Társila Samille Santos da Silveira

Análise Empírica de Algoritmos de Ordenação

Brasil
2020, v-1.0

Társila Samille Santos da Silveira

Análise empírica de algoritmos

Relatório técnico apresentado à disciplina de Estrutura de Dados Básicas I, como requisito parcial para obtenção de nota referente à unidade I.

Universidade Federal do Rio Grande do Norte - UFRN
Instituto Metrópole Digital - IMD
Bacharelado em Tecnologia da Informação

Brasil
2020, v-1.0

Conteúdo

1 Introdução

Este relatório objetiva realizar análises e comparações de algoritmos de ordenação eles foram executadas em um mesmo computador, no sistema operacional Ubutu. Todos os valores foram registrados em tabelas e plotados em gráficos, permitindo fácil comparação.

Nas seções seguintes, é apresentado o método seguido, abrangendo os materiais e ferramentas utilizadas; depois, são mostrados os resultados obtidos e, por fim, as discussões geradas a partir deles. O único apêndice traz a implementação em C++ dos algoritmos escolhidos.

2 Metodologia

2.1 Materiais utilizados

2.1.1 Computador

- Computador: Acer Aspire-A315-42G
- Especificações:

  Processador IAMD® Ryzen 5 3500u gfx × 8

  2 x 8GB DDR4

  HD de 1T GB

  Radeon 540X Series (POLARIS12, DRM 3.35.0, 5.4.0-47-generic, LLVM 10.0.0) / AMD® Raven

- Sistema Operacional e suas Especificações:

  GNU/Linux

  Ubuntu 20.04.1 LTS(x86-64) Kernel 3.36.3

2.1.2 Ferramentas de programação

Os quatro algoritmos escolhidos foram implementados na linguagem C++, padrão ISO/IEC 14882:2011, ou simplesmente C++11.

Os códigos foram compilados pelo CMake, uma família de ferramentas de plataforma cruzada de código aberto projetada para construir, testar e empacotar software. CMake é usado para controlar o processo de compilação do software usando uma plataforma simples e arquivos de configuração independentes do compilador, e gerar makefiles e espaços de trabalho nativos que podem ser usados no ambiente de copilação de sua escolha. Foi utilizado:

```
>  cmake -S . -Bbuild
> cd build
> make
```

A biblioteca chrono3 foi responsável pelas medições do tempo de execução.

2.2 Método de comparações

Os algoritmos foram comparados segundo o critério de tempo de execução em relação ao tamanho do array.

Para gerar o gráfico de 25 pontos, o vetor inicial começou com 10.000 e foi incrementado de 40mil em 40mil até chegar em 1.010.000 elementos.

Foi implementado e analisado sete algoritmos. São eles: insertion sort, selection sort, bubble sort, shell sort, quick sort, merge sort e radix sort (LSD).

Com relação a organização das amostras foi simulado seis situações:

1. arranjos com elementos em ordem não decrescente,
2. arranjos com elemento em ordem não crescente,
3. arranjos com elementos 100% aleatórios,
4. arranjos com 75% de seus elementos em sua posição definitiva,
5. arranjos com 25% de seus elementos em sua posição definitiva, e
6. arranjos com 50% de seus elementos em sua posição definitiva.

3 Resultados

3.1 Tabelas - por algoritmo

3.1.1 Insertion

A Tabela 1 e 2 apresenta os resultados dos testes realizados com o algoritmo Insertion sort em diferentes situações de amostra, o resultado de tempo está em milissegundos.

Tabela 1: Insertion Sort (tempo em ms)

Tamanho do Array	**Crescente**	**Decrescente**	**Aleatório**
10000	550,377	101,956	75,482
50000	1077,17	2.421,970	1.583,800
90000	3469,59	7.833,780	5.404,500
130000	7462,2	16.256,000	11.225,900
170000	12557,3	27.856,500	18.535,200
210000	18880,7	42.301,900	28.711,700
250000	27008,6	59.965,400	41.137,200
290000	36740,1	80.531,800	56.253,300
330000	47595,7	104.420,000	71.916,000
370000	59177,2	131.021,000	98.030,400
410000	72527,3	160.739,000	119.197,000
450000	88845,4	193.791,000	145.967,000
490000	104.553,000	229.605,000	173.334,000
530000	122.178,000	268.542,000	203.260,000
570000	142.547,000	315.684,000	226.165,000
610000	163.120,000	394.911,000	249.170,000
650000	183.942,000	465.972,000	275.138,000
690000	208.621,000	522.556,000	316.797,000
730000	232.146,000	590.953,000	377.368,000
770000	260.515,000	628.866,000	421.278,000
810000	288.769,000	693.121,000	447.360,000
850000	305.281,000	771.121,000	524.716,000
890000	335.134,000	818.266,000	649.251,000
930000	366.052,000	904.231,000	723.784,000
970000	399.564,000	928.554,000	741.000,000

Tabela 2: Insertion Sort (tempo em ms)

Tamanho do Array	**75% Aleatório**	**50% Aleatório**	**25% Aleatório**
10000	71,9663	83,2349	44,6875
50000	1575,7400	2097,1300	1087,77
90000	4963,9100	6730,4000	3515,69
130000	10505,3000	14155,9000	7416,56
170000	18269,3000	20408,5000	12614,7
210000	27249,7000	23833,7000	19225,9
250000	38286,6000	41619,8000	27512,7
290000	51730,9000	51795,8000	37160,9
330000	66350,6000	63412,8000	48283,8
370000	86469,8000	90323,5000	60906,1
410000	107734,0000	108305,0000	74446,6
450000	133323,0000	147413,0000	89390,6
490000	159320,0000	178700,0000	105529
530000	186991,0000	220332,0000	124057
570000	219251,0000	255489,0000	143506
610000	257619,0000	278879,0000	164648
650000	292809,0000	337602,0000	209307
690000	345894,0000	408861,0000	233766
730000	434744,0000	481184,0000	261210
770000	571000,0000	490971,0000	280444
810000	657000,0000	438604,0000	310181
850000	698000,0000	502607,0000	339117
890000	822000,0000	452795,0000	379192
930000	890000,0000	476458,0000	417594
970000	926000,0000	552236,0000	4,52E+05

3.1.2 Selection

A Tabela 3 e 4 apresenta os resultados dos testes realizados com o algoritmo Selection sort em diferentes situações de amostra, o resultado de tempo está em milissegundos.

Tabela 3: Selection Sort (tempo em ms)

Tamanho do Array	Crescente	Decrescente	Aleatório
10000	37,609	251,551	43,698
50000	667,644	6.326,880	844,853
90000	2.152,050	19.557,300	2.804,390
130000	4.342,840	40.984,400	5.675,910
170000	7.769,350	75.251,500	9.105,960
210000	11.987,100	118.257,000	13.858,000
250000	16.443,600	175.041,000	17.900,500
290000	23.194,300	238.020,000	23.277,200
330000	30.984,100	323.680,000	30.136,300
370000	39.758,700	390.678,000	39.653,400
410000	46.609,100	485.030,000	49.330,800
450000	57.374,500	513.363,000	62.570,100
490000	65.690,400	629.288,000	71.472,800
530000	75.679,300	673.056,000	83.609,700
570000	90.300,400	659.361,000	97.260,400
610000	101.356,000	105.012,000	110.160,000
650000	123.253,000	114.622,000	128.671,000
690000	150.585,000	133.372,000	165.380,000
730000	160.535,000	135.360,000	184.990,000
770000	195.979,000	150.631,000	212.639,000
810000	208.087,000	175.798,000	232.121,000
850000	223.211,000	213.962,000	242.870,000

890000	247.716,000	231.065,000	271.252,000
930000	266.839,000	261.714,000	294.587,000
970000	278.371,000	396.680,000	315.804,000

Tabela 4: Selection Sort (tempo em ms)

Tamanho do Array	**75% Aleatório**	**50% Aleatório**	**25% Aleatório**
10000	37,6521	33,4927	38,4499
50000	718,287	669,0210	713,19
90000	2326,7	2161,1100	2367,58
130000	5085,31	4636,3800	5007,98
170000	9109,59	8009,9700	8692,24
210000	13868,2	11957,9000	13378,6
250000	20437,2	18897,2000	20264,7
290000	29357,2	24041,7000	25862,9
330000	37036,7	29003,1000	32907,9
370000	48364,7	36731,0000	39409,2
410000	58017,6	46782,3000	51058,9
450000	61948,2	55707,5000	61474,9
490000	72947,3	65936,1000	72703,2
530000	85506,8	94190,8000	87830,4
570000	101491	104036,0000	98085,9
610000	114963	113317,0000	114881
650000	131645	126524,0000	131305
690000	149140	146973,0000	146701
730000	163804	164827,0000	167859
770000	1,85E+05	188794,0000	185998
810000	2,06E+05	208127,0000	203345

850000	2,25E+05	225143,0000	225362
890000	2,73E+05	239874,0000	247785
930000	2,88E+05	254858,0000	272509
970000	2,97E+05	269000,0000	2,960E+05

3.1.3 Bubble

A Tabela 5 e 6 apresenta os resultados dos testes realizados com o algoritmo Bubble sort em diferentes situações de amostra, o resultado de tempo está em milissegundos.

Tabela 5: Bubble Sort (tempo em ms)

Tamanho do Array	**Crescente**	**Decrescente**	**Aleatório**
10000	0,022424	106,753	744,671
50000	0,107498	2.448,490	3612,32
90000	0,193374	8.009,900	11997
130000	0,279271	16.593,000	25243,6
170000	0,365337	28.866,500	43637,4
210000	0,451113	43.650,400	66699,8
250000	0,549444	62.157,100	95124,8
290000	0,622847	83.610,200	128264
330000	0,935331	112.238,000	166221
370000	0,794930	141.223,000	209526
410000	0,528623	174.459,000	257547
450000	0,477564	216.846,000	311703
490000	0,523333	266.344,000	368725
530000	0,524486	323.430,000	431170
570000	0,535467	399.596,000	499702
610000	0,570104	418.145,000	571088
650000	0,561848	482.144,000	648969

690000	0,593710	551.190,000	733277
730000	0,622886	620.718,000	822037
770000	0,653485	684.272,000	912443
810000	0,737178	743.385,000	1,01E+06
850000	0,751024	784.500,000	1,11E+06
890000	1,227140	877.607,000	1,22E+06
930000	0,861118	906.153,000	1,33E+06
970000	1,159530	1.010.000,000	1,45E+06

Tabela 6: Bubble Sort (tempo em ms)

Tamanho do Array	**75% Aleatório**	**50% Aleatório**	**25% Aleatório**
10000	105,066	81,5117	41,4325
50000	2689,47	1781,89	736,401
90000	9631,35	6453,36	3707,38
130000	20961,8	13996,9	7767,99
170000	34961,3	22839,8	9728,61
210000	52981,2	34842,5	14047
250000	75972,3	50641,7	30556,4
290000	108289	69439,4	41044,3
330000	134829	87064,3	52433,3
370000	172928	112132	63955,7
410000	212924	139999	85407,8
450000	248535	168463	104720
490000	295738	198536	124587
530000	358047	230181	146553
570000	418542	267654	165060
610000	473960	312648	189147

650000	523900	352109	223827
690000	565002	398290	260618
730000	686203	460597	295028
770000	1,03E+06	547589	315196
810000	1,10E+06	626248	353801
850000	1,20E+06	663418	372166
890000	1,37E+06	705965	432554
930000	1,09E+06	794918	457822
970000	1,31E+06	8,89E+05	494771

3.1.4 Shell

A Tabela 7 e 8 apresenta os resultados dos testes realizados com o algoritmo Shell sort em diferentes situações de amostra, o resultado de tempo está em milissegundos.

Tabela 7: Shell Sort (tempo em ms)

Tamanho do Array	**Crescente**	**Decrescente**	**Aleatório**
10000	0,141934	0,523717	7,35771
50000	0,856916	3,182520	44,615100
90000	1,588370	5,854350	93,149600
130000	2,318410	5,418880	144,951000
170000	3,197750	4,214860	195,324000
210000	4,194180	4,851580	232,732000
250000	5,452360	5,791540	272,606000
290000	5,810690	6,949090	331,653000
330000	7,068030	7,592670	362,909000
370000	7,397750	8,463980	422,356000
410000	7,947250	10,787900	468,770000
450000	9,596600	10,854100	542,353000

490000	10,163400	12,955400	621,599000
530000	12,771100	12,830200	685,300000
570000	11,787800	22,619700	734,905000
610000	13,639700	16,416900	829,777000
650000	13,702200	17,871500	865,399000
690000	15,729900	20,338100	926,432000
730000	16,158100	18,753100	916,665000
770000	16,846300	18,970400	1.032,630000
810000	25,642200	20,440000	1.025,040000
850000	19,920500	21,290000	1.158,550000
890000	23,400600	24,184700	1.185,680000
930000	21,082400	21,561300	1.236,120000
970000	23,473300	27,453600	1.400,000000

Tabela 8: Shell Sort (tempo em ms)

Tamanho do Array	**75% Aleatório**	**50% Aleatório**	**25% Aleatório**
10000	1,69108	1,14954	0,641407
50000	6,64847	5,93433	4,07795
90000	7,72793	5,49962	9,95476
130000	12,66720	7,84125	10,2829
170000	16,77220	11,81700	8,17809
210000	19,92270	13,25740	8,67102
250000	25,32670	16,71950	11,5839
290000	29,58900	21,65760	15,8813
330000	34,03590	24,26660	17,7484
370000	39,19720	29,64430	19,2139
410000	44,07610	31,37050	24,0502

450000	50,91630	36,15880	26,9892
490000	53,61650	39,76620	30,1645
530000	58,91190	42,14160	34,0697
570000	61,79880	45,50250	32,4792
610000	67,13650	50,62130	37,9639
650000	75,71080	56,11120	42,3954
690000	76,72010	59,40220	44,02
730000	81,72860	63,36920	47,2162
770000	90,97260	72,73350	57,7582
810000	92,06470	72,30000	56,5178
850000	97,34640	78,70000	57,7914
890000	101,78200	77,50000	59,3907
930000	107,52300	80,70000	65,3049
970000	111,00000	88,40000	68,5997

3.1.5 Quick

A Tabela 9 e 10 apresenta os resultados dos testes realizados com o algoritmo Quick sort em diferentes situações de amostra, o resultado de tempo está em milissegundos.

