)	0.000000(s)
100000	27.533001(s)	8.222000(s)	0.023000(s)	0.011000(s)
500000	716.258972(s)	203.104004(s)	0.111000(s)	Error
1000000	2881.845947(s)	814.495972(s)	0.226000(s)	Error

Bubble Sort: é um algoritmo de ordenação mais simples onde percorre o vetor varias vezes, e cada passagem fazer com que o numero maior fique no final da sequencia. Exemplo:

inicial : [3] [0] [7] [1] [5] [2] [4] [6] comparar o 3 com 0 para ver quem e o maior assim:

1: [0] [3] [7] [1] [5] [2] [4] [6] comparar se o 3 e maior que 7, como é menor continua assim:
2: [0] [3] [7] [1] [5] [2] [4] [6] agora comparar o 7 com 1, assim:
3: [0] [3] [1] [7] [5] [2] [4] [6] comparar 7 com 5:
4: [0] [3] [1] [5] [7] [2] [4] [6] comparar 7 com 2:
5: [0] [3] [1] [5] [2] [7] [4] [6] comparar 7 com 4:
6: [0] [3] [1] [5] [2] [4] [7] [6] comparar 7 com 6:
7: [0] [3] [1] [5] [2] [4] [6] [7] e repetir tudo de novo com 0 comparando com 3 ate ficar ordenado.

Selection Sort: é um algoritmo de ordenação baseado em se passar sempre o menor valor do vetor para primeira posição, depois o de segundo menor valor para a segunda posição e assim sucessivamente(n – 1).

inicio: [3] [0] [7] [1] [5] [2] [4] [6] compar o menor valor com a primeira posicao que 3:

1: [0] [3] [7] [1] [5] [2] [4] [6] pegar o segundo menor valor que e 1 e trocar com 3:
2: [0] [1] [7] [3] [5] [2] [4] [6] pegar o terceiro menor valor que e 2 e trocar com 7:
3: [0] [1] [2] [3] [5] [7] [4] [6] pegar o quarto menor valor que e 3 e trocar com próprio 3:
4: [0] [1] [2] [3] [5] [7] [4] [6] pegar o quinto menor valor que e 4 e trocar com 5:
5: [0] [1] [2] [3] [4] [7] [5] [6] pegar o sexto menor valor que e 5 e trocar com 7:
6: [0] [1] [2] [3] [4] [5] [7] [6] pegar o setimo menor valor que e 6 e trocar com 7:
7: [0] [1] [2] [3] [4] [5] [6] [7] ate ficar ordenado dessa forma.

Merge Sort: é um algoritmo de ordenação por mistura, do tipo dividir para conquistar. Exemplo:

- dividir/conquista

DIVIDIR

left m+1 rigth
(a)inicio[3 0 7 1] [5 2 4 6]fim
(b)inicio[3 0][7 1]fim inicio[5 2][4 6]fim

CONQUISTA

(c)(pegar o (b) e comparar quem e maior e colocar o menor sempre na esquerda)
[0 3] [1 7] [2 5] [4 6]
(d)(pegar o (c) e comparar quem e menor)[0 1 3 7] [2 4 5 6]
(e)(ordernar o (d))[0 1 2 3 4 5 6 7]

Quick Sort: é um algoritmo eficiente de ordenação. Onde há um pivo e o posiciona no array de uma maneira em que os elementos menores ou igual a do pivo fique a sua esquerda e o maiores ou igual na sua direita. Exemplo:

[3] [0] [7] [1] [5] [2] [4] [6]

pivo principal é o **[3]**, assim:

(a)[0] [1] [2] **[3]** (b) [7] [5] [4] [6]

no (a) o pivo é o **[0]**, mas como já esta ordenado não precisa mudar nada.

Já no (b) o pivo é o **[7]**, assim:

(c) [5] [4] [6] **[7]**

no (c) o pivo é o **[5]**, assim:

[4] **[5]** [6] [7]

com isso juntando o (a) e o (c) fica:

[0] [1] [2] **[3]** [4] [5] [6] [7] de forma ordenada, onde o pivo principal que e o [3], os menores valores que o 3 fica do lado esquerdo e maior que 3 fica na direita.