МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Программирование»
Тема: Обзор стандартной библиотеки

Студент гр. 9381 Колованов Р.А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Работа с функциями стандартной библиотеки языка С; получение навыков работы с ними.

Задание.

Вариант 2.

Напишите программу, на вход которой подается массив целых чисел длины 1000**,** при этом число 0 либо встречается один раз, либо не встречается.

Программа должна совершать следующие действия:

- отсортировать массив, используя алгоритм быстрой сортировки (см. функции стандартной библиотеки**)**
- определить, присутствует ли в массиве число 0, используя алгоритм двоичного поиска (для реализации алгоритма двоичного поиска используйте функцию стандартной библиотеки**)**
- посчитать время, за которое совершен поиск числа 0, используя при этом функцию стандартной библиотеки
- вывести строку "exists", если ноль в массиве есть и "doesn't exist" в противном случае
- вывести время, за которое был совершен двоичный поиск
- определить, присутствует ли в массиве число 0, используя перебор всех чисел массива
- посчитать время, за которое совершен поиск числа 0 перебором, используя при этом функцию стандартной библиотеки
- вывести строку "exists", если **0** в массиве есть и "doesn't exist" в противном случае
- вывести время, за которое была совершен поиск перебором.

Выполнение работы.

Для того, чтобы определить время работы алгоритма, используется функция *clock()* из стандартной библиотек. Сохранив значения функции *clock()* до и после выполнения алгоритма, получим число тактов с начала работы программы до и после выполнения алгоритма. Тогда если найти их разность, то можно получить количество тактов, за которое выполнился алгоритм. Для того, чтобы найти время выполнения алгоритма в секундах, требуется поделить количество тактов на константу *CLOCKS_PER_SEC*.

Для поиска элемента при помощи функции *bsearch()* стандартной библиотеки для начала требуется отсортировать массив. Для сортировки используется функция *qsort()*, которая требует для себя функцию-компаратор. Была написана следующая функция-компаратор:

```
int comparator(const void* x1, const void* x2) {
     const int* v1 = (const int*)x1;
    const int* v2 = (const int*)x2;

    if (*v1 > *v2) {
        return 1;
    } else if (*v1 < *v2) {
        return -1;
    } else {
        return 0;
    }
}
```

Функция-компаратор принимает на вход два указателя на неопределенный тип, которые нужно преобразовать к нужному типу и сравнить числа по эти указателям. Если первое число больше второго, то нужно вернуть положительное число, если второе число больше первого – нужно вернуть -1. Если числа равны, то нужно вернуть 0.

Дальше используя функцию-компаратор выполняется сортировка массива входных данных *array*. Далее производится получение количества тактов в *time1* до выполнения сортировки функцией *bsearch(),* после выполняется поиск при помощи *bsearch(),* после получается количество тактов *time2* после выполнения функции. Если функция bsearch() вернет NULL, то это

значит, что элемент не найдет в тексте. В таком случае выводим *“doesn't exist”*, иначе выводим *“exists”*. После этого выводим полученное время:

```
printf("%f\n", (time2 - time1) / CLOCKS_PER_SEC);
```

Аналогичные действия выполняем для поиска элемента при помощи перебора элементов. Выводим полученные результаты на экран.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	399 439 45 285 476 311 251 328 425 309 281 182 410 278 241 91 236 144 341 236 252 77 58 331 198 12 260 457 176 307 117 482 6 483 233 105 306 312 498 249 73 157 115 450 42 74 10 244 198 61 137 408 187 104 78 358 440 237 129 206 90 152 165 126 479 287 234 445 418 270 80 426 463 407 3 9 20 173 50 480 3 241 380 338 171 254 238 52 322 287 426 116 279 415 321 261 495 157 284 493 197 107 157 118 92 88 122 315 234 315 173 433 378 253 141 278 245 390 217 116 373 214 221 363 168 220 352 16 41 207 309 123 400 279 279 169 427 150 444 479 68 269 402 101 282 262 347 232 165 424 393 423 249 78 397 129 247 152 38 13 440 20 410 217 269 485 237 15 336 457 338 322 237 374 320 101 24 450 208 266 413 273 43 78 180 126 158 10 263 284 235 284 325 391 392 469 448 348 72 481 145 325 193 418 256 148 363 290 309 495 1 393 158 48 358 203 215 126 301 80 194 55 267 377 212 497 26 7 182 89 152 373 320 139 173 39 226 386 317 102 289 350 154 79 75 110 413 386 120 154 366 358 447 283 287 14 381 484 344 369 437 197 194 174 201 105 415 334 419 277 258 447 212 328 258 365 425 390 8 170 230 90 440 351 323 427 315 238 475 325 456 359 186 159 39 242 34 86 143 58 496 317 373 34 461 371 32 406 381 67 136 198 92 338 131 496 472 124 121 208 15 301 79 305 389 451 424 308 480 427 419 85 100 417 386 225 190 371 180 156 131 91 156 67 213 31 81 321 294 250 485 378 352 176 339 373 374 136 407 77 45 431 334 8 476 486 445 386 467 444 118 150 453 287 242 315 41 397 392 303 213 164 94 488 302 410 222 231 211 325 99 421 278 382 483 292 9 497 186 104 26 99 478 261 27 206 367 340 100 327 463 142 236 282 433 158 203 308 121 97 122 304 374 286 452 453 243 97 341 260 437 251 217 15 103 240 381 210 443 452 432 252 185 52 9 288 390 27 375 36 416 263 258 276 214 446 323 67 299 407 59 459 28 464 426	exists

