1 Ordenacao e ordenação invertida

Todos os algoritmos e suas variações estão implementados no codigo a seguir.

Para testramos a funcionalidade dos algoritmos utilizamos um argumento na main para escolher uma das possibilidades de algoritmos.

Parâmetros:

- 1 – Shellsort
- 12 – Shellsort Reverso
- 2 – Quicksort
- 22 – Quicksort Reverso
- 3 – Heapsort
- 32 – Heapsort Reverso
- 4 – Mergesort
- 42 – Mergesort Reverso

Resultados: Printamos as primeiras 30 posições do vetor após a ordenação

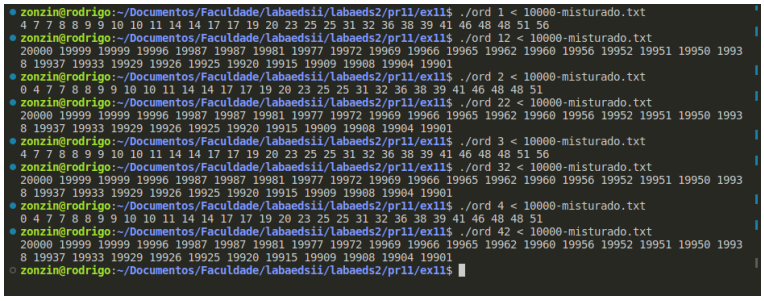

Figura 1: Todos os algoritmos

2 Tempos

Usando o script em Python, obtemos os seguintes tempos:

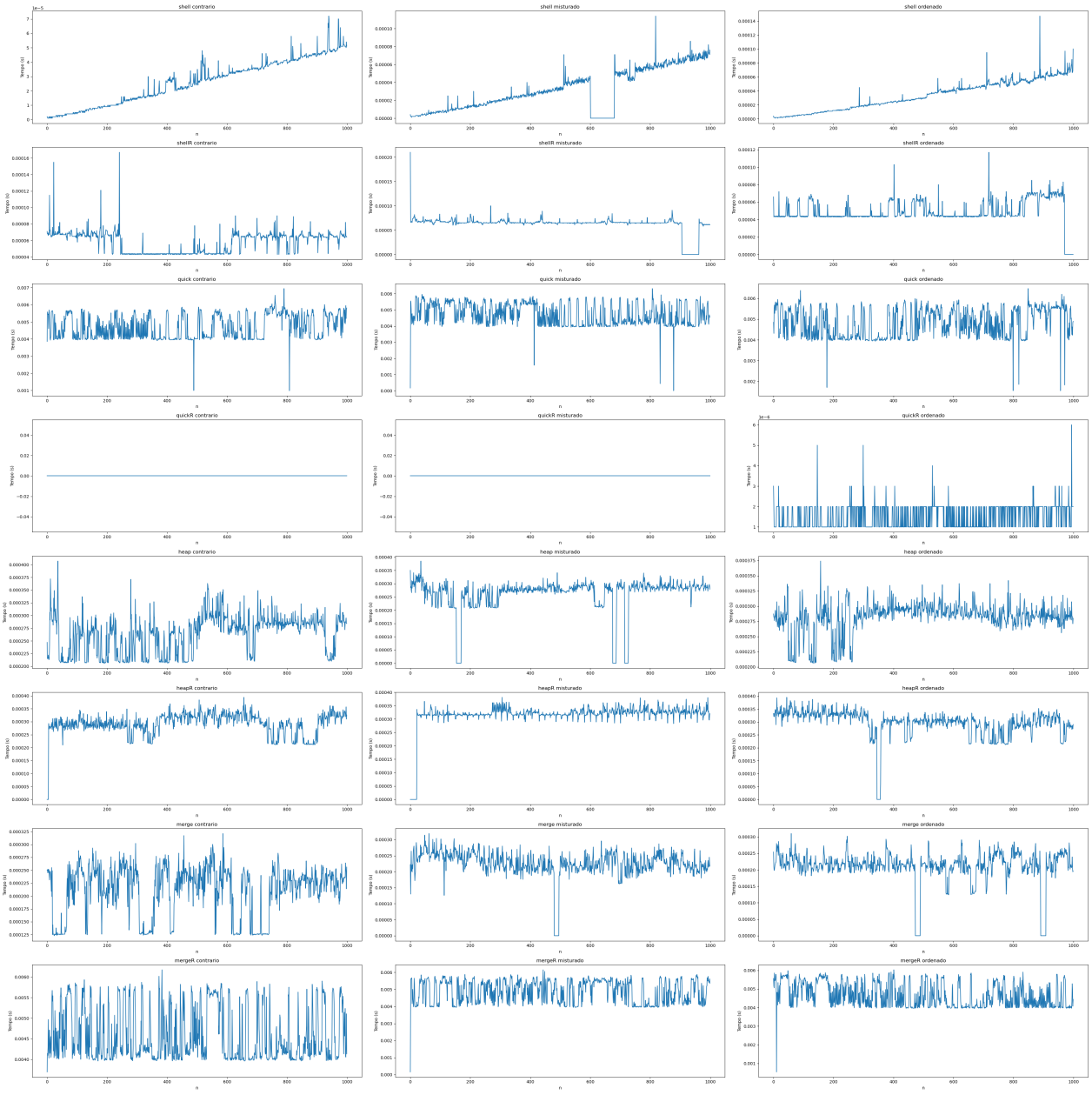

Figura 2: Algoritmos vs tempo vs n

3 Códigos

```c
#include <stdlib.h>
#include <stdio.h>
#include <sys/resource.h>
#include <sys/time.h>

```

```
6
7  void shellsort(int*, int);
8  void shellsortReverse(int*, int);
9
10 void quicksort(int*, int, int);
11 void quicksortReverse(int*, int, int);
12
13 void mergesort(int*, int, int);
14 void mergesortReverse(int*, int, int);
15
16 void heapsort(int*, int);
17 void heapsortReverse(int*, int);
18
19 void shellsort(int *vetor, int n){
20     int intervalo, i, j, temp;
21     for(intervalo = n / 2; intervalo > 0; intervalo /= 2){
22         for(i = intervalo; i < n; i += 1){
23             temp = vetor[i];
24             for(j = i; j >= intervalo && vetor[j - intervalo] > temp; j -=
                   intervalo){
25                 vetor[j] = vetor[j - intervalo];
26             }
27             vetor[j] = temp;
28         }
29     }
30 }
31
32 void shellsortReverse(int *vetor, int n) {
33     int intervalo, i, j, temp;
34     for (intervalo = n / 2; intervalo > 0; intervalo /= 2) {
35         for (i = intervalo; i < n; i += 1) {
36             temp = vetor[i];
37             for (j = i; j >= intervalo && vetor[j - intervalo] < temp; j -=
                   intervalo) {
38                 vetor[j] = vetor[j - intervalo];
39             }
40             vetor[j] = temp;
41         }
42     }
43 }
44
45
46 void troca(int *a, int *b) {
47     int temp = *a;
48     *a = *b;
49     *b = temp;
50 }
51
52 int particiona(int vetor[], int inicio, int fim) {
53     int pivo = vetor[fim];
54     int i = inicio - 1;
55
56     for (int j = inicio; j < fim; j++) {
57         if (vetor[j] <= pivo) {
58             i++;
59             troca(&vetor[i], &vetor[j]);
60         }
61     }
62     troca(&vetor[i + 1], &vetor[fim]);
```

```
63      return i + 1;
64  }
65
66  void quicksort(int *vetor, int inicio, int fim) {
67      if (inicio < fim) {
68          int indice_pivo = particiona(vetor, inicio, fim);
69
70          quicksort(vetor, inicio, indice_pivo - 1);
71          quicksort(vetor, indice_pivo + 1, fim);
72      }
73  }
74
75  int particionaReverse(int vetor[], int inicio, int fim) {
76      int pivo = vetor[fim];
77      int i = inicio - 1;
78
79      for (int j = inicio; j < fim; j++) {
80          if (vetor[j] >= pivo) {
81              i++;
82              troca(&vetor[i], &vetor[j]);
83          }
84      }
85      troca(&vetor[i + 1], &vetor[fim]);
86      return i + 1;
87  }
88
89  void quicksortReverse(int vetor[], int inicio, int fim) {
90      if (inicio < fim) {
91          int indice_pivo = particionaReverse(vetor, inicio, fim);
92
93          quicksortReverse(vetor, inicio, indice_pivo - 1);
94          quicksortReverse(vetor, indice_pivo + 1, fim);
95      }
96  }
97
98  // HEAPSORT
99  void maxHeapify(int vetor[], int tamanho, int indice) {
100     int maior = indice;
101     int filho_esq = 2 * indice + 1;
102     int filho_dir = 2 * indice + 2;
103
104     if (filho_esq < tamanho && vetor[filho_esq] > vetor[maior])
105         maior = filho_esq;
106
107     if (filho_dir < tamanho && vetor[filho_dir] > vetor[maior])
108         maior = filho_dir;
109
110     if (maior != indice) {
111         troca(&vetor[indice], &vetor[maior]);
112         maxHeapify(vetor, tamanho, maior);
113     }
114 }
115
116 void heapsort(int vetor[], int tamanho) {
117     for (int i = tamanho / 2 - 1; i >= 0; i--)