Tabela 9: Quick Sort (tempo em ms)

Tamanho do Array	**Crescente**	**Decrescente**	**Aleatório**
10000	38,439	56,124	1,37187
50000	896,361	1.062,460	6,4013
90000	3.049,340	3.409,320	5,74139
130000	6.549,280	7.592,170	8,68235
170000	11.126,400	12.901,200	12,1732
210000	16.527,600	19.826,400	14,997
250000	25.352,100	29.123,600	18,4564

290000	32.999,700	39.968,600	20,9403
330000	47.413,300	51.827,800	23,6509
370000	66.119,800	65.741,900	26,4366
410000	75.248,800	81.022,100	29,3219
450000	87.027,600	96.126,300	31,554
490000	98.586,300	108.192,000	34,7285
530000	116.402,000	126.368,000	37,8392
570000	134.467,000	157.527,000	39,9053
610000	152.380,000	183.631,000	43,381
650000	167.082,000	204.218,000	46,3887
690000	185.430,000	231.449,000	49,61
730000	208.585,000	266.976,000	54,1953
770000	249.621,000	306.143,000	55,1586
810000	276.822,000	335.396,000	57,5653
850000	302.914,000	369.251,000	59,7177
890000	329.387,000	416.312,000	64,1731
930000	369.468,000	461.634,000	70,2149
970000	442.935,000	471.350,000	71,9924

Tabela 10: Quick Sort (tempo em ms)

Tamanho do Array	**75% Aleatório**	**50% Aleatório**	**25% Aleatório**
10000	1,40905	1,45423	1,3815
50000	7,94807	6,79341	5,7543
90000	7,62955	6,44315	5,2858
130000	17,1286	8,14445	8,8323
170000	13,468	10,6635	10,8639
210000	14,8065	13,7061	13,0859
250000	17,7589	15,8521	27,1547

290000	22,6626	18,7975	24,6716
330000	23,5924	22,0336	22,3685
370000	32,9361	34,7816	22,8828
410000	32,4829	26,8391	26,2544
450000	32,0686	30,9759	27,0561
490000	34,0769	33,4554	30,1636
530000	37,3972	36,1347	32,3819
570000	40,2315	38,4495	35,8989
610000	43,0951	39,7197	37,4928
650000	45,7639	42,8598	40,2120
690000	50,4749	46,3042	45,3899
730000	52,8868	48,8535	49,4176
770000	56,6467	51,3839	51,1772
810000	59,0648	58,0092	59,6217
850000	63,2095	58,3855	57,2204
890000	65,5121	64,8364	56,0862
930000	68,4372	62,8559	61,8559
970000	73,0338	64,7771	68,7971

3.1.6 Merge

A Tabela 11 e 12 apresenta os resultados dos testes realizados com o algoritmo Merge sort em diferentes situações de amostra, o resultado de tempo está em milissegundos.

Tabela 11: Merge Sort (tempo em ms)

Tamanho do Array	**Crescente**	**Decrescente**	**Aleatório**
10000	0,831938	0,789048	9,24844
50000	4,28844	4,177600	45,2281
90000	3,98192	6,201410	84,8571

130000	3,4759	7,400880	122,284
170000	5,69468	6,680200	159,797
210000	6,87792	7,259700	195,756
250000	7,73427	8,919320	235,997
290000	9,56693	9,688910	284,329
330000	11,558	20,417100	316,247
370000	12,2998	12,934000	369,546
410000	13,9059	13,313400	409,37
450000	15,8798	14,059400	465,244
490000	16,5206	18,762200	495,485
530000	17,7825	19,855300	541,499
570000	21,1192	27,974700	582,658
610000	20,3087	20,009700	631,695
650000	25,7103	20,682000	646,727
690000	23,8981	25,101500	694,036
730000	31,0487	24,512600	760,506
770000	25,6626	26,404000	781,451
810000	28,3349	25,631100	811,852
850000	29,9325	28,272800	835,472
890000	29,6335	31,001500	906,818
930000	31,3074	31,700000	924,722
970000	30,0276	32,527300	9,83E+02

Tabela 12: Merge Sort (tempo em ms)

Tamanho do Array	**75% Aleatório**	**50% Aleatório**	**25% Aleatório**
10000	1,66901	1,57221	0,999526
50000	8,47195	7,67271	4,7733

90000	9,52492	8,12475	7,64743
130000	10,4264	8,37098	6,46336
170000	14,4231	12,4251	8,74087
210000	17,0069	15,3763	11,5736
250000	21,7212	17,7885	13,645
290000	24,6108	21,3079	15,4634
330000	27,2183	23,2263	17,2189
370000	31,1577	25,1644	20,9858
410000	34,8153	27,9632	21,1328
450000	37,5267	31,8033	23,1439
490000	40,6836	33,853	24,8705
530000	43,3928	35,4764	26,9197
570000	48,6762	38,7347	29,6399
610000	52,4619	44,3055	32,8406
650000	55,9544	45,8265	33,3826
690000	59,229	48,6191	36,9185
730000	62,5811	50,9146	39,4938
770000	65,9231	56,8765	40,4992
810000	69,7567	56,2739	44,1344
850000	73,3342	59,0107	44,3991
890000	79,6538	63,3542	47,865
930000	81,0907	64,5547	49,6476
970000	85,3933	70,6367	52,5072

3.1.7 Radix

A Tabela 13 e 14 apresenta os resultados dos testes realizados com o algoritmo Radix sorte em diferentes situações de amostra, o resultado de tempo está em milissegundos.

Tabela 13: Radix Sort (tempo em ms)

Tamanho do Array	Crescente	Decrescente	Aleatório
10000	1,06669	1,21837	1,32586
50000	5,63104	7,59981	7,52265
90000	9,07509	5,13239	5,77492
130000	8,87866	8,71697	12,5726
170000	15,0095	16,1068	11,1341
210000	20,1975	19,1271	16,3798
250000	16,089	20,3002	26,0519
290000	21,3857	29,0642	26,1971
330000	31,417	39,4336	29,6249
370000	35,3998	35,7185	35,4817
410000	37,5829	42,564	43,115
450000	42,4596	50,0124	47,7867
490000	43,6295	51,6474	53,9599
530000	54,3095	48,7403	55,0018
570000	59,4458	55,6399	62,7936
610000	61,9212	65,2542	65,8268
650000	67,2574	60,4038	65,9438
690000	68,8658	67,7286	71,7584
730000	73,843	77,5445	71,2754
770000	82,1858	96,5155	82,4063
810000	81,6243	87,2034	96,6291
850000	90,2094	86,5011	92,0958
890000	100,05	91,2127	91,6926
930000	98,6862	109,933	100,774
970000	101,347	96,7173	105,548

Tabela 14:

Tamanho do Array	75% Aleatório	50% Aleatório	25% Aleatório
10000	1,0528	1,02896	1,2753
50000	5,835	4,53697	5,76415
90000	5,99769	5,04388	7,11213
130000	8,42169	10,5381	8,43051
170000	11,9533	15,5484	13,1388
210000	19,7669	18,4242	20,1131
250000	19,8516	23,9704	25,2876
290000	25,6012	29,7053	29,1213
330000	26,9539	30,1615	35,4637
370000	28,3068	40,6771	37,3114
410000	42,0241	42,0839	44,379
450000	50,0832	48,1648	47,5019
490000	53,2877	49,8581	50,3318
530000	57,6196	56,1351	64,1493
570000	56,6466	56,6386	69,2543
610000	66,7071	64,1567	69,4745
650000	69,2882	63,7088	71,7552
690000	69,5621	68,9464	77,3572
730000	75,4068	79,9859	84,8757
770000	83,9746	78,7453	86,798
810000	76,4008	88,1493	93,0883
850000	67,107	87,8286	94,1368
890000	77,5358	94,2925	100,723
930000	88,768	100,436	107,305
970000	101,277	109,015	109,578

3.2 Tabelas - por tipo de amostra

3.2.1 Crescente

A Tabela 15 e 16 apresenta os resultados dos testes realizados com o algoritmos com o array em ordem crescente, o resultado de tempo está em milissegundos.

Tabela 15: Crescente (tempo em ms)

Tamanho do Array	Insertion	Selection	Bubble
10000	50,331	37,609	0,022424
50000	1.171,200	667,644	0,107498
90000	4.005,900	2.152,050	0,193374
130000	7.590,030	4.342,840	0,279271
170000	13.110,200	7.769,350	0,365337
210000	19.184,900	11.987,100	0,451113
250000	29.050,700	16.443,600	0,549444
290000	38.312,700	23.194,300	0,622847
330000	45.691,400	30.984,100	0,935331
370000	61.361,600	39.758,700	0,794930
410000	75.184,200	46.609,100	0,528623
450000	87.541,100	57.374,500	0,477564
490000	107.386,000	65.690,400	0,523333
530000	138.373,000	75.679,300	0,524486
570000	140.454,000	90.300,400	0,535467
610000	122.955,000	101.356,000	0,570104
650000	135.323,000	123.253,000	0,561848
690000	146.200,000	150.585,000	0,593710
730000	161.398,000	160.535,000	0,622886
770000	201.366,000	195.979,000	0,653485
810000	218.224,000	208.087,000	0,737178

850000	229.712,000	223.211,000	0,751024
890000	261.962,000	247.716,000	1,227140
930000	310.386,000	266.839,000	0,861118
970000	328.501,000	278.371,000	1,159530

Tabela 16: Crescente (tempo em ms)

Tamanho do Array	**Shell**	**Quick**	**Merge**	**Radix**
10000	0,141934	38,439	0,831938	1,06669
50000	0,856916	896,361	4,28844	5,63104
90000	1,588370	3.049,340	3,98192	9,07509
130000	2,318410	6.549,280	3,4759	8,87866
170000	3,197750	11.126,400	5,69468	15,0095
210000	4,194180	16.527,600	6,87792	20,1975
250000	5,452360	25.352,100	7,73427	16,089
290000	5,810690	32.999,700	9,56693	21,3857
330000	7,068030	47.413,300	11,558	31,417
370000	7,397750	66.119,800	12,2998	35,3998
410000	7,947250	75.248,800	13,9059	37,5829
450000	9,596600	87.027,600	15,8798	42,4596
490000	10,163400	98.586,300	16,5206	43,6295
530000	12,771100	116.402,000	17,7825	54,3095
570000	11,787800	134.467,000	21,1192	59,4458
610000	13,639700	152.380,000	20,3087	61,9212
650000	13,702200	167.082,000	25,7103	67,2574
690000	15,729900	185.430,000	23,8981	68,8658
730000	16,158100	208.585,000	31,0487	73,843
770000	16,846300	249.621,000	25,6626	82,1858
810000	25,642200	276.822,000	28,3349	81,6243

850000	19,920500	302.914,000	29,9325	90,2094
890000	23,400600	329.387,000	29,6335	100,05
930000	21,082400	369.468,000	31,3074	98,6862
970000	23,473300	442.935,000	30,0276	101,347

3.2.2 Decrescente

A Tabela 17 e 18 apresenta os resultados dos testes realizados com o algoritmos com o array em ordem decrescente, o resultado de tempo está em milissegundos.

Tabela 17: Decrescente (tempo em ms)

Tamanho do Array	**Insertion**	**Selection**	**Bublee**
10000	101,956	251,551	106,753
50000	2.421,970	6.326,880	2.448,490
90000	7.833,780	19.557,300	8.009,900
130000	16.256,000	40.984,400	16.593,000
170000	27.856,500	75.251,500	28.866,500
210000	42.301,900	118.257,000	43.650,400
250000	59.965,400	175.041,000	62.157,100
290000	80.531,800	238.020,000	83.610,200
330000	104.420,000	323.680,000	112.238,000
370000	131.021,000	390.678,000	141.223,000
410000	160.739,000	485.030,000	174.459,000
450000	193.791,000	513.363,000	216.846,000
490000	229.605,000	629.288,000	266.344,000
530000	268.542,000	673.056,000	323.430,000
570000	315.684,000	659.361,000	399.596,000
610000	394.911,000	105.012,000	418.145,000
650000	465.972,000	114.622,000	482.144,000
690000	522.556,000	133.372,000	551.190,000
730000	590.953,000	135.360,000	620.718,000
770000	628.866,000	150.631,000	684.272,000
810000	693.121,000	175.798,000	743.385,000
850000	771.121,000	213.962,000	784.500,000
890000	818.266,000	231.065,000	877.607,000
930000	904.231,000	261.714,000	906.153,000

970000	928.554,000	396.680,000	1.010.000,000

Tabela 18: Decrescente (tempo em ms)

Tamanho do Array	**Shell**	**Quick**	**Merge**	**Radix**
10000	0,523717	56,124	0,789048	1,21837
50000	3,182520	1.062,460	4,177600	7,59981
90000	5,854350	3.409,320	6,201410	5,13239
130000	5,418880	7.592,170	7,400880	8,71697
170000	4,214860	12.901,200	6,680200	16,1068
210000	4,851580	19.826,400	7,259700	19,1271
250000	5,791540	29.123,600	8,919320	20,3002
290000	6,949090	39.968,600	9,688910	29,0642
330000	7,592670	51.827,800	20,417100	39,4336
370000	8,463980	65.741,900	12,934000	35,7185
410000	10,787900	81.022,100	13,313400	42,564
450000	10,854100	96.126,300	14,059400	50,0124
490000	12,955400	108.192,000	18,762200	51,6474
530000	12,830200	126.368,000	19,855300	48,7403
570000	22,619700	157.527,000	27,974700	55,6399
610000	16,416900	183.631,000	20,009700	65,2542
650000	17,871500	204.218,000	20,682000	60,4038
690000	20,338100	231.449,000	25,101500	67,7286
730000	18,753100	266.976,000	24,512600	77,5445
770000	18,970400	306.143,000	26,404000	96,5155
810000	20,440000	335.396,000	25,631100	87,2034
850000	21,290000	369.251,000	28,272800	86,5011
890000	24,184700	416.312,000	31,001500	91,2127
930000	21,561300	461.634,000	31,700000	109,933

970000	27,453600	471.350,000	32,527300	96,7173

3.2.3 100% Aleatória

A Tabela 19 e 20 apresenta os resultados dos testes realizados com o algoritmos com o array em ordem 100% Aleatória, o resultado de tempo está em milissegundos.

