Белгородский Государственный Технологический Университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №3 по теме: «Сравнительный анализ методов сортировки»

Выполнил:  
студент группы ПВ-21  
Адаменко И. И.

Проверил:  
профессор  
Синюк В. Г.

Белгород  
2013

Цель работы: изучение методов сортировки массивов и приобретение навыков в проведении сравнительного анализа различных методов сортировки.

## Задания

1. Изучить временные характеристики алгоритмов.
2. Изучить методы сортировки:
   1. вставками;
   2. выбором;
   3. обменом;
      1. улучшенная обменом (первая);
      2. улучшенная обменом (вторая);
   4. Шелла;
   5. Хоара;
   6. пирамидальная.
3. Программно реализовать методы сортировки массивов.
4. Разработать и программно реализовать средство для проведения экспериментов по определению временных характеристик алгоритмов сортировки.
5. Провести эксперименты по определению временных характеристик алгоритмов сортировки. Результаты экспериментов представить в виде таблицы 9, клетки которой содержат время выполнения алгоритма сортировки массива с заданным количеством элементов. Провести эксперимент для упорядоченных, неупорядоченных и упорядоченных в обратном порядке массивов (для каждого типа массива заполнить отдельную таблицу).
6. Построить графики функций временной сложности алгоритмов сортировки.
7. Определить аналитическое выражение функции временной сложности алгоритмов сортировки.
8. Определить порядок функций временной сложности алгоритмов сортировки при сортировке упорядоченных, неупорядоченных и упорядоченных в обратном порядке массивов.

