

Algoritmos de Ordenação

1.

↙
Baseados em comparações

↘
não baseados em
comparação

↓
Elementares

- insertion sort
- selection sort
- bubble sort
- merge sort

↓
Não elementares

- quick sort
- heap sort

Algoritmos baseados em comparação

2.

↳ Para dados distribuídos de forma eletrônica, não é possível ter um tempo de execução inferior ao tempo linear $O(N \times \log_2 N)$

→ N é o número de elementos do array a ordenar

Alg. não baseados em comparação

↳ embora não se apliquem sempre, quando se podem aplicar, sob determinadas condições, conseguem executar em tempo linear $O(N)$

Insertion sort

3.

A SORTING EXAMPLE

A 5 ORTING ...

A 0 S RTING ...

A 0 R S TING ...

✓

zona onde

vai inserir

elemento corrente

por ordem. No fim

de cada iteração, esta

zona fica ordenada

Funções auxiliares

```
fun less(x: Chan, y: Chan):
```

```
Boolean = x < y
```

```
fun exch(a: ChanArray,  
        i: Int, j: Int) {
```

```
    val aux = a[i]
```

```
    a[i] = a[j]
```

```
    a[j] = aux  
}
```

main

val arr: CharArray(5)

arr[0] = 'A'

arr[1] = 'S'

arr[2] = 'O'

arr[3] = 'R'

arr[4] = 'T'

print(arr)

Não altera
o array

fun print(a: CharArray)

{
fun ...

← função recebe
uma cópia de arr,

ou recebe o array
original?

Recebe o array original,

passado numa nova referência
ou ponteiro ←

↓
endereço do array na
memória heap

Memória

STACK

arr

0x123

HEAP

0

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

```

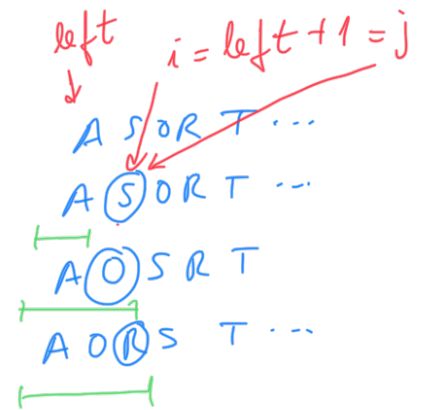
fun lessExch (a: ChanArray, i: Int, j: Int) {
    if (less(a[i], a[j]))
        exch(a, i, j)
}

```

```

fun insertionSort1(a: ChanArray, left: Int, right: Int)
{
    var i = left + 1
    while (i <= right) {
        var j = i
        while (j > left) {
            lessExch(a, j, j-1)
        }
        ++i
    }
}

```



```

{
    var optimization:
        insertionSort2
}

```

Complexity temporal: $O(n^2)$

first case:

$$n = \text{right} - \text{left} + 1$$

ordendo de forma decrecente

1 compensação para trocar \rightarrow janela já ordenada

+ " " "

2

+ " "

3

+

:

i

\sim soma dos n termos de uma
série aritmética

$$= \sum_{i=1}^n i = \frac{n \times (n+1)}{2} = \frac{n^2}{2} + \frac{n}{2} \sim n^2$$

\downarrow
 $n \rightarrow \infty$

melhor caso: ordenado de forma crescente

7.

1 comparações para três

+
1 " " "

+
1 " " "

+

;

1

" $O(n)$