

UFSJ - Ciências da Computação

Laboratório de Programação 2

Roteiro 11

Nome: Geraldo Arthur Detomi

1.1) Faça um programa para ler um valor N e em seguida N inteiros, armazenando esses inteiros em um vetor. Em seguida, ordenar esses valores utilizando os metodos de ordenacao vistos: ShellSort, QuickSort, MergeSort e HeapSort.

roteiro_11/1-1.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void liberar_vetor(int **v) {
5      free(*v);
6      *v = NULL;
7  }
8
9  int *copiaVetor(int *v, int n) {
10     int i;
11     int *v2;
12     v2 = (int *)malloc(n * sizeof(int));
13     for (i = 0; i < n; i++)
14         v2[i] = v[i];
15     return v2;
16 }
17
18 void imprimeVetor(int *v, int n) {
19     int i, prim = 1;
20     printf("[");
21     for (i = 0; i < n; i++)
22         if (prim) {
23             printf("%d", v[i]);
24             prim = 0;
25         } else
26             printf(", %d", v[i]);
27     printf("]\n");
28 }
29
30 void troca(int *a, int *b) {
31     int aux = *a;
32     *a = *b;
33     *b = aux;
34 }
35
36 void shellSort(int *v, int n) {
37     int i, j, atual;
38     int h = 1;
39     while (h < n)
40         h = 3 * h + 1;
41     while (h > 0) {
42         for (i = h; i < n; i++) {
43             atual = v[i];
44             j = i;
45             while (j > h - 1 && atual <= v[j - h]) {
46                 v[j] = v[j - h];
47                 j = j - h;
48             }
49             v[j] = atual;
50         }
51         h = h / 3;
52     }
53 }
54
55 void criaHeap(int *v, int pai, int fim) {
56     int aux = v[pai];
57     int filho = 2 * pai + 1;
58     while (filho <= fim) {
59         if (filho < fim)
60             if (v[filho] < v[filho + 1])
61                 filho++;
62         if (aux < v[filho]) {
63             v[pai] = v[filho];
64             pai = filho;
```

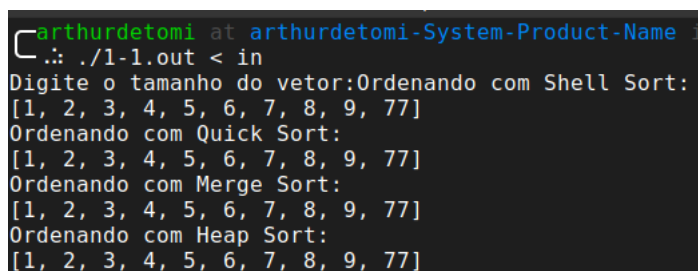
```
65     filho = 2 * pai + 1;
66 } else
67     filho = fim + 1;
68 }
69 v[pai] = aux;
70 }
71
72 void heapSort(int *v, int n) {
73     int i;
74     for (i = (n - 1) / 2; i >= 0; i--)
75         criaHeap(v, i, n - 1);
76     for (i = n - 1; i >= 1; i--) {
77         troca(&v[0], &v[i]);
78         criaHeap(v, 0, i - 1);
79     }
80 }
81
82 int particao(int *v, int ini, int fim, int *comp, int *mov) {
83     int i = ini, j = fim;
84     int pivo = v[(ini + fim) / 2];
85     while (1) {
86         (*comp)++;
87         while (v[i] < pivo) {
88             i++;
89             (*comp)++;
90         }
91
92         (*comp)++;
93         while (v[j] > pivo) {
94             j--;
95             (*comp)++;
96         }
97
98         if (i < j) {
99             troca(&v[i], &v[j]);
100             (*mov)++;
101             i++;
102             j--;
103         } else {
104             return j;
105         }
106     }
107 }
108
109 void quickSort(int *v, int ini, int fim, int n, int *comp, int *mov) {
110     if (ini < fim) {
111         int q = particao(v, ini, fim, comp, mov);
112         quickSort(v, ini, q, n, comp, mov);
113         quickSort(v, q + 1, fim, n, comp, mov);
114     }
115 }
116
117 void merge(int *v, int ini, int meio, int fim) {
118     int tam = fim - ini + 1;
119     // Vetor Auxiliar - A
120     int *A = (int *)malloc(tam * sizeof(int));
121     int i = ini, j = meio + 1, k = 0;
122     while (i <= meio && j <= fim) {
123         if (v[i] < v[j]) {
124             A[k] = v[i];
125             i++;
126         } else {
127             A[k] = v[j];
128             j++;
129         }
130         k++;
131     }
132     while (i <= meio) {
133         A[k] = v[i];
134         i++;
135         k++;
136     }
137     while (j <= fim) {
138         A[k] = v[j];
139         j++;
140         k++;
141     }
142     for (i = ini, k = 0; i <= fim; i++, k++)
143         v[i] = A[k];
144     free(A);
145 }
```