## Программная реализация методов сортировки массивов (модуль)

|  |  |
| --- | --- |
| 001 | **UNIT** SORT; |
| 002 |  |
| 003 | *{------------------------------------------------------------------------}* |
| 004 | INTERFACE |
| 005 | *{------------------------------------------------------------------------}* |
| 006 |  |
| 007 | **uses** Windows, CRT; |
| 008 |  |
| 009 | **const** |
| 010 | MAXSTACK = 1024; |
| 011 | RANDLEFT = 0; |
| 012 | RANDRIGHT = 65000; |
| 013 |  |
| 014 | **type** |
| 015 | TBase = **word**; |
| 016 | TSize = **word**; |
| 017 | TArray = **array of** TBase; |
| 018 | T**Procedure** = **procedure**(**var** a: TArray; size: TSize); |
| 019 |  |
| 020 | **function** countTime(**var** a: TArray; size: TSize; sort: TProcedure):  **cardinal**; |
| 021 | **function** checkSort(**var** a: TArray; size: TSize): **boolean**; |
| 022 |  |
| 023 | **procedure** generateUnsort(**var** a: TArray; size: TSize); |
| 024 | **procedure** generateSort(**var** a: TArray; size: TSize); |
| 025 | **procedure** generateSortReverse(**var** a: TArray; size: TSize); |
| 026 | **procedure** insertSort(**var** a: TArray; size: TSize); |
| 027 | **procedure** selectSort(**var** a: TArray; size: TSize); |
| 028 | **procedure** bubbleSort(**var** a: TArray; size: TSize); |
| 029 | **procedure** bubbleSortImprovedOne(**var** a: TArray; size: TSize); |
| 030 | **procedure** bubbleSortImprovedTwo(**var** a: TArray; size: TSize); |
| 031 | **procedure** ShellSort(**var** a: TArray; size: TSize); |
| 032 | **procedure** quickSort(**var** a: TArray; size: TSize); |
| 033 | **procedure** heapSort(**var** a: TArray; size: TSize); |
| 034 | **procedure** outputArray(**var** a: TArray; size: TSize;   nameOfFile: **string**); |
| 035 |  |
| 036 | *{------------------------------------------------------------------------}* |
| 037 | IMPLEMENTATION |
| 038 | *{------------------------------------------------------------------------}* |
| 039 |  |
| 040 | **function** countTime(**var** a: TArray; size: TSize; sort: T**Procedure**):  **cardinal**; |
| 041 | **var** |
| 042 | start: **cardinal**; |
| 043 |  |
| 044 | **begin** |
| 045 |  |
| 046 | start := GetCurrentTime; |
| 047 | sort(a, size); |
| 048 | countTime := GetCurrentTime - start; |
| 049 |  |
| 050 | **end**; |
| 051 |  |
| 052 | **function** checkSort(**var** a: TArray; size: TSize): **boolean**; |
| 053 | **var** |
| 054 | i: TSize; |
| 055 | flag: **boolean**; |
| 056 |  |
| 057 | **begin** |
| 058 |  |
| 059 | flag := TRUE; |
| 060 | i := 1; |
| 061 |  |
| 062 | **while** ((i <= size) **and** flag) **do** |
| 063 | **begin** |
| 064 | flag := (a[i] >= a[i - 1]); |
| 065 | inc(i); |
| 066 | **end**; |
| 067 |  |
| 068 | checkSort := flag; |
| 069 |  |
| 070 | **end**; |
| 071 |  |
| 072 | **procedure** generateUnsort(**var** a: TArray; size: TSize); |
| 073 | **var** |
| 074 | i: TSize; |
| 075 |  |
| 076 | **begin** |
| 077 |  |
| 078 | randomize; |
| 079 | **for** i := 0 **to** size **do** |
| 080 | a[i] := random(RANDRIGHT - RANDLEFT) + RANDLEFT; |
| 081 |  |
| 082 | **end**; |
| 083 |  |
| 084 | **procedure** generateSort(**var** a: TArray; size: TSize); |
| 085 | **var** |
| 086 | i: TSize; |
| 087 |  |
| 088 | **begin** |
| 089 |  |
| 090 | **for** i := 0 **to** size **do** |
| 091 | a[i] := i; |
| 092 |  |
| 093 | **end**; |
| 094 |  |
| 095 | **procedure** generateSortReverse(**var** a: TArray; size: TSize); |
| 096 | **var** |