	50 499 90 177 499 172 175 298 126 4 417 154 90 22 264 14 160 335 378 260 264 468 487 490 444 339 260 166 109 471 155 59 295 473 235 95 75 471 381 16 462 387 360 443 99 112 120 397 266 312 413 131 298 277 420 263 59 222 130 182 447 21 278 61 282 383 151 210 152 101 11 19 259 165 161 192 119 40 486 247 141 135 308 349 461 213 183 143 162 429 255 182 38 128 310 389 359 426 402 421 294 45 78 60 362 248 379 300 177 79 378 275 221 222 276 413 334 230 490 291 191 84 285 363 456 441 458 173 428 250 153 211 336 179 480 378 357 334 420 273 299 404 349 377 349 424 312 205 49 139 329 301 67 85 484 183 372 322 338 276 466 468 146 455 395 221 422 369 356 276 357 75 96 249 491 102 312 131 35 436 259 346 121 285 201 62 496 274 120 202 485 64 453 72 465 21 238 402 421 321 365 376 76 247 102 378 457 365 47 146 208 264 279 364 79 35 348 373 122 281 107 164 479 370 243 420 155 202 370 425 397 78 133 190 441 7 225 476 423 185 294 92 158 362 365 477 26 181 230 28 197 300 442 245 346 219 141 188 421 240 250 437 372 316 349 33 12 434 10 465 398 493 479 477 156 149 342 435 277 226 342 154 398 206 439 293 98 332 205 96 458 263 135 139 379 203 108 235 248 18 295 381 332 380 428 459 111 248 53 122 315 319 152 409 285 284 406 253 7 411 8 117 368 55 16 351 326 108 365 202 433 188 423 126 244 37 369 137 436 483 196 194 271 150 185 471 199 374 471 379 138 325 449 289 88 367 116 235 349 227 321 403 359 312 114 166 472 280 284 23 90 131 7 198 158 290 123 6 44 162 95 252 489 233 353 146 195 317 303 298 262 28 62 477 360 102 375 121 173 122 442 332 97 1 477 84 454 70 152 253 263 444 219 242 280 92 465 206 387 51 187 438 316 319 427 104 119 286 268 169 240 413 97 325 485 183 420 400 51 363 300 244 131 415 386 194 140 400 64 242 278 174 290 450 52 157 365 329 296 419 493 191 201 490 235 420 448 237 492 373 493 165 464 9 225 444 276 5 498 19 313 362 331 413 91 448 169 238 206 163 230 487 228 185 232 113 264 289 98 473 316 13 444 57 354 103 174 214 371 496 43 222 460 389 432 88 8 382 248 126 286 374 211 449 210 404 445 165 366 22 38 229 233 418 0	

Выводы.

Были изучены основные функции стандартной библиотеки; получены навыки работы с ними.

Разработана программа, на вход которой подается 1000 чисел, после чего программа выводит время, за которое число 0 будет найдено при помощи функции bsearch и при помощи обычного перебора. Если числа 0 нет в массиве чисел, то программа выведет *“doesn't exist”*.

Для поиска и сортировки, а также для нахождения времени выполнения использовались функции *bsearch()*, *qsort()* и *clock()*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

int comparator(const void* x1, const void* x2) {
     const int* v1 = (const int*)x1;
    const int* v2 = (const int*)x2;

    if (*v1 > *v2) {
        return 1;
    } else if (*v1 < *v2) {
        return -1;
    } else {
        return 0;
    }
}

int main() {
     int array[1000];

     for (int i = 0; i < 1000; i++) {
          scanf("%d", &(array[i]));
     }

     qsort(array, 1000, sizeof(int), comparator);

     clock_t time1 = clock();

     int value = 0;
     int* element = bsearch(&value, array, 1000, sizeof(int),
comparator);

     clock_t time2 = clock();

     if (element != NULL) {
          printf("exists\n");
     } else {
          printf("doesn't exist\n");
     }

     printf("%f\n", (time2 - time1) / CLOCKS_PER_SEC);

     time1 = clock();

    element = NULL;
     for (int i = 0; i < 1000; i++) {
          if (array[i] == 0) {
               element = array + i;
               break;
          }
     }
```

```
	time2 = clock();

	if (element != NULL) {
		printf("exists\n");
	} else {
		printf("doesn't exist\n");
	}

	printf("%f\n", (time2 - time1) / CLOCKS_PER_SEC);

	return 0;
}
```