**Теоретическая часть.**

Дек (от англ. deq - double ended queue, т.е. очередь с двумя концами), представляет собой структуру данных, в которой можно добавлять и удалять элементы с двух сторон.

Задачи, требующие структуры дека, встречаются в вычислительной технике и программировании гораздо реже, чем задачи, реализуемые на структуре стека или очереди. Как правило, вся организация дека выполняется программистом без каких-либо специальных средств системной поддержки.

Примером дека может быть, например, некий терминал, в который вводятся команды, каждая из которых выполняется какое-то время. Если ввести следующую команду, не дождавшись, пока закончится выполнение предыдущей, то она встанет в очередь и начнет выполняться, как только освободится терминал. Это FIFO очередь. Если же дополнительно ввести операцию отмены последней введенной команды, то получается дек.

**Реализация дека в программе.**

В программе для реализации дека создаётся абстрактный тип данных Deque, который содержит размер дека, указатель на первый и последний элементы дека. Элемент состоит из значения, указателя на следующий и предыдущий элементы. Изначально хвост, голова и размер дека равны нулю.

При добавлении элементы слева, если дек не имеет элементов, то:

1. динамически выделяется память на новый элемент.
2. поля следующего и предыдущего элемента равны нулю.
3. значение элемента становится равным заданному
4. голова дека указывает на новый элемент
5. размер дека увеличивается на еденицу

При добавлении элементы слева, если дек имеет элементы, то:

1. динамически выделяется память на новый элемент.
2. поле следующего элемента равно голове, а предыдущего элемента равно нулю.
3. значение элемента становится равным заданному
4. указатель головы на предыдущий элемент равен новому элементу.
5. голова дека указывает на новый элемент
6. размер дека увеличивается на единицу

Операции добавления элементов справа аналогичны вышеприведённым.

При проверке дека на пустоту, проверяется не равен ли нулю размер дека.

При удалении элементов слева, проверяем дек на пустоту. Если голова равна хвосту, то голову дека приравниваем к нулю и уменьшаем его размер. Если голова не равна хвосту, то объявляем новый указатель на элемент, присваиваем ему значение головы. Присваиваем голове значение поля некст указателя и удаляем указатель, уменьшаем размер дека.

Операции удаления элементов справа аналогичны вышеприведённым.

**Анализ трудоёмкости работы программы.**

Анализируя набор операций выполняемых над деком 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 методы удаления и добавления элемента справа в дек.

Добавление элемента в дек справа.

**void** AddR(**int** el)//6

* 1. {
     1. nop++;
     2. **if** (head==0)//1
     3. {
     4. head = **new** element;//1
     5. tail=head;//1
     6. head->value=el;//1
     7. head->next=0;//1
     8. nop+=4;
  2. }
  3. **else**{
     1. tail->next=**new** element;//1
     2. tail->next->value=el;//1
     3. tail->next->next=0;//1
     4. tail=tail->next;//1
     5. nop+=4;
  4. }
  5. size++;
  6. nop++;
  7. };

В строке a.ii 1 операция – сравнение. Далее 2 ветки алгоритма – if и else. Рассмотрим каждую.

1. if) в строке a.iv 1 операция – создание элемента, в строке a.v 1 операция – присваивание, в строке a.vi 1 операция – присваивание, в строке a.vii 1 операция – присваивание. Итого: 4 операции.

2. else) в строке c.i 1 операция – создание элемента, в строке c.ii 1 операция – присваивание, в строке c.iii 1 операция – присваивание, в строке c.iv 1 операция – присваивание. Итого: 4 операции.

В обоих ветках получено одинаковое количество операций.

В строке e 1 операция – инкремент.

Подведём итоги: 1+4+1=6. nop+=6; Количество операций увеличиваем на 6.

Удаление элемента справа.