118         maxHeapify(vetor, tamanho, i);
119
120     for (int i = tamanho - 1; i >= 0; i--) {
121         troca(&vetor[0], &vetor[i]);
```

```
122         maxHeapify(vetor, i, 0);
123     }
124 }
125
126
127 void maxHeapifyReverse(int vetor[], int tamanho, int indice) {
128     int maior = indice;
129     int filho_esq = 2 * indice + 1;
130     int filho_dir = 2 * indice + 2;
131
132     if (filho_esq < tamanho && vetor[filho_esq] < vetor[maior])
133         maior = filho_esq;
134
135     if (filho_dir < tamanho && vetor[filho_dir] < vetor[maior])
136         maior = filho_dir;
137
138     if (maior != indice) {
139         troca(&vetor[indice], &vetor[maior]);
140         maxHeapifyReverse(vetor, tamanho, maior);
141     }
142 }
143
144 void heapsortReverse(int vetor[], int tamanho) {
145     for (int i = tamanho / 2 - 1; i >= 0; i--)
146         maxHeapifyReverse(vetor, tamanho, i);
147
148     for (int i = tamanho - 1; i >= 0; i--) {
149         troca(&vetor[0], &vetor[i]);
150         maxHeapifyReverse(vetor, i, 0);
151     }
152 }
153
154 //MERGE
155 void merge(int vetor[], int esquerda, int meio, int direita) {
156     int n1 = meio - esquerda + 1;
157     int n2 = direita - meio;
158
159     int *esq = (int *)malloc(n1 * sizeof(int));
160     int *dir = (int *)malloc(n2 * sizeof(int));
161
162     for (int i = 0; i < n1; i++)
163         esq[i] = vetor[esquerda + i];
164     for (int j = 0; j < n2; j++)
165         dir[j] = vetor[meio + 1 + j];
166
167     int i = 0, j = 0, k = esquerda;
168     while (i < n1 && j < n2) {
169         if (esq[i] <= dir[j]) {
170             vetor[k] = esq[i];
171             i++;
172         } else {
173             vetor[k] = dir[j];
174             j++;
175         }
176         k++;
177     }
178
179     while (i < n1) {
180         vetor[k] = esq[i];
```

```
181         i++;
182         k++;
183     }
184 
185     while (j < n2) {
186         vetor[k] = dir[j];
187         j++;
188         k++;
189     }
190 
191     free(esq);
192     free(dir);
193 }
194 
195 void mergesort(int vetor[], int esquerda, int direita) {
196     if (esquerda < direita) {
197         int meio = esquerda + (direita - esquerda) / 2;
198 
199         mergesort(vetor, esquerda, meio);
200         mergesort(vetor, meio + 1, direita);
201 
202         merge(vetor, esquerda, meio, direita);
203     }
204 }
205 
206 void mergeReverse(int vetor[], int esquerda, int meio, int direita) {
207     int n1 = meio - esquerda + 1;
208     int n2 = direita - meio;
209 
210     int *esq = (int *)malloc(n1 * sizeof(int));
211     int *dir = (int *)malloc(n2 * sizeof(int));
212 
213     for (int i = 0; i < n1; i++)
214         esq[i] = vetor[esquerda + i];
215     for (int j = 0; j < n2; j++)
216         dir[j] = vetor[meio + 1 + j];
217 
218     int i = 0, j = 0, k = esquerda;
219     while (i < n1 && j < n2) {
220         if (esq[i] >= dir[j]) {
221             vetor[k] = esq[i];
222             i++;
223         } else {
224             vetor[k] = dir[j];
225             j++;
226         }
227         k++;
228     }
229 
230     while (i < n1) {
231         vetor[k] = esq[i];
232         i++;
233         k++;
234     }
235 
236     while (j < n2) {
237         vetor[k] = dir[j];
238         j++;
239         k++;
```

```
240     }
241
242     free(esq);
243     free(dir);
244 }
245
246 void mergesortReverse(int vetor[], int esquerda, int direita) {
247     if (esquerda < direita) {
248         int meio = esquerda + (direita - esquerda) / 2;
249
250         mergesortReverse(vetor, esquerda, meio);
251         mergesortReverse(vetor, meio + 1, direita);
252
253         mergeReverse(vetor, esquerda, meio, direita);
254     }
255 }
256
257 /*
258 int main(int argc, char** argv){
259     int N = 0;
260     struct rusage inicio, fim;
261     struct timeval tempo_inicial, tempo_final;
262
263     fscanf(stdin, "%d", &N);
264     int *vet  =(int*)malloc(sizeof(int)*N);
265
266     for(int i =0; i<N; i++){
267         fscanf(stdin, "%d", &vet[i]);
268     }
269
270     int alg = atoi(argv[1]);
271
272
273     if(alg == 1){
274         for(int i=0; i< N; i++){
275             getrusage(RUSAGE_SELF, &inicio);
276                 shellsort(vet, i);
277             getrusage(RUSAGE_SELF, &fim);
278             tempo_inicial = inicio.ru_utime;
279             tempo_final = fim.ru_utime;
280             double tempo_execucao = (tempo_final.tv_sec - tempo_inicial.tv_sec) +
281                                     (tempo_final.tv_usec - tempo_inicial.tv_usec)
                                        / 1000000.0;
282             printf("%d,%.10lf\n", i, tempo_execucao);
283         }
284     }
285
286     if(alg == 12){
287         shellsortReverse(vet, N);
288     }
289
290     if(alg == 2){
291         quicksort(vet, 0, N);
292     }
293
294     if(alg == 22){
295         quicksortReverse(vet, 0, N);
296     }
297
```

```
298     if(alg == 3){
299         heapsort(vet, N);
300     }
301 
302     if(alg == 32){
303         heapsortReverse(vet, N);
304     }
305 
306     if(alg == 4){
307         mergesort(vet, 0, N);
308     }
309     if(alg == 42){
310         mergesortReverse(vet, 0, N);
311     }
312 }
313 */
314 
315 int main(int argc, char** argv) {
316     int N = 0;
317     fscanf(stdin, "%d", &N);
318     int *vet = (int*)malloc(sizeof(int) * N);
319 
320     for (int i = 0; i < N; i++) {
321         fscanf(stdin, "%d", &vet[i]);
322     }
323 
324     int alg = atoi(argv[1]);
325 
326     struct rusage inicio, fim;
327     struct timeval tempo_inicial, tempo_final;
328 
329     N = 1000;
330     for (int i = 0; i < N; i++){
331         getrusage(RUSAGE_SELF, &inicio);
332         if (alg == 1) {
333             shellsort(vet, i);
334         } else if (alg == 12) {
335             shellsortReverse(vet, N);
336         } else if (alg == 2) {
337             quicksort(vet, 0, N);
338         } else if (alg == 22) {
339             quicksortReverse(vet, 0, N);
340         } else if (alg == 3) {
341             heapsort(vet, N);
342         } else if (alg == 32) {
343             heapsortReverse(vet, N);
344         } else if (alg == 4) {
345             mergesort(vet, 0, N);
346         } else if (alg == 42) {
347             mergesortReverse(vet, 0, N);
348         }
349         getrusage(RUSAGE_SELF, &fim);
350         tempo_inicial = inicio.ru_utime;
351         tempo_final = fim.ru_utime;
352         double tempo_execucao = (tempo_final.tv_sec - tempo_inicial.tv_sec) +
353                                 (tempo_final.tv_usec - tempo_inicial.tv_usec) /
                                     1000000.0;
354         printf("%d,%.10lf\n", i, tempo_execucao);
355     }
```

```
356
357     if(alg == -1){
358         for(int i = 0; i<30; i++){
359             printf("%d ", vet[i]);
360         }
361         printf("\n");
362     }
363
364
365     free(vet);
366     return 0;
367 }
```

../ex11/ordenacao.c