```

146
147 void mergeSort(int *v, int ini, int fim) {
148     if (ini < fim) {
149         int meio = (ini + fim) / 2;
150         mergeSort(v, ini, meio);
151         mergeSort(v, meio + 1, fim);
152         merge(v, ini, meio, fim);
153     }
154 }
155
156 enum algoritmos_para_teste {
157     SHELL_SORT = 0,
158     QUICK_SORT,
159     MERGE_SORT,
160     HEAP_SORT
161 };
162
163 #define QTD_ALGORITMOS_TESTE 4
164
165 int main() {
166     int n;
167
168     printf("Digite o tamanho do vetor:");
169     scanf("%d", &n);
170
171     int *array_original = (int *)malloc(n * sizeof(int));
172
173     for (int i = 0; i < n; i++) {
174         scanf("%d", &array_original[i]);
175     }
176
177     for (int i = 0; i < QTD_ALGORITMOS_TESTE; i++) {
178         int *array_copia = copiaVetor(array_original, n);
179
180         int comp = 0, mov = 0;
181
182         switch (i) {
183             case SHELL_SORT:
184                 printf("Ordenando com Shell Sort:\n");
185                 shellSort(array_copia, n);
186                 imprimeVetor(array_copia, n);
187                 break;
188             case QUICK_SORT:
189                 printf("Ordenando com Quick Sort:\n");
190                 quickSort(array_copia, 0, n - 1, n, &comp, &mov);
191                 imprimeVetor(array_copia, n);
192                 break;
193             case MERGE_SORT:
194                 printf("Ordenando com Merge Sort:\n");
195                 mergeSort(array_copia, 0, n - 1);
196                 imprimeVetor(array_copia, n);
197                 break;
198             case HEAP_SORT:
199                 printf("Ordenando com Heap Sort:\n");
200                 heapSort(array_copia, n);
201                 imprimeVetor(array_copia, n);
202                 break;
203         }
204
205         liberar_vetor(&array_copia);
206     }
207
208     liberar_vetor(&array_original);
209     return 0;
210
211     return 0;
212 }

```

Saída do terminal:



```

C:\arthurdetomi at arthurdetomi-System-Product-Name
.: ./1-1.out < in
Digite o tamanho do vetor:Ordenando com Shell Sort:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 77]
Ordenando com Quick Sort:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 77]
Ordenando com Merge Sort:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 77]
Ordenando com Heap Sort:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 77]

```

1-2) Modifique os algoritmos de ordenacao anteriores, para que ordenem de forma decrescente os numeros (do maior para o menor) e teste os novos metodos.

roteiro_11/1-2.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void liberar_vetor(int **v) {
5      free(*v);
6      *v = NULL;
7  }
8
9  int *copiaVetor(int *v, int n) {
10     int i;
11     int *v2;
12     v2 = (int *)malloc(n * sizeof(int));
13     for (i = 0; i < n; i++)
14         v2[i] = v[i];
15     return v2;
16 }
17
18 void imprimeVetor(int *v, int n) {
19     int i, prim = 1;
20     printf("[");
21     for (i = 0; i < n; i++)
22         if (prim) {
23             printf("%d", v[i]);
24             prim = 0;
25         } else
26             printf(", %d", v[i]);
27     printf("]\n");
28 }
29
30 void troca(int *a, int *b) {
31     int aux = *a;
32     *a = *b;
33     *b = aux;
34 }
35
36 void shellSort(int *v, int n) {
37     int i, j, atual;
38     int h = 1;
39     while (h < n)
40         h = 3 * h + 1;
41     while (h > 0) {
42         for (i = h; i < n; i++) {
43             atual = v[i];
44             j = i;
45             while (j > h - 1 && atual >= v[j - h]) {
46                 v[j] = v[j - h];
47                 j = j - h;
48             }
49             v[j] = atual;
50         }
51         h = h / 3;
52     }
53 }
54
55 void criaHeap(int *v, int pai, int fim) {
56     int aux = v[pai];
57     int filho = 2 * pai + 1;
58     while (filho <= fim) {
59         if (filho < fim)
60             if (v[filho] > v[filho + 1])
61                 filho++;
62         if (aux > v[filho]) {
63             v[pai] = v[filho];
64             pai = filho;
65             filho = 2 * pai + 1;
66         } else
67             filho = fim + 1;
68     }
69     v[pai] = aux;
70 }
71
72 void heapSort(int *v, int n) {
73     int i;
74     for (i = (n - 1) / 2; i >= 0; i--)
75         criaHeap(v, i, n - 1);
76     for (i = n - 1; i >= 1; i--) {
```

```
77     troca(&v[0], &v[i]);
78     criaHeap(v, 0, i - 1);
79 }
80 }
81
82 int particao(int *v, int ini, int fim, int *comp, int *mov) {
83     int i = ini, j = fim;
84     int pivo = v[(ini + fim) / 2];
85     while (1) {
86         (*comp)++;
87         while (v[i] > pivo) {
88             i++;
89             (*comp)++;
90         }
91
92         (*comp)++;
93         while (v[j] < pivo) {
94             j--;
95             (*comp)++;
96         }
97
98         if (i < j) {
99             troca(&v[i], &v[j]);
100             (*mov)++;
101             i++;
102             j--;
103         } else {
104             return j;
105         }
106     }
107 }
108
109 void quickSort(int *v, int ini, int fim, int n, int *comp, int *mov) {
110     if (ini < fim) {
111         int q = particao(v, ini, fim, comp, mov);
112         quickSort(v, ini, q, n, comp, mov);
113         quickSort(v, q + 1, fim, n, comp, mov);
114     }
115 }
116
117 void merge(int *v, int ini, int meio, int fim) {
118     int tam = fim - ini + 1;
119     // Vetor Auxiliar - A
120     int *A = (int *)malloc(tam * sizeof(int));
121     int i = ini, j = meio + 1, k = 0;
122     while (i <= meio && j <= fim) {
123         if (v[i] > v[j]) {
124             A[k] = v[i];
125             i++;
126         } else {
127             A[k] = v[j];
128             j++;
129         }
130         k++;
131     }
132     while (i <= meio) {
133         A[k] = v[i];
134         i++;
135         k++;
136     }
137     while (j <= fim) {
138         A[k] = v[j];
139         j++;
140         k++;
141     }
142     for (i = ini, k = 0; i <= fim; i++, k++)
143         v[i] = A[k];
144     free(A);
145 }
146
147 void mergeSort(int *v, int ini, int fim) {
148     if (ini < fim) {
149         int meio = (ini + fim) / 2;
150         mergeSort(v, ini, meio);
151         mergeSort(v, meio + 1, fim);
152         merge(v, ini, meio, fim);
153     }
154 }
155
156 enum algoritmos_para_teste {
157     SHELL_SORT = 0,
```

```

158     QUICK_SORT,
159     MERGE_SORT,
160     HEAP_SORT
161 };
162
163 #define QTD_ALGORITMOS_TESTE 4
164
165 int main() {
166     int n;
167
168     printf("Digite o tamanho do vetor:");
169     scanf("%d", &n);
170
171     int *array_original = (int *)malloc(n * sizeof(int));
172
173     for (int i = 0; i < n; i++) {
174         scanf("%d", &array_original[i]);
175     }
176
177     for (int i = 0; i < QTD_ALGORITMOS_TESTE; i++) {
178         int *array_copia = copiaVetor(array_original, n);
179
180         int comp = 0, mov = 0;
181
182         switch (i) {
183             case SHELL_SORT:
184                 printf("Ordenando com Shell Sort:\n");
185                 shellSort(array_copia, n);
186                 imprimeVetor(array_copia, n);
187                 break;
188             case QUICK_SORT:
189                 printf("Ordenando com Quick Sort:\n");
190                 quickSort(array_copia, 0, n - 1, n, &comp, &mov);
191                 imprimeVetor(array_copia, n);
192                 break;
193             case MERGE_SORT:
194                 printf("Ordenando com Merge Sort:\n");
195                 mergeSort(array_copia, 0, n - 1);
196                 imprimeVetor(array_copia, n);
197                 break;
198             case HEAP_SORT:
199                 printf("Ordenando com Heap Sort:\n");
200                 heapSort(array_copia, n);
201                 imprimeVetor(array_copia, n);
202                 break;
203         }
204
205         liberar_vetor(&array_copia);
206     }
207
208     liberar_vetor(&array_original);
209     return 0;
210
211     return 0;
212 }

