Biblioteca Digital de Arendelle

Ciência da Computação

Eduardo de Paiva Dias - 2018126657

- ## Introdução

A ordenação consiste em rearranjar os elementos de modo que se possa facilitar a análise desses dados. Temos diversos métodos de ordenação, contudo neste trabalho iremos abordar o quicksort e variações do mesmo, como:

1. Quicksort Clássico: A seleção do pivô é o elemento central.
2. Quicksort Mediana de três: A implementação do quicksort mediana de três é semelhante com o do clássico, exceto pela seleção do pivô que é feita uma mediana entre o primeiro o último elemento e o elemento do meio.
3. Quicksort Primeiro elemento: Implementação parecida com o do quicksort clássico, exceto pelo pivô que é sempre o primeiro elemento do vetor e de seus subvetores.
4. Quicksort inserção 1%: A seleção do pivô nesta variação é o mesmo do quicksort mediana de três, contudo quando um subvetor está com 1% do tamanho original é usado o método de ordenação por inserção.
5. Quicksort inserção 5%: Semelhante ao quicksort 1%, contudo o subvetor está com 5% do tamanho original.
6. Quicksort inserção 10%: Semelhante ao quicksort 1%, contudo o subvetor está com 10% do tamanho original.
7. Quicksort não recursivo: A seleção do pivô é igual ao quicksort clássico, contudo usa o sistema de pilhas para simular as chamadas recursivas.

- ## Implementação

Para a implementação variação dos quicksort, foi usado uma estrutura para cada um deles para facilitar o uso e correção de erros, apesar de algumas funções tenham a mesma funcionalidade.

QuickSortClassico(), QuickSortMediana(), QuickSortPrimeiroElemento(), QuickSortNaoRec(), QuickSortInsercao1(), QuickSortInsercao5(), QuickSortInsercao10(): Inicializa os quicksort.

OrdenaClassico(), OrdenaMediana(), OrdenaPrimeiroElemento(), OrdenaInsercao1(), OrdenaInsercao5(), OrdenaInsercao10(): Contém a partição e as chamadas recursivas do quicksort.

ParticaoInsercao10(), ParticaoInsercao5(), ParticaoInsercao1(), ParticaoNaoRec(), ParticaoPrimeiroElemento(), ParticaoMediana(), ParticaoClassico(): Particiona o quicksort e obtém o pivô.

obterMediana(), obterMedianaInsercao1(), obterMedianaInsercao5(), obterMedianaInsercao10(): Obtém a mediana para selecionar o pivô nos casos da mediana de três, criando um vetor e ordenado para pegar o elemento do meio.

InsercaoMediana(), InsercaoMediana1(), InsercaoMediana5(), InsercaoMediana10(): Para a seleção do pivô, foi usado a inserção para ordenar o pequeno vetor.

Insercao1(), Insercao5(), Insercao10(): Ordena o vetor com inserção quando o subvetor está igual a %1 ou %5 ou %10.

TipoItem, TipoApontador, TipoCelula, TipoPilha, FPVazia(), Vazia(), Empilha(), Desempilha(): Estruturas e funções usadas para a ordenação não recursiva do quicksort com pivô central.

preencher_arquivo(): Preenche o arquivo com a saída sem o vetor.

preencher_arquivo_vetor(): Preenche o arquivo com a saída do vetor.

Para a execução do programa temos possíveis entradas:

<variação> <tipo> <tamanho>
<variação> <tipo> <tamanho> -p
<variação> <tipo> <tamanho> <nomedoarquivo.txt>
<variação> <tipo> <tamanho> <nomedoarquivo.txt> -´p
Sendo variação:

- QC para quicksort clássico
- QM3 para quicksort mediana de três
- QPE para quicksort primeiro elemento

QI1P para quicksort inserção 1%

QI5P para quicksort inserção 5%

QI10P para quicksort inserção 10%

QNR para quicksort não recursivo

tipo:

OrdC para quicksort crescente

OrdD para quicksort decrescente

Ale para quicksort Aleatório

Saida:

<variação> <tipo> <tamanho> <n_comparação> <n_movimentação> <tempo execução>

- **Análise Experimental**

Para fazer a análise experimental, foi pego tamanho de 5 vetores diferentes de 50mil até 250mil(não foi usado de 250mil até 500mil, pois as variáveis em muitos casos estavam acontecendo overflow), analisando a comparação e movimentação com o tempo com os diferentes quicksort. As especificações básicas do computador de teste foram, 16 GB de memória ram e processador i7 e sistema operacional ubuntu.

Legenda:

QC: Quicksort clássico.
QM3: Quicksort mediana de três.
QPE: Quicksort primeiro elemento.
QI1P: Quicksort inserção 1%.
QI5P: Quicksort inserção 5%.
QI10P: Quicksort inserção 10%
QNR: Quicksort não recursivo.

Vetor Ordenado em ordem crescente

A figura 1 apresenta a quantidade de comparações e movimentos para o vetor ordenado em ordem crescente, enquanto a figura 1.1 apresenta os resultados em função do tempo.

VETOR CRESCENTE										
	50000		100000		150000		200000		250000	
	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.
QC	750015	32767	1600016	65535	2456803	84464	3400017	131071	4250017	131071
QM3	750015	32767	1600016	65535	2456803	84464	3400017	131071	4250017	131071
QPE	1250074998	49999	705182702	99999	-	-	-	-	-	-
QI1P	399752	127	799752	127	1199752	127	1599752	127	1999752	127
QI5P	299942	31	599942	31	899942	31	1199942	31	1499942	31
QI10P	249973	15	499973	15	749973	15	999973	15	1249973	15
QNR	750015	32767	1600016	65535	2456803	84464	3400017	131071	4250017	131071

Figura 1: Quantidade de comparações e movimentos dos testes no vetor OrdC.

VETOR CRESCENTE					
	50000	100000	150000	200000	250000
QC	2566	3617	5406	7630	9392
QM3	2173	4548	6646	9649	10933
QPE	2164676	8661283	-	-	-
QI1P	806	1603	2417	3212	3986
QI5P	596	1193	1835	2412	3019
QI10P	532	1054	1577	2109	2553
QNR	2741	5591	8092	11822	14856

Figura 1.1: Tempo gasto pelos testes no vetor OrdC.

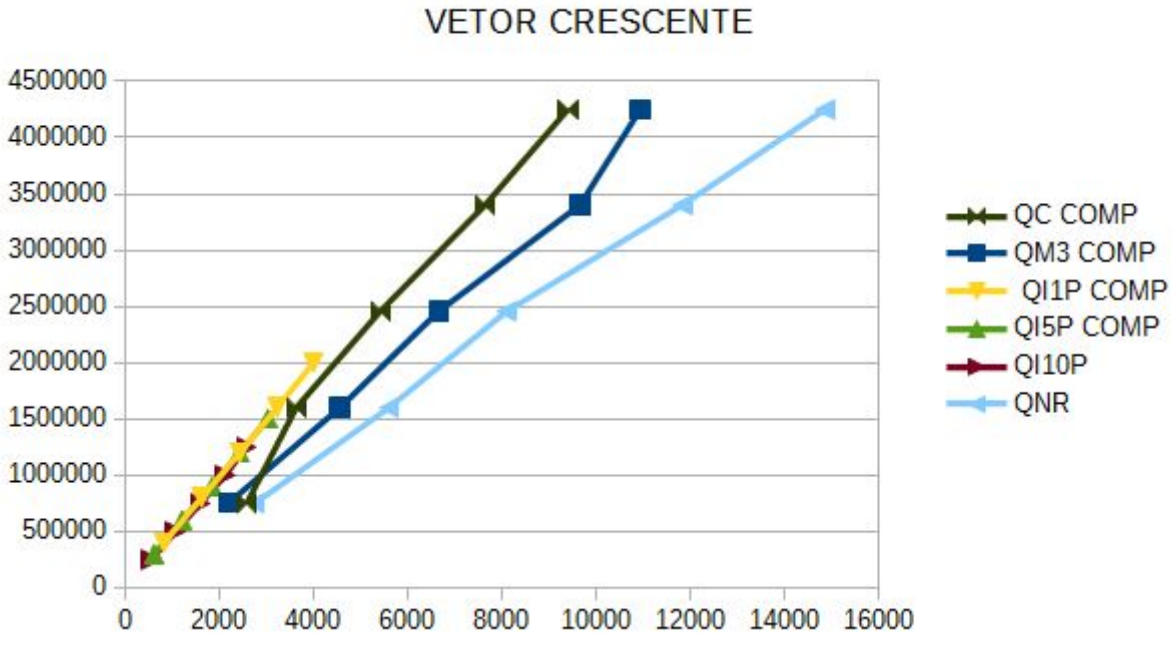

Figura 1.2: Número de comparações (Y) por tempo de execução (X).

Baseado na tabela e nos valores dos gráficos percebemos que QPE estava assumindo valores altos tanto de movimentação quanto de tempo que ele tornou-se inviável

de ser mostrado no gráfico. Além disso o QI10P se tornou o mais viável entre os 7 tipos de ordenação do quicksort tanto de movimentação quanto de execução, devido ao método de inserção contido neste quicksort, pois a inserção em vetores crescentes é o método de ordenação mais eficiente.

Observação: Não foi possível pegar alguns valores de comparação, movimentação e tempo da variação de quicksort primeiro elemento, devido a demora de execução desse algoritmo e a quantidade de memória sendo usada. Além disso, os resultados que foram obtidos, continham overflow. Isso só ocorre, pois o quicksort primeiro elemento sempre entra no pior caso de ordenação.

Vetor Ordenado em ordem Decrescente

A figura 2 apresenta a quantidade de comparações e movimentos para o vetor ordenado em ordem decrescente, enquanto a figura 2.1 apresenta os resultados em função do tempo.

VETOR DECRESCENTE										
	50000		100000		150000		200000		250000	
	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.
QC	750028	57766	1600030	115534	2456820	159464	3400032	231070	4250032	256070
QM3	750028	57766	1600030	115534	2456820	159464	3400032	231070	4250032	256070
QPE	1250099996	49999	705232700	99999	-	-	-	-	-	-
QI1P	399758	25126	799758	50126	1199758	75126	1599758	100126	1999758	125126
QI5P	299946	25030	599946	50030	899946	75030	1199946	100030	1499946	125030
QI10P	249976	25014	499976	50014	749976	75014	999976	100014	1249976	125014
QNR	750028	57766	1600030	115534	2456820	159464	3400032	231070	4250032	256070

Figura 2: Quantidade de comparações e movimentos dos testes no vetor OrdD.

VETOR DECRESCENTE					
	50000	100000	150000	200000	250000
QC	1805	3790	5558	8101	9549
QM3	2197	4524	6549	9434	11355
QPE	2156013	8625935	-	-	-
QI1P	915	1864	2702	3517	4421
QI5P	684	1355	2033	2716	3426
QI10P	592	1184	1781	2366	2983
QNR	2766	5677	8739	11708	13625

Figura 2.1: Tempo gasto pelos testes no vetor OrdD.

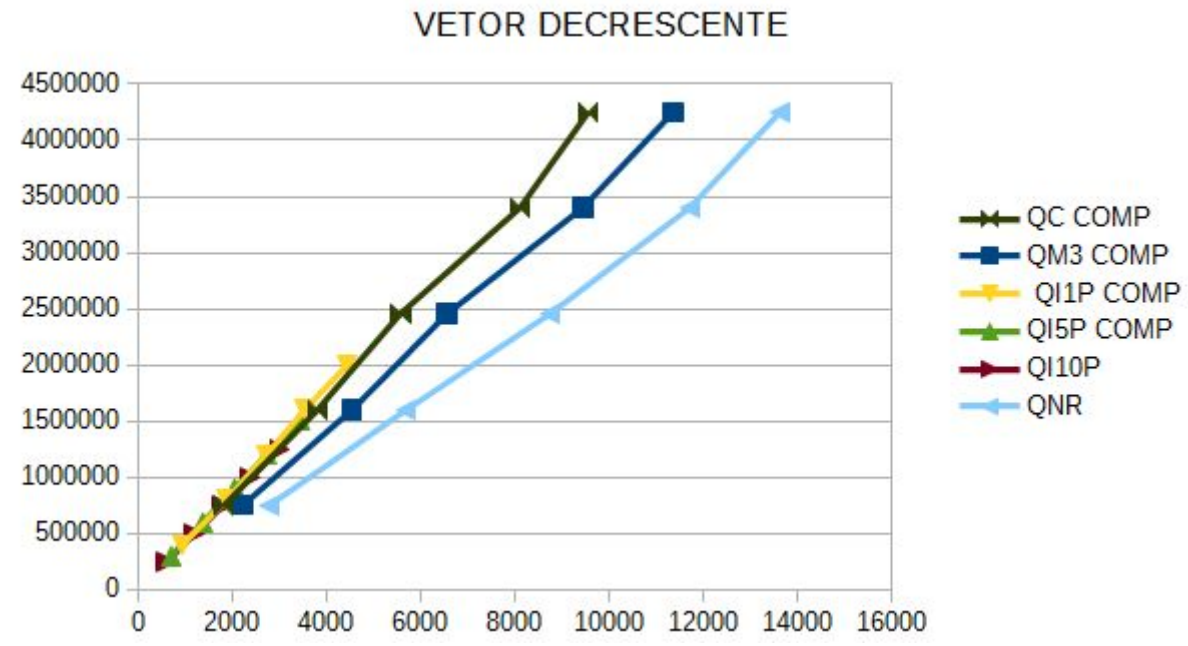

Figura 2.2: Número de comparações (Y) por tempo de execução (X).

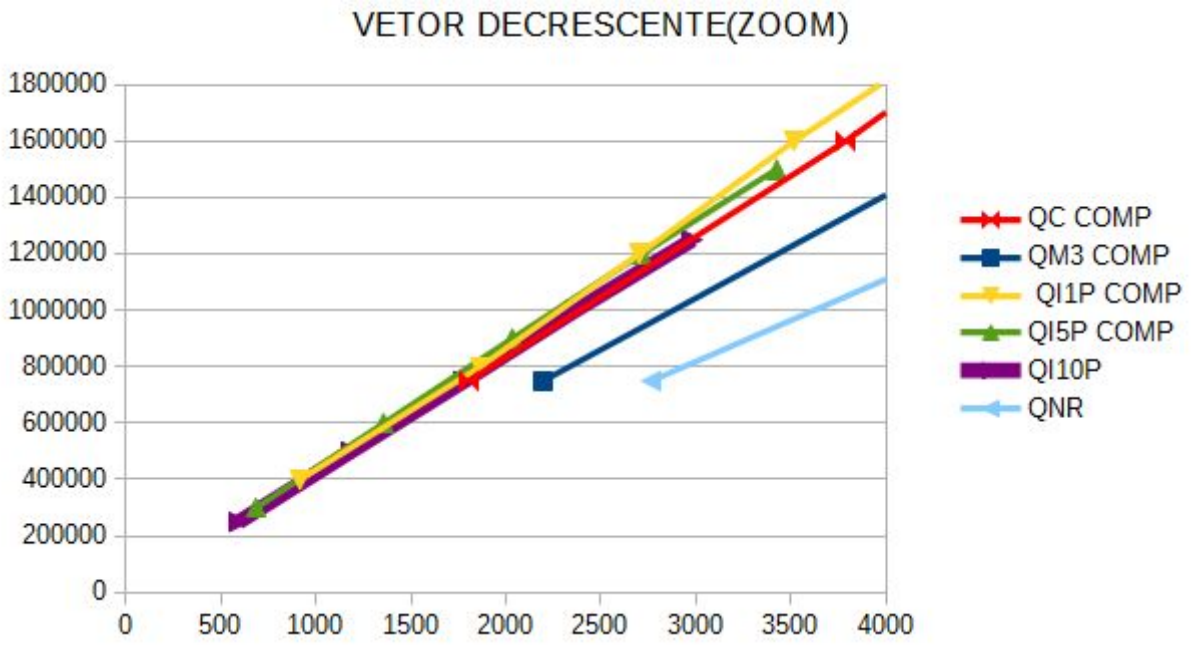

Figura ampliada 2.2.1: Número de comparações (Y) por tempo de execução (X).

Baseado na tabela e nos valores dos gráficos percebemos que QPE estava assumindo valores altos tanto de movimentação quanto de tempo que ele tornou-se inviável de ser mostrado no gráfico. Além disso o QI10P se tornou o mais viável entre os 7 tipos de ordenação do quicksort tanto de movimentação quanto de execução. Contudo, o QI1P e o

QC se tornaria mais viáveis se fosse usado vetores maiores pois ele acabaria se tornando mais eficiente. Pois, em vetores não crescente e vetores grandes o método de inserção não é muito eficiente, logo o QI1P se tornaria com melhor resultado com o tempo.

Observação: Não foi possível pegar alguns valores de comparação, movimentação e tempo da variação de quicksort primeiro elemento, devido a demora de execução desse algoritmo e a quantidade de memória sendo usada. Além disso, os resultados que foram obtidos, continham overflow. Isso só ocorre, pois o quicksort primeiro elemento sempre entra no pior caso de ordenação.

Vetor Aleatório

VETOR ALEATÓRIO										
	50000		100000		150000		200000		250000	
	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.
QC	1032338	199686	2235727	421544	3473573	653174	4746392	889904	6002760	1131189
QM3	949368	204871	2033483	433144	3142042	671164	4296871	914266	5521851	1159116
QPE	1171109	200612	2509905	424214	3915992	655426	5310853	895175	6794653	1137164
QI1P	4708653	4299185	18071164	17244228	39935175	38705743	70006917	68377020	109321412	107246912
QI5P	21720830	21429344	85839084	85249506	192319420	191428726	338243568	337038282	535264323	533768116
QI10P	43281751	43032123	171008930	170520389	381581880	380855578	673876111	672887476	1064192676	1062961864
QNR	1037990	199512	2232743	421731	3418376	654821	4727578	890836	6036503	1131275

Figura 3: Quantidade de comparações e movimentos dos testes no vetor aleatório

VETOR ALEATÓRIO					
	50000	100000	150000	200000	250000
QC	6636	13977	21928	29403	36841
QM3	7132	14901	23251	31520	40731
QPE	6774	13965	21519	29339	37211
QI1P	12144	42430	91760	155234	239858
QI5P	47478	182084	408625	709494	1109393
QI10P	91347	367615	811559	1395026	2166363
QNR	7796	16252	24994	33954	43826

Figura 3.1: Tempo gasto pelos testes no vetor aleatório.

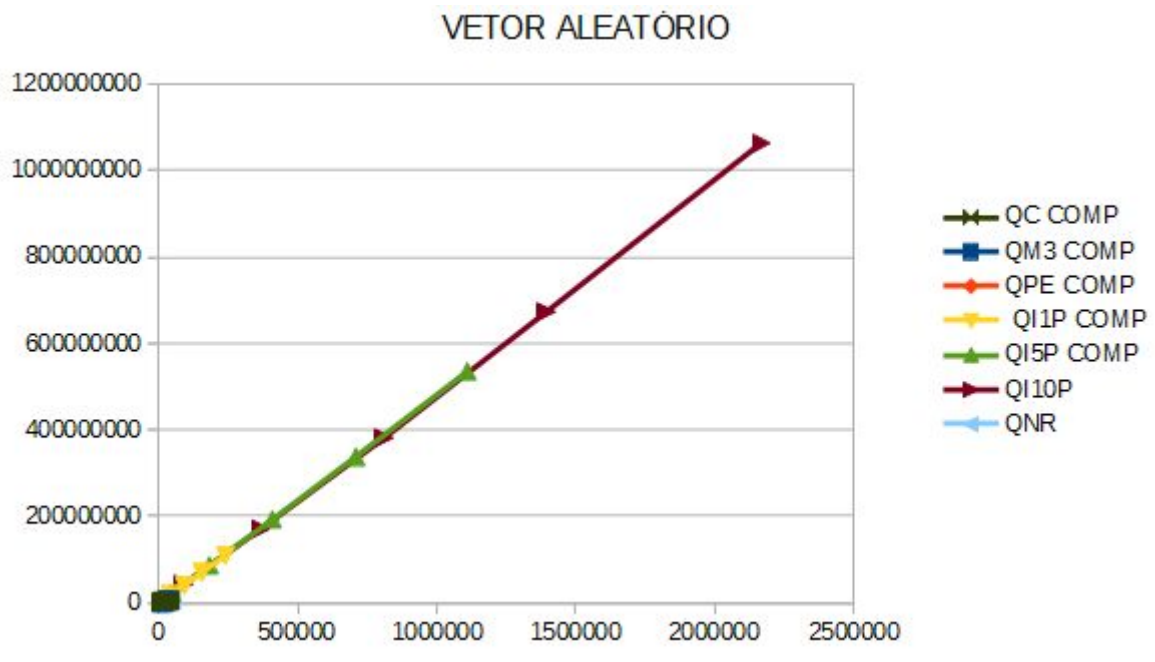

Figura 2.2: Número de comparações (Y) por tempo de execução (X).

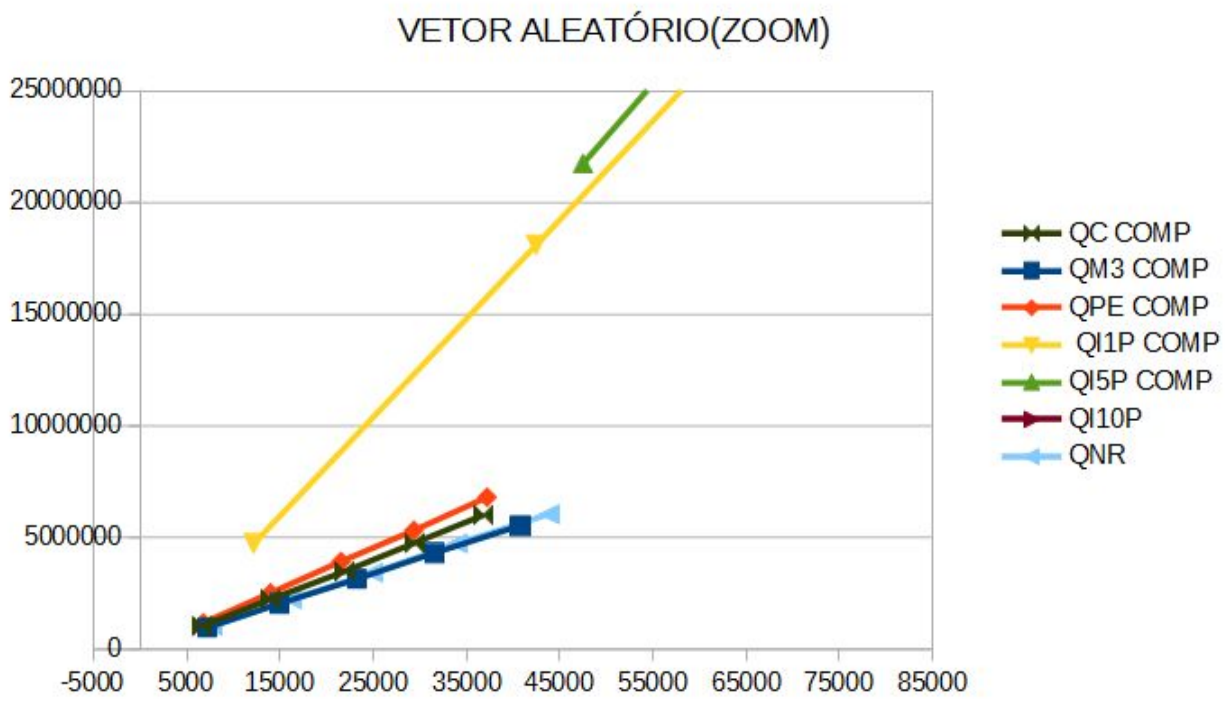

Figura ampliada 2.2.1: Número de comparações (Y) por tempo de execução (X).

Baseado na tabela e nos valores dos gráficos percebemos que QC era o mais viável dos 7 métodos de ordenação, mostrando tanto número de movimentações comparações e tempo de execuções boas. Além disso, o QI1P, QI5P e o QI10P demonstraram um número de comparações muito alto fazendo que esses métodos se tornassem totalmente inviáveis.

- **Conclusão**

Foram apresentados 7 métodos de ordenação da variação do quicksort. Após os testes e o estudo dos resultados e dos algoritmos, são apresentados observações sobre cada um dos métodos.

1. Quicksort Clássico

Apresentou um ótimo desempenho em ordenação de vetores aleatórios.

2. Quicksort mediana de três

Em geral esse método apresentou bons resultados se adequando bem as situações.

3. Quicksort primeiro elemento

Apresentou o pior desempenho de ordenação tanto crescente quanto decrescente em relação a comparação e tempo de execução. Isso se deve porque ele sempre cai no pior caso de ordenação.

4. Quicksort inserção 1%

Apresentou um bom desempenho no vetor de ordenação em ordem crescente, apesar de não ter sido o melhor por causa do tamanho do subvetor. No método de ordenação aleatória ele não teve um bom desempenho

5. Quicksort inserção 5%

Seu desempenho foi bom nos métodos de ordenação em ordem crescente e decrescente, porém na ordenação aleatória ele não teve um desempenho bom.

6. Quicksort inserção 10%

Apresentou bons resultados nos métodos de ordenação de vetores crescente e decrescente, funcionando bem em tamanho de subvetores pequenos.

7. Quicksort não recursivo

Sua eficiência nos métodos de vetor de ordenação em ordem decrescente e ordem crescente não foram boas e no método de ordenação aleatória teve um bom desempenho. Esse método se equipara ao quicksort clássico, contudo se torna melhor, pois a memória auxiliar para pilha de recursão é baixa.

Podemos concluir com isso, que ficou evidente a discrepância de alguns métodos como o quicksort primeiro elemento que se tornou uma péssima escolha. Contudo, em casos que o vetor está crescente métodos que continham inserção se tornaram melhores, vetores decrescentes e aleatórios o quicksort com pivô central se tornaria mais eficiente em vetores grandes e em vetores pequenos os que contém inserção são mais eficientes. Não foi possível verificar o resultado com vetores gigantes, devido ao overflow com as variáveis e o tempo de execução muito longo.

- ## Bibliografia

CHAIMOWICZ, Luiz. *Ordenação: Quicksort*. Belo Horizonte, 2019/1.