Introdução

Os algoritmos de ordenação são um conjunto de instruções que tem como função rearranjar os componentes de uma lista ou um array para melhorar a apresentação delas, melhorar o desempenho de algoritmo, reduzir a complexidade de algum problema computacional, entre outras coisas que eles podem fazer. Existem quatro tipos de algoritmos de ordenação, os de ordenação por troca (Bubble Sort e Quick Sort), ordenação por Seleção (Selection Sort e HeapSort), ordenação por inserção (Insertion Sort e Shell Sort) e ordenação por intercalação (Merge Sort). A seguir há uma pequena explicação de como funciona cada algoritmo de ordenação:

- Selection Sort: Esse algoritmo pega o menor número e leva para a primeira posição e depois o segundo menor e leva para a segunda posição e assim sucessivamente.
- Insertion Sort: Neste uma lista de elementos é percorrida e os elementos são inseridos na posição em relação aos elementos já posicionados anteriormente na lista.
- Bubble Sort: Compara pares de elementos de uma lista ou array e os troca se estiverem fora de ordem e assim vai ocorrendo está troca até todos os elementos da lista serem comparados e estiver ordenados em seus lugares.
- Quicksort: É baseado em dividir para conquistar, primeiro é escolhido um elemento da lista denominado (pivot) e depois sua lista é dividida em duas, sendo uma com elementos menores que o elemento escolhido (pivot) e outra com os maiores e assim se ordena a lista e por fim ela se junta novamente.
- Heapsort: Organiza os elementos em uma heap ou árvore heap(como se fosse uma pliha), em seguida ele pega o maior número e coloca-o no início da lista e assim ele repete o processo até que todos estejam ordenados:
- Merge Sort: Divide uma lista em duas partes iguais e ordena cada metade para depois agrupa-las e estarem ordenadas.
- Shell Sort: É uma evolução do Insertion Sort, ele utiliza-se do dividir para conquistar, dividindo sua lista em pequenos subgrupos e nesses subgrupos ocorre o processo de ordenação do Insertion Sort, onde os elementos são posicionados de acordo com os que já estão posicionados.

Esses são alguns dos vários algoritmos de ordenação que existem, mas esses são os mais conhecidos e utilizados.

Neste projeto será utilizado o Bubble Sort, Insertion Sort e Selection Sort para verificar seus comportamentos em diversas situações com bases de dados de mil a quinhentas mil linhas.

Desenvolvimento

Sore a ferramenta

A ferramenta utilizada para a realização dos testes é baseada na linguagem C, tendo construção simples, em console, para manter a leveza da aplicação e o foco apenas nos resultados. Conta com lógicas de cada um dos algoritmos testados separados de seus programas principais para facilitação da manutenção e adaptação a novas bases de dados; e programa para construção do boletim de resultados.

A mensura do tempo se dá com a implementação da biblioteca time.h, tendo sua função clock() chamada para preencher uma variável (início) do tipo clock_t antes da chamada do método qual consiste na execução do algoritmo de ordenação. Após chamarmos o método de ordenação, mais uma vez passamos pelo processo de chamar a função clock() e guardar seu retorno numa variável (fim). O próximo passo consiste na subtração da variável fim pela variável início, obtendo os clocks de CPU necessários para a execução da ordenação. Para convertermos este resultado em segundos, basta dividirmos pela função CLOCKS_PER_SECOND ou multiplicarmos o mesmo por 0.001.

Sua construção modular conta com os programas principais, ResultInsertSort.c, ResultBubbleSort.c e ResultSelectionSort.c, responsáveis pelas junções de todas as bibliotecas utilizadas, declaração dos métodos e suas execuções.

Desenvolvidos como bibliotecas, também conhecidos como headers (arquivos .h), temos os programas InsertionSort.h, BubbleSort.h e SelectionSort.h, responsáveis pela implementação das lógicas dos algoritmos de ordenação de mesmo nome, bem como abertura dos arquivos que serviram como base de dados e gravação de novos arquivos, desta vez com os dados já ordenados; Report.h, responsável pela construção de um boletim, como forma de apresentar os resultados da execução dos programas principais numa leitura mais legível; Gerador.h, responsável pela criação de novas bases de dados, tendo customização fácil para a atingir diferentes formas de organização dos dados gerados (crescente, decrescente e aleatória), podendo também ter a quantidade de dados escolhida, bem como opção de repetição dos dados; e Proximo.h, responsável pelo controle do início e fim da execução do programa.

Sobre os dados

Os dados utilizados foram das bases de dados disponibilizadas pelo Prof. Luís Forçán, sendo estas listas com diferentes quantidades de números dispostos de forma aleatória (1.000, 5.000, 10.000, 50.000, 100.000 e 500.000 números). Também foi preciso criar um programa o Gerador.h, já mencionado anteriormente, que foi construído para auxiliar algumas destas bases, sendo utilizado para criação de dados apenas em casos em que as bases previamente fornecidas estavam incompletas, como na base de dados de 500.000 números. Segue abaixo o código do Gerador.h:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MIN 1
#define MAX 1000000
#define QTDE 100000  //precisa ser menor que MAX

Comment Code
void shuffle(int *array) {
    for (int i = MAX - MIN - 1; i > 0; i--) {
        int j = rand() % (i + 1);
        int tmp = array[j];
        array[j] = array[i];
        array[i] = tmp;
    }
}

