Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

Тульский государственный университет

КАФЕДРА ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

ИПОЛЬЗОВАНИЕ ТАЙМЕРА

Лабораторная работа № 3
по курсу «Структуры и алгоритмы обработки данных»

Вариант № 4

Выполнил: студент группы 220601 _____ Белым А.А.
(подпись)
Проверил: д. ф.-м.н, проф.каф. ИБ _____ Двоенко С.Д.
(подпись)

Тула 2013

Цель работы

Изучить функции работы со временем в языке C/C++. Написать программу, использующую функции таймера библиотеки C/C++.

Задание

Написать программу, которая определяет время упорядочивания исходной последовательности чисел любым известным методом сортировки прямым обменом. Для “засечки” времени использовать функцию DIFFTIME.

Теоретическая справка

Во многих программах требуется следить за временем или выполнять какие-либо периодические действия. Для этих целей используются таймеры.

Например, программы MS-DOS для работы с таймером перехватывают аппаратное прерывание таймера, встраивая свой собственный обработчик для прерывания INT 8h.

Программы обработки прерываний, или обработчики прерываний, как их обычно называют, являются важнейшей составной частью большинства программных продуктов. Структура обработчика прерываний и его взаимодействие с остальными компонентами программного комплекса определяется рядом признаков, которые в данной работе рассматриваться не будут.

В отличие от программ MS-DOS обычные приложения WINDOWS не могут самостоятельно обрабатывать прерывания таймера, поэтому для работы с ним используют другие способы. Операционная система WINDOWS позволяет для каждого приложения создать несколько виртуальных таймеров. Все эти таймеры работают по прерываниям одного физического таймера.

Как видно, практическое значение таймера для согласования работы прикладных программ и ОС очень велико. Следует заметить также, что использование таймера не ограничивается только этим. Например, таймер может использоваться для “засечки” времени работы программы, с целью определения

времени ее исполнения. Очевидно, что быстрота работы программы один из критериев ее эффективности.

Таким образом, применение таймеров достаточно широко распространено в тех областях деятельности, где необходимо следить за временем, учитывать время выполнения каких-то действий, согласовывать работу разрозненных частей – механизмов, управляемых компьютером, различных программ – для выполнения какой-либо задачи и многое другое, что хоть как-то связано со временем.

В языке C/C++ для реализации возможности работы со временем предусмотрена целая библиотека под названием time.h. Помимо функций описанных в данной библиотеке существует еще и ряд других функций, работающих со временем, которые описаны в других библиотеках – bios.h, dos.h, sys\timeb.h и utime.h.

Схема алгоритма

На рисунках 1 и 2 представлена схема алгоритма сортировки массива по методу Хоара.

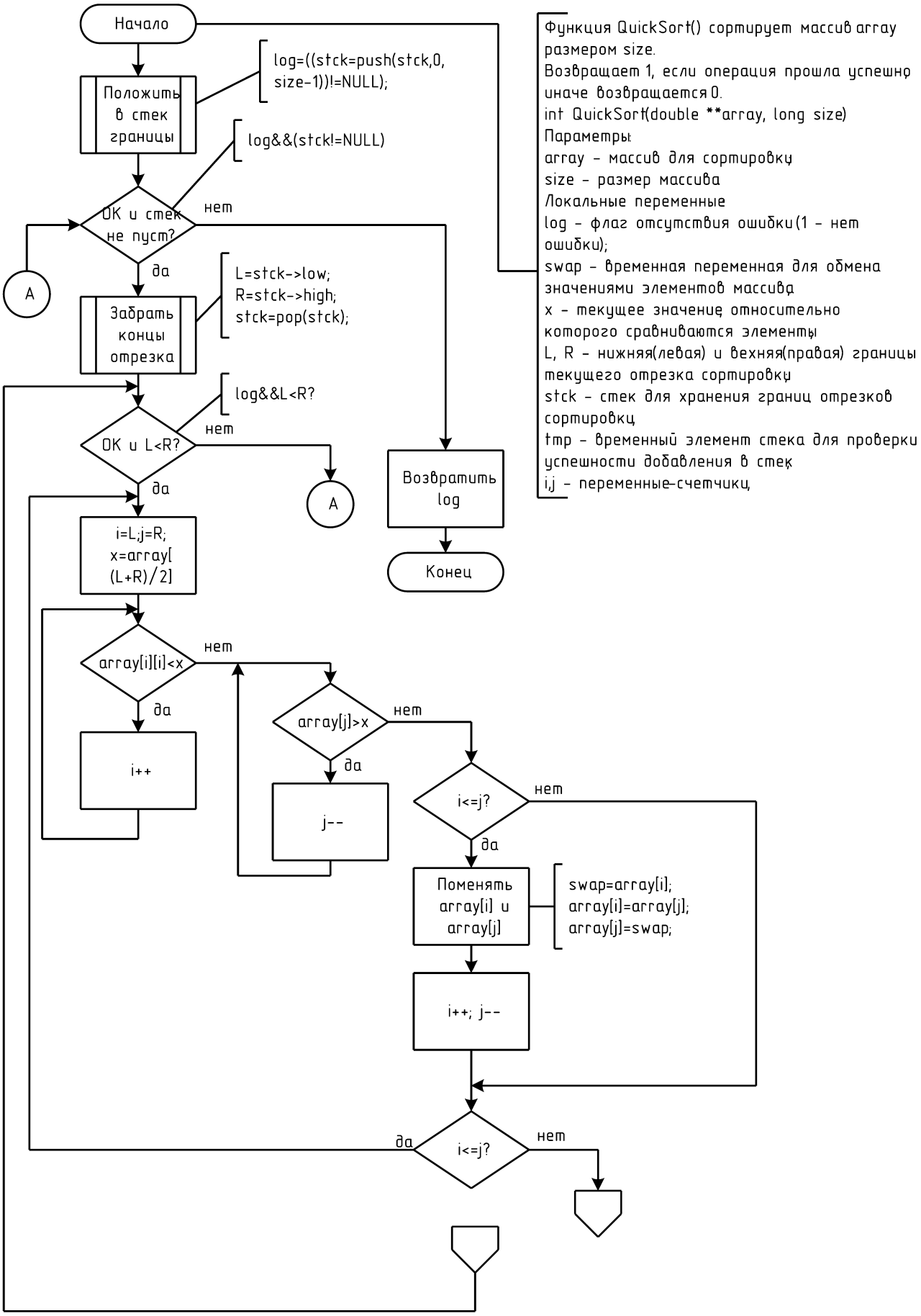

Рисунок 1 - Схема алгоритма сортировки Хоара

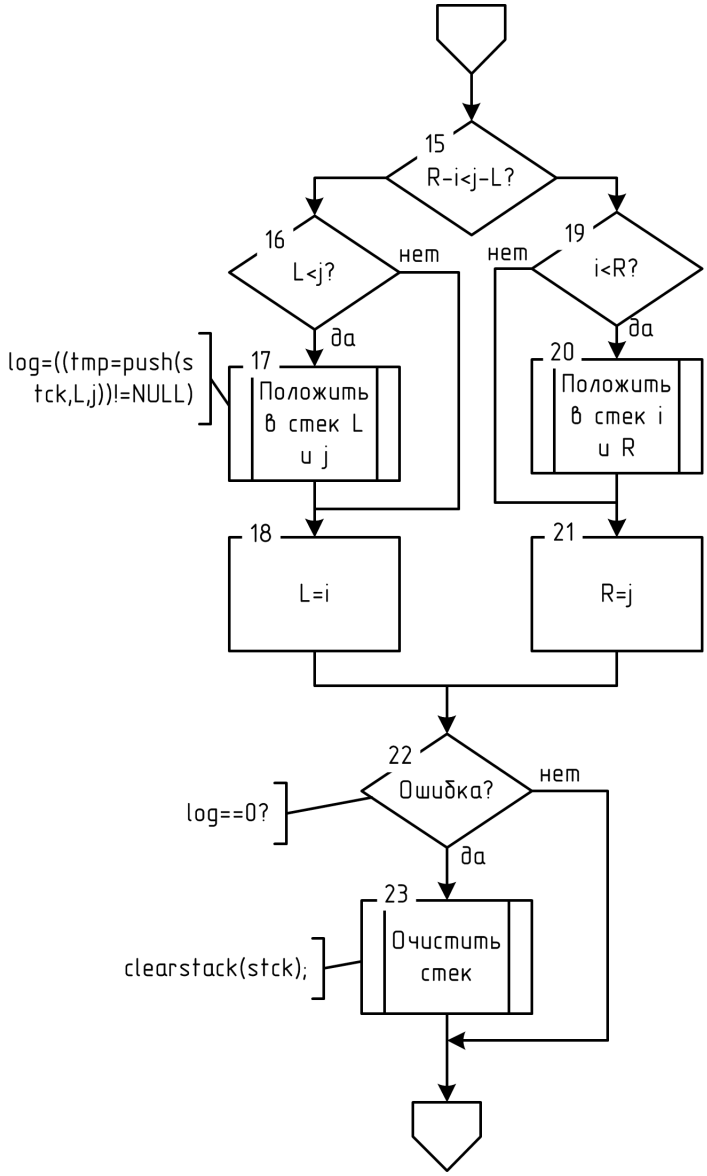

Рисунок 2 - Схема алгоритма сортировки Хоара (продолжение)

Данный вариант реализации использует стек, построенный с помощью связного списка. Для работы с такой структурой данных необходимы отдельные процедуры добавления в стек, удаления из стека, и очистки стека, и они соответственно представлены на рисунках 3, 4 и 5.

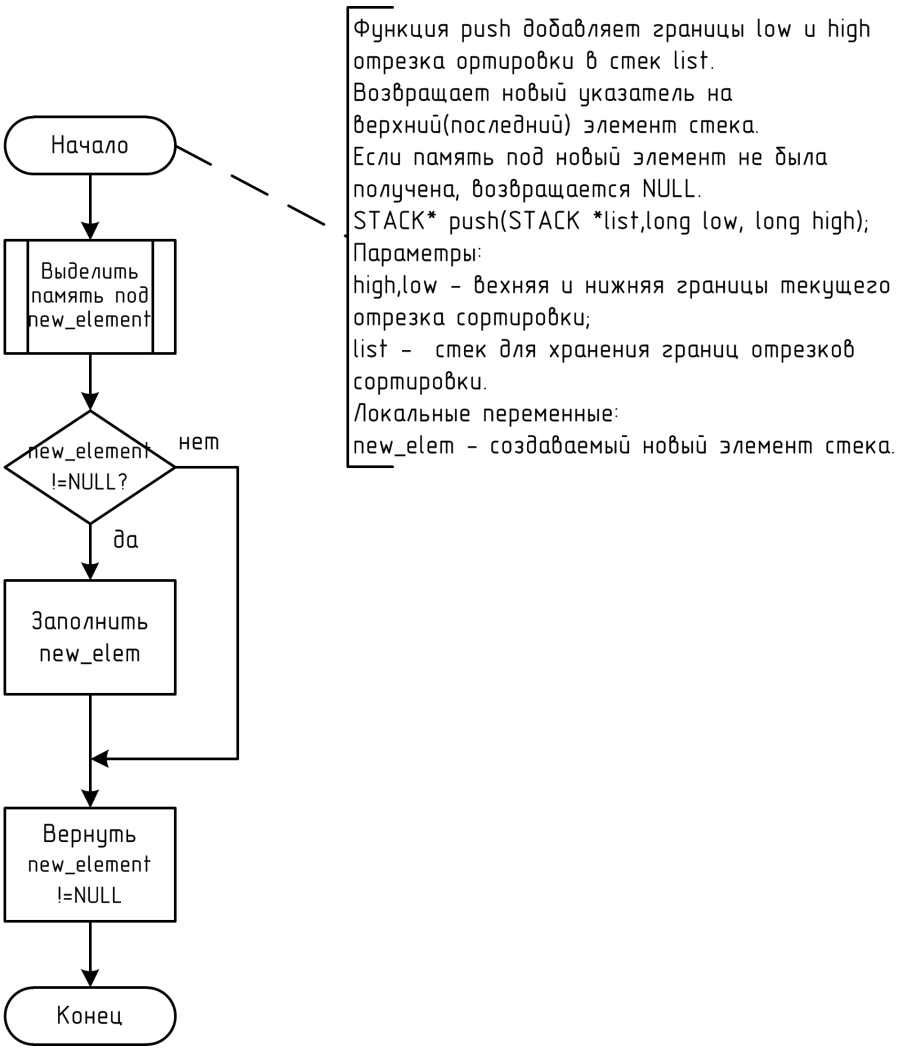

Рисунок 3 - Схема алгоритма вставки элемента в стек

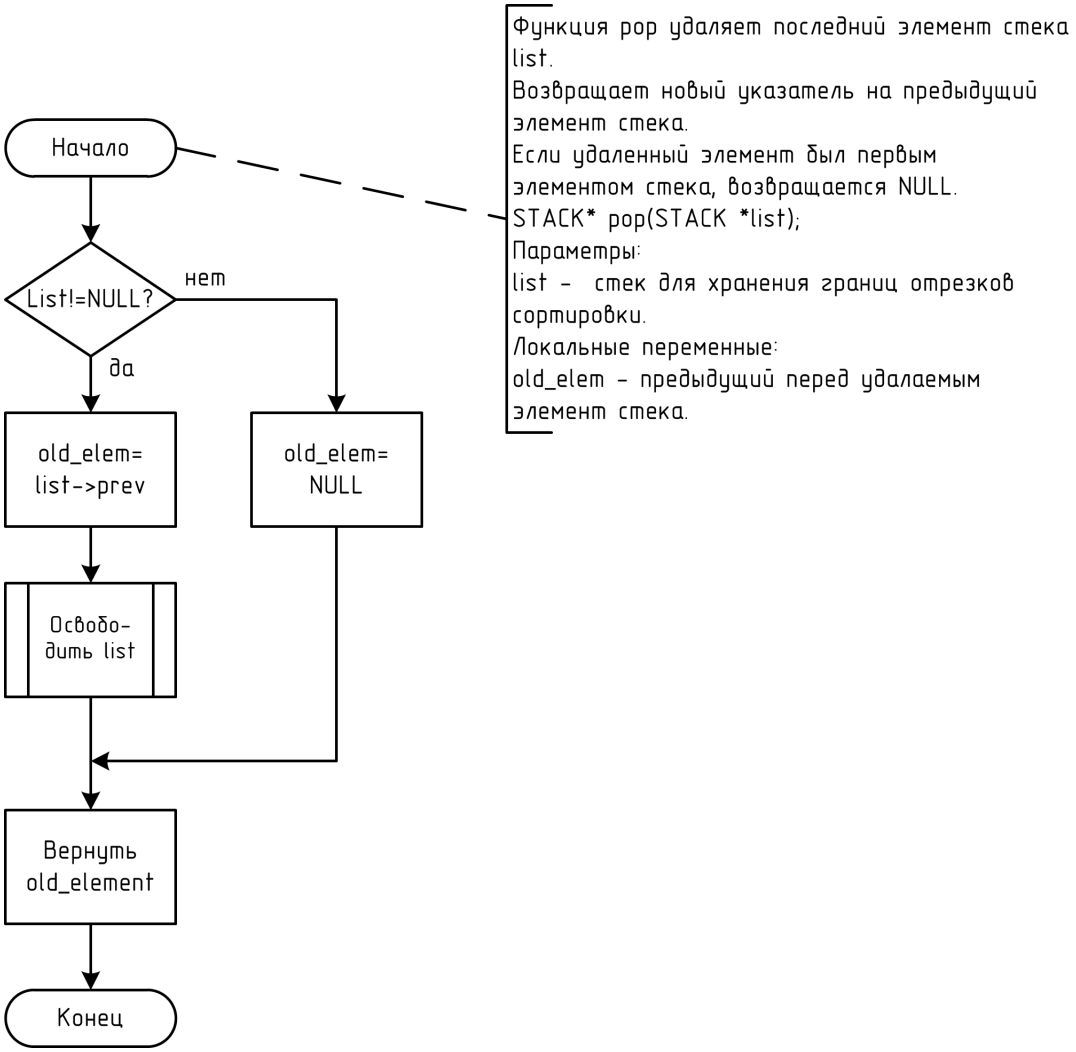

Рисунок 4 - Схема алгоритма удаления элемента из стека

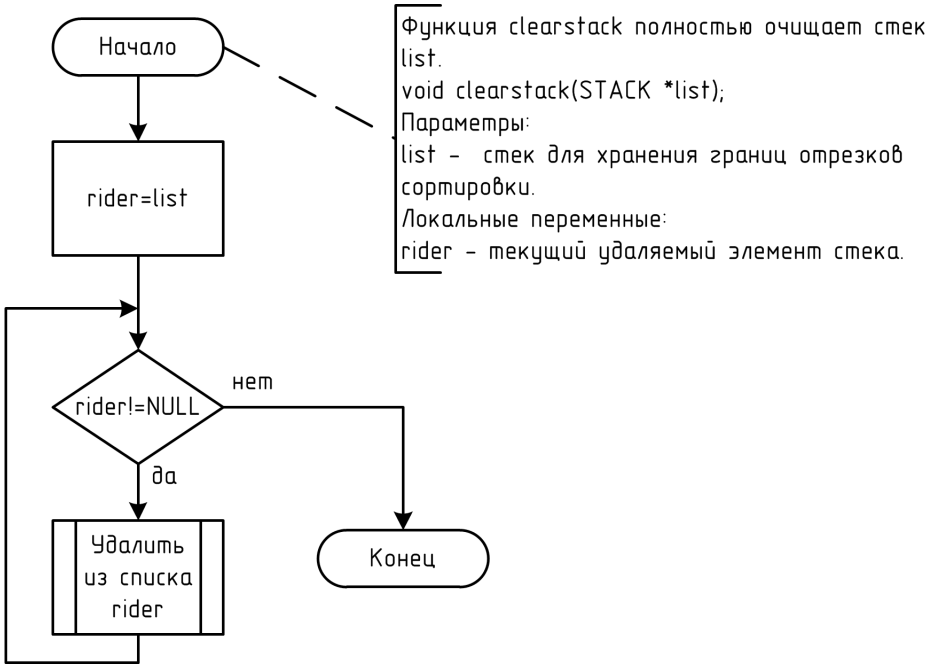

Рисунок 5 - Схема алгоритма очистки стека

Текст программы

Далее представлен текст программы на языке C, реализующей замер времени выполнения быстрой сортировки Хоара.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
//связный список-стек для функции сортировки
typedef struct stack {
    long high,low; //вехняя и нижняя границы текущего отрезка сортировки
    struct stack *prev; //предыдущий элемент стека
} STACK;

STACK* push(STACK *list,long low, long high){
  STACK *new_elem;
    new_elem=(STACK*)malloc(sizeof(STACK));
    if (new_elem!=NULL){
        new_elem->low=low;
        new_elem->high=high;
        new_elem->prev=list;
    }
    return new_elem;
}
STACK* pop(STACK *list){
  STACK *old_elem;
    if (list!=NULL){
        old_elem=list->prev;
        free(list);
    } else old_elem=NULL;
    return old_elem;
};

void clearstack(STACK *list){
  STACK *rider;
    rider=list;
    while(rider!=NULL){
        rider=pop(rider);
    }
};

int QuickSort(int *array, long size) {

  int log;

  int  swap,x;
  STACK *stck=NULL,*tmp=NULL;
  long i,j, L, R;

    log=((stck=push(stck,0,size-1))!=NULL);
    while (log&&(stck!=NULL)){
        L=stck->low; R=stck->high;
        stck=pop(stck);
        while (log&&L<R){
            i=L;j=R;x=array[(L+R)/2];
            do{
                while (array[i]<x) i++;
                while (x<array[j]) j--;
                if (i<=j){
                    swap=array[i];array[i]=array[j];array[j]=swap;
                    i++;j--;
                }
            }while(i<=j);
            if (R-i<j-L){
                if (L<j){
```

```
                    log=((tmp=push(stck,L,j))!=NULL);
                    if (log)
                        stck=tmp;
                    else
                        clearstack(stck);
                }
                L=i;

            } else {
                if (i<R){
                    log=((tmp=push(stck,i,R))!=NULL);
                    if (log)
                        stck=tmp;
                    else
                        clearstack(stck);
                }
                R=j;
            }
        }
    };
    return log;
}

int main(){
    int n;
    int *X;
    int i=0;
    time_t t1,t2;

    srand(time(NULL));
    printf("Введите размер массива.\n");
    scanf("%d",&n);
    n+=3;
    X=malloc(n*sizeof(int));
    for(i=0;i<n;++i){
        X[i]=rand();
    }
//    printf("Исходный массив:\n");
//    for(i=0;i<n;++i){
//      printf("%d ",X[i]);
//    }
    printf("\n");
    time(&t1);
    QuickSort(X,n);
    time(&t2);
//    printf("Отсортированный массив:\n");
//    for(i=0;i<n;++i){
//      printf("%d ",X[i]);
//    }
    printf("Затраченное время: %f сек.\n",difftime(t2,t1));
    printf("\n");
    return 0;
}
```

Тестовый пример

На рисунке 6 представлен пример работы программы, реализующей замер времени выполнения быстрой сортировки Хоара.

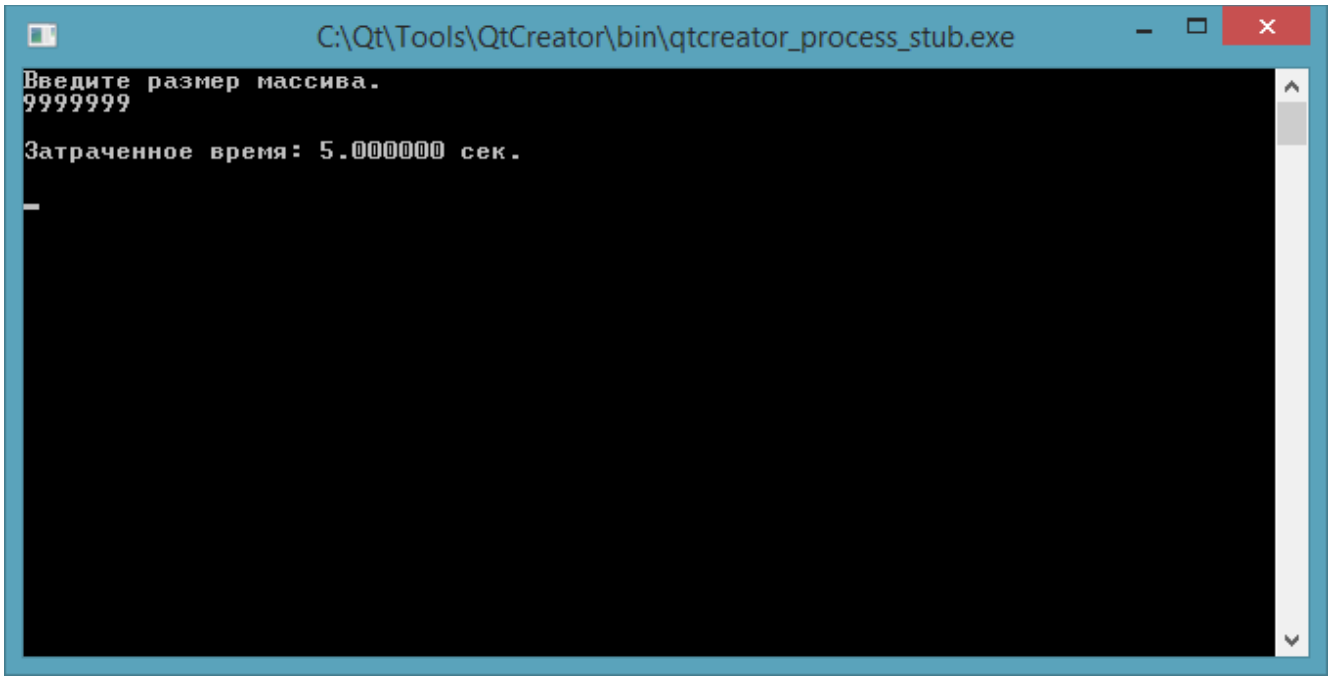

Рисунок 6 — Пример работы программы, замеряющей время сортировки

Вывод

В данной работе я познакомился с функциями для работы со временем в языках C/C++. В данном языке доступны многочисленные функции, которые можно использовать для получения текущего времени, часового пояса, замера прошедшего времени.