**void** DeleteR()//1+1+1+2(n-2)+2+1=2n+2

* 1. {
     1. nop++;
  2. element \*f = head;//1
  3. nop++;

1. **if** (f == tail)//1
2. {
   * 1. **delete** f;
     2. head = tail = NULL;//2
     3. nop+=3;
3. }
4. **else**
5. {
   1. nop++;
   2. **while** (f->next != tail)//1
      1. {
      2. f = f->next;//1
      3. nop+=2;
      4. }
   3. **delete** tail;
   4. tail = f;
   5. tail->next = NULL;
   6. nop+=2;
6. }
7. size--;
8. nop++;
   1. };

В строке b—1 операция(создание элемента), в строке 2—1 оп.(сравнение). Рассмотрим ветки if и else в строке 2 по отдельности.

1. if) В строке 3.ii – 2 операции (двойное присваивание)

2. else) В строке 6b – 1 операция(сравнение). В строке 6.b цикл выполнится n-2 раза. В цикле будет происходить сравнение(6b) – 1 операция и присваивание(6bii) – 1 операция. Следовательно общее количество операций цикла равно (n-2)2. В строке 6d –присваивание(1 опер.), в строке 6e –присваивание(1 опер.). Итого: 1+2(n-2)+2. Следовательно, в ветке else операций больше, при расчёте формулы учитываем это.

В строке 8 1 операция – уменьшение на 1.

Итого: F(n)= 1+1+1+2(n-2)+2+1=***2n+2***

**Листинг программы.**

#include <iostream>

#include <cstdlib>

//#include <ctime>

#include <time.h>

#include <windows.h>

#include <fstream>

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

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

**class** element{

**public**:

**int** value;

element \*next;

element \*prev;

};

**class** Deque{

**public**:

**int** size;

element\* head;

element\* tail;

Deque(){

size=0;

head=0;

tail=0;

nop++;

}

**void** AddL(**int** el)//8

{

nop++;//ïðîâåðêà óñëîâèÿ

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

{

element \*a=**new** element;//1

a->next=0;//1

a->prev=0;//1

a->value=el;//1

head=a;//1

size++;//1

nop+=6;

}

**else**{

element \*a = **new** element;//1

a->next = head;//1

a->prev = 0;//1

a->value=el;//1

head->prev=a;//1

head=a;//1

size++;//1

nop+=7;

}

};

**void** AddR(**int** el)//6

{

nop++;

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

{

head = **new** element;//1

tail=head;//1

head->value=el;//1

head->next=0;//1

nop+=4;

}

**else**{

tail->next=**new** element;//1

tail->next->value=el;//1

tail->next->next=0;//1

tail=tail->next;//1

nop+=4;

}

size++;

nop++;

};

**void** DeleteR()//1+1+1+2(n-2)+2+1=2n+2

{

nop++;

element \*f = head;//1

nop++;

**if** (f == tail)//1

{

**delete** f;

head = tail = NULL;//2

nop+=3;

}

**else**

{

nop++;

**while** (f->next != tail)//1

{

f = f->next;//1

nop+=2;

}

**delete** tail;

tail = f;

tail->next = NULL;

nop+=2;

}

size--;

nop++;

};

**void** DeleteL()//7

{

nop++;

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

**return**;

**else**

{nop++;

**if**(head==tail)//1

{

head=0;//1

nop++;

}

**else**{

element \*f;//1

f=head;//1

head=f->next;//1

**delete** f;

nop+=3;

}

}

size--;

nop++;

};

**int** getQueueSize()

{

**return** size;

};

**bool** isEmpty()//1

{

nop++;

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

**return** 1;

**return** 0;

};

**int** Value()//2

{

element\* f3;//1

f3 = head;//1

nop+=2;

**return** f3->value;

};

**int** ValueR()//2

{

element\* f3;

f3 = tail;

nop+=2;

**return** f3->value;

};

};

**void** Prohod(Deque &q)//17

{

**int** temp;//1

temp = q.Value();//1+2

nop+=2;

q.DeleteL();//7

q.AddR(temp);//6

temp=0;//1

nop++;

}

**void** Show(Deque &q){

cout << "Âàø ìàññèâ - " << endl;

cout << endl;

**for** (**int** i=0; i<q.getQueueSize(); i++){

cout << q.Value() << " ";

Prohod(q);

}

cout << endl;

cout << endl;

}

**void** set(Deque &q, **int** n, **int** ne)//20n-2

{

**for** (**int** i=0; i<q.getQueueSize(); i++)//(n-1)\*(3+17)+(3+7+6)

{

nop+=3;