Comment Code
int main(void) {
    srand(time(NULL));
    int * numeros = malloc((MAX - MIN) * sizeof(int));
    if (!numeros) exit(EXIT_FAILURE);
    FILE *lerascii;
    lerascii = fopen("dtaleat100kuni0.txt", "w");

    for (int i = 0; i < MAX - MIN; i++) {
        numeros[i] = i + MIN;
        fprintf(lerascii, "%d\n", numeros[i]);
    }
    fclose(lerascii);
    shuffle(numeros);
    lerascii = fopen("dtaleat100kuni0.txt", "w");

    for (int i = 0; i < QTDE; i++) {
        printf("%d\n", numeros[i]);
        fprintf(lerascii, "%d\n", numeros[i]);
    }
    return 0;
}
```

Essas bases de dados foram utilizadas para analisar o comportamento do Bubble Sort, Selection Sort e Insertion Sort em determinados tipos de situações com pequenas e grandes quantidades de dados a serem ordenadas, onde cada um apresentou um tipo de comportamento.

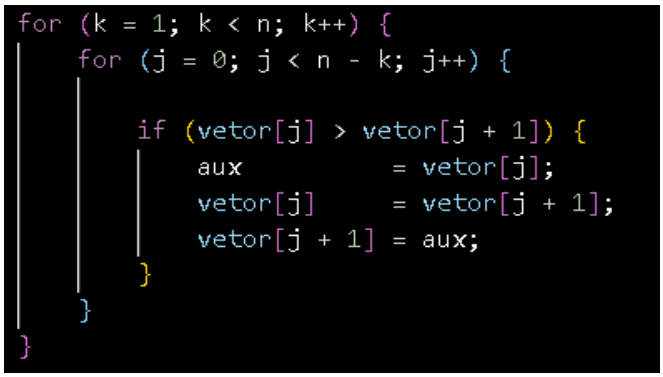
Processos de ordenação e resultados

Cada um dos três algoritmos testados possui formas próprias de implementação, e tais formas afetam drasticamente o desempenho deste. Os testes foram capazes de evidenciar o desempenho por tempo de cada um dos algoritmos propostos sobre uma mesma seleção de base de dados, já descritas anteriormente, representando vetores contendo a mesma quantidade de números aleatórios e não repetidos.

Processo de ordenação e resultado do Bubble Sort

O processo de ordenação de cada algoritmo ocorre de maneira diferente, no Bubble Sort os dados são comparados com o seu sucessor um a um verificando se seus sucessores possuem valores maiores que eles e caso possuam os dados são trocados de lugares e isso vai ocorrendo até que toda base de dados seja ordenada de forma crescente ou decrescente.

Na programação isso ocorre com o auxílio de três variáveis, duas para verificarem e armazenarem os valores que iram ser comparados e uma para realizar a troca das posições caso o valor sucessor seja maior que o antecessor. Veja como é este código:



Em relação ao resultado pós ordenação das bases de dados o algoritmo Bubble Sort apresentou o pior resultado geral, ficando em último na comparação da avaliação de desempenho em 5 das 6 bases testadas, sendo seu único resultado positivo a aplicação em base de 1.000 linhas, onde superou o algoritmo Selection Sort por 0.853s e o Insertion Sort por apenas 0.001s. O teste evidenciou o esperado para a estrutura do código deste algoritmo, que mesmo apresentando resultado efetivo num vetor pequeno, acaba sendo pior a cada aumento na base de dados, pois necessita de muitas comparações para atingir seu objetivo sendo um algoritmo quadrático.

Processo de ordenação e resultado do Selection Sort

O processo de ordenação do Selection Sort funciona da seguinte forma, o menor ou o mair dado da base de dados é selecionado e colocado na nova lista assim sucessivamente até que todos os dados sejam colocados em ordem. O algoritmo do Selection percorre toda a base de dados até encontrar o menor ou maior dado. Veja a seguir como é a estrutura do algoritmo:

```
for (i = 0; i < (tam - 1); i++) {
    min = i;
    for (j = (i + 1); j < tam; j++) {
        if (v[j] < v[min]) {
            min = j;
        }
    }
    if (i != min) {
        int swap = v[i];
        v[i] = v[min];
        v[min] = swap;
    }
}
```

Em relação ao resultado pós ordenação das bases de dados o algoritmo Selection Sort apresentou o resultado médio, tendo seus melhores desempenhos

nas bases de 5.000 e 10.000, respectivamente. Na base de 1.000 dados o algoritmo apresentou o pior resultado na comparação geral, fato que se justifica pela necessidade de comparação, mesmo que o vetor já esteja ordenado, verificando, portanto, todos os elementos. Em comparação com o Bubble Sort, afastando a referida base de 1.000 linhas, apresentou resultado duas vezes mais rápidos em 4 das 5 bases restantes, sendo que na última base, de 500.000 dados, apresentou resultado semelhante, superando seu concorrente em apenas 0.603s.

Processo de ordenação e resultado do Insertion Sort

A ordenação de dados com Insertion Sort consiste em percorrer uma base de dados da esquerda para a direita e à medida que vai avançando ordenar os dados a esquerda. Percorrendo a base de dados ele seleciona um dado e o insere em um local apropriado para ele.

Na programação o código para a ordenação funciona com uma variável denominada chave que contém o número seleciona e ela será comparada com seus antecessores para verificar se ela for maior ou menor que eles caso seja maior ficará na posição em que está, caso seja menor irá ser transferida a posição que ela pertence. Veja a seguir o código:

```
for (j = 1; j < tam; j++) {
    chave = v[j];
    i = j - 1;
    while ((i >= 0) && (v[i] > chave)) {
        v[i + 1] = v[i];
        i--;
    }
    v[i + 1] = chave;
}
```

Em relação ao resultado pós ordenação das bases de dados o algoritmo Insertion Sort apresentou o melhor resultado geral, superando em comparação, o tempo em 5 das 6 bases testadas. Como esperado pela mecânica realizada em sua estrutura, alcança desempenho extremo em comparação com seus concorrentes de

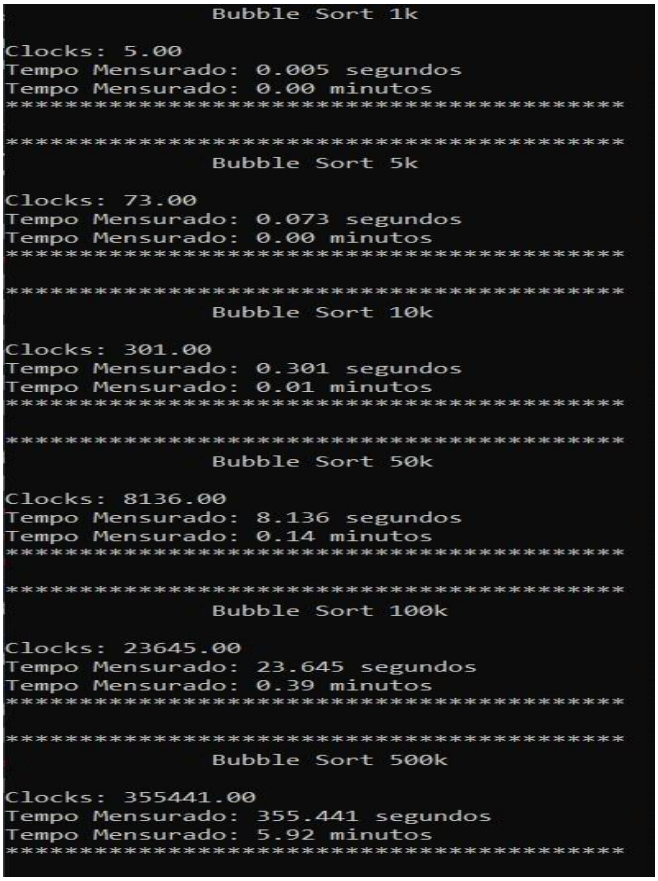
teste, pela inferior necessidade de comparações, mesmo que ainda assim, tenha uma alta necessidade de trocas.

A seguir está a tabela com o tempo que cada algoritmo de ordenação demorou para realizar a ordenação das respectivas bases de dados:

Algoritmos	1k	5k	10k	50k	100k	500k
Insertion Sort	0.004s	0.008s	0.025s	0.086s	0.318s	1.215s
Selection Sort	0.856s	0.039s	0.130s	3.180s	12.53s	344.9s
Bubble Sort	0.003s	0.065s	0.286s	7.939s	23.19s	345.5s

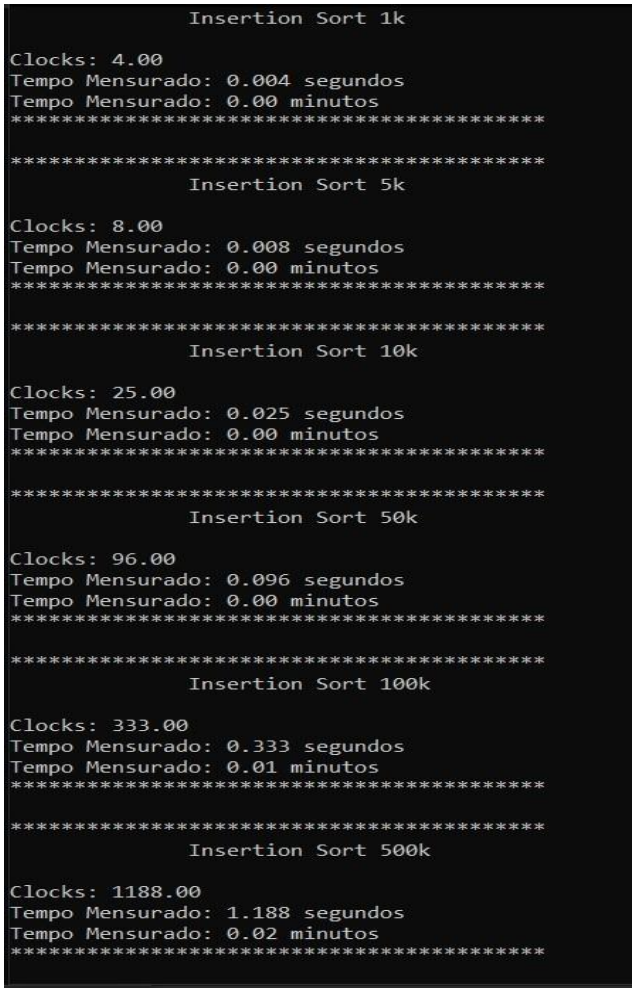
Dados extraídos de testes próprios, realizados em 14/10/2023 às 16:07.

Como informado anteriormente o Bubble Sort entre os três algoritmos apresentou a pior performance na ordenação e ficou em evidência que ele apenas tem um comportamento de alta performance em base de dados pequenas. Segue a seguir uma imagem com os dados obtidos em uma das execuções do sistema com o Bubble Sort.



Em relação ao Selection Sort, seu comportamento já era esperado por conta dele revisar a base de dados mesmo após ela está ordenada. Segue a seguir uma imagem com os dados obtidos em uma das execuções do sistema com o Selection Sort.

O Insertion Sort ele obteve a maior performance em na ordenação, conseguiu organizar todas as bases de dados em segundos, obtendo a melhor performance entre os três algoritmos. Segue a seguir uma imagem com os dados obtidos em uma das execuções do sistema com o Insertion Sort.



Além de comparar os dados por tempo, pode-se comparar o pior, melhor e médio caso do algoritmo por quantas trocas eles realizaram no processo de ordenação. Os cálculos são realizados de acordo com a quantidade de vezes que foi realizada a troca, quanto mais trocas maiores serão os valores e assim podendo encaixar ou em melhor ou em médio ou pior caso.

Algoritmo	Complexidade		
	Melhor	Médio	Pior
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$

Análise de algoritmos de ordenação por comparação e troca – pt.wikipedia.org/wiki/Insertion_sort. Acesso em 14/10/2023 às 16:21.

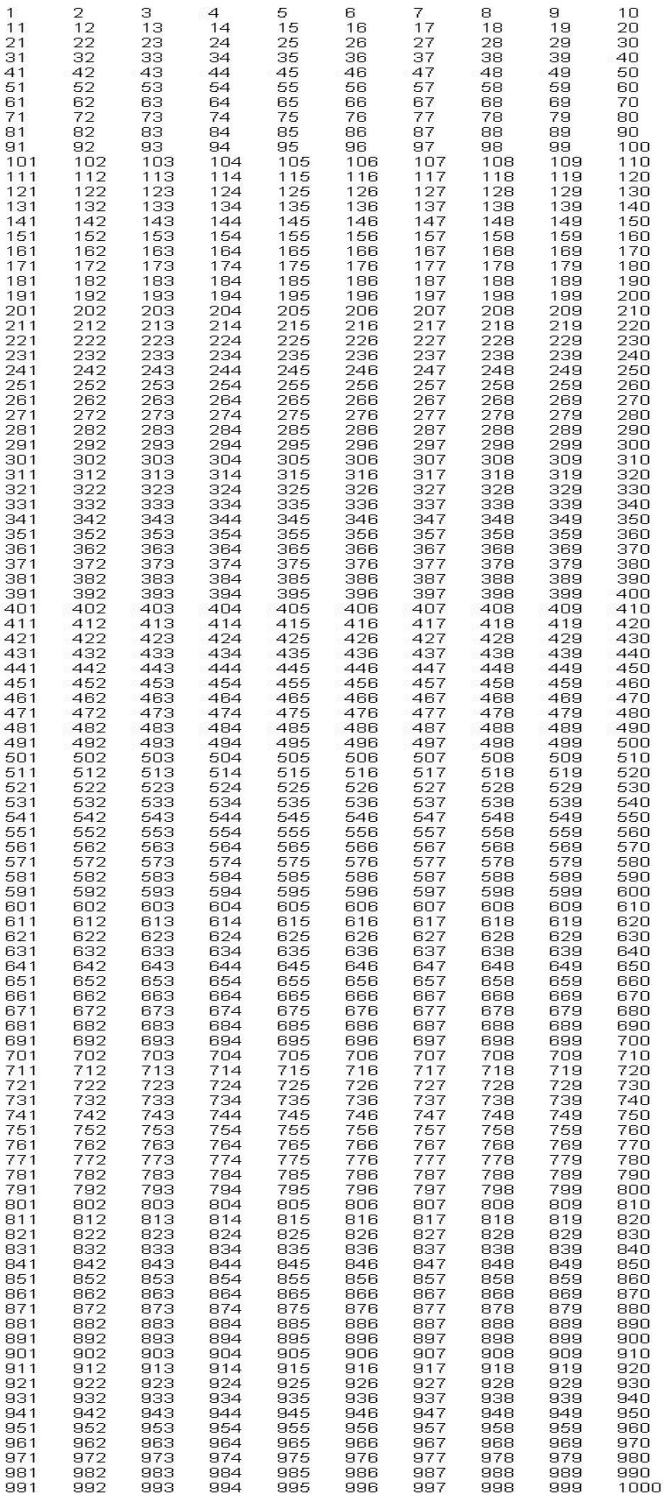
Dados não ordenados e ordenados