Tabela 19: 100% Aleatória (tempo em ms)

Tamanho do Array	**Insertion**	**Selection**	**Bublee**
10000	75,482	43,698	744,671
50000	1.583,800	844,853	3.612,320
90000	5.404,500	2.804,390	11.997,000
130000	11.225,900	5.675,910	25.243,600
170000	18.535,200	9.105,960	43.637,400
210000	28.711,700	13.858,000	66.699,800
250000	41.137,200	17.900,500	95.124,800
290000	56.253,300	23.277,200	128.264,000
330000	71.916,000	30.136,300	166.221,000
370000	98.030,400	39.653,400	209.526,000
410000	119.197,000	49.330,800	257.547,000
450000	145.967,000	62.570,100	311.703,000
490000	173.334,000	71.472,800	368.725,000
530000	203.260,000	83.609,700	431.170,000
570000	226.165,000	97.260,400	499.702,000
610000	249.170,000	110.160,000	571.088,000
650000	275.138,000	128.671,000	648.969,000
690000	316.797,000	165.380,000	733.277,000
730000	377.368,000	184.990,000	822.037,000
770000	421.278,000	212.639,000	912.443,000
810000	447.360,000	232.121,000	1.010.000,000
850000	524.716,000	242.870,000	1.110.000,000
890000	649.251,000	271.252,000	1.220.000,000
930000	723.784,000	294.587,000	1.330.000,000
970000	741.000,000	315.804,000	1.450.000,000

Tabela 20: 100% Aleatória (tempo em ms)

Tamanho do Array	Shell	Quick	Merge	Radix
10000	2,66471	1,73032	1,685170	1,32586
50000	8,971450	9,24955	7,598790	7,52265
90000	15,222000	8,75612	8,632500	5,77492
130000	23,597100	12,90820	13,800800	12,5726
170000	32,181100	17,47580	17,228200	11,1341
210000	39,942300	21,63220	20,814300	16,3798
250000	45,240300	36,86790	24,772400	26,0519
290000	54,089800	34,22670	31,285900	26,1971
330000	61,973200	34,80950	33,319200	29,6249
370000	69,056900	39,89390	37,434000	35,4817
410000	79,863600	43,91160	41,449800	43,115
450000	90,836000	50,65230	45,189000	47,7867
490000	99,340400	54,34100	50,673700	53,9599
530000	105,528000	61,67970	54,418600	55,0018
570000	111,751000	64,23310	58,596100	62,7936
610000	123,036000	69,43410	66,014600	65,8268
650000	140,968000	74,87830	77,029200	65,9438
690000	139,786000	77,45850	77,417500	71,7584
730000	147,154000	82,45060	75,426000	71,2754
770000	167,645000	86,52560	79,025300	82,4063
810000	171,427000	89,30480	82,611800	96,6291
850000	183,507000	93,67870	90,737500	92,0958
890000	186,377000	99,22620	94,396700	91,6926
930000	196,277000	102,94200	101,729000	100,774
970000	204,762000	112,02100	100,628000	105,548

3.2.4 75% Aleatória

A Tabela 21 e 22 apresenta os resultados dos testes realizados com o algoritmos com o array em ordem 75% Aleatória, o resultado de tempo está em milissegundos.

Tabela 21: 75% Aleatória (tempo em ms)

Tamanho do Array	Insertion	Selection	Bublee
10000	71,9663	37,6521	105,066
50000	1575,7400	718,287	2689,470
90000	4963,9100	2326,7	9631,350
130000	10505,3000	5085,31	20961,800
170000	18269,3000	9109,59	34961,300
210000	27249,7000	13868,2	52981,200
250000	38286,6000	20437,2	75972,300
290000	51730,9000	29357,2	108289,000
330000	66350,6000	37036,7	134829,000
370000	86469,8000	48364,7	172928,000
410000	107734,0000	58017,6	212924,000
450000	133323,0000	61948,2	248535,000
490000	159320,0000	72947,3	295738,000
530000	186991,0000	85506,8	358047,000
570000	219251,0000	101491	418542,000
610000	257619,0000	114963	473960,000
650000	292809,0000	131645	523900,000
690000	345894,0000	149140	565002,000
730000	434744,0000	163804	686203,000
770000	571000,0000	1,85E+05	1030000,000
810000	657000,0000	2,06E+05	1100000,000
850000	698000,0000	2,25E+05	1200000,000
890000	822000,0000	2,73E+05	1370000,000
930000	890000,0000	2,88E+05	1090000,000
970000	926000,0000	2,97E+05	1310000,000

Tabela 22: 75% Aleatória (tempo em ms)

Tamanho do Array	Shell	Quick	Merge	Radix
10000	1,69108	1,40905	1,66901	1,0528

50000	6,64847	7,94807	8,47195	5,835
90000	7,72793	7,62955	9,52492	5,99769
130000	12,66720	17,1286	10,4264	8,42169
170000	16,77220	13,468	14,4231	11,9533
210000	19,92270	14,8065	17,0069	19,7669
250000	25,32670	17,7589	21,7212	19,8516
290000	29,58900	22,6626	24,6108	25,6012
330000	34,03590	23,5924	27,2183	26,9539
370000	39,19720	32,9361	31,1577	28,3068
410000	44,07610	32,4829	34,8153	42,0241
450000	50,91630	32,0686	37,5267	50,0832
490000	53,61650	34,0769	40,6836	53,2877
530000	58,91190	37,3972	43,3928	57,6196
570000	61,79880	40,2315	48,6762	56,6466
610000	67,13650	43,0951	52,4619	66,7071
650000	75,71080	45,7639	55,9544	69,2882
690000	76,72010	50,4749	59,229	69,5621
730000	81,72860	52,8868	62,5811	75,4068
770000	90,97260	56,6467	65,9231	83,9746
810000	92,06470	59,0648	69,7567	76,4008
850000	97,34640	63,2095	73,3342	67,107
890000	101,78200	65,5121	79,6538	77,5358
930000	107,52300	68,4372	81,0907	88,768
970000	111,00000	73,0338	85,3933	101,277

3.2.5 50% Aleatória

A Tabela 23 e 24 apresenta os resultados dos testes realizados com o algoritmos com o array em ordem 50% Aleatória, o resultado de tempo está em milissegundos.

Tabela 23: 50% Aleatória (tempo em ms)

Tamanho do Array	Insertion	Selection	Bublee
10000	83,2349	33,493	81,5117
50000	2097,1300	669,021	1781,8900
90000	6730,4000	2.161,110	6453,3600
130000	14155,9000	4.636,380	13996,9000
170000	20408,5000	8.009,970	22839,8000
210000	23833,7000	11.957,900	34842,5000
250000	41619,8000	18.897,200	50641,7000
290000	51795,8000	24.041,700	69439,4000
330000	63412,8000	29.003,100	87064,3000
370000	90323,5000	36.731,000	112132,0000
410000	108305,0000	46.782,300	139999,0000
450000	147413,0000	55.707,500	168463,0000
490000	178700,0000	65.936,100	198536,0000
530000	220332,0000	94.190,800	230181,0000
570000	255489,0000	104.036,000	267654,0000
610000	278879,0000	113.317,000	312648,0000
650000	337602,0000	126.524,000	352109,0000
690000	408861,0000	146.973,000	398290,0000
730000	481184,0000	164.827,000	460597,0000
770000	490971,0000	188.794,000	547589,0000
810000	438604,0000	208.127,000	626248,0000
850000	502607,0000	225.143,000	663418,0000
890000	452795,0000	239.874,000	705965,0000
930000	476458,0000	254.858,000	794918,0000
970000	552236,0000	269.000,000	889000,0000

Tabela 24: 50% Aleatória (tempo em ms)

Tamanho do Array	Shell	Quick	Merge	Radix
10000	1,14954	1,45423	1,572210	1,02896
50000	5,934330	6,79341	7,672710	4,53697
90000	5,499620	6,44315	8,124750	5,04388
130000	7,841250	8,14445	8,370980	10,5381
170000	11,817000	10,6635	12,425100	15,5484

210000	13,257400	13,7061	15,376300	18,4242
250000	16,719500	15,8521	17,788500	23,9704
290000	21,657600	18,7975	21,307900	29,7053
330000	24,266600	22,0336	23,226300	30,1615
370000	29,644300	34,7816	25,164400	40,6771
410000	31,370500	26,8391	27,963200	42,0839
450000	36,158800	30,9759	31,803300	48,1648
490000	39,766200	33,4554	33,853000	49,8581
530000	42,141600	36,1347	35,476400	56,1351
570000	45,502500	38,4495	38,734700	56,6386
610000	50,621300	39,7197	44,305500	64,1567
650000	56,111200	42,8598	45,826500	63,7088
690000	59,402200	46,3042	48,619100	68,9464
730000	63,369200	48,8535	50,914600	79,9859
770000	72,733500	51,3839	56,876500	78,7453
810000	72,300000	58,0092	56,273900	88,1493
850000	78,700000	58,3855	59,010700	87,8286
890000	77,500000	64,8364	63,354200	94,2925
930000	80,700000	62,8559	64,554700	100,436
970000	88,400000	64,7771	70,636700	109,015

3.2.6 25% Aleatória

A Tabela 25 e 26 apresenta os resultados dos testes realizados com o algoritmos com o array em ordem 25% Aleatória, o resultado de tempo está em milissegundos.

Tabela 25: 25% Aleatória (tempo em ms)

Tamanho do Array	**Insertion**	**Selection**	**Bublee**
10000	44,688	38,4499	41,433
50000	1.087,770	713,19	736,401

90000	3.515,690	2367,58	3.707,380
130000	7.416,560	5007,98	7.767,990
170000	12.614,700	8692,24	9.728,610
210000	19.225,900	13378,6	14.047,000
250000	27.512,700	20264,7	30.556,400
290000	37.160,900	25862,9	41.044,300
330000	48.283,800	32907,9	52.433,300
370000	60.906,100	39409,2	63.955,700
410000	74.446,600	51058,9	85.407,800
450000	89.390,600	61474,9	104.720,000
490000	105.529,000	72703,2	124.587,000
530000	124.057,000	87830,4	146.553,000
570000	143.506,000	98085,9	165.060,000
610000	164.648,000	114881	189.147,000
650000	209.307,000	131305	223.827,000
690000	233.766,000	146701	260.618,000
730000	261.210,000	167859	295.028,000
770000	280.444,000	185998	315.196,000
810000	310.181,000	203345	353.801,000
850000	339.117,000	225362	372.166,000
890000	379.192,000	247785	432.554,000
930000	417.594,000	272509	457.822,000
970000	452.000,000	2,960E+05	494.771,000

Tabela 26: 25% Aleatória (tempo em ms)

Tamanho do Array	**Shell**	**Quick**	**Merge**	**Radix**
10000	0,641407	1,3815	0,999526	1,2753
50000	4,07795	5,7543	4,7733	5,76415
90000	9,95476	5,2858	7,64743	7,11213
130000	10,2829	8,8323	6,46336	8,43051
170000	8,17809	10,8639	8,74087	13,1388
210000	8,67102	13,0859	11,5736	20,1131
250000	11,5839	27,1547	13,645	25,2876
290000	15,8813	24,6716	15,4634	29,1213

330000	17,7484	22,3685	17,2189	35,4637
370000	19,2139	22,8828	20,9858	37,3114
410000	24,0502	26,2544	21,1328	44,379
450000	26,9892	27,0561	23,1439	47,5019
490000	30,1645	30,1636	24,8705	50,3318
530000	34,0697	32,3819	26,9197	64,1493
570000	32,4792	35,8989	29,6399	69,2543
610000	37,9639	37,4928	32,8406	69,4745
650000	42,3954	40,2120	33,3826	71,7552
690000	44,02	45,3899	36,9185	77,3572
730000	47,2162	49,4176	39,4938	84,8757
770000	57,7582	51,1772	40,4992	86,798
810000	56,5178	59,6217	44,1344	93,0883
850000	57,7914	57,2204	44,3991	94,1368
890000	59,3907	56,0862	47,865	100,723
930000	65,3049	61,8559	49,6476	107,305
970000	68,5997	68,7971	52,5072	109,578

3.3 Gráficos - por algoritmo

3.3.1 Insertion Sort

Figura 1: Gráfico de barras

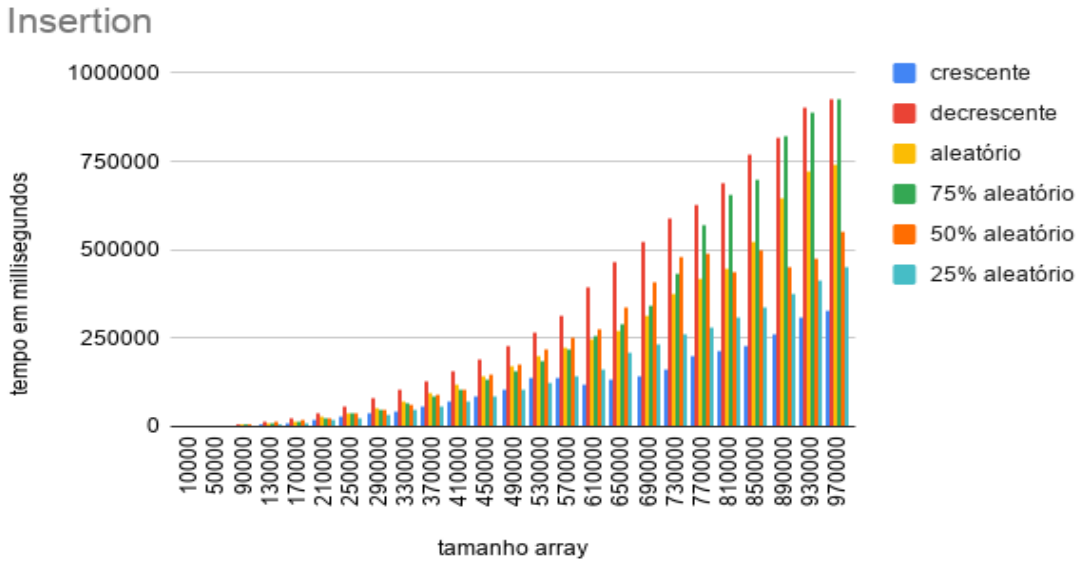
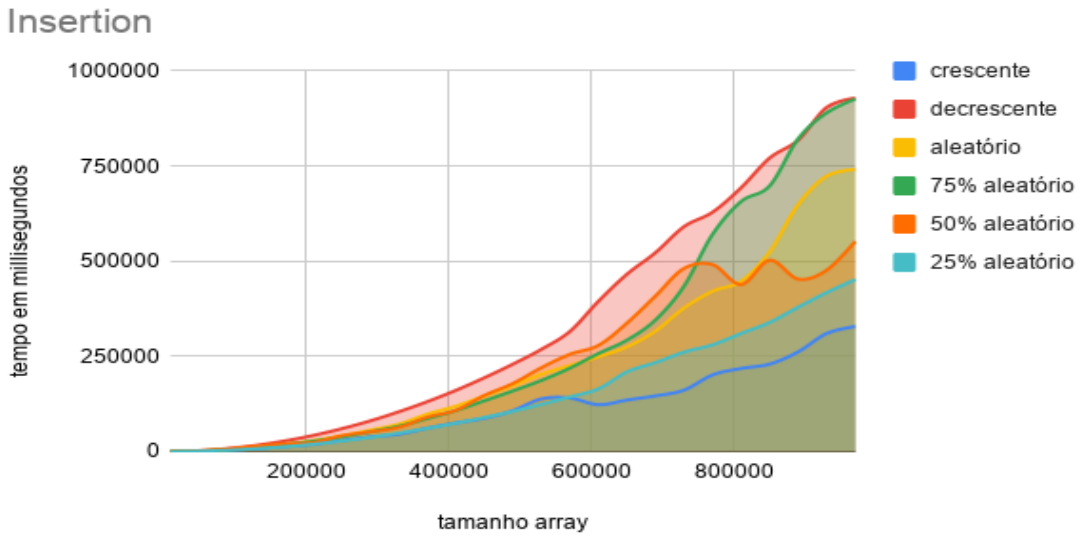


Figura 2: Gráfico de linhas



3.3.2 Selection Sort

Análise de tempo para o algoritmo de ordenação Selection Sort

Figura 3: Gráfico de barras

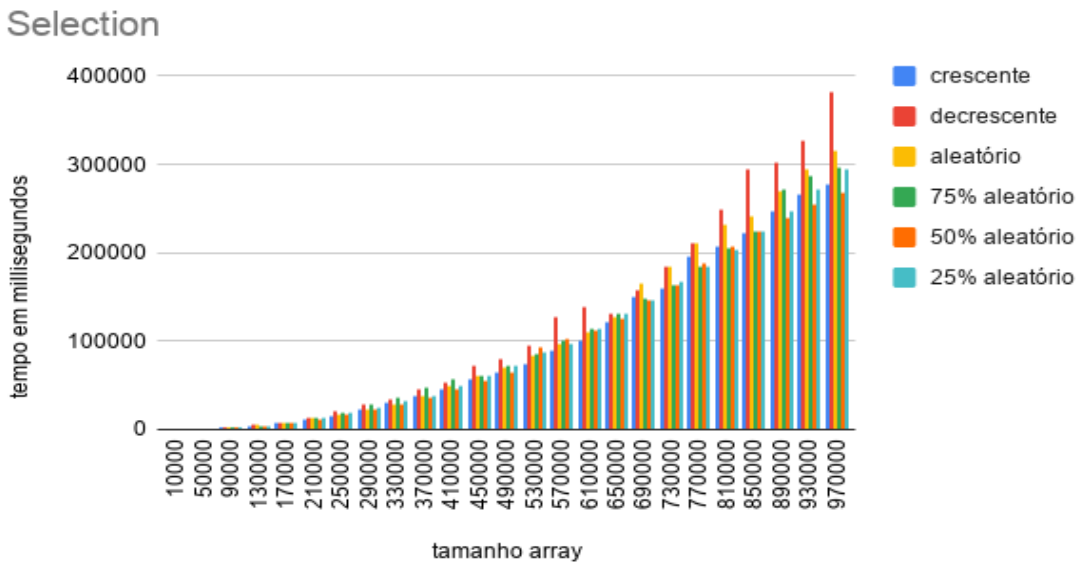
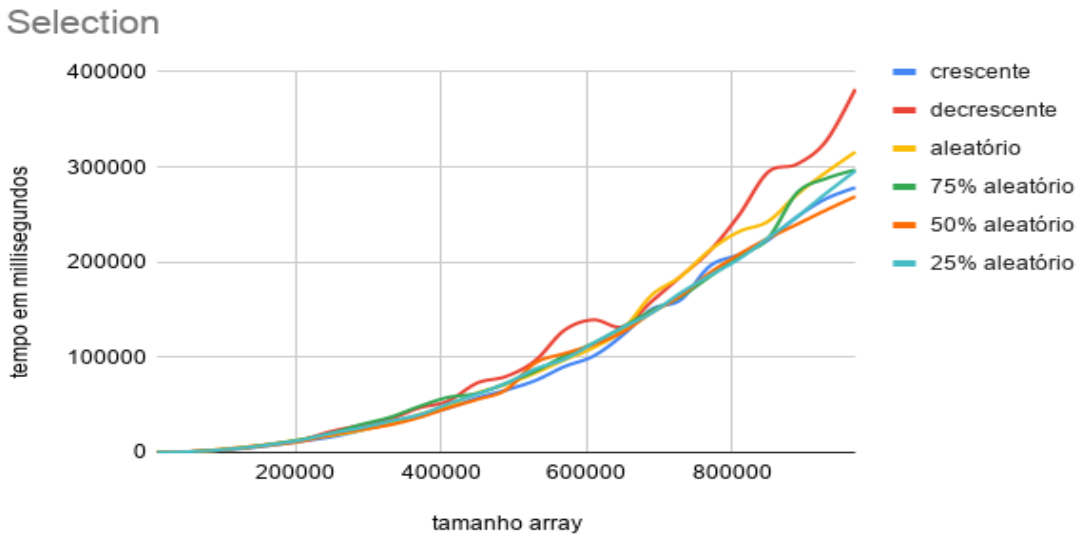


Figura 4: Gráfico de linhas



3.3.3 Bubble Sort

Análise de tempo para o algoritmo de ordenação Bubble Sort

Figura 5: Gráfico de barras

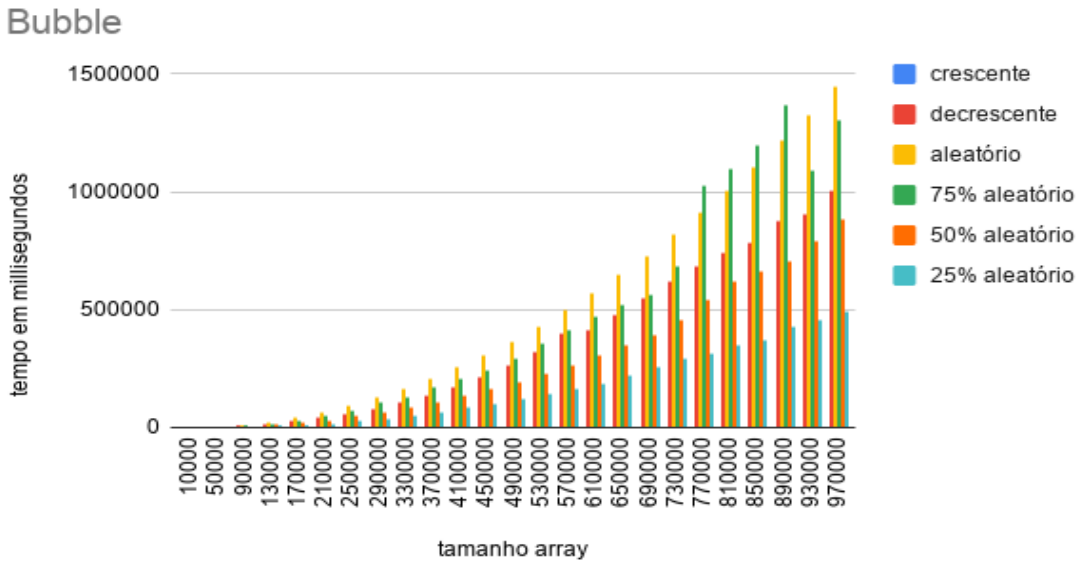
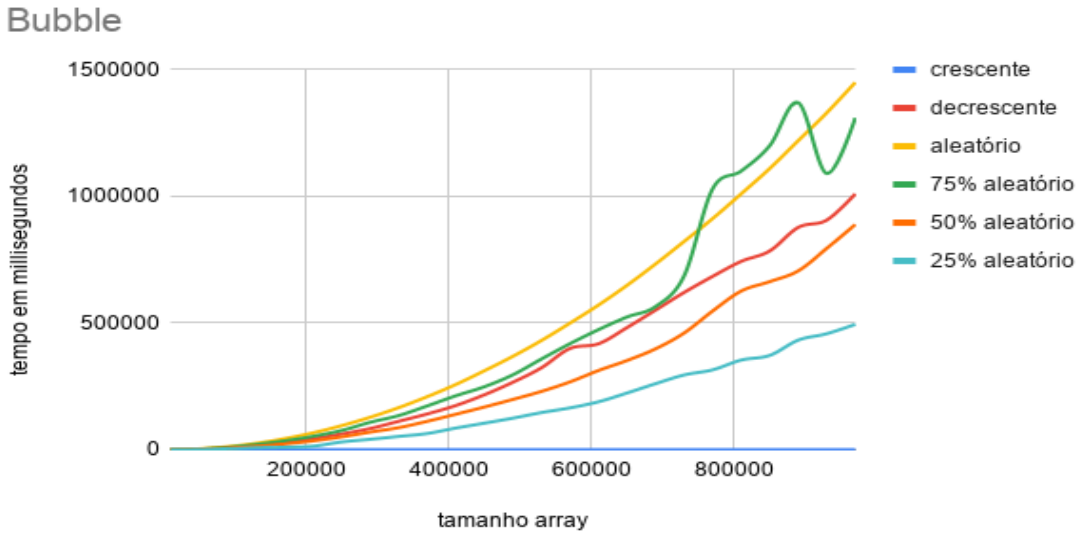


Figura 6: Gráfico de linhas



3.3.4 Shell Sort

Análise de tempo para o algoritmo de ordenação Shell Sort

Figura 7: Gráfico de barras

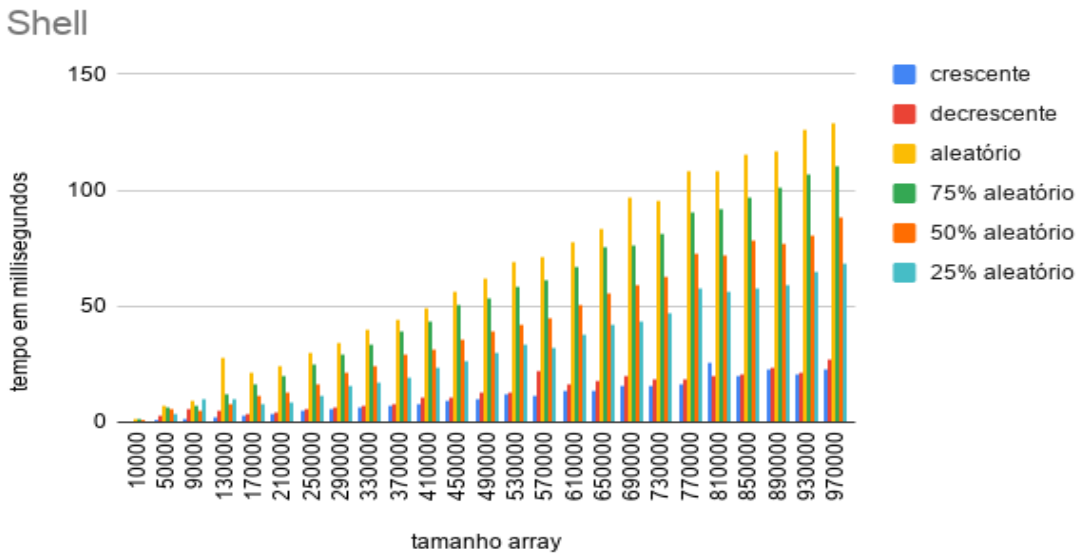
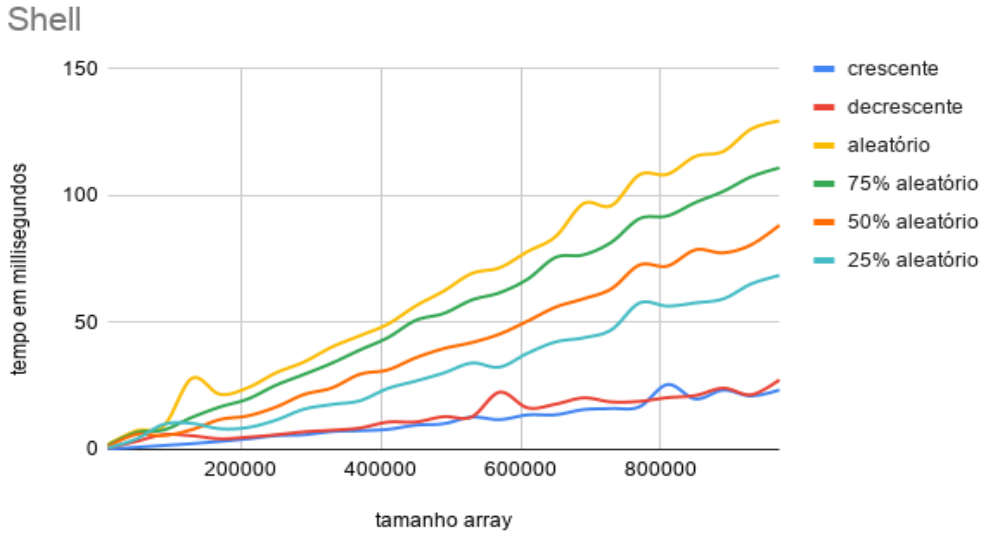


Figura 8: Gráfico de linhas



3.3.5 Quick Sort

Análise de tempo para o algoritmo de ordenação Quick Sort

Figura 9: Gráfico de barras

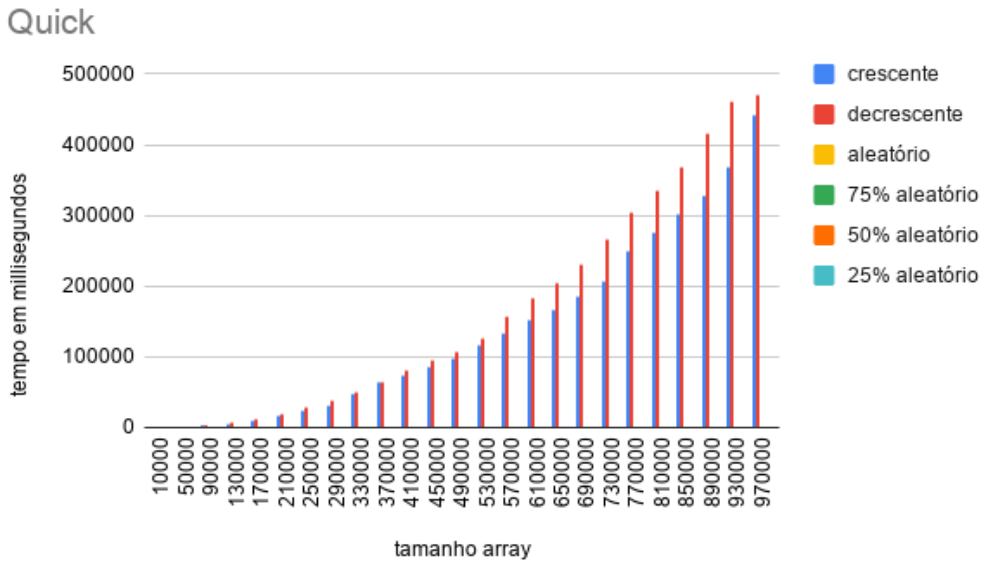
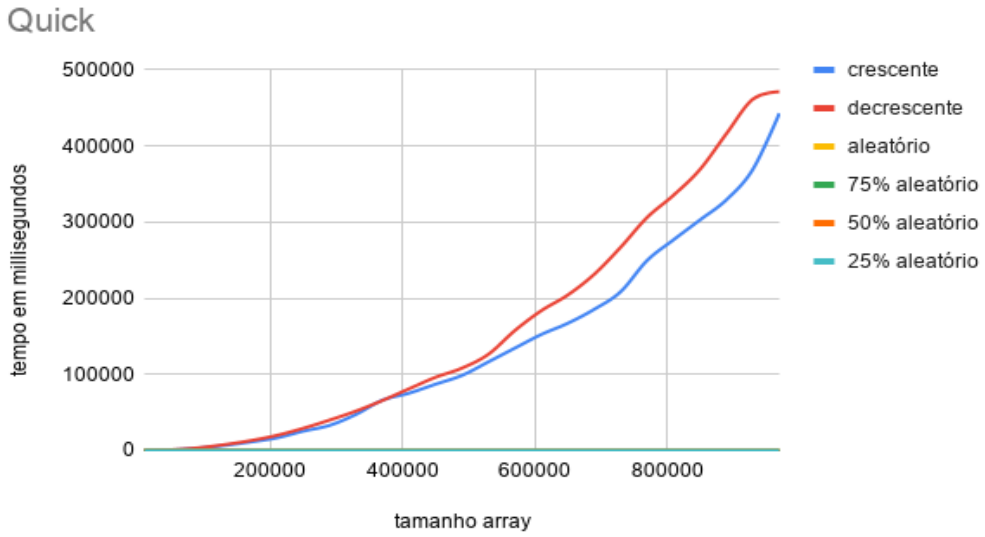


Figura 10: Gráfico de linhas



3.3.6 Merge Sort

Análise de tempo para o algoritmo de ordenação Merge Sort

Figura 11: Gráfico de barras

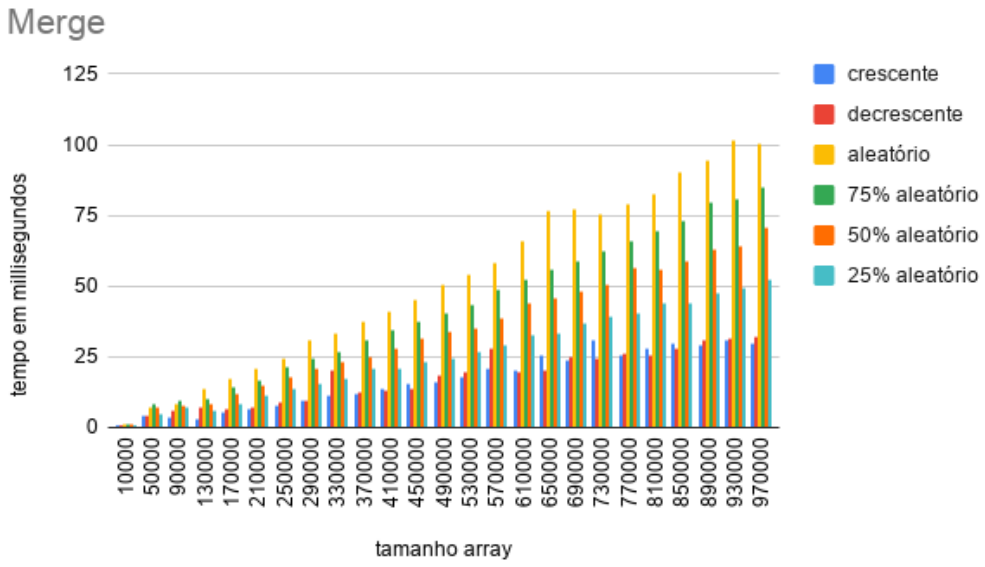
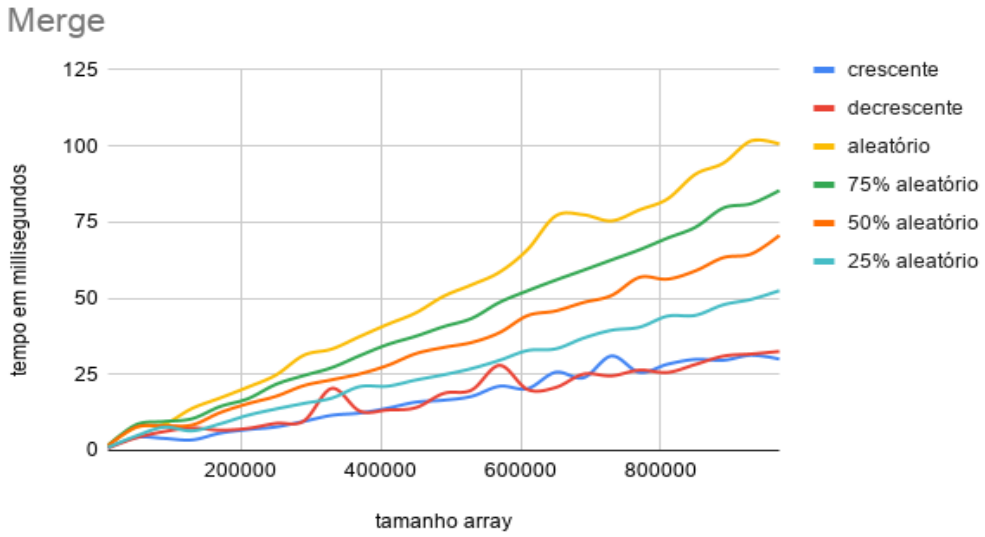


Figura 12: Gráfico de linhas



3.3.7 Radix Sort

Análise de tempo para o algoritmo de ordenação Radix Sort

Figura 13: Gráfico de barras

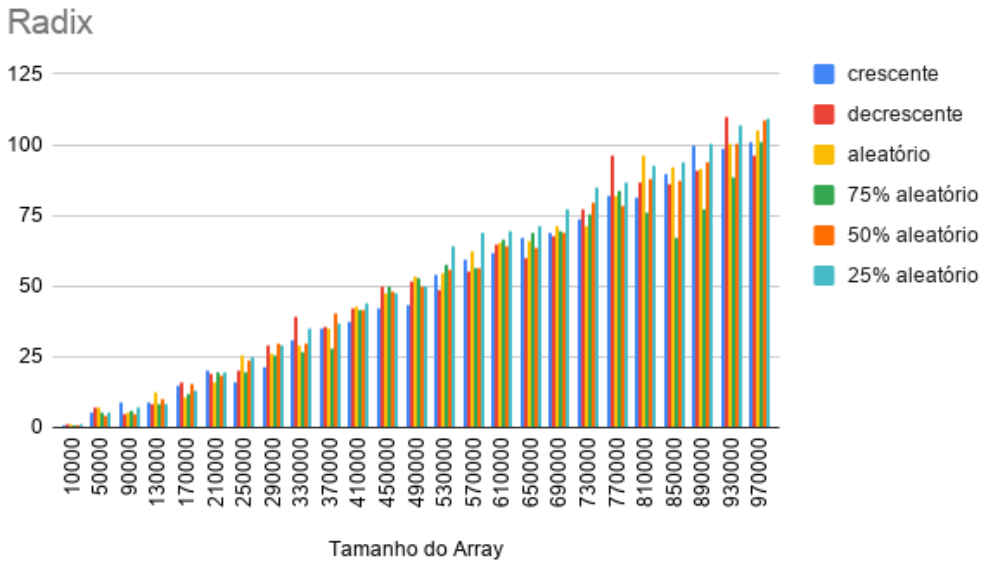
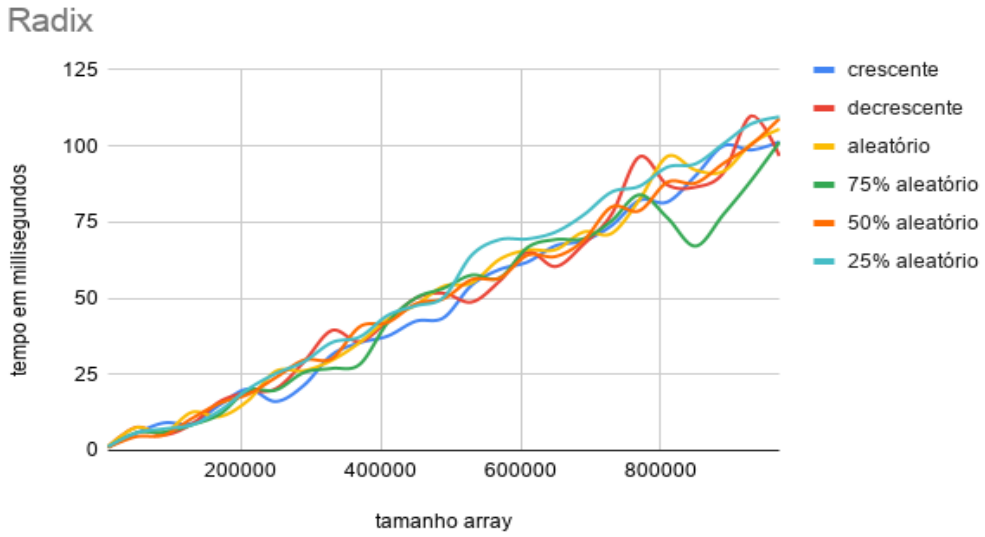