**if** (i==n)

{

q.DeleteL();//7

q.AddR(ne);//6

}

**else**{

Prohod(q);//17

}

}

nop+=2;

}

**int** get(Deque &q, **int** n)//20n-20+19+1+2=20n+2

{

**int** res=0;//1

nop++;

**for** (**int** i=0; i<q.getQueueSize(); i++)//(n-1)\*(3+17)+(3+3+7+6)

{

nop+=3;

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

{

nop++;

res=q.Value();//1+2

q.DeleteL();//7

q.AddR(res);//6

}

**else**{

Prohod(q);//17

}

}

nop+=2;

**return** res;

}

**void** SortirovkaVstavkoy(Deque &q, **int** n){

**int** temp, // âðåìåííàÿ ïåðåìåííàÿ äëÿ õðàíåíèÿ çíà÷åíèÿ ýëåìåíòà ñîðòèðóåìîãî ìàññèâà

item; // èíäåêñ ïðåäûäóùåãî ýëåìåíòà

nop+=2;

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

{

temp = get(q , i); // èíèöèàëèçèðóåì âðåìåííóþ ïåðåìåííóþ òåêóùèì çíà÷åíèåì ýëåìåíòà ìàññèâà

//1

item = i-1; // çàïîìèíàåì èíäåêñ ïðåäûäóùåãî ýëåìåíòà ìàññèâà

//2

nop+=3;

nop+=2;

**while**(item >= 0 && get( q , item ) > temp) //2

// ïîêà èíäåêñ íå ðàâåí 0 è ïðåäûäóùèé ýëåìåíò ìàññèâà áîëüøå òåêóùåãî

{

set(q , item+1 , get( q , item )); //1

// ïåðåñòàíîâêà ýëåìåíòîâ ìàññèâà

item--;//1

nop+=4;

}

set(q , item+1 , temp);

nop++;

}

}

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

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

q.AddR(-100+rand()%200);

}

**void** UdalenieVsego(Deque &q){

**int** n=q.getQueueSize();

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

q.DeleteL();

}

/\*\*/

**void** menu(){

cout << "0 - çàïîëíèòü ñëó÷àéíûìè ýëåìåíòàìè" << endl;

cout << "1 - äîáàâèòü ýëåìåíò ñëåâà" << endl;

cout << "2 - äîáàâèòü ýëåìåíò ñïðàâà" << endl;

cout << "3 - óäàëèòü ýëåìåíò ñëåâà" << endl;

cout << "4 - óäàëèòü ýëåìåíò ñïðàâà" << endl;

cout << "5 - âûâåñòè ïåðâûé ýëåìåíò íà óäàëåíèå ñëåâà" << endl;

cout << "6 - âûâåñòè ïåðâûé ýëåìåíò íà óäàëåíèå ñïðàâà" << endl;

cout << "7 - ñîðòèðîâêà" << endl;

cout << "8 - î÷èñòèòü âñå" << endl;

cout << "9 - âûõîä" << endl;

};

**int** main(){

srand(time(NULL));

setlocale(LC\_ALL, "Rus");

**int** h=0;

Deque q;

**while** (h!=9){

menu();

**do**{

cin >> h;

**if** (h<-1 **or** h>9)

cout << "îøèáêà" << endl;

}

**while** (h<0 **and** h>9);

**switch** (h){

**case** 0:

**int** nu;

cout << "�'êîëüêî ýëåìåíòîâ ââåñòè?" << endl;

cin >> nu;

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

cout << "Î÷åðåäü óæå ÷åì-òî çàïîëíåíà" << endl;

system("pause");

}

**else**

RandomnoeZapolnenie(q, nu);

system("CLS");

Show(q);

**break**;

**case** 1:

**int** el;

cout << "Ââåäèòå ýëåìåíò" << endl;

cin >> el;

q.AddL(el);

system("CLS");

Show(q);

**break**;

**case** 2:

**int** el1;

cout << "Ââåäèòå ýëåìåíò" << endl;

cin >> el1;

q.AddR(el1);

system("CLS");

Show(q);

**break**;

**case** 3:

q.DeleteL();

system("CLS");

Show(q);

**break**;

**case** 4:

q.DeleteR();

system("CLS");

Show(q);

**break**;

**case** 5:

cout << endl;

cout << "Ïåðâûé ýëåìåíò íà óäàëåíèå - " <<q.Value();

cout << endl;

system("pause");

system("CLS");

Show(q);

**break**;

**case** 6:

cout << endl;

cout << "Ïåðâûé ýëåìåíò íà óäàëåíèå - " <<q.ValueR();

cout << endl;

system("pause");

system("CLS");

Show(q);

**break**;

**case** 7:

SortirovkaVstavkoy(q, q.getQueueSize());

system("CLS");

Show(q);

**break**;

**case** 8:

UdalenieVsego(q);

system("CLS");

Show(q);

**break**;

}

}

**return** 1;

}

/\*int main()

{

srand(time(NULL));

setlocale(LC\_ALL, "Rus");

int h=0;

Deque q;

ofstream fl("andr.txt");

for (int i=1;i<2001;i++)

{

//fl<<(rand());

//fl<<-i;

fl<<i;

//fl<<1;

fl<<" ";

}

fl.close();

int i, N,b;

for (N=200;N<2001;N+=200)

{

ifstream f("andr.txt");

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

{

if(f >> b)

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

q.AddR(b);

}

}

int t1,t2;

nop=0;

t1=GetTickCount();

SortirovkaVstavkoy(q, q.getQueueSize());

//cout<<endl<<nop;

t2=GetTickCount();

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

cout<<N/200<<") NOP "<<nop<<endl;

UdalenieVsego(q);

f.close();

}

}

\*/

**Расчёты трудоёмкости и выводы функции роста.**

Сложность алгоритма измеряется числом сравнений и равна O(n^2).

Наилучший случай - когда исходный список уже отсортирован.

Тогда на i-ом проходе вставка производится в точке A[i],

а общее число сравнений равно n-1, т.е. сложность составляет O(n).

Наихудший случай возникает, когда список отсортирован по убыванию.

Тогда каждая вставка происходит в точке A[0] и требует i сравнений.

**Подсчёт операций в сортировке.**

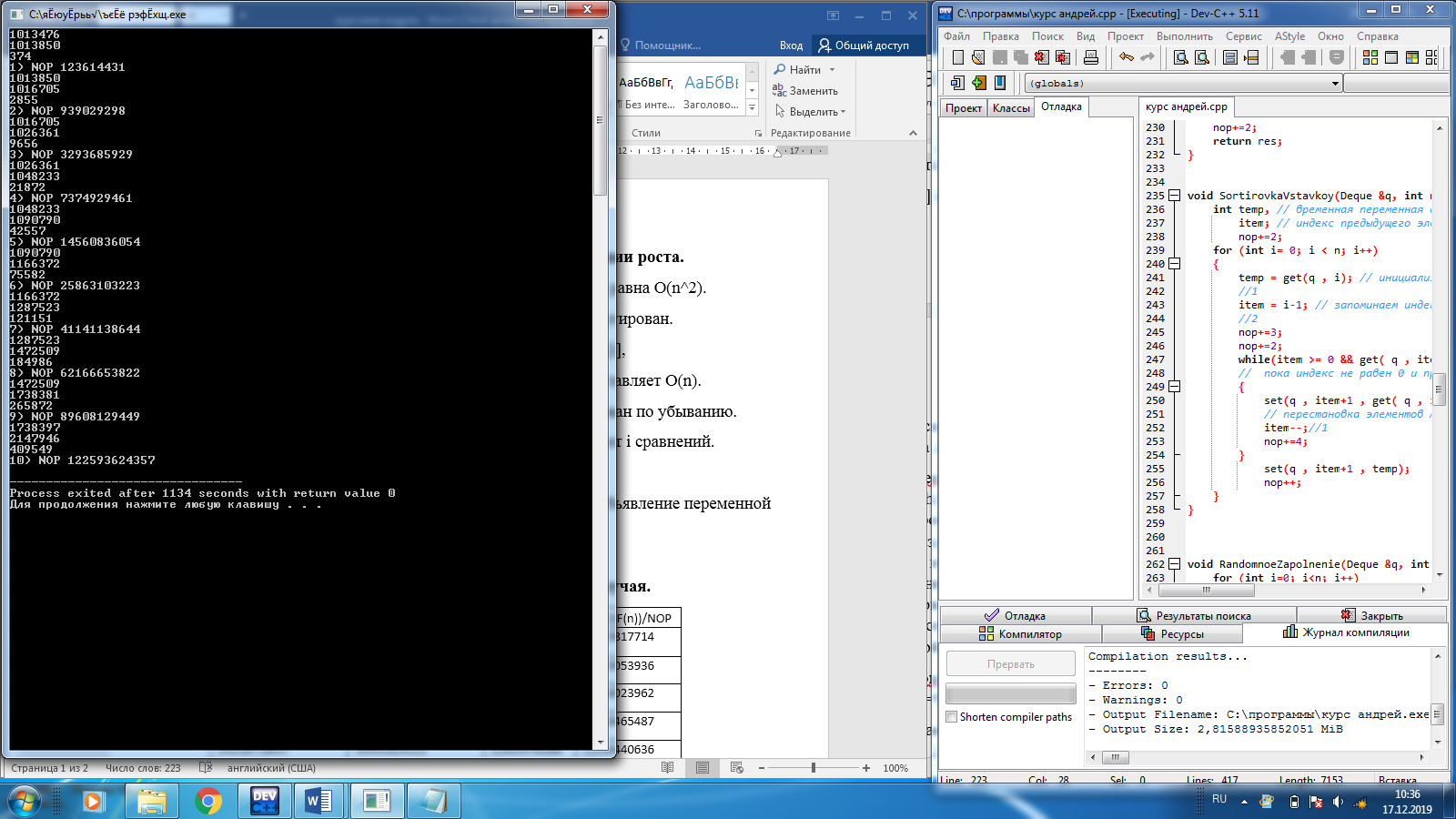
F(n)=2(*int temp,item*;)+(*цикл for (int i= 0; i < n; i++)* )n \* ( *(объявление переменной temp*) 1 + (*get(q,i);)* (20n+2) + *(item=i-1;)* 2*+( while(item >= 0 && get( q , item ) > temp))* (20n+2+2) + *(set(q , item+1 , temp);)* (20n-2+1))+ *(цикл while(item >= 0 && get( q , item ) > temp) )* ∑n-1i=0 i\*( (*проверки и get в цикле)* (2+20n+2) + *(set(q , item+1 , get( q , item ));)* (20n-2+1+20n+2) +(*item--;)* 1)=

=2+n(1+20n+2+2+2+20n+2+1+20n-2)+∑n-1i=0 i\*(2+20n+2+20n+2+20n-2+2)=

=2+n(60n+8)+(n^2-n)(6+60n)/2=2+60n^2+8n+3n^2+30n^3-3n-30n^2=**30n^3+33n^2+5n+2**

**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 |
| 645243,3 | 1283422 | 1,952207 | 3,883042 |
| 674354,5 | 1345009 | 2,05029 | 4,089329 |
| 672316 | 1342171 | 1,971009 | 3,934801 |
| 703233,5 | 1404535 | 2,085596 | 4,165464 |
| 705712,5 | 1409874 | 2,062588 | 4,120643 |
| 686506,4 | 1371755 | 2,006237 | 4,0088 |
| 680016,6 | 1358965 | 2,002489 | 4,001834 |
| 664723,2 | 1328533 | 1,977981 | 3,953245 |
| 658463,2 | 1316122 | 1,953695 | 3,905003 |
| 586332,8 | 1172021 | 1,958764 | 3,915375 |



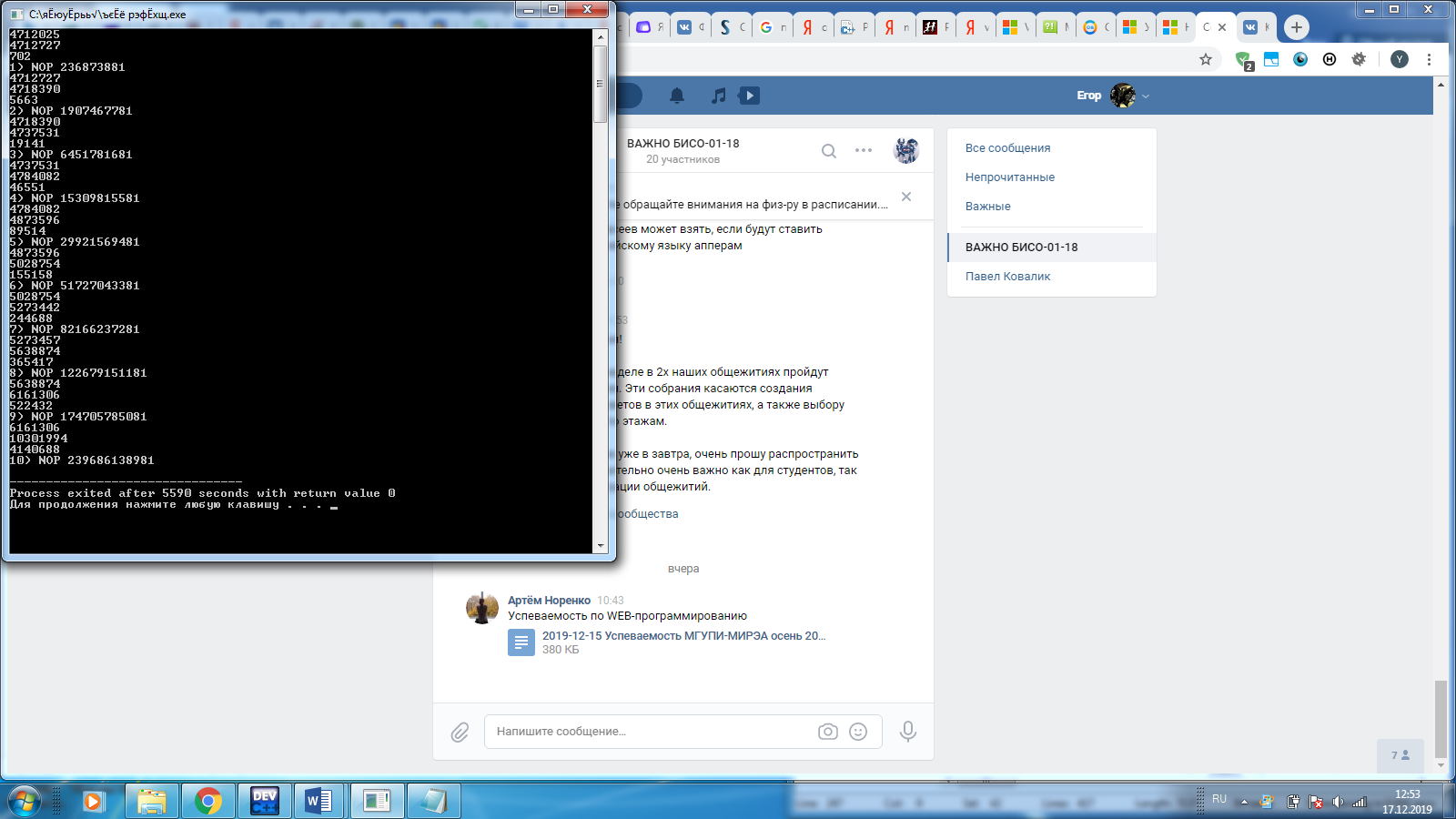
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | T nach | T kon | T kon- T nach | NOP | O(F(n))=60n^3 | F(n) |
| 200 | 1013476 | 1013850 | 374 | 123614431 | 480000000 | 241321002 |
| 400 | 1013850 | 1016705 | 2855 | 939029298 | 3840000000 | 1,925E+09 |
| 600 | 1016705 | 1026361 | 9656 | 3293685929 | 1,296E+10 | 6,492E+09 |
| 800 | 1026361 | 1048233 | 21872 | 7374929461 | 3,072E+10 | 1,538E+10 |
| 1000 | 1048233 | 1090790 | 42557 | 14560836054 | 6E+10 | 3,003E+10 |
| 1200 | 1090790 | 1166372 | 75582 | 25863103223 | 1,0368E+11 | 5,189E+10 |
| 1400 | 1166372 | 1287523 | 121151 | 41141138644 | 1,6464E+11 | 8,238E+10 |
| 1600 | 1287523 | 1472509 | 184986 | 62166653822 | 2,4576E+11 | 1,23E+11 |
| 1800 | 1472509 | 1738381 | 265872 | 89608129449 | 3,4992E+11 | 1,751E+11 |
| 2000 | 1738397 | 2147946 | 409549 | 122593624357 | 4,8E+11 | 2,401E+11 |

По графику c3 видно, что значения формулы и варианта со случайными числами сильно отличаются (больше чем в 2 раза). Докажем, что формула верна рассмотрев худший случай – элементы дека упорядочены в обратном порядке.

Нас интересует только зависимость F(n) от NOP, для того чтобы убедиться, что формула верна.

**Таблица и график для худшего случая.**

|  |  |  |  |
| --- | --- | --- | --- |
| n | NOP | C3 = F(n)/NOP | F(n) |
| 200 | 236873881 | 1,018774 | 241321002 |
| 400 | 1907467781 | 1,009339 | 1,925E+09 |
| 600 | 6451781681 | 1,006216 | 6,492E+09 |
| 800 | 15309815581 | 1,004658 | 1,538E+10 |
| 1000 | 29921569481 | 1,003724 | 3,003E+10 |
| 1200 | 51727043381 | 1,003102 | 5,189E+10 |
| 1400 | 82166237281 | 1,002659 | 8,238E+10 |
| 1600 | 122679151181 | 1,002326 | 1,23E+11 |
| 1800 | 174705785081 | 1,002067 | 1,751E+11 |
| 2000 | 239686138981 | 1,00186 | 2,401E+11 |



По графику видно, что коэффициент стремится к 1, значит формула верна. Делаем вывод: случай со случайными значениями сильно отличается от худшего случая, этим объясняется “странный” вид графика c3 для случайных значений.

**Таблицы и графики для особого случая.**

Массив уже отсортирован по убыванию или все элементы массива одинаковы.

**Подсчёт операций в сортировке.**

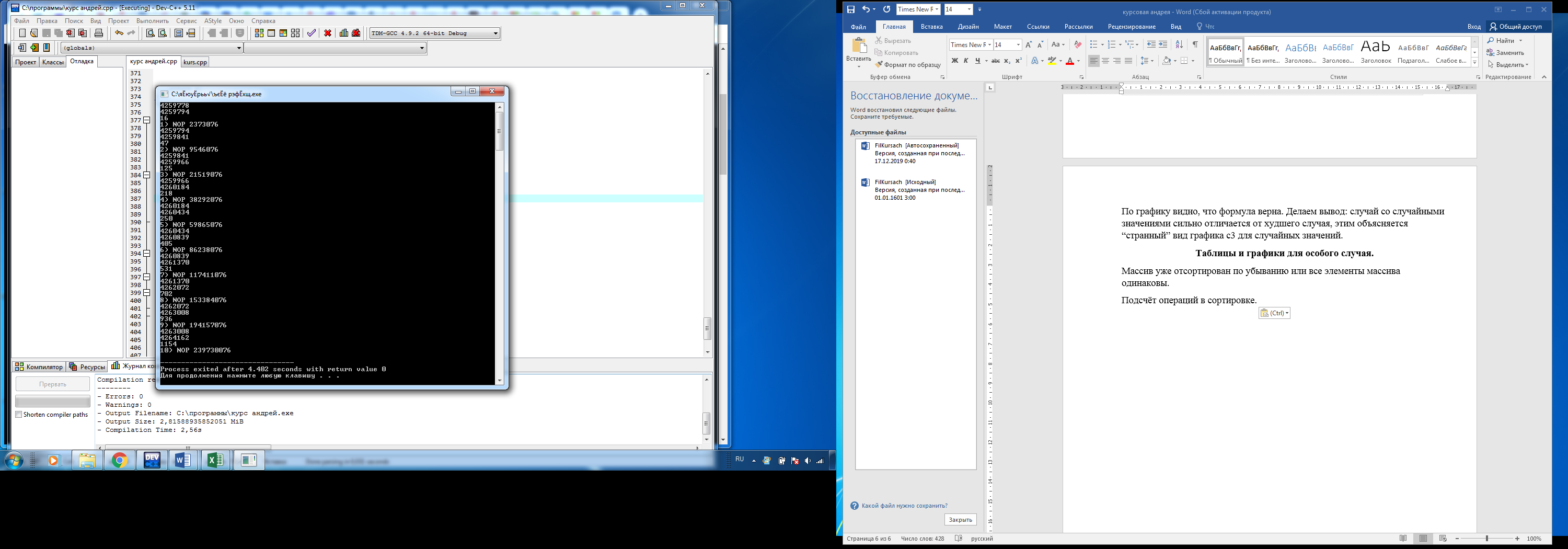
В лучшем случае перестановок элементов не требуется, поэтому в цикл(*while(item >= 0 && get( q , item ) > temp)*) не происходит ни одного входа.

F(n)=2(*int temp,item*;)+(*цикл for (int i= 0; i < n; i++)* )n \* ( *(объявление переменной temp*) 1 + (*get(q,i);)* (20n+2) + *(item=i-1;)* 2*+( while(item >= 0 && get( q , item ) > temp))* (20n+2+2) + *(set(q , item+1 , temp);)* (20n-2+1))=

=2+n(1+20n+2+2+2+20n+2+1+20n-2)=

=2+n(60n+8) =**60n^2+8n+2**

|  |  |  |  |
| --- | --- | --- | --- |
| C1 = F(n)/T(n) | C2 = O(F(n))/T(n) | C3 = F(n)/NOP | C4=O(F(n))/NOP |
| 150100,1 | 300000 | 1,012021 | 2,022691 |
| 204323,4 | 408510,6 | 1,005984 | 2,011298 |
| 172838,4 | 345600 | 1,003984 | 2,007521 |
| 176176,2 | 352293,6 | 1,002986 | 2,005637 |
| 240032 | 480000 | 1,002387 | 2,004508 |
| 213357 | 426666,7 | 1,001989 | 2,003755 |
| 221490 | 442937,9 | 1,001704 | 2,003218 |
| 218821,7 | 437606,8 | 1,001491 | 2,002815 |
| 207707,7 | 415384,6 | 1,001325 | 2,002502 |
| 207986,1 | 415944,5 | 1,001193 | 2,002252 |



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | T nach | T kon | T kon- T nach | NOP | O(F(n))=60n^3 | F(n) |
| 200 | 4259778 | 4259794 | 16 | 2373076 | 4800000 | 2401602 |
| 400 | 4259794 | 4259841 | 47 | 9546076 | 19200000 | 9603202 |
| 600 | 4259841 | 4259966 | 125 | 21519076 | 43200000 | 21604802 |
| 800 | 4259966 | 4260184 | 218 | 38292076 | 76800000 | 38406402 |
| 1000 | 4260184 | 4260434 | 250 | 59865076 | 120000000 | 60008002 |
| 1200 | 4260434 | 4260839 | 405 | 86238076 | 172800000 | 86409602 |
| 1400 | 4260839 | 4261370 | 531 | 117411076 | 235200000 | 117611202 |
| 1600 | 4261370 | 4262072 | 702 | 153384076 | 307200000 | 153612802 |
| 1800 | 4262072 | 4263008 | 936 | 194157076 | 388800000 | 194414402 |
| 2000 | 4263008 | 4264162 | 1154 | 239730076 | 480000000 | 240016002 |

**Выводы.**

Реализована сортировка простой вставкой линейного списка типа дек. Этот алгоритм реализован в виде программы на языке C++, с использованием методов объектно-ориентированного программирования. При этом создан абстрактный тип данных Deque. 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(N3), 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. Сложность алгоритма стала равной O(n^2), так как не требовалось переставлять элементы местами. При этом коэффициент пропорциональности между экспериментальной функцией роста примерно равен 2,0 для варианта случайного заполнения очереди и от 1,