As bases de dados utilizadas como dito anteriormente tem diversos tamanhos com diversos elementos, no caso o que foi utilizado no projeto foram números nessas bases de dados, que precisavam ser organizados pelos algoritmos para verificar a

atuação deles em diversos casos. A seguir pode ser ver um pequeno trecho da base de dados que foi utilizada:

524	830	173	610	213	495	772	220	133	39
470	578	534	63	808	506	689	195	952	258
205	793	112	59	295	350	728	914	991	278
153	382	377	124	62	596	633	869	696	550
467	48	366	131	371	618	463	832	583	284
347	234	851	208	574	28	335	825	536	908
345	477	911	554	462	985	215	360	635	117
533	185	263	32	204	318	522	216	848	314
585	323	356	214	702	649	222	751	493	223
445	528	162	376	604	8	52	615	465	748
165	682	276	777	939	887	927	324	394	572
799	152	361	652	249	405	209	762	567	74
291	946	979	895	58	194	835	580	598	64
542	219	817	65	126	721	147	956	871	666
531	67	60	231	40	717	166	862	154	44
253	452	141	108	120	976	992	288	661	881
638	478	510	597	575	714	546	665	297	903
217	446	774	357	433	221	606	229	122	720
450	484	5	797	619	26	418	789	408	84
698	327	971	237	454	568	684	813	53	310
552	161	54	988	538	368	593	138	525	401
676	187	282	266	691	907	19	672	352	2
479	443	516	573	587	629	853	938	464	412
711	134	921	250	425	207	438	293	614	549
645	872	157	458	359	930	874	292	783	163
197	419	135	829	964	440	731	725	171	547
948	770	42	818	496	15	718	968	865	632
937	673	389	409	819	592	160	518	556	243
954	337	622	732	924	841	624	803	306	259
392	659	212	226	77	990	582	90	729	902
36	675	571	497	786	391	637	514	591	530
227	200	317	683	893	25	30	634	861	539
246	444	254	456	307	680	321	736	486	256
642	588	701	708	232	61	260	72	203	379
18	767	631	996	365	47	333	71	690	771
750	537	294	805	891	850	707	448	882	595
202	852	845	934	268	686	540	188	828	589
329	298	228	302	308	644	875	102	519	23
411	654	737	918	272	296	584	98	577	20
406	884	7	662	658	925	286	668	91	242
566	431	149	354	487	372	765	103	46	651
791	378	380	613	626	239	150	395	621	834
101	782	143	441	716	265	167	132	400	80
489	983	745	879	481	640	926	602	31	795
289	820	423	248	563	502	713	130	919	119
422	740	332	915	206	104	316	967	381	12
710	625	168	594	420	277	998	145	910	628
37	498	78	34	775	541	70	73	280	535
824	796	158	863	175	815	678	565	16	932
641	697	369	432	281	873	115	334	790	833
374	11	320	313	169	351	515	692	760	826
849	311	358	245	564	844	766	687	700	981
322	55	427	857	473	961	430	56	76	114
963	877	864	838	490	170	183	111	823	386
196	500	105	24	837	557	14	601	636	483
688	742	300	33	370	375	429	738	706	349
35	336	600	779	886	247	898	407	181	255
842	807	86	471	978	127	191	99	504	724
393	579	129	943	933	973	109	511	485	469
189	889	733	671	82	975	944	890	609	299
414	816	755	404	123	987	660	883	723	193
503	455	888	68	435	977	384	980	269	83
94	599	480	66	941	752	192	802	785	905
225	218	121	957	416	570	643	901	301	679
966	955	756	43	172	180	410	923	436	398
699	997	93	982	726	950	403	935	527	761
106	801	396	670	388	949	719	936	92	140
655	434	611	746	867	139	545	182	447	41
508	353	753	798	264	156	198	428	507	491
315	551	75	325	788	57	85	773	913	715
856	261	9	831	252	210	870	630	709	466
608	984	148	781	669	794	780	211	279	847
763	413	312	21	562	776	235	769	974	4
757	800	520	339	555	764	331	810	257	605
897	822	110	273	703	22	274	999	747	836
449	650	607	962	151	482	743	784	164	96
958	348	970	144	89	759	424	95	792	960
620	262	125	453	128	994	993	754	460	922
928	543	917	667	387	341	827	1	287	741
617	238	390	616	303	426	38	49	10	45
739	385	695	892	811	190	159	51	501	415
1000	517	581	821	330	474	894	461	155	3
653	397	576	118	468	439	663	986	526	906

Os dados mostrados acima foram retirados da base de dados com 1.000 elementos, no caso números.



1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130
131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170
171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190
191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210
211	212	213	214	215	216	217	218	219	220
221	222	223	224	225	226	227	228	229	230
231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250
251	252	253	254	255	256	257	258	259	260
261	262	263	264	265	266	267	268	269	270
271	272	273	274	275	276	277	278	279	280
281	282	283	284	285	286	287	288	289	290
291	292	293	294	295	296	297	298	299	300
301	302	303	304	305	306	307	308	309	310
311	312	313	314	315	316	317	318	319	320
321	322	323	324	325	326	327	328	329	330
331	332	333	334	335	336	337	338	339	340
341	342	343	344	345	346	347	348	349	350
351	352	353	354	355	356	357	358	359	360
361	362	363	364	365	366	367	368	369	370
371	372	373	374	375	376	377	378	379	380
381	382	383	384	385	386	387	388	389	390
391	392	393	394	395	396	397	398	399	400
401	402	403	404	405	406	407	408	409	410
411	412	413	414	415	416	417	418	419	420
421	422	423	424	425	426	427	428	429	430
431	432	433	434	435	436	437	438	439	440
441	442	443	444	445	446	447	448	449	450
451	452	453	454	455	456	457	458	459	460
461	462	463	464	465	466	467	468	469	470
471	472	473	474	475	476	477	478	479	480
481	482	483	484	485	486	487	488	489	490
491	492	493	494	495	496	497	498	499	500
501	502	503	504	505	506	507	508	509	510
511	512	513	514	515	516	517	518	519	520
521	522	523	524	525	526	527	528	529	530
531	532	533	534	535	536	537	538	539	540
541	542	543	544	545	546	547	548	549	550
551	552	553	554	555	556	557	558	559	560
561	562	563	564	565	566	567	568	569	570
571	572	573	574	575	576	577	578	579	580
581	582	583	584	585	586	587	588	589	590
591	592	593	594	595	596	597	598	599	600
601	602	603	604	605	606	607	608	609	610
611	612	613	614	615	616	617	618	619	620
621	622	623	624	625	626	627	628	629	630
631	632	633	634	635	636	637	638	639	640
641	642	643	644	645	646	647	648	649	650
651	652	653	654	655	656	657	658	659	660
661	662	663	664	665	666	667	668	669	670
671	672	673	674	675	676	677	678	679	680
681	682	683	684	685	686	687	688	689	690
691	692	693	694	695	696	697	698	699	700
701	702	703	704	705	706	707	708	709	710
711	712	713	714	715	716	717	718	719	720
721	722	723	724	725	726	727	728	729	730
731	732	733	734	735	736	737	738	739	740
741	742	743	744	745	746	747	748	749	750
751	752	753	754	755	756	757	758	759	760
761	762	763	764	765	766	767	768	769	770
771	772	773	774	775	776	777	778	779	780
781	782	783	784	785	786	787	788	789	790
791	792	793	794	795	796	797	798	799	800
801	802	803	804	805	806	807	808	809	810
811	812	813	814	815	816	817	818	819	820
821	822	823	824	825	826	827	828	829	830
831	832	833	834	835	836	837	838	839	840
841	842	843	844	845	846	847	848	849	850
851	852	853	854	855	856	857	858	859	860
861	862	863	864	865	866	867	868	869	870
871	872	873	874	875	876	877	878	879	880
881	882	883	884	885	886	887	888	889	890
891	892	893	894	895	896	897	898	899	900
901	902	903	904	905	906	907	908	909	910
911	912	913	914	915	916	917	918	919	920
921	922	923	924	925	926	927	928	929	930
931	932	933	934	935	936	937	938	939	940
941	942	943	944	945	946	947	948	949	950
951	952	953	954	955	956	957	958	959	960
961	962	963	964	965	966	967	968	969	970
971	972	973	974	975	976	977	978	979	980
981	982	983	984	985	986	987	988	989	990
991	992	993	994	995	996	997	998	999	1000

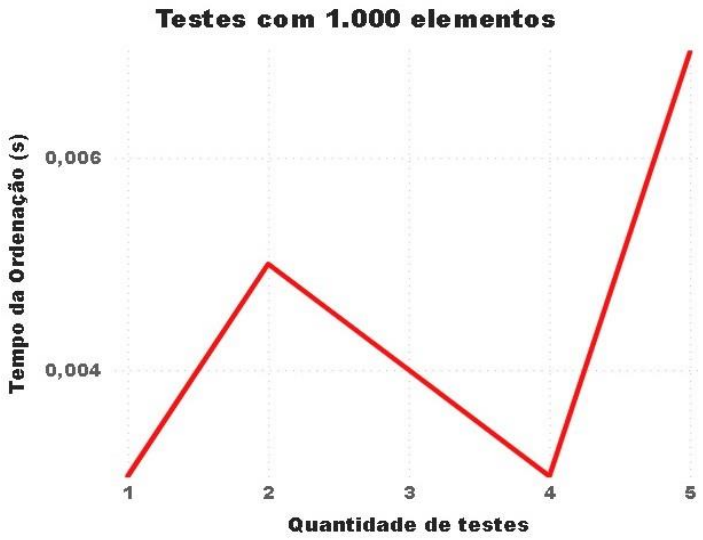
Agora nesta próxima imagem pode-se observar como os dados ficaram após serem organizados pelos algoritmos de ordenação.

Resultados e Discussão

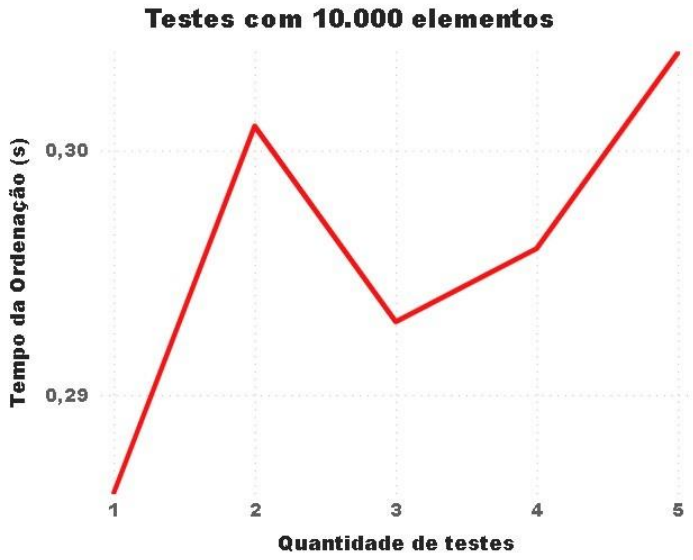
Após diversas execuções do sistema pode se obter diversos resultados e pode se observar como cada algoritmo de ordenação se comporta em determinada situação.

A primeira situação que os algoritmos foram colocados foi com a base de dados aleatória com 1.000 elementos, no caso números. Ao serem testados com essa base de dados cada um se comportou de uma forma diferente um obtive dados já esperados e os outros obtiveram dados que não eram esperados.

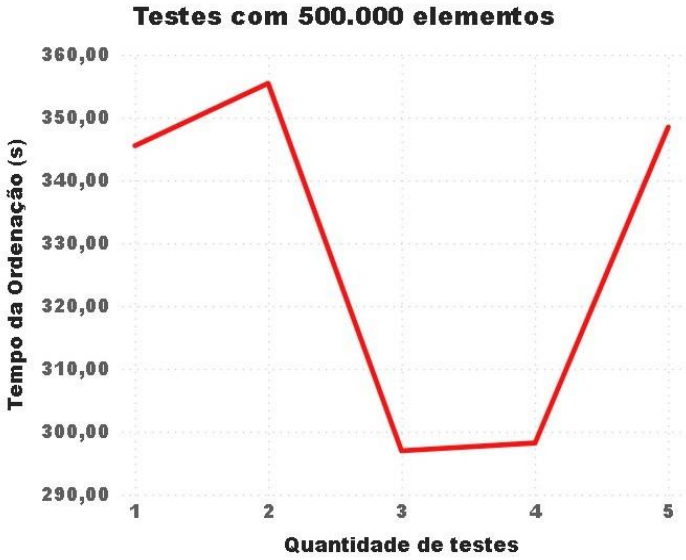
O Bubble Sort como dito anteriormente mesmo em apenas uma execução do programa ele obteve a pior performance na ordenação das bases de dados em comparação com o Selection Sort e o Insertion Sort, e isso se repetiu após o programa ser executado mais vezes. O Bubble Sort apenas conseguiu superar os outros algoritmos em alguns casos na base de dados com 1.000 elementos, realizando testes com 5 execuções seguidas o Bubble Sort consegui superar somente 3 vezes e com uma pequena diferença de tempo em relação aos outros algoritmos de ordenação. A seguir pode se observar um gráfico com a performance em tempo de como o Bubble Sort se comportou com a base de dados de 1.000 elementos.



Nas outras bases de dados o Bubble Sort apresentou resultados péssimos, o que já era de se esperar, pois o Bubble Sort apresenta bons resultados apenas com bases de dados semi-ordenadas. Com a base de dados com 10.000 elementos (números), o Bubble Sort por ser um algoritmo quadrático (O(n^2)) era de se esperar que seu comportamento fosse performático, mas o que ocorreu foi o contrário ele começou a apresentar lentidão ao começar a aumentar as bases de dados, em relação aos outros algoritmos apresentou os piores resultados. A seguir pode se observar um gráfico com a performance em tempo de como o Bubble Sort se comportou com a base de dados de 10.000 elementos.

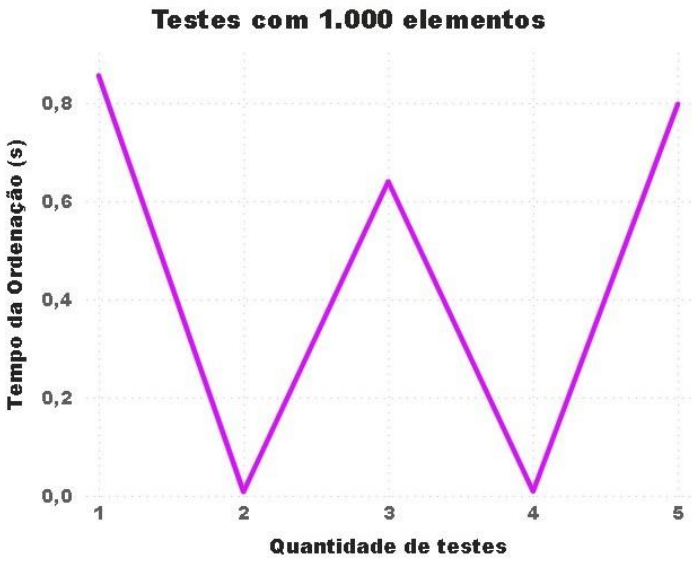


Com a base de dados com 500.000 elementos (números) não foi diferente, o Bubble Sort aumentou ainda mais seu tempo para ordenar os dados e assim continuando tendo apenas resultados ruins em ordenação por tempo. Assim podemos ver no gráfico a seguir como ficou seu tempo de processamento nas 5 execuções de teste que foi feita.

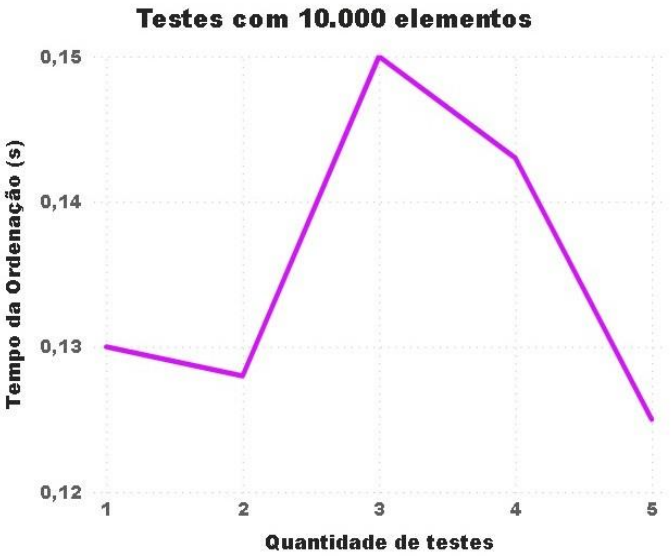


O Selection Sort, por sua vez, apresentou resultados que eram de se esperar em alguns casos por conta da sua necessidade de percorrer a base de dados mesmo depois dela já estar organizada. Na base de dados com 1.000 elementos, o Selection Sort apresentou os piores resultados em geral, dentro das 5 execuções testes seu tempo de ordenação foram os maiores em relação ao primeiro teste

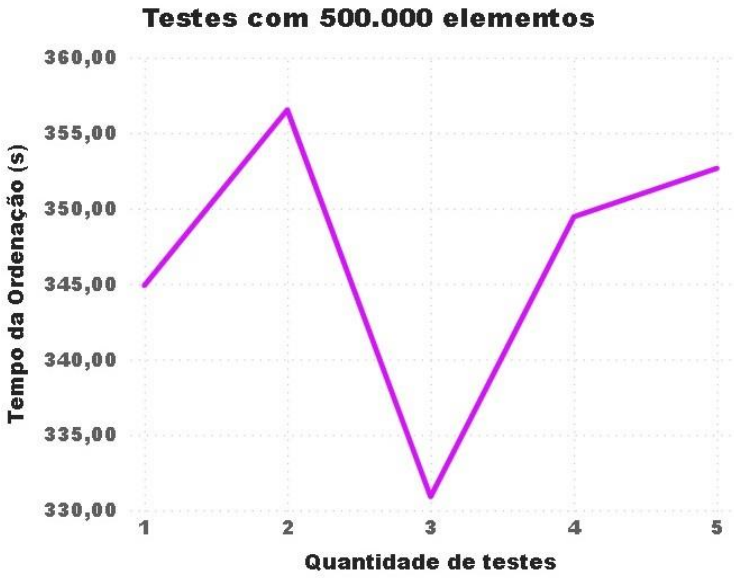
enquanto o Bubble Sort realizou a ordenação em 0.003s, o Selection Sort demorou cerca de 200 vezes mais de tempo em relação ao Bubble Sort. Como pode ser observado no gráfico a seguir 3 de seus resultados demandaram muito tempo para conseguirem realizar as ordenações, enquanto os outros 2 demandaram um número menor de tempo.



Com a base de dados de 10.000 elementos o Selection Sort conseguiu obter tempos relativamente bons em comparação ao tempo que ele demandou para organizar a base de dados de 1.000 elementos, mas em relação aos outros algoritmos seus resultados foram médios, pois o Insertion Sort ainda conseguiu te superar em tempo de ordenação, mas conseguiu superar o Bubble Sort que obteve o dobre de seu tempo de execução. A seguir pode se observar um gráfico com a performance em tempo de como o Selection Sort se comportou com a base de dados de 10.000 elementos.

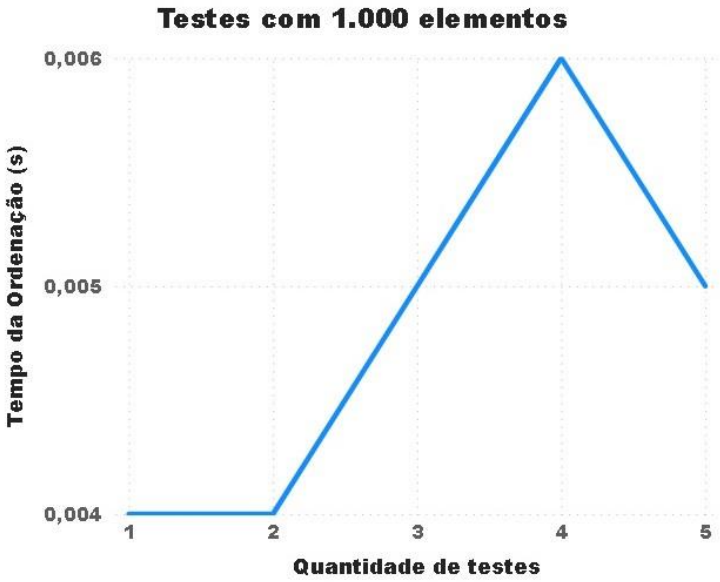


Com a base de dados de 500.000 elementos não foi diferente da de 1.000 elementos, o Selection Sort obteve os piores resultados em tempo de ordenação em relação aos outros algoritmos de ordenação, demorou em todos os 5 testes cerca de 4 a 6 minutos para conseguir ordenar a base de dados. Veja seus dados no gráfico a seguir.

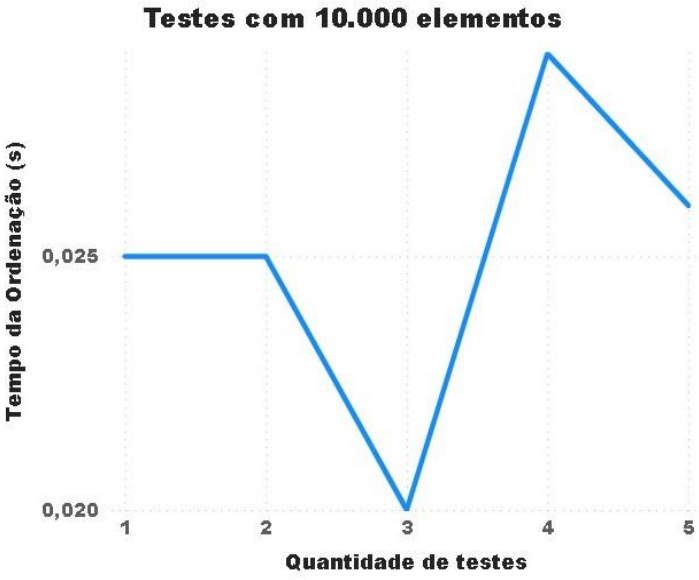


O Insertion Sort em relação ao Selection Sort e o Bubble Sort obteve os melhores resultados em todos os testes, somente na base de dados com 1.000 elementos que ele foi superado pelo Bubble Sort em alguns testes, mas os tempos em que ele foi superado não foram nada exuberante foram questões de alguns

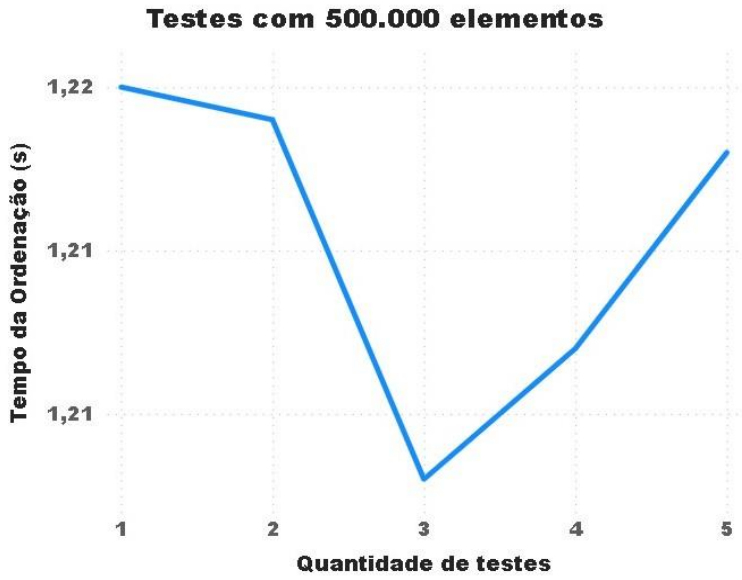
milésimos de segundos, como no primeiro teste que ele foi superado por apenas 0.001s. Veja no gráfico a seguir como foi seu comportamento nessa base de dados.



Na base de dados de 10.000 elementos não foi diferente, como dito anteriormente o Insertion Sort foi o que apresentou melhor performance nas ordenações de dados. A seguir pode se observar no gráfico seu comportamento.



O comportamento do Insertion Sort na base de dados de 500.000 elementos foi o que mais surpreendeu, pois, colocando-o em comparação com os outros algoritmos era de se pensar que ele iria demorar um grande tempo para conseguir organizar os dados, mas ele conseguiu organizar os dados com cerca de 300 vezes mais agilidade do que os outros algoritmos. A seguir está expresso no gráfico os resultados obtidos pelo Insertion Sort.



Como se pode observar através dos dados coletados, pode se deduzir que o Bubble Sort apresenta melhores resultados com bases de dados menores e não com grandes bases de dados. Quanto ao Selection Sort, o que mais influência em sua demora para ordenação seria além de sua forma de ordenação, mas sim a sua última conferida para ver se tudo está no lugar. E o Insertion Sort neste cenário que utilizamos para os testes ele não possui desvantagem alguma, pois como mostra os resultados ele consegue manipular os dados mesmo de forma aleatória com perfeição.