| 097 | i: TSize; |
| 098 |  |
| 099 | **begin** |
| 100 |  |
| 101 | **for** i := 0 **to** size **do** |
| 102 | a[i] := size - i; |
| 103 |  |
| 104 | **end**; |
| 105 |  |
| 106 | **procedure** exchange(**var** a, b: TBase); |
| 107 | **var** |
| 108 | tmp: TBase; |
| 109 |  |
| 110 | **begin** |
| 111 |  |
| 112 | tmp := a; |
| 113 | a := b; |
| 114 | b := tmp; |
| 115 |  |
| 116 | **end**; |
| 117 |  |
| 118 | **procedure** compareExchange(**var** a, b: TBase); |
| 119 | **begin** |
| 120 |  |
| 121 | **if** (b < a) **then** |
| 122 | exchange(a, b); |
| 123 |  |
| 124 | **end**; |
| 125 |  |
| 126 | **function** bCompareExchange(**var** a, b: TBase): **boolean**; |
| 127 | **begin** |
| 128 |  |
| 129 | bCompareExchange := FALSE; |
| 130 | **if** (b < a) **then** |
| 131 | **begin** |
| 132 | exchange(a, b); |
| 133 | bCompareExchange := TRUE; |
| 134 | **end**; |
| 135 |  |
| 136 | **end**; |
| 137 |  |
| 138 | **procedure** insertSort(**var** a: TArray; size: TSize); |
| 139 | **var** |
| 140 | i, j: TSize; |
| 141 | tmp: TBase; |
| 142 |  |
| 143 | **begin** |
| 144 |  |
| 145 | **for** i := size **downto** 1 **do** |
| 146 | **if** (a[i] < a[i - 1]) **then** |
| 147 | compareExchange(a[i - 1], a[i]); |
| 148 |  |
| 149 | **for** i := 2 **to** size **do** |
| 150 | **begin** |
| 151 | j := i; |
| 152 | tmp := a[i]; |
| 153 |  |
| 154 | **while** (tmp < a[j - 1]) **do** |
| 155 | **begin** |
| 156 | a[j] := a[j - 1]; |
| 157 | dec(j); |
| 158 | **end**; |
| 159 |  |
| 160 | a[j] := tmp; |
| 161 | **end**; |
| 162 |  |
| 163 | **end**; |
| 164 |  |
| 165 | **procedure** selectSort(**var** a: TArray; size: TSize); |
| 166 | **var** |
| 167 | i, min, j: TSize; |
| 168 |  |
| 169 | **begin** |
| 170 |  |
| 171 | **for** i := 0 **to** size - 1 **do** |
| 172 | **begin** |
| 173 | min := i; |
| 174 |  |
| 175 | **for** j := i + 1 **to** size **do** |
| 176 | **begin** |
| 177 | **if** (a[j] < a[min]) **then** |
| 178 | min := j; |
| 179 |  |
| 180 | exchange(a[i], a[min]); |
| 181 | **end**; |
| 182 | **end**; |
| 183 |  |
| 184 | **end**; |
| 185 |  |
| 186 | **procedure** bubbleSort(**var** a: TArray; size: TSize); |
| 187 | **var** |
| 188 | i, j: TSize; |
| 189 |  |
| 190 | **begin** |
| 191 |  |
| 192 | **for** i := 0 **to** size - 1 **do** |
| 193 | **for** j := size **downto** 1 **do** |
| 194 | compareExchange(a[j - 1], a[j]); |
| 195 |  |
| 196 | **end**; |
| 197 |  |
| 198 | **procedure** bubbleSortImprovedOne(**var** a: TArray; size: TSize); |
| 199 | **var** |
| 200 | flag: **boolean**; |
| 201 | i, j: TSize; |
| 202 |  |
| 203 | **begin** |
| 204 |  |
| 205 | **for** i := 0 **to** size - 1 **do** |
| 206 | **begin** |
| 207 | flag := FALSE; |
| 208 |  |
| 209 | **for** j := size **downto** 1 **do** |
| 210 | **if** (bCompareExchange(a[j - 1], a[j]))   **then** |
| 211 | flag := TRUE; |
| 212 |  |
| 213 | **if** (**not** flag) **then** break; |
| 214 | **end**; |
| 215 |  |
| 216 | **end**; |
| 217 |  |
| 218 | **procedure** bubbleSortImprovedTwo(**var** a: TArray; size: TSize); |
| 219 | **var** |
| 220 | flag: **boolean**; |
| 221 | i, j, last: TSize; |
| 222 |  |
| 223 | **begin** |
| 224 |  |
| 225 | last := 1; |
| 226 | **for** i := 0 **to** size - 1 **do** |
| 227 | **begin** |
| 228 | flag := FALSE; |
| 229 |  |
| 230 | **for** j := size **downto** last **do** |
| 231 | **if** (bCompareExchange(a[j - 1], a[j]))   **then** |
| 232 | **begin** |
| 233 | last := j - 1; |
| 234 | flag := TRUE; |
| 235 | **end**; |
| 236 |  |
| 237 | **if** (**not** flag) **then** break; |
| 238 | **end**; |
| 239 |  |
| 240 | **end**; |
| 241 |  |
| 242 | **procedure** ShellSort(**var** a: TArray; size: TSize); |
| 243 | **var** |
| 244 | h, i, j: TSize; |
| 245 | tmp: TBase; |
| 246 |  |
| 247 | **begin** |
| 248 |  |
| 249 | h := 1; |
| 250 | **while** (h <= (size **div** 9)) **do** |
| 251 | h := 3 \* h + 1; |
| 252 |  |
| 253 | **while** (h > 0) **do** |
| 254 | **begin** |
| 255 | h := h **div** 3; |
| 256 |  |
| 257 | **for** i := h **to** size **do** |
| 258 | **begin** |
| 259 | j := i; |
| 260 | tmp := a[i]; |
| 261 | **while** ((j >= h)   **and** (tmp < a[j - h])) **do** |
| 262 | **begin** |
| 263 | a[j] := a[j - h]; |
| 264 | dec(j, h); |
| 265 | **end**; |
| 266 |  |
| 267 | a[j] := tmp; |
| 268 | **end**; |
| 269 | **end**; |
| 270 |  |
| 271 | **end**; |
| 272 |  |
| 273 | **procedure** quickSort(**var** a: TArray; size: TSize); |
| 274 | **procedure** quickSortR(left, right: longint); |
| 275 | **var** |
| 276 | i, j: longint; |
| 277 | pivot: TBase; |
| 278 |  |
| 279 | **begin** |
| 280 |  |
| 281 | i := left; |
| 282 | j := right; |
| 283 | pivot := a[(left + right) **div** 2]; |
| 284 |  |
| 285 | **repeat** |
| 286 |  |
| 287 | **while** (a[i] < pivot) **do** inc(i); |
| 288 | **while** (pivot < a[j]) **do** dec(j); |
| 289 |  |
| 290 | **if** (i <= j) **then** |
| 291 | **begin** |
| 292 | exchange(a[i], a[j]); |
| 293 | inc(i); |
| 294 | dec(j); |
| 295 | **end**; |
| 296 | **until** (i > j); |
| 297 |  |
| 298 | **if** (left < j) **then** quickSortR(left, j); |
| 299 | **if** (i < right) **then** quickSortR(i, right); |
| 300 |  |
| 301 | **end**; |
| 302 |  |
| 303 | **begin** |
| 304 |  |
| 305 | randomize; |
| 306 | quickSortR(0, size); |
| 307 |  |
| 308 | **end**; |
| 309 |  |
| 310 | **procedure** heapSort(**var** a: TArray; size: TSize); |
| 311 | **procedure** downHeap(left, right: TSize); |
| 312 | **var** |
| 313 | newEl: TBase; |
| 314 | child: TSize; |
| 315 |  |
| 316 | **begin** |
| 317 |  |
| 318 | newEl := a[left]; |
| 319 |  |
| 320 | **while** (left <= (right **div** 2)) **do** |
| 321 | **begin** |
| 322 | child := 2 \* left; |
| 323 |  |
| 324 | **if** ((child < right)  **and** (a[child] < a[child + 1])) **then** |
| 325 | inc(child); |
| 326 |  |
| 327 | **if** (newEl >= a[child]) **then** break; |
| 328 |  |
| 329 | a[left] := a[child]; |
| 330 | left := child; |
| 331 | **end**; |
| 332 | a[left] := newEl; |
| 333 |  |
| 334 | **end**; |
| 335 |  |
| 336 | **var** |
| 337 | i: TBase; |
| 338 |  |
| 339 | **begin** |
| 340 |  |
| 341 | **for** i := (size **div** 2 - 1) **downto** 0 **do** |
| 342 | downHeap(i, size); |
| 343 |  |
| 344 | **for** i := size **downto** 1 **do** |
| 345 | **begin** |
| 346 | exchange(a[i], a[0]); |
| 347 | downHeap(0, i - 1); |
| 348 | **end**; |
| 349 |  |
| 350 | **end**; |
| 351 |  |
| 352 | **procedure** outputArray(**var** a: TArray; size: **word**; nameOfFile: **string**); |
| 353 | **var** |
| 354 | f: **text**; |
| 355 | i: **word**; |
| 356 |  |
| 357 | **begin** |
| 358 |  |
| 359 | assign(f, nameOfFile); |
| 360 | rewrite(f); |
| 361 |  |
| 362 | **for** i := 0 **to** size **do** |
| 363 | write(f, a[i], ' '); |
| 364 |  |
| 365 | writeln(*'Array is written to '*, nameOfFile); |
| 366 |  |
| 367 | close(f); |
| 368 |  |
| 369 | **end**; |
| 370 |  |
| 371 | **end**. |

## Программная реализация методов сортировки массивов (главная программа)

|  |  |
| --- | --- |
| 01 | **program** sd\_lab\_3; |
| 02 |  |
| 03 | **uses** SORT, CRT; |
| 04 |  |
| 05 | **var** |
| 06 | a: TArray; |
| 07 | size: TSize; |
| 08 |  |
| 09 | **begin** |
| 10 |  |
| 11 | size := 4500; |
| 12 | setlength(a, size); |
| 13 | dec(size); |
| 14 |  |
| 15 | generateUnsort(a, size); |
| 16 |  |
| 17 | outputArray(a, size, *'beforeSort.txt'*); |
| 18 | writeln(*'Time: '*, countTime(a, size, @heapSort) / 1000 : 0 : 3); |
| 19 |  |
| 20 | writeln(*'Sorted: '*, checkSort(a, size)); |
| 21 | outputArray(a, size, *'afterSort.txt'*); |
| 22 | readln; |
| 23 |  |
| 24 | **end**. |

## Результаты экспериментов

### Неупорядоченные массивы

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сортировка | Количество элементов в массиве | | | | | | | | |
| 500 | 1000 | 1500 | 2000 | 2500 | 3000 | 3500 | 4000 | 4500 |
| Включением | 0.001 | 0.007 | 0.016 | 0.016 | 0.018 | 0.023 | 0.031 | 0.035 | 0.038 |
| Выбором | 0.001 | 0.009 | 0.016 | 0.031 | 0.062 | 0.078 | 0.110 | 0.156 | 0.172 |
| Обменом | 0.015 | 0.023 | 0.031 | 0.047 | 0.078 | 0.109 | 0.156 | 0.172 | 0.234 |
| Обменом (улучшен. первая) | 0.010 | 0.016 | 0.031 | 0.063 | 0.094 | 0.125 | 0.172 | 0.204 | 0.250 |
| Обменом (улучшен. вторая) | 0.008 | 0.016 | 0.032 | 0.040 | 0.046 | 0.078 | 0.110 | 0.156 | 0.203 |
| Шелла | 0.001 | 0.001 | 0.001 | 0.001 | 0.002 | 0.005 | 0.011 | 0.015 | 0.023 |
| Хоара | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.002 | 0.002 |
| Пирамидальная | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.002 | 0.002 | 0.003 |

### Упорядоченные массивы

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сортировка | Количество элементов в массиве | | | | | | | | |
| 500 | 1000 | 1500 | 2000 | 2500 | 3000 | 3500 | 4000 | 4500 |
| Включением | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| Выбором | 0.001 | 0.009 | 0.016 | 0.031 | 0.060 | 0.075 | 0.101 | 0.120 | 0.141 |
| Обменом | 0.005 | 0.010 | 0.014 | 0.023 | 0.042 | 0.061 | 0.084 | 0.101 | 0.125 |
| Обменом (улучшен. первая) | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| Обменом (улучшен. вторая) | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| Шелла | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| Хоара | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| Пирамидальная | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |

### Упорядоченные в обратном порядке массивы

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сортировка | Количество элементов в массиве | | | | | | | | |
| 500 | 1000 | 1500 | 2000 | 2500 | 3000 | 3500 | 4000 | 4500 |
| Включением | 0.001 | 0.007 | 0.016 | 0.018 | 0.021 | 0.027 | 0.040 | 0.062 | 0.072 |
| Выбором | 0.001 | 0.009 | 0.016 | 0.031 | 0.052 | 0.061 | 0.106 | 0.121 | 0.140 |
| Обменом | 0.015 | 0.023 | 0.031 | 0.047 | 0.078 | 0.109 | 0.156 | 0.172 | 0.234 |
| Обменом (улучшен. первая) | 0.010 | 0.016 | 0.031 | 0.063 | 0.094 | 0.125 | 0.172 | 0.204 | 0.250 |
| Обменом (улучшен. вторая) | 0.008 | 0.016 | 0.032 | 0.040 | 0.046 | 0.078 | 0.110 | 0.156 | 0.203 |
| Шелла | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| Хоара | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| Пирамидальная | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |

## Аналитические выражения

* Сортировка включением:
* Сортировка выбором:
* Сортировка обменом:
* Улучшенная сортировка обменом (первая):
* Улучшенная сортировка обменом (вторая):
* Сортировка Шелла:
* Сортировка Хоара:
* Пирамидальная сортировка:

## Порядок функций временной сложности

* Сортировка включением:
* Сортировка выбором:
* Сортировка обменом:
* Улучшенная сортировка обменом (первая):
* Улучшенная сортировка обменом (вторая):
* Сортировка Шелла:
* Сортировка Хоара:
* Сортировка включением: