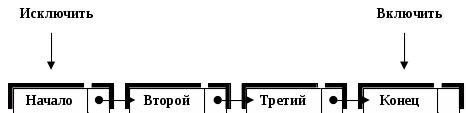
**Теоретическая часть.**

**Тип линейного списка – очередь.**

Очередь – структура данных типа «список», позволяющая добавлять элементы лишь в конец списка, и извлекать их из его начала. Она функционирует по принципу FIFO (First In, First Out — «первым пришёл — первым вышел»), для которого характерно, что все элементы a1, a2, …, an-1, an, добавленные раньше элемента an+1, должны быть удалены прежде, чем будет удален элемент an+1. Также очередь может быть определена как частный случай односвязного списка, который обслуживает элементы в порядке их поступления. Как и в «живой» очереди, здесь первым будет обслужен тот, кто пришел первым.



*Очередь*

Стандартный набор операций (часто у разных авторов он не идентичен), выполняемых над очередями, совпадает с тем, что используется при обработке стеков:

* добавление элемента;
* удаление элемента;
* чтение первого элемента.

Только, если в отношении стека в момент добавления или удаления элемента допустимо задействование лишь его вершины, то касательно очереди эти две операции должны быть применены так, как это регламентировано в определении этой структуры данных, т. е. добавление – в конец, удаление – из начала. Далее, при реализации интерфейса очереди, список стандартных операций будет расширен.

**Реализация очереди в программе.**

В программе для реализации очереди создаётся АТД Queue, который содержит Указатель на начало-Счетчик, методы добавления, вывода массива, конструктор, проверка на пустоту, вывод значения на выходе, вывод размера очереди, помещение элемента в очередь, просеивания через кучу, сортировки на куче, вывода массива и извлечения элемента из очереди.

Рассмотрим, для примера методы извлечения элемента из очереди и помещения элемента в очередь.

Метод помещения элемента в очередь:

Проверяем очередь на переполнение, сравнивая счётчик с размером массива. Если очередь не полна, то присваиваем элементу массива с индексом счётчика переданное в метод значение и увеличиваем счётчик элементов на 1.

Метод извлечения элемента из очереди:

Проверяем счётчик элементов, если он не равен нулю,то в цикле от индекса равного нулю до счётчика минус один выполняем смещение элемента на один вперёд. Уменьшаем счётчик на 1.

**Анализ трудоёмкости работы программы.**

Анализируя набор операций выполняемых над очередью считаем сколько в нем содержится элементарных операций. Базовыми операциями для вычисления F(n) будем считать следующие операции:

1. Сложение, умножение и.т.п
2. Сравнения (>,<,==, и.т.п)
3. Объявление переменной (int a;)
4. Обращение к элементу массива по индексу (a[i])

Для примера возьмём метод Sortirovka().

**void** Sortirovka(**int** \*numbers, **int** array\_size)

{

// Формируем нижний ряд пирамиды

nop+=4;

1. **for** (**int** i = (array\_size / 2) - 1; i >= 0; i--)//2
2. {
3. nop+=2;
4. proseivanie(numbers, i, array\_size - 1);//1
5. nop++;
6. }
7. // Просеиваем через пирамиду остальные элементы
8. nop+=3;
9. **for** (**int** i = array\_size - 1; i >= 1; i--)//2
10. {
    1. nop+=2;
11. **int** temp = numbers[0];//2
12. numbers[0] = numbers[i];//3
13. numbers[i] = temp;//2
14. proseivanie(numbers, 0, i - 1);//1
15. nop+=8;
16. }

}

};

В строке 1 при первом входе в цикл прибавляются 4 операции(сравнение, присваивание, деление, вычитание), в каждом последующем входе прибавляются 2 операции(вычитание и сравнение). В строке 4 добавляется 1 операция(вычитание). В строке 9 при первом входе в цикл прибавляются 3 операции(сравнение, присваивание, вычитание), в каждом последующем входе прибавляются 2 операции(вычитание и сравнение). В строке 11 – 2 операции(присваивание, обращение к элементу массива). В строке 12 – 3 операции(присваивание, два обращения к элементу массива). В строке 13 – 2 операции(присваивание, обращение к элементу массива). В строке 14 добавляется 1 операция(вычитание).

**Листинг программы.**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <windows.h>

#include <fstream>

/\*

    �'азовыми операциями для вычисления F(n) будем считать следующие операции:

        Сложение, умножение и тп

        Сравнения (>,<,==, и тп)

        Объявление переменной (int a;)

\*/

**unsigned** **long** **long** **int** nop = 0;

**using** **namespace** std;

**int** SIZ=10000000000;//Размер массива

**int** \*a;//массив

**class** Queue {

**public** :

**int** count;//Указатель на начало-Счетчик

**void** pop();//функция добавления

**void** show();//вывод массива

Queue() {//конструктор

//выделение динамической памяти для массива

a = **new** **int**[SIZ];//1

count =0 ;//1

//+1 создание объекта класса

nop+=3;

}

**bool** isempty(){ //Пуст ли массив?

nop++;

**if** (SIZ == 0) //1

**return** 0;

**else** **return** 1;

};

**int** value(){ //�'ывод значения на выходе

**return** a[0];

};

**void** getQueueSize(){ //�'ывод размера очереди

cout<<"Кол-во элементов в массиве:"<<SIZ<<endl;

};

**int** get(**int** n){

**int** val;//1

**int** val\_pos;//1

nop+=3;

**if** (n != 1)//1

{

nop++;//присваивание int i=0;

**for** (**int** i = 0; i != count; )//1

{

nop++;

val = value();//1

pop();//5n+4

push (val);

i++;

**if** (i == n - 1)

{

val\_pos = value();

}

}

}

**else** {

val\_pos = value();

}

**return** val\_pos;

}

**void** set (**int** pos, **int** val){

**int** valu;

**if** (pos==1) {

a[0] = val;

}

**else**{

**for** (**int** i = 0; i != count; )

{

valu = value();

pop();

push (valu);

i++;

**if** (i == (pos - 1))

{

a[0] = val;

}

}

}

}

**void** push ( **int** num ){ //Помещение элемента в очередь

**if** (count == SIZ)

cout <<"Очередь полна" <<endl ;

**else**

{

a[count]=num;

count++;

}

}

// Функция "просеивания" через кучу - формирование кучи

**void** proseivanie(**int** \*numbers, **int** root, **int** bottom)

{

**int** maxChild; //1// индекс максимального потомка

**int** done = 0; //1// флаг того, что куча сформирована

// Пока не дошли до последнего ряда

nop+=5;

**while** ((root \* 2 <= bottom) && (!done)) //3

{

nop+=3;

nop+=2;

**if** (root \* 2 == bottom)//2 // если мы в последнем ряду,

{maxChild = root \* 2;//2

nop+=2;} // запоминаем левый потомок

// иначе запоминаем больший потомок из двух

**else** {nop+=6;

**if** (numbers[root \* 2] > numbers[root \* 2 + 1])//6

{maxChild = root \* 2;//2

nop+=2;

}

**else**

{maxChild = root \* 2 + 1;//3

nop+=3;

}

}

// если элемент вершины меньше максимального потомка

nop+=3;

**if** (numbers[root] < numbers[maxChild]) //3

{

**int** temp = numbers[root];//2 // меняем их местами

numbers[root] = numbers[maxChild];//3

numbers[maxChild] = temp;//2

root = maxChild;//1

nop+=8;

}

**else** // иначе

{done = 1; // пирамида сформирована

nop++;

}

}

}

// Функция сортировки на куче

**void** Sortirovka(**int** \*numbers, **int** array\_size)

{

// Формируем нижний ряд пирамиды

nop+=4;

**for** (**int** i = (array\_size / 2) - 1; i >= 0; i--)//2

{

nop+=2;

proseivanie(numbers, i, array\_size - 1);//1

nop++;

}

// Просеиваем через пирамиду остальные элементы

nop+=3;

**for** (**int** i = array\_size - 1; i >= 1; i--)//2

{

nop+=2;

**int** temp = numbers[0];//2

numbers[0] = numbers[i];//3

numbers[i] = temp;//2

proseivanie(numbers, 0, i - 1);//1

nop+=8;

}

}

};

**void** Queue::show() {//�'ывод массива

**for**(**int** i=0; i<count;i++)

cout<<" "<<a[i];

cout<<endl;

}

**void** RandomnoeZapolnenie(Queue &q, **int** n){

**for** (**int** i=0; i<n; i++)

{

q.push(-100+rand()%200);}

}

**void** Queue::pop()//1+2+5\*n+1=5n+4

{// �звлечение элемента из очереди

nop++;

**if** ( count==0) //1

{

cout << "очередь пуста" <<endl ;

**return** ;

}

nop+=2;

**for** (**int** i=0; i<count ; i++)//2 //смещение элементов

{

a[i]=a[i+1];//3

nop+=5;

}

count--;

nop++;

}

**void** menu(){

cout << "0 - �-аполнение массива" << endl;

cout << "1 - �"обавление элемента в массив" << endl***;***

cout << "2 - Удаление элемента" << endl***;***

cout << "3 - �-начение (value)" << endl***;***

cout << "4 - Сортировка" << endl***;***

cout << "5 - Set" << endl***;***

cout << "6 - Get"<<endl***;***

cout << "7 - �'ыход" << endl***;***

***}***

/\*\****/***

int main() ***{***

setlocale(LC\_ALL, "Rus")***;***

srand(time(NULL))***;***

  Queue q***;***

  int h=0***;***

while (h!=7)***{***

menu()***;***

do***{***

cin >> h***;***

if (h<-1 or h>6***)***

cout << "�'ыход" << endl***;***

***}***

while (h<0 and h>6)***;***

switch (h)***{***

case 0***:***

cout << "�'веди размер массива" << endl***;***

cin >> SIZ***;***

if (q.isempty() == 0)***{***

cout << "Очередь пуста" << endl***;***

system("pause")***;***

***}***

els***e***

RandomnoeZapolnenie(q, SIZ)***;***

system("CLS")***;***

q.show()***;***

break***;***

case 1***:***

int el***;***

cout << "�'веди элемент" << endl***;***

cin >> el***;***

SIZ++***;***

q.push(el)***;***

system("CLS")***;***

q.getQueueSize()***;***

q.show()***;***

break***;***

case 2***:***

q.pop()***;***

SIZ--***;***

system("CLS")***;***

q.getQueueSize()***;***

q.show()***;***

break***;***

case 3***:***

cout << endl***;***

cout << "�-начение элемента - " <<q.value()***;***

cout << endl***;***

system("pause")***;***

system("CLS")***;***

q.show()***;***

break***;***

case 4***:***

q.Sortirovka(a,SIZ)***;***

system("CLS")***;***

q.getQueueSize()***;***

q.show()***;***

break***;***

case 5***:***

int numb,elem***;***

cout<<"�'веди номер позиции"<<endl***;***

cin>>numb***;***

cout<<"�'веди элемент"<<endl***;***

cin>>elem***;***

q.set(numb,elem)***;***

system("CLS")***;***

q.show()***;***

break***;***

case 6***:***

int number***;***

cout<<"�'веди номер позиции"<<endl***;***

cin>>number***;***

system("CLS")***;***

cout<<"Элемент в этой позиции: "***;***

cout<< q.get(number)<<endl***;***

q.show()***;***

break***;***

***}***

***}***

return 1***;***

  cout<<endl***;***

  return 0***;***

***}***

/\*int main(***)***

***{***

srand(time(NULL))***;***

setlocale(LC\_ALL, "Rus")***;***

int h=0***;***

ofstream fl("andr.txt")***;***

for (int i=1;i<100000001;i++***)***

***{***

//fl<<(rand())***;***

//fl<<-i***;***

//fl<<i***;***

fl<<1***;***

fl<<" "***;***

***}***

fl.close()***;***

int i, N,b***;***

for (N=10000000;N<100000001;N+=10000000***)***

***{***

Queue q***;***

ifstream f("andr.txt")***;***

for (i=1;i<N;i++***)***

***{***

if(f >> b***)***

{//cout << b << std::endl;

q.push(b)***;***

***}***

***}***

int t1,t2***;***

nop=0***;***

t1=GetTickCount()***;***

q.Sortirovka(a,N-1)***;***

//cout<<endl<<nop***;***

t2=GetTickCount()***;***

cout<<t1<<endl<<t2<<endl<<t2-t1<<endl***;***

cout<<N/10000000<<"***)*** NOP "<<nop<<endl***;***

delete a***;***

f.close()***;***

***}***

}\****/***

**Расчёты трудоёмкости и выводы функции роста.**

Общая идея пирамидальной сортировки заключается в том, что сначала строится пирамида из элементов исходного массива, а затем осуществляется сортировка элементов.

Выполнение алгоритма разбивается на два этапа.

1 этап Построение пирамиды. Определяем правую часть дерева, начиная с n/2-1 (нижний уровень дерева). Берем элемент левее этой части массива и просеиваем его сквозь пирамиду по пути, где находятся меньшие его элементы, которые одновременно поднимаются вверх; из двух возможных путей выбираем путь через меньший элемент.

Разумеется, полученный массив еще не упорядочен. Однако процедура просеивания является основой для пирамидальной сортировки. В итоге просеивания наименьший элемент оказывается на вершине пирамиды.

2 этап Сортировка на построенной пирамиде. Берем последний элемент массива в качестве текущего. Меняем верхний (наименьший) элемент массива и текущий местами. Текущий элемент (он теперь верхний) просеиваем сквозь n-1 элементную пирамиду. Затем берем предпоследний элемент и т.д.

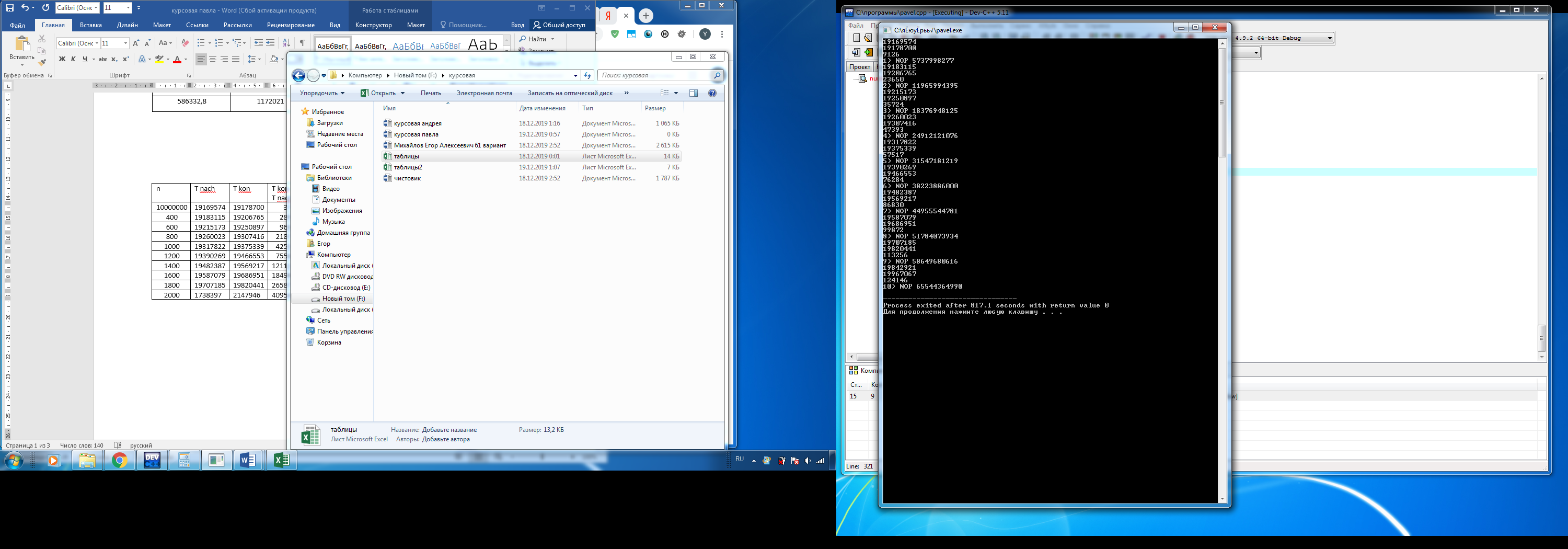
**Подсчёт операций в сортировке.**

F(n)=4*(for (int i = (array\_size / 2) - 1; i >= 0; i--) – присваивание, деление, вычитание, сравнение)* + n/2(*цикл for (int i = (array\_size / 2) - 1; i >= 0; i--)*)\*(2(*цикл for (int i = (array\_size / 2) - 1; i >= 0; i--) – вычитание, сравнение*)+1(*proseivanie(numbers, i, array\_size - 1); )*)+

+(Строка proseivanie(numbers, i, array\_size - 1); выполняется с i-1,i-2,i-3,…0 элементом. Значит это можно представить как ∑n/2-1i=0\*(сумма операций i-ого элемента в методе proseivanie()). Произведём подсчёт операций метода proseivanie() при i-ом элементе.A(i)=2(*int maxChild; //1 int done = 0; //1*) +3*(while ((root \* 2 <= bottom) && (!done)))*+log2n/i(*while ((root \* 2 <= bottom) && (!done)), где bottom=n-1, а root=i)*\*(3*(while ((root \* 2 <= bottom) && (!done)))*+2*(if (root \* 2 == bottom))*+6*(if (numbers[root \* 2] > numbers[root \* 2 + 1]))*+3*(maxChild = root \* 2 + 1;)*+3*(if (numbers[root] < numbers[maxChild]))*+8*(int temp = numbers[root];numbers[root] = numbers[maxChild];numbers[maxChild] = temp;root = maxChild;)*). ∑n/2-1i=0 A[i]= ∑n/2-1i=0 (2+3+log2n/i(3+2+6+3+3+8))= ∑n/2-1i=0(5+25log2n/i). Так как деление на 0 невозможно, то в формуле заменяем 0 на 1 , а n/2-1 на n/2. ∑n/2i=1((log2n/i)\*25+5)=5n/2+25log2n∑n/2i=1(1/i)=5n/2+25log2n\*ln(n/2))+3*(for (int i = array\_size - 1; i >= 1; i--))*+(n-1)\*(( *for (int i = array\_size - 1; i >= 1; i--), где array\_size = n*) (25log2n)*(так как просеивание выполняется с одним тем же нулевым элементом )*+10)=4+3n/2+5n/2+25log2n\*n/2+3+25nlog2n-25log2n+10n-10= -28+14n+25nlog2n+25log2n\*n/2-25log2n

**Таблицы и графики для варианта со случайными значениями.**

|  |  |  |  |
| --- | --- | --- | --- |
| C1 = F(n)/T(n) | C2 = O(F(n))/T(n) | C3 = F(n)/NOP | C4=O(F(n))/NOP |
| 649614,6 | 1274025 | 1,03318 | 2,026273 |
| 522484,5 | 1025518 | 1,032656 | 2,026868 |
| 531123,4 | 1042932 | 1,032481 | 2,027414 |
| 542559,2 | 1065706 | 1,032169 | 2,027406 |
| 565820,4 | 1111646 | 1,031607 | 2,02676 |
| 517116,3 | 1016142 | 1,032017 | 2,02793 |
| 534510 | 1050478 | 1,032387 | 2,02896 |
| 534954,8 | 1051486 | 1,031727 | 2,027921 |
| 534079,5 | 1049882 | 1,031339 | 2,027384 |
| 544428,1 | 1070329 | 1,031188 | 2,027285 |



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | T nach | T kon | T kon- T nach | NOP | O(F(n))=50n\*log2n | F(n) |
| 10000000 | 19169574 | 19178700 | 9126 | 5737998277 | 480000000 | 241321002 |
| 20000000 | 19183115 | 19206765 | 23650 | 11965994395 | 3840000000 | 1,925E+09 |
| 30000000 | 19215173 | 19250897 | 35724 | 18376948125 | 1,296E+10 | 6,492E+09 |
| 40000000 | 19260023 | 19307416 | 47393 | 24912121076 | 3,072E+10 | 1,538E+10 |
| 50000000 | 19317822 | 19375339 | 57517 | 31547181219 | 6E+10 | 3,003E+10 |
| 60000000 | 19390269 | 19466553 | 76284 | 38223886000 | 1,0368E+11 | 5,189E+10 |
| 70000000 | 19482387 | 19569217 | 86830 | 44955544781 | 1,6464E+11 | 8,238E+10 |
| 80000000 | 19587079 | 19686951 | 99872 | 51784073934 | 2,4576E+11 | 1,23E+11 |
| 90000000 | 19707185 | 19820441 | 113256 | 58649680616 | 3,4992E+11 | 1,751E+11 |
| 1E+08 | 19842921 | 19967067 | 124146 | 65544364990 | 4,8E+11 | 2,401E+11 |

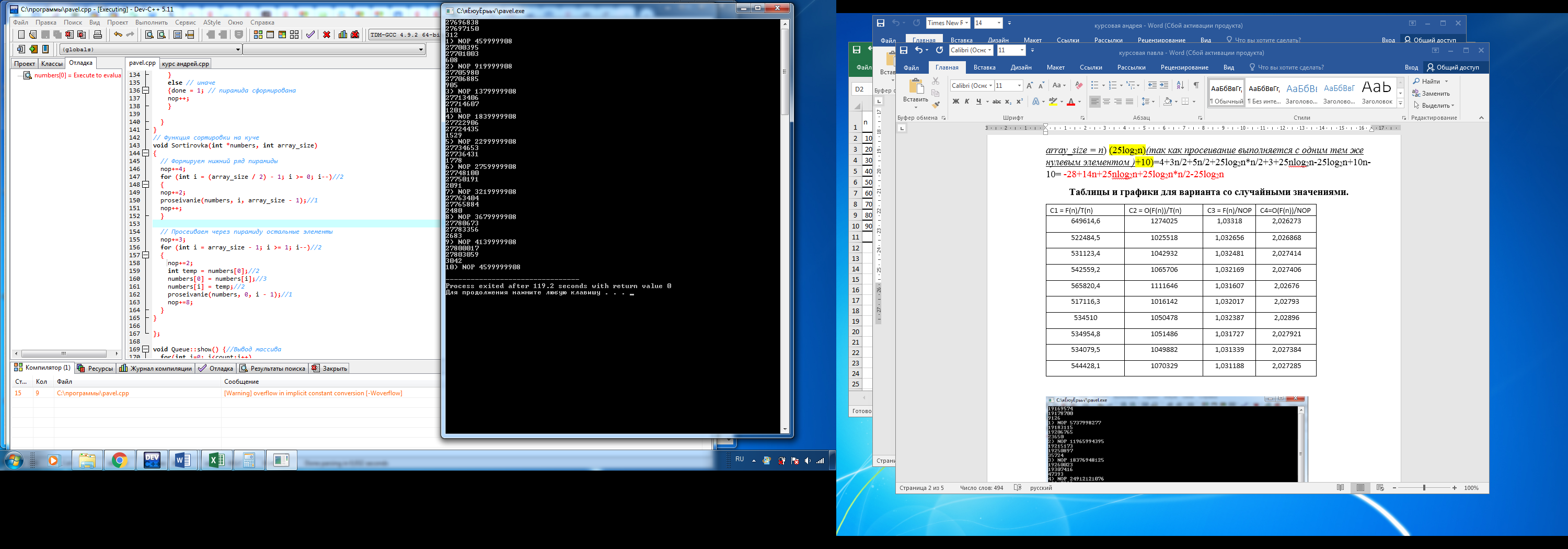
**Таблицы и графики для особого случая.**

Особый случай возникает, тогда когда все элементы массива одинаковы. При этом при просеивании очередного элемента цикл(пока не дошли до последнего ряда) прекращается после первого прохода, так как срабатывает условие того, что куча уже сформирована. Таким образом вместо log 2 n цикл происходит 1 раз и сложность алгоритма становится равной O(n\*log n)=O(n\*1)=O(n).

**Подсчёт операций в сортировке.**

F(n)=4*(for (int i = (array\_size / 2) - 1; i >= 0; i--) – присваивание, деление, вычитание, сравнение)* + n/2(*цикл for (int i = (array\_size / 2) - 1; i >= 0; i--)*)\*(2(*цикл for (int i = (array\_size / 2) - 1; i >= 0; i--) – вычитание, сравнение*)+1(*proseivanie(numbers, i, array\_size - 1); )*) +(5+3+2+6+3+3+1)n/2(складываем все nop у метода просеивание. Умножаем на n/2, так как цикл(*for (int i = (array\_size / 2) - 1; i >= 0; i--)*) выполняется n/2 раз)+(2+8)\*n(складываем nop в цикле *for (int i = array\_size - 1; i >= 1; i--)*)+(5+3+2+6+3+3+1)n(складываем все nop у метода просеивание. Умножаем на n, так как цикл(*for (int i = (array\_size / 2) - 1; i >= 0; i--)*) выполняется n раз)=23n+10n+3n/2+23n/2=***46n+7***

|  |  |  |  |
| --- | --- | --- | --- |
| C1 = F(n)/T(n) | C2 = O(F(n))/T(n) | C3 = F(n)/NOP | C4=O(F(n))/NOP |
| 1474359 | 2884615 | 1 | 1,956522 |
| 1513158 | 2960526 | 1 | 1,956522 |
| 1524862 | 2983425 | 1 | 1,956522 |
| 1532057 | 2997502 | 1 | 1,956522 |
| 1504251 | 2943100 | 1 | 1,956522 |
| 1552306 | 3037120 | 1 | 1,956522 |
| 1539933 | 3012912 | 1 | 1,956522 |
| 1483871 | 2903226 | 1 | 1,956522 |
| 1543049 | 3019009 | 1 | 1,956522 |
| 1512163 | 2958580 | 1 | 1,956522 |



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | T nach | T kon | T kon- T nach | NOP | O(F(n))=90n | F(n) |
| 10000000 | 27696838 | 27697150 | 312 | 4,6E+08 | 900000000 | 460000007 |
| 20000000 | 27700395 | 27701003 | 608 | 9,2E+08 | 1800000000 | 920000007 |
| 30000000 | 27705980 | 27706885 | 905 | 1,38E+09 | 2700000000 | 1380000007 |
| 40000000 | 27713406 | 27714607 | 1201 | 1,84E+09 | 3600000000 | 1840000007 |
| 50000000 | 27722906 | 27724435 | 1529 | 2,3E+09 | 4500000000 | 2300000007 |
| 60000000 | 27734653 | 27736431 | 1778 | 2,76E+09 | 5400000000 | 2760000007 |
| 70000000 | 27748100 | 27750191 | 2091 | 3,22E+09 | 6300000000 | 3220000007 |
| 80000000 | 27763404 | 27765884 | 2480 | 3,68E+09 | 7200000000 | 3680000007 |
| 90000000 | 27780673 | 27783356 | 2683 | 4,14E+09 | 8100000000 | 4140000007 |
| 1E+08 | 27800017 | 27803059 | 3042 | 4,6E+09 | 9000000000 | 4600000007 |

**Выводы.**

Реализована сортировка простой вставкой линейного списка типа массив. Этот алгоритм реализован в виде программы на языке C++, с использованием методов объектно-ориентированного программирования. При этом создан абстрактный тип данных Queue. Корректность работы алгоритма сортировки, подтверждена посредством тестирования произвольными наборами входных данных, вводимых с клавиатуры.

Произведен расчёт трудоёмкости предложенного алгоритма посредством построения функции роста, которая оценивалась прямым подсчетом операций, выполняемых программой в худшем случае(наиболее неблагоприятная комбинация входных данных). Установлено, что функция роста имеет асимптотический характер O(N\*log2 N), где N – длина очереди.

При сортировке элементов линейного списка был обнаружен особый случай, при котором время программы и количество операций существенно отличались от случайных значений. Это случай когда все сортируемые элементы были равны по значению. При этом сложность алгоритма стала равной O(n), так как операция “просеивания” элемента в куче выполнялась не log2 N, а только 1 раз. Коэффициент пропорциональности между экспериментальной функцией роста примерно равен от 1,031 до 1,033 для варианта случайного заполнения очереди и 1 для особого случая.

**Список источников.**

1)Кнут Д.Е. Искусство программирования. В 3-х томах. – М.: "Вильямс", 2000.

2) Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ. – М.: МЦНМО, 2005.

3) Левитин А.В. Алгоритмы: введение в разработку и анализ.: Пер. с англ.– М.:Издательский дом «Вильямс», 2006. – 576.