```

Saída do terminal:

```

└─ arthurdetomi at arthurdetomi-System-Product-Name
└─ ./1-2.out < in
Digite o tamanho do vetor:Ordenando com Shell Sort:
[77, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Ordenando com Quick Sort:
[77, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Ordenando com Merge Sort:
[77, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Ordenando com Heap Sort:
[77, 9, 8, 7, 6, 5, 4, 3, 2, 1]

```

1-3) Considerando os 3 metodos vistos, utilize um programa para ordenar grande quantidade de valores e verifique o aumento do numero de comparacoes, movimentacoes e tempo de execucao conforme se aumenta o volume de dados processados

roteiro_11/tempo.h

```

1  #ifndef TEMPO_H
2  #define TEMPO_H
3
4  #include <stdio.h>
5  #include <sys/resource.h>
6  #include <sys/time.h>
7

```

```

8 // Estrutura para armazenar os tempos de execução
9 typedef struct Temporizador {
10     struct timeval start_tv, end_tv;
11     struct rusage start_usage, end_usage;
12 } Temporizador;
13
14 // Inicia a contagem do temporizador
15 void iniciarTemporizador(Temporizador *t);
16
17 // Finaliza a contagem do temporizador
18 void finalizarTemporizador(Temporizador *t);
19
20 // Calcula o tempo real decorrido em segundos
21 double calcularTempoReal(Temporizador *t);
22
23 // Calcula o tempo de CPU em modo usuário em segundos
24 double calcularTempoUsuario(Temporizador *t);
25
26 // Calcula o tempo de CPU em modo sistema/kernel em segundos
27 double calcularTempoSistema(Temporizador *t);
28
29 // Imprime todos os tempos medidos de forma formatada
30 void imprimirTempos(Temporizador *t);
31
32 #endif

```

roteiro_11/tempo.c

```

1 #include "tempo.h"
2
3 // Inicia a medição do tempo, armazenando os valores atuais
4 void iniciarTemporizador(Temporizador *t) {
5     gettimeofday(&t->start_tv, NULL);
6     getrusage(RUSAGE_SELF, &t->start_usage);
7 }
8
9 // Finaliza a medição do tempo, armazenando os valores finais
10 void finalizarTemporizador(Temporizador *t) {
11     gettimeofday(&t->end_tv, NULL);
12     getrusage(RUSAGE_SELF, &t->end_usage);
13 }
14
15 double calcularTempo(struct timeval inicio, struct timeval fim) {
16     time_t seg = fim.tv_sec - inicio.tv_sec;
17     suseconds_t microseg = fim.tv_usec - inicio.tv_usec;
18
19     // Ajusta caso microsegundos do fim sejam menores que os do início
20     if (microseg < 0) {
21         seg -= 1;
22         microseg += 1000000;
23     }
24
25     return (double)seg + (double)microseg / 1e6;
26 }
27
28 // Calcula diferença entre tempos reais (start e end) em segundos
29 double calcularTempoReal(Temporizador *t) {
30     return calcularTempo(t->start_tv, t->end_tv);
31 }
32
33 // Calcula diferença entre tempos de sistema (start e end) em segundos
34 double calcularTempoSistema(Temporizador *t) {
35     return calcularTempo(t->start_usage.ru_stime, t->end_usage.ru_stime);
36 }
37
38 // Imprime todos os tempos de execução de forma legível
39 void imprimirTempos(Temporizador *t) {
40     printf("Tempo de execução:\n");
41     printf("Tempo Real: %.8f segundos\n", calcularTempoReal(t));
42     printf("Tempo Sistema: %.8f segundos\n", calcularTempoSistema(t));
43 }

```

roteiro_11/1-3.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "tempo.h"
5
6 void liberar_vetor(int **v) {
7     free(*v);
8     *v = NULL;

```

```
9 | }
10
11 int *copiaVetor(int *v, int n) {
12     int i;
13     int *v2 = (int *)malloc(n * sizeof(int));
14     for (i = 0; i < n; i++)
15         v2[i] = v[i];
16     return v2;
17 }
18
19 void imprimeVetor(int *v, int n) {
20     int i, prim = 1;
21     printf("[");
22     for (i = 0; i < n; i++)
23         if (prim) {
24             printf("%d", v[i]);
25             prim = 0;
26         } else
27             printf(", %d", v[i]);
28     printf("]\n");
29 }
30
31 void troca(int *a, int *b) {
32     int aux = *a;
33     *a = *b;
34     *b = aux;
35 }
36
37 void shellSort(int *v, int n, int *comp, int *mov) {
38     int i, j, atual;
39     int h = 1;
40     while (h < n)
41         h = 3 * h + 1;
42     while (h > 0) {
43         for (i = h; i < n; i++) {
44             atual = v[i];
45             j = i;
46             while (j > h - 1 && atual <= v[j - h]) {
47                 (*comp)++;
48                 (*mov)++;
49                 v[j] = v[j - h];
50                 j = j - h;
51             }
52             (*comp)++;
53             v[j] = atual;
54             (*mov)++;
55         }
56         h = h / 3;
57     }
58 }
59
60 void criaHeap(int *v, int pai, int fim, int *comp, int *mov) {
61     int aux = v[pai];
62     int filho = 2 * pai + 1;
63     while (filho <= fim) {
64         if (filho < fim) {
65             (*comp)++;
66             if (v[filho] < v[filho + 1]) {
67                 filho++;
68             }
69         }
70         (*comp)++;
71         if (aux < v[filho]) {
72             v[pai] = v[filho];
73             (*mov)++;
74             pai = filho;
75             filho = 2 * pai + 1;
76         } else {
77             break;
78         }
79     }
80     v[pai] = aux;
81     (*mov)++;
82 }
83
84 void heapSort(int *v, int n, int *comp, int *mov) {
85     for (int i = (n - 1) / 2; i >= 0; i--)
86         criaHeap(v, i, n - 1, comp, mov);
87     for (int i = n - 1; i >= 1; i--) {
88         troca(&v[0], &v[i]);
89         (*mov) += 2;
```



```
90     criaHeap(v, 0, i - 1, comp, mov);
91 }
92 }
93
94 int particao(int *v, int ini, int fim, int *comp, int *mov) {
95     int i = ini, j = fim;
96     int pivo = v[(ini + fim) / 2];
97     while (1) {
98         while (v[i] < pivo) {
99             i++;
100             (*comp)++;
101         }
102
103         while (v[j] > pivo) {
104             j--;
105             (*comp)++;
106         }
107
108         (*comp)++;
109         if (i < j) {
110             troca(&v[i], &v[j]);
111             (*mov)++;
112             i++;
113             j--;
114         } else {
115             return j;
116         }
117     }
118 }
119
120 void quickSort(int *v, int ini, int fim, int *comp, int *mov) {
121     if (ini < fim) {
122         int q = particao(v, ini, fim, comp, mov);
123         quickSort(v, ini, q, comp, mov);
124         quickSort(v, q + 1, fim, comp, mov);
125     }
126 }
127
128 void merge(int *v, int ini, int meio, int fim, int *comp, int *mov) {
129     int tam = fim - ini + 1;
130     int *A = (int *)malloc(tam * sizeof(int));
131     int i = ini, j = meio + 1, k = 0;
132     while (i <= meio && j <= fim) {
133         (*comp)++;
134         if (v[i] < v[j]) {
135             A[k++] = v[i++];
136             (*mov)++;
137         } else {
138             A[k++] = v[j++];
139             (*mov)++;
140         }
141     }
142     while (i <= meio) {
143         A[k++] = v[i++];
144         (*mov)++;
145     }
146     while (j <= fim) {
147         A[k++] = v[j++];
148         (*mov)++;
149     }
150     for (i = ini, k = 0; i <= fim; i++, k++) {
151         v[i] = A[k];
152         (*mov)++;
153     }
154     free(A);
155 }
156
157 void mergeSort(int *v, int ini, int fim, int *comp, int *mov) {
158     if (ini < fim) {
159         int meio = (ini + fim) / 2;
160         mergeSort(v, ini, meio, comp, mov);
161         mergeSort(v, meio + 1, fim, comp, mov);
162         merge(v, ini, meio, fim, comp, mov);
163     }
164 }
165
166 enum algoritmos_para_teste {
167     SHELL_SORT = 0,
168     QUICK_SORT,
169     MERGE_SORT,
170     HEAP_SORT
```

```
171 };
172
173 #define QTD_ALGORITMOS_TESTE 4
174
175 int main() {
176     int n;
177     printf("Digite o tamanho do vetor:");
178     scanf("%d", &n);
179
180     int *array_original = (int *)malloc(n * sizeof(int));
181
182     for (int i = 0; i < n; i++) {
183         scanf("%d", &array_original[i]);
184     }
185
186     for (int i = 0; i < QTD_ALGORITMOS_TESTE; i++) {
187         int *array_copia = copiaVetor(array_original, n);
188         int comp = 0, mov = 0;
189
190         Temporizador tempo_teste;
191         iniciarTemporizador(&tempo_teste);
192
193         switch (i) {
194             case SHELL_SORT:
195                 printf("\nOrdenando com Shell Sort:\n");
196                 shellSort(array_copia, n, &comp, &mov);
197                 break;
198             case QUICK_SORT:
199                 printf("\nOrdenando com Quick Sort:\n");
200                 quickSort(array_copia, 0, n - 1, &comp, &mov);
201                 break;
202             case MERGE_SORT:
203                 printf("\nOrdenando com Merge Sort:\n");
204                 mergeSort(array_copia, 0, n - 1, &comp, &mov);
205                 break;
206             case HEAP_SORT:
207                 printf("\nOrdenando com Heap Sort:\n");
208                 heapSort(array_copia, n, &comp, &mov);
209                 break;
210         }
211
212         finalizarTemporizador(&tempo_teste);
213
214         imprimeVetor(array_copia, n);
215         printf("Quantidade de movimentações realizadas: %d\n", mov);
216         printf("Quantidade de comparações realizadas: %d\n", comp);
217         imprimirTempos(&tempo_teste);
218         printf("-----\n");
219
220         liberar_vetor(&array_copia);
221     }
222
223     liberar_vetor(&array_original);
224     return 0;
225 }
226
```

Saída do terminal:

```

[arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2/roteiro_11 on mainxxx 25-06-16]
$ ./1-3.out < input/100-misturado.txt
Digite o tamanho do vetor:
Ordenando com Shell Sort:
[2, 3, 4, 4, 9, 10, 12, 12, 14, 17, 18, 21, 25, 25, 27, 28, 29, 29, 33, 34, 38, 38, 39, 40, 42, 43, 43, 51, 52, 58, 62, 66, 67, 67, 68, 69, 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]
Quantidade de movimentações realizadas: 793
Quantidade de comparações realizadas: 793
Tempo de execução:
Tempo Real: 0.00001300 segundos
Tempo Sistema: 0.00001300 segundos
-----

Ordenando com Quick Sort:
[2, 3, 4, 4, 9, 10, 12, 12, 14, 17, 18, 21, 25, 25, 27, 28, 29, 29, 33, 34, 38, 38, 39, 40, 42, 43, 43, 51, 52, 58, 62, 66, 67, 67, 68, 69, 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]
Quantidade de movimentações realizadas: 164
Quantidade de comparações realizadas: 674
Tempo de execução:
Tempo Real: 0.00001100 segundos
Tempo Sistema: 0.00001100 segundos
-----

Ordenando com Merge Sort:
[2, 3, 4, 4, 9, 10, 12, 12, 14, 17, 18, 21, 25, 25, 27, 28, 29, 29, 33, 34, 38, 38, 39, 40, 42, 43, 43, 51, 52, 58, 62, 66, 67, 67, 68, 69, 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]
Quantidade de movimentações realizadas: 1344
Quantidade de comparações realizadas: 542
Tempo de execução:
Tempo Real: 0.00001400 segundos
Tempo Sistema: 0.00001400 segundos
-----

Ordenando com Heap Sort:
[2, 3, 4, 4, 9, 10, 12, 12, 14, 17, 18, 21, 25, 25, 27, 28, 29, 29, 33, 34, 38, 38, 39, 40, 42, 43, 43, 51, 52, 58, 62, 66, 67, 67, 68, 69, 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]
Quantidade de movimentações realizadas: 824
Quantidade de comparações realizadas: 1025
Tempo de execução:
Tempo Real: 0.00001000 segundos
Tempo Sistema: 0.00000900 segundos
-----

[arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2/roteiro_11 on mainxxx 25-06-16]
$ ./1-3.out < input/100-ordenado.txt
Digite o tamanho do vetor:
Ordenando com Shell Sort:
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 100]
Quantidade de movimentações realizadas: 342
Quantidade de comparações realizadas: 342
Tempo de execução:
Tempo Real: 0.00001100 segundos
Tempo Sistema: 0.00001100 segundos
-----

Ordenando com Quick Sort:
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 100]
Quantidade de movimentações realizadas: 0
Quantidade de comparações realizadas: 672
Tempo de execução:
Tempo Real: 0.00000800 segundos
Tempo Sistema: 0.00000800 segundos
-----

Ordenando com Merge Sort:
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 100]
Quantidade de movimentações realizadas: 1344
Quantidade de comparações realizadas: 356
Tempo de execução:
Tempo Real: 0.00001300 segundos
Tempo Sistema: 0.00001300 segundos
-----

Ordenando com Heap Sort:
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 100]
Quantidade de movimentações realizadas: 888
Quantidade de comparações realizadas: 1081
Tempo de execução:
Tempo Real: 0.00002700 segundos
Tempo Sistema: 0.00000000 segundos
-----

```