Figura 14: Gráfico de linhas



3.4 Gráficos - por tipo de amostra

3.4.1 Crescente

Figura 15: Gráfico de linhas

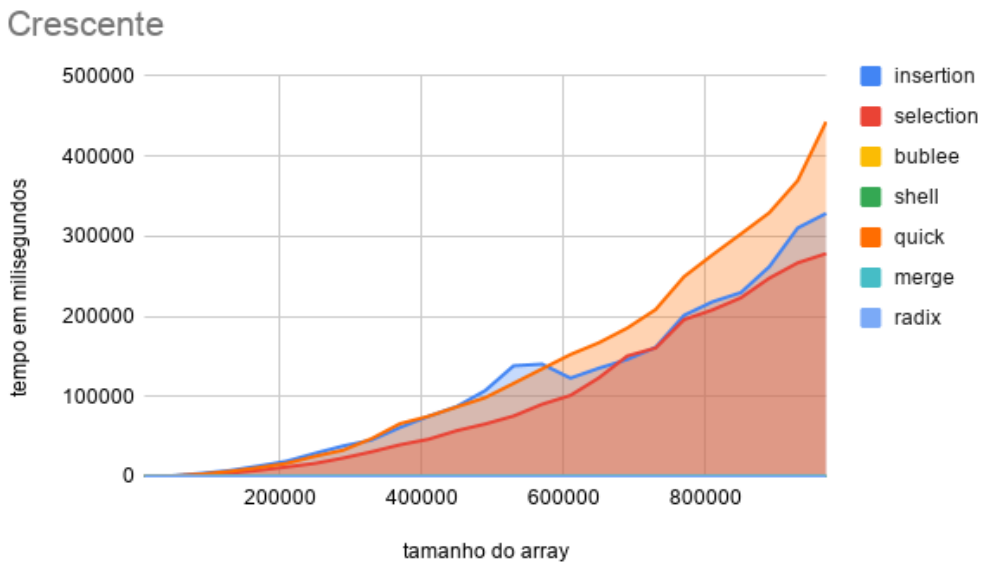


Figura 16: Gráfico de linhas

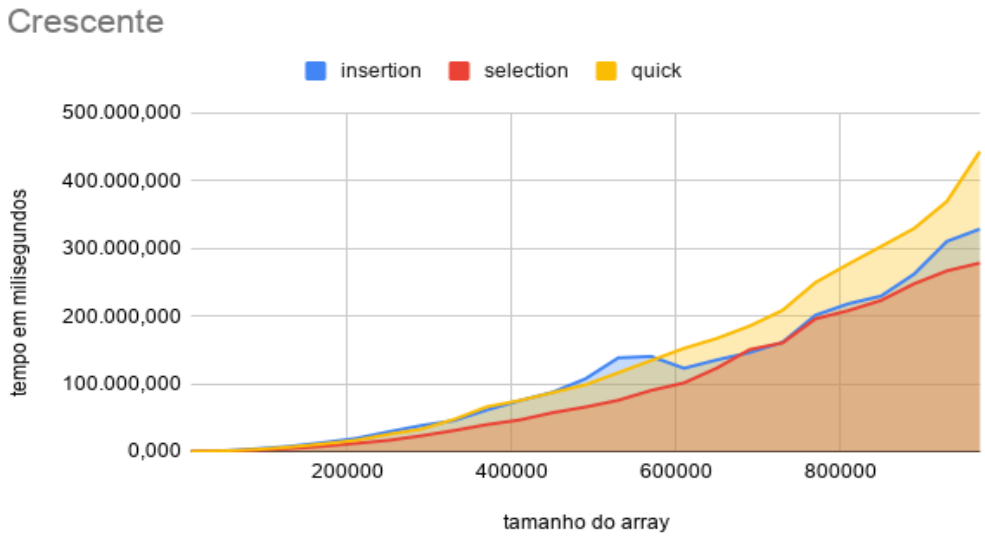
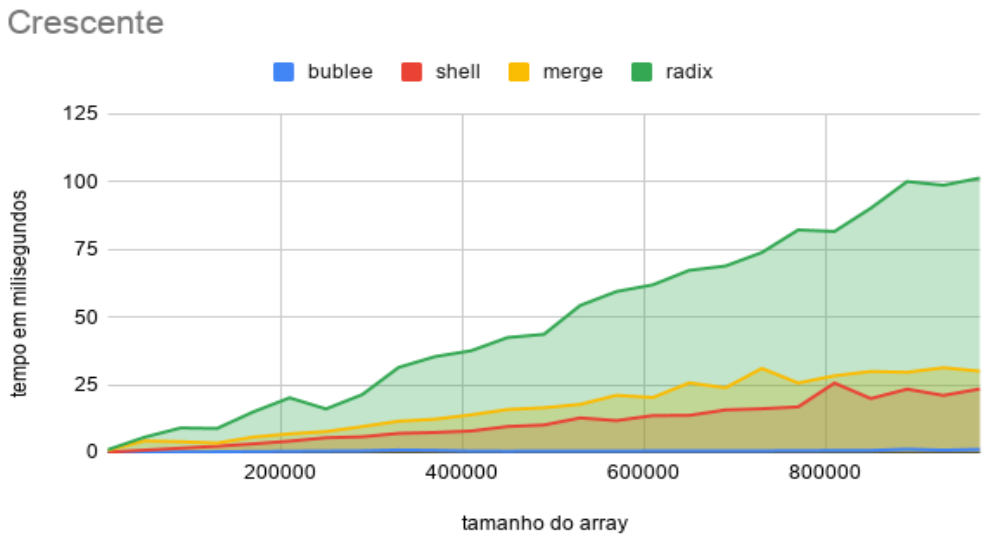


Figura 17: Gráfico de linhas



3.4.2 Decrescente

Figura 18: Gráfico de linhas

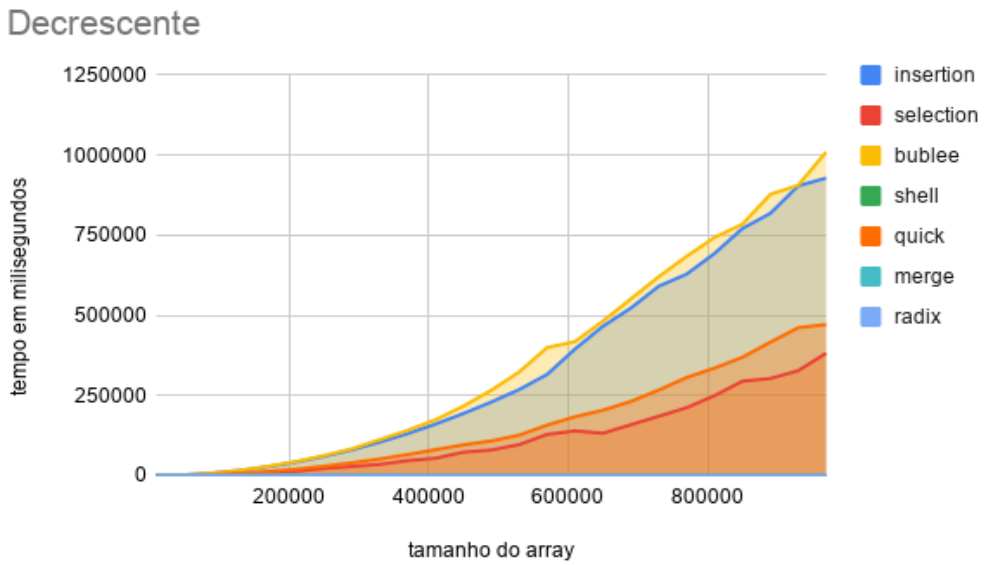


Figura 19: Gráfico de linhas

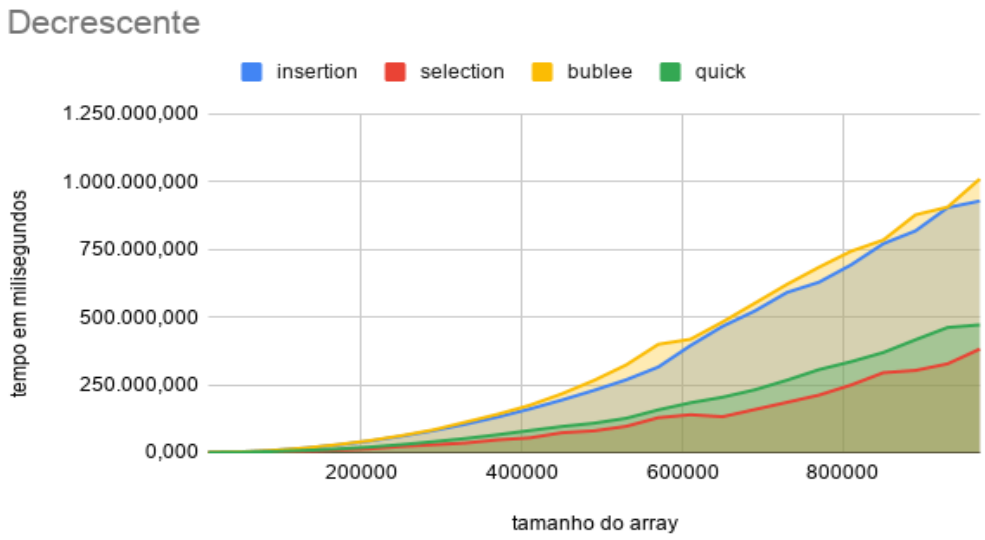
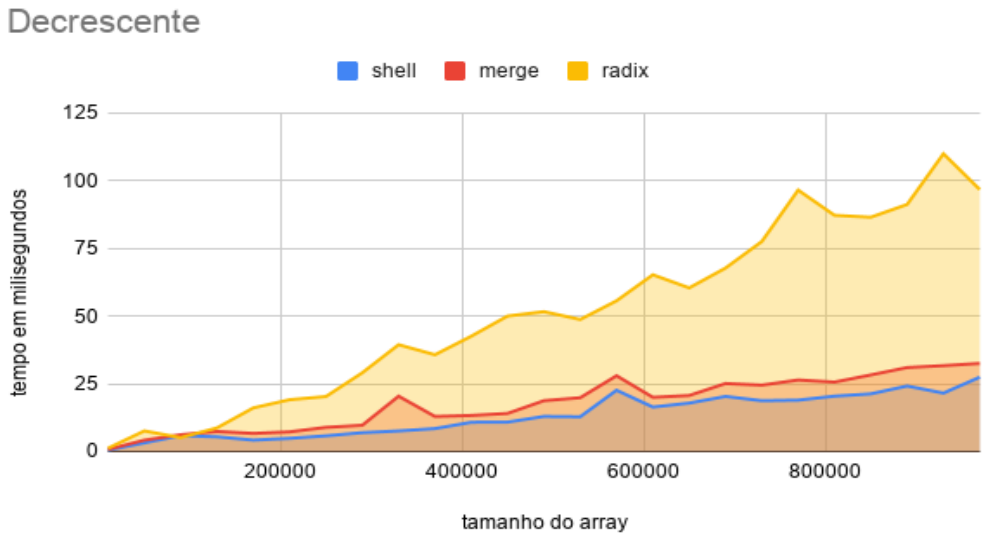


Figura 20: Gráfico de linhas



3.4.3 100% Aleatória

Figura 21: Gráfico de linhas

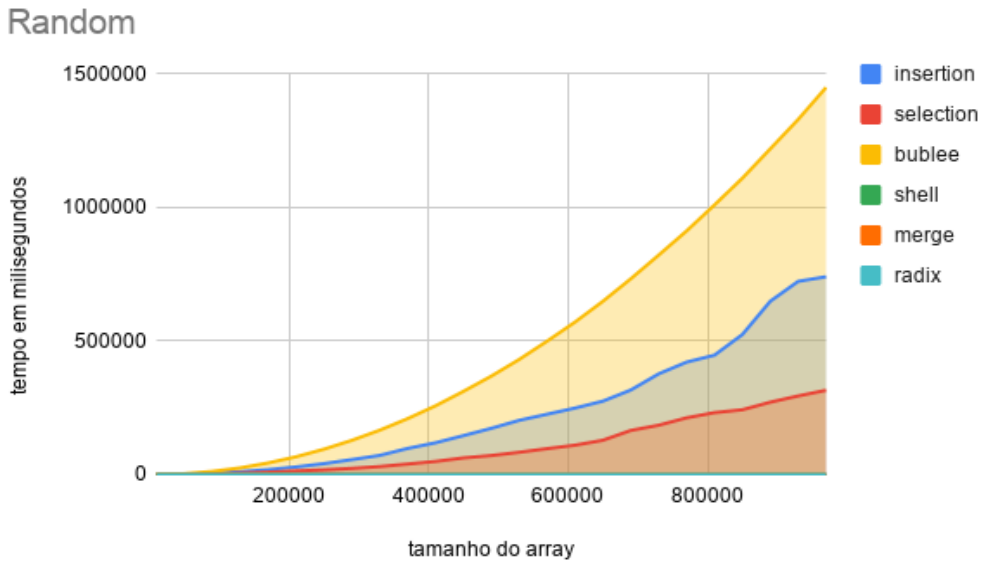


Figura 22: Gráfico de linhas

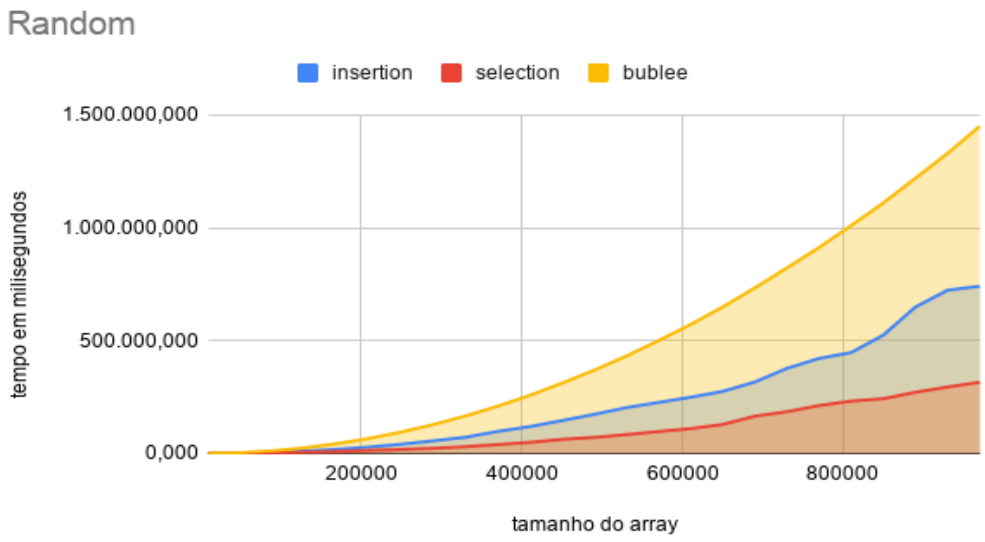
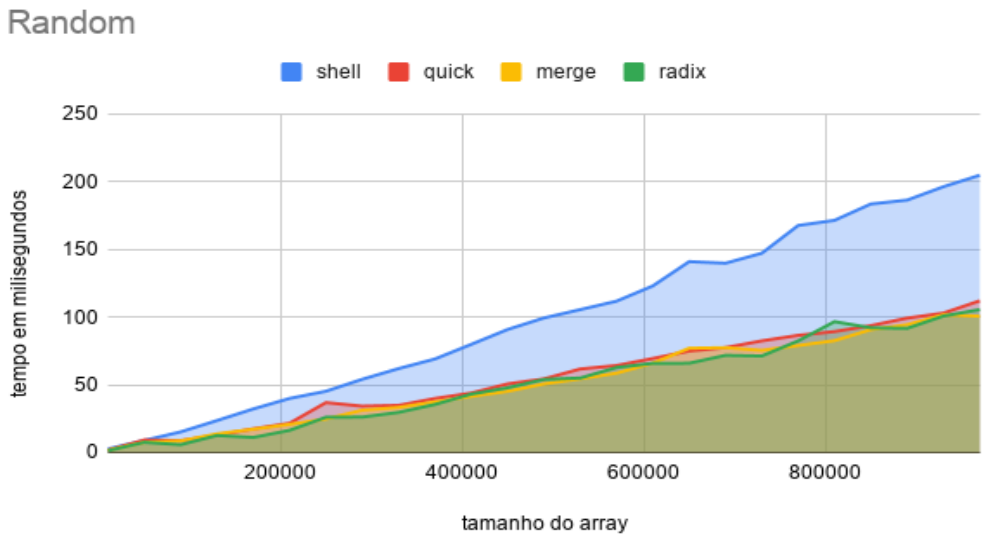


Figura 23: Gráfico de linhas



3.4.4 75% Aleatória

Figura 24: Gráfico de linhas

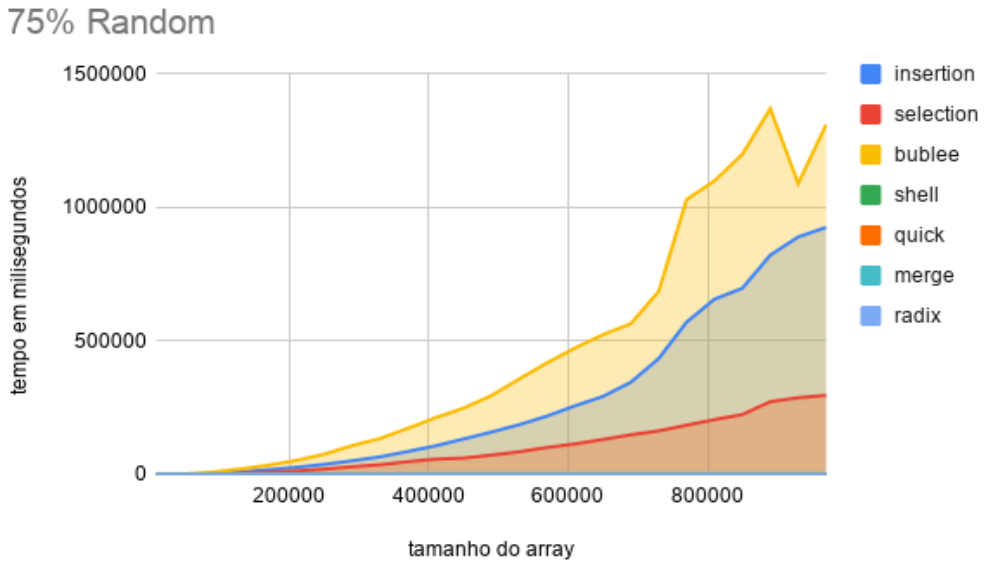


Figura 25: Gráfico de linhas

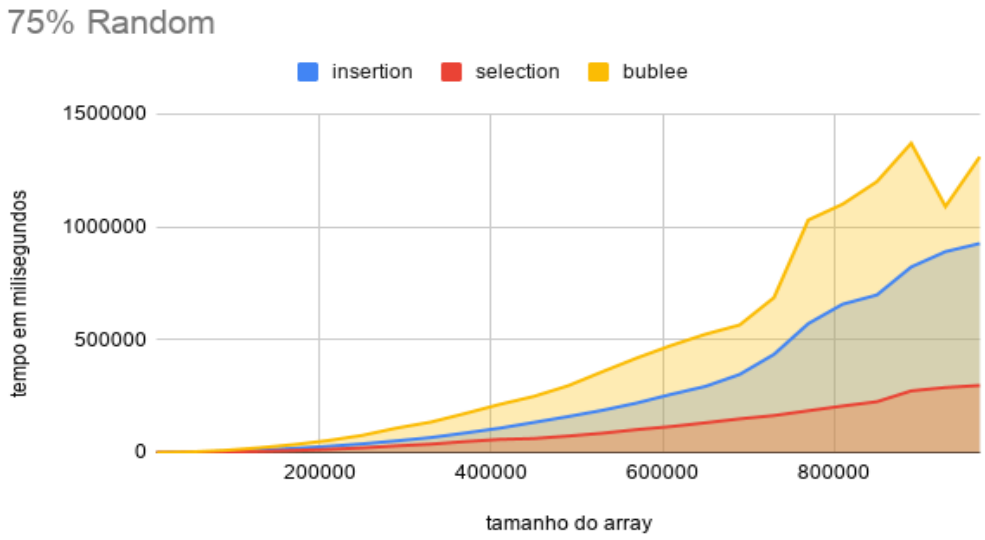
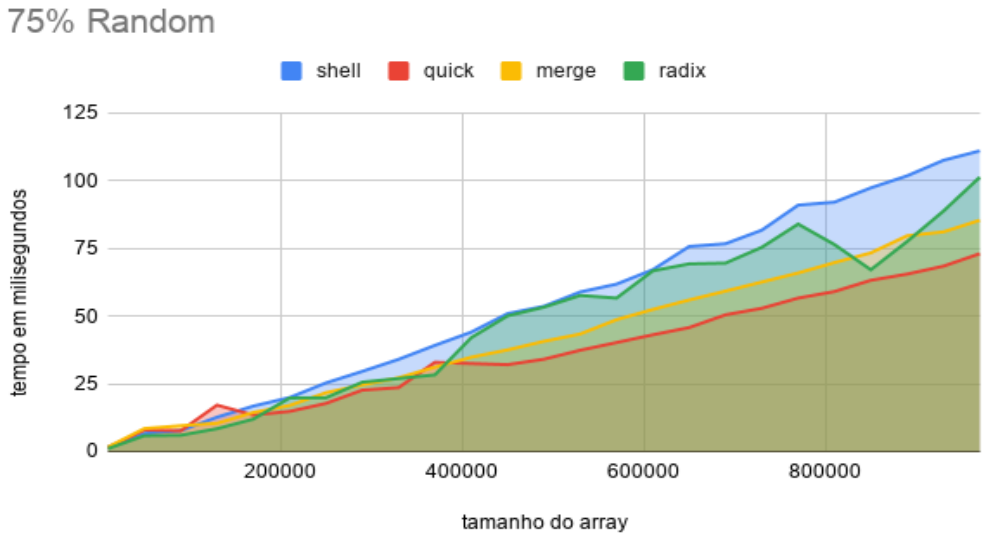


Figura 26: Gráfico de linhas



3.4.5 50% Aleatória

Figura 27: Gráfico de linhas

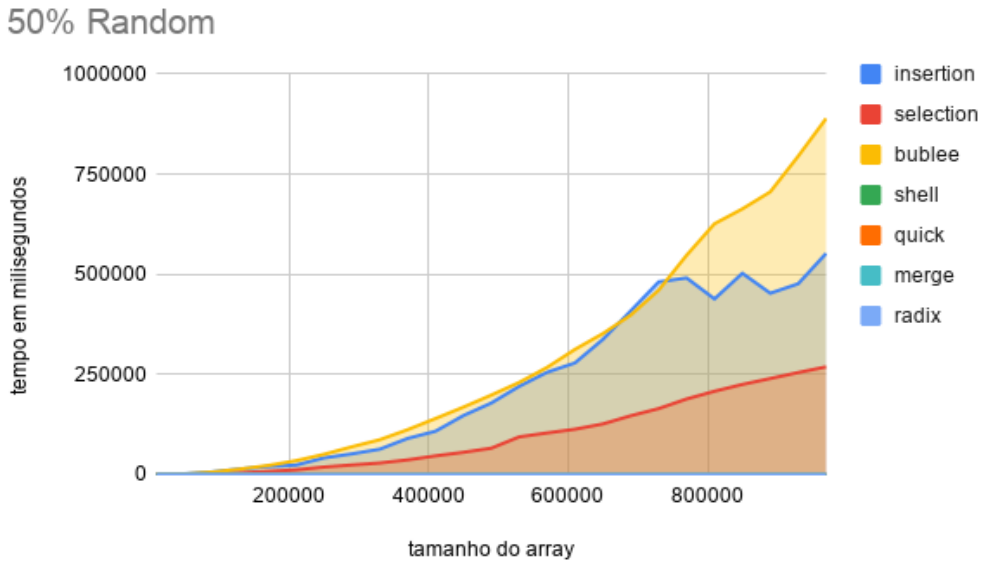


Figura 28: Gráfico de linhas

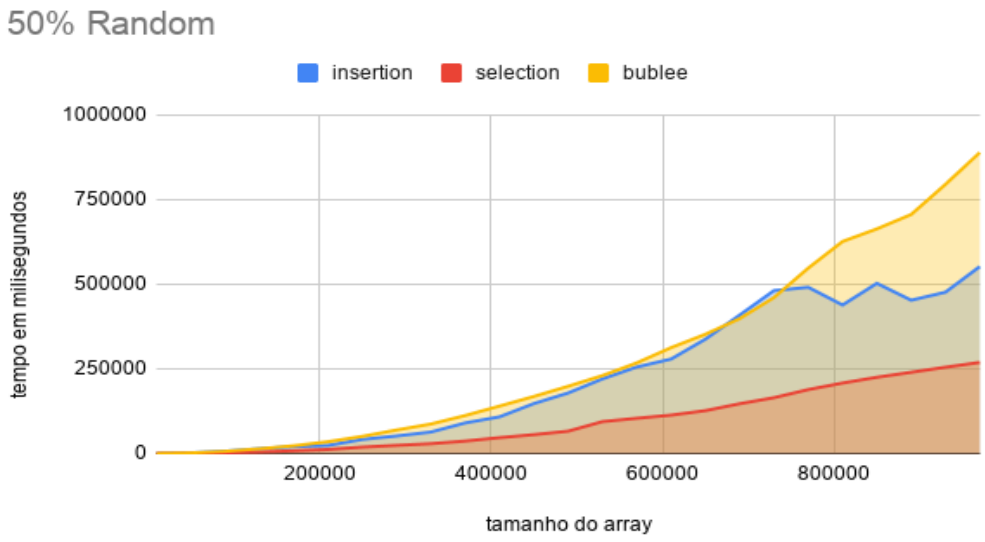
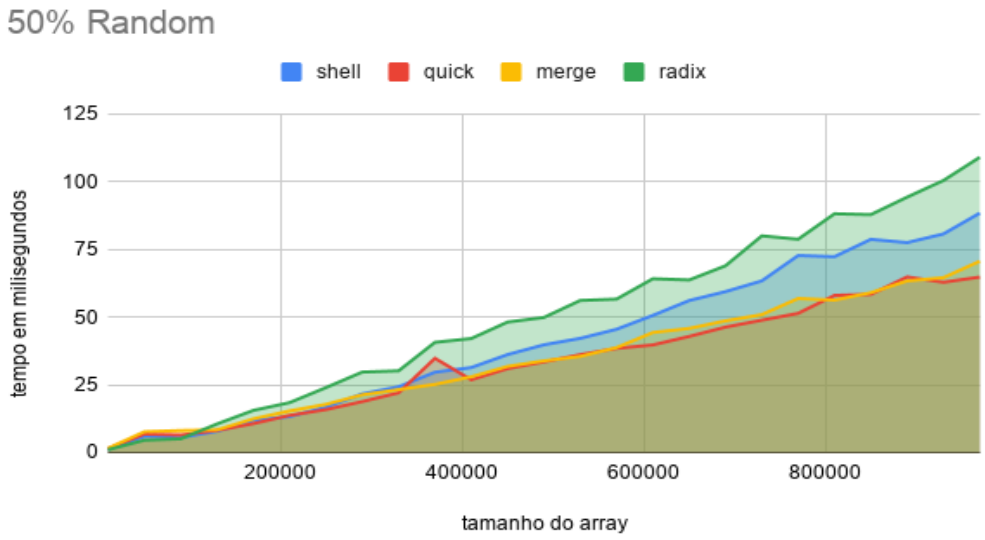


Figura 29: Gráfico de linhas



3.4.6 25% Aleatória

Figura 30: Gráfico de linhas

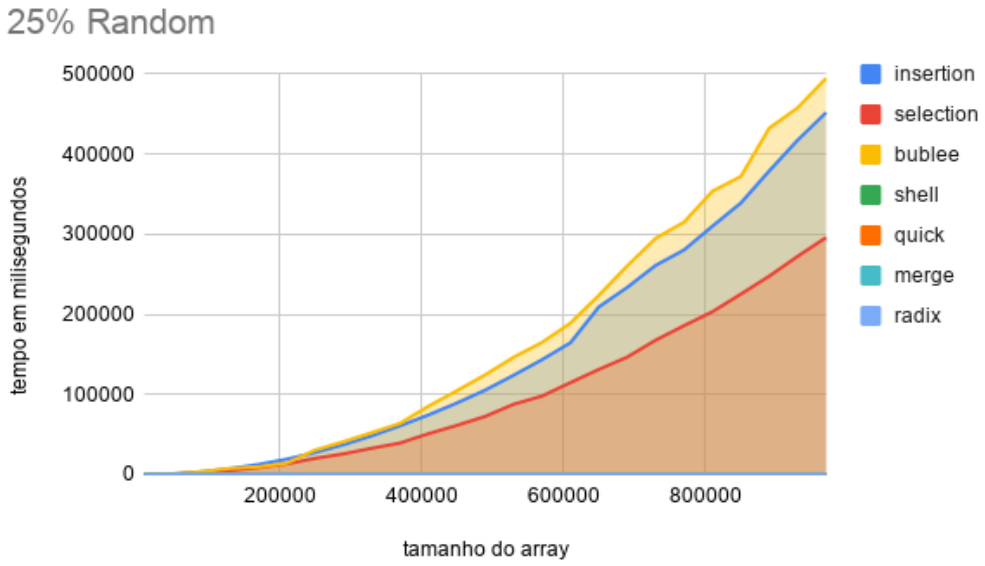


Figura 31: Gráfico de linhas

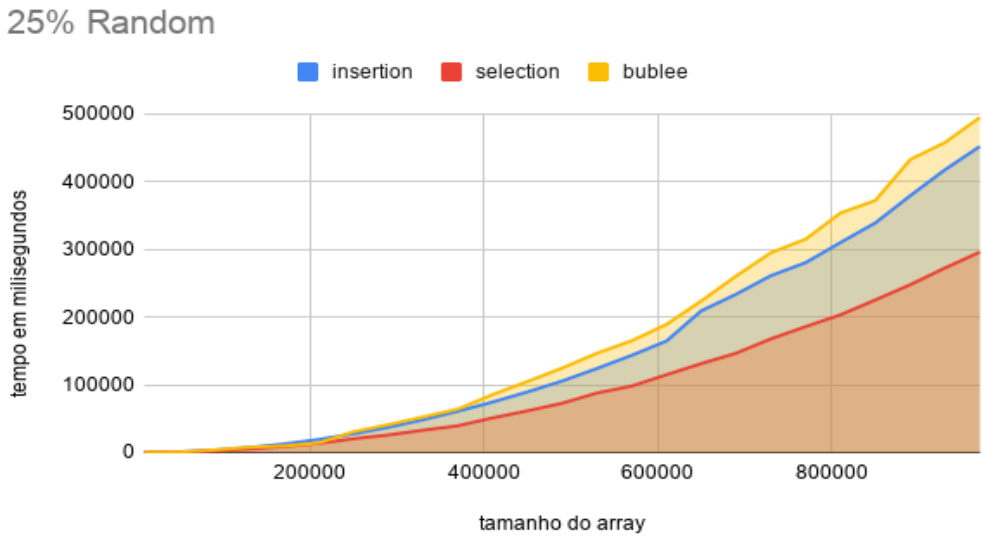
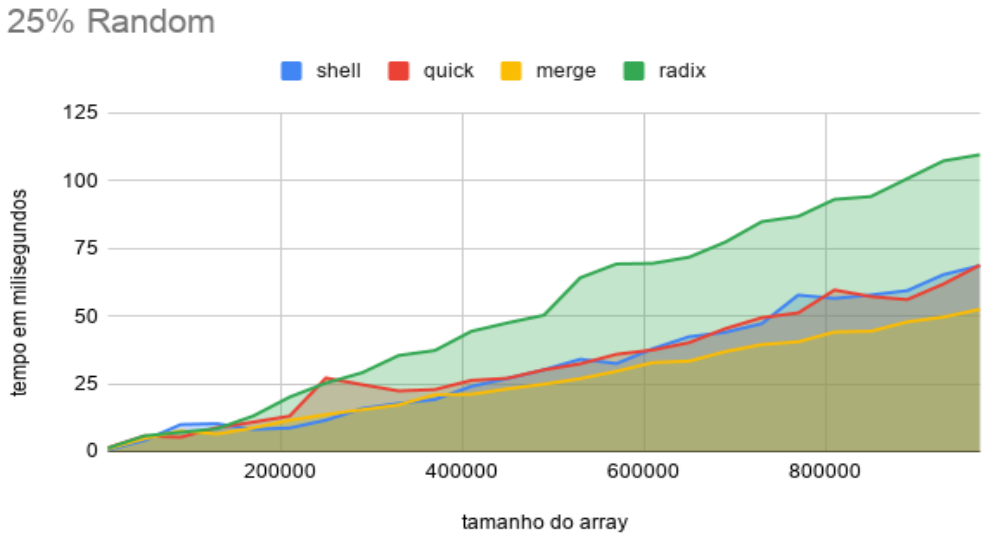


Figura 32: Gráfico de linhas



4 Discussão

4.1 Geral

Para amostras crescentes, o insertion, selection e quick tem algorítimos ineficientes quando comparados com shell, merge, radix, e bubble sendo este ultimo claramente o mais eficiente para este caso. Já para amostras decrescentes, o insertion, bubble, selection e quick tem algorítimos ineficientes quando comparados com shell, merge e radix, como podemos ver o bubble passa de mais eficiente no caso de arrays em ordem crescente para estar entre os piores em ordem decrescente.

Para amostras em ordem aleatória, sendo totalmente aleatórias ou parcialmente, o insertion, selection e bubble tem algorítimos ineficientes quando comparados com shell, merge, radix, e quick .

4.2 Quais algoritmos para qual cenários?

1. elementos em ordem não decrescente - bubble
2. elemento em ordem não crescente - shell
3. elementos 100% aleatórios - radix
4. 75% de seus elementos em sua posição definitiva - quick
5. 25% de seus elementos em sua posição definitiva - quick
6. 50% de seus elementos em sua posição definitiva - merge

4.3 Radix vs. Outros

O radix tem performance parecida em todos os tipos de amostra, ao contrario dos outros, com performances tão boas quanto, que variam dependendo da amostra.

4.4 Quick sort vs. Merge sort

Nas amostras crescente e decrescente o Merge sorte tem melhor performance. Já nas amostras aleatórias, pelos gráficos 23, 26, 29, 32, é possível ver que quanto mais aleatório é a amostra a performance do Merge se torna ligeiramente pior que a do Quick sort.

4.5 Picos e Vales

Sim, provavelmente por causa de sobrecargas no sistema por estar executando tarefas paralelas.

4.6 Análise empírica vs. Análise matemática

Ambas são compatíveis.

Bibliografia

https://cmake.org/

Apêndice - Implementação dos algoritmos em CPP

4.7 Função auxiliar para troca de posição

```
  void swap(int * p1, int * p2) {
        int temp = * p1;
        * p1 = * p2;
        * p2 = temp;
}
```

4.8 Insertion Sort

```
  void insertionsort(value_type * array, int size) {
        for (int i = 1; i < size; i++) {
                for (int j = i; j > 0; j--) {
                        if (array[j - 1] > array[j]) {
                                swap( & array[j], & array[j - 1]);
                        }
                }
        }
}
```

4.9 Selection Sort

```
  void selectionsort(value_type * array, int size) {
        int i, j, min_value;
        for (i = 0; i < size - 1; i++) {
                min_value = i;
                for (j = i + 1; j < size; j++)
                if (array[j] < array[min_value])
                min_value = j;
                swap( & array[min_value], & array[i]);
        }
}
```

4.10 Bubble Sort

```
void bubblesort(value_type * array, int size) {
        bool changed = false;
        int ordained = 0;
        do {
                changed = false;
                ordained++;
                for (int i = 0; i < (size - ordained); i++) {
                        if (array[i + 1] < array[i]) {
                                changed = true;
```

```
                                swap( & array[i], & array[i + 1]);
                        }
                }
        } while (changed);
}
```

4.11 Shell Sort

```
void shellsort(value_type * array, int size) {
        for (int gap = size / 2; gap > 0; gap /= 2) {
          for (int i = gap; i < size; i += 1) {
                int temp = array[i];
                int j;
                for (j = i; j >= gap && array[j - gap] > temp; j -= gap)
                  array[j] = array[j - gap];
                array[j] = temp;
            }
        }
}
```

4.12 Quick Sort

4.12.1 Passa do formato padrão (array, size) para o do Quick

```
  void quicksort(value_type * array, int size) {
        quicksort(array, 0, size - 1);
}
```

4.12.2 Particiona

```
int partition(value_type * array, int low, int high) {
        int pivot = array[high];
        int i = (low - 1);
        for (int j = low; j <= high - 1; j++) {
                if (array[j] < pivot) {
                        i++;
                        swap( & array[i], & array[j]);
                }
        }
        swap( & array[i + 1], & array[high]);
        return (i + 1);
}
```

4.12.3 Pricipal

```
  void quicksort(value_type * array, int l, int h) {
        int stack[h - l + 1];
        int top = -1;
```

```
        stack[++top] = l;
        stack[++top] = h;

        while (top >= 0) {
                h = stack[top--];
                l = stack[top--];

                int p = partition(array, l, h);
                if (p - 1 > l) {
                        stack[++top] = l;
                        stack[++top] = p - 1;
                }
                if (p + 1 < h) {
                        stack[++top] = p + 1;
                        stack[++top] = h;
                }
        }
}
```

4.13 Merge Sort

4.13.1 Passa do formato padrão (array, size) para o do merge

```
        void mergesort(value_type * array, int size) {
                mergesort(array, 0, size - 1);
        }
```

4.13.2 Pricipal que divide em 2 subarrays

```
        void mergesort(value_type * array, int l, int r) {
                if (l < r) {
                        int m = l + (r - l) / 2;
                        mergesort(array, l, m);
                        mergesort(array, m + 1, r);
                        merge(array, l, m, r);
                }
        }
```

4.13.3 Função que mistura (merge) os 2 subarrays

```
        void merge(value_type * array, int l, int m, int r) {
                int i, j, k;
                int n1 = m - l + 1;
                int n2 = r - m;
                int left[n1], rigth[n2];

                for (i = 0; i < n1; i++)
                left[i] = array[l + i];
                for (j = 0; j < n2; j++)
                rigth[j] = array[m + 1 + j];
```

```
                i = 0;
                j = 0;
                k = l;
                while (i < n1 && j < n2) {
                        if (left[i] <= rigth[j]) {
                                array[k] = left[i];
                                i++;
                        } else {
                                array[k] = rigth[j];
                                j++;
                        }
                        k++;
                }

                while (i < n1) {
                        array[k] = left[i];
                        i++;
                        k++;
                }
                while (j < n2) {
                        array[k] = rigth[j];
                        j++;
                        k++;
                }
        }

}
```

4.14 Radix Sort

```
  void radixsort(value_type * array, int size) {
        int i;
        value_type * b = new value_type[size];
        value_type maior = array[0];
        int exp = 1;
        for (i = 0; i < size; i++) {
                if (array[i] > maior)
                maior = array[i];
        }
        while (maior / exp > 0) {
                int count[10] = {
                        0
                };
                for (i = 0; i < size; i++)
                count[(array[i] / exp) % 10]++;
                for (i = 1; i < 10; i++)
                count[i] += count[i - 1];
                for (i = size - 1; i >= 0; i--)
```

```
                b[--count[(array[i] / exp) % 10]] = array[i];
                for (i = 0; i < size; i++)
                array[i] = b[i];
                exp *= 10;
        }
        delete[] b;
}
```

4.15 Codifo Gerar Array

4.15.1 Na ordem crescente

```
        for( auto i{0ull} ; i < tamanho_array ; ++i ){
                crescente[i]  = i;
        }
```

4.15.2 Na ordem decrescente

```
        for( auto i{0ull} ; i < tamanho_array ; ++i ){
                decrescente[i]  = tamanho_array - i;
        }
```

4.15.3 Aleatória

```
        for( auto i{0ull} ; i < tamanho_array ; ++i ){
                aleatorio[i]  = rand() % tamanho_array;
        }
```

4.15.4 50% aleatória

```
    for( auto i{0ull} ; i < tamanho_array ; ++i ){
          aleatorio50[i] =  (i%2==0)  ? i :  rand() % tamanho_array;
    }
```

4.15.5 75% aleatória

```
    for( auto i{0ull} ; i < tamanho_array ; ++i ){
                aleatorio75[i] =  (i%4==0)  ? i :  rand() % tamanho_array;
        }
```

4.15.6 25% aleatória

```
    for( auto i{0ull} ; i < tamanho_array ; ++i ){
                aleatorio25[i] =  (i%4!=0)  ? i :  rand() % tamanho_array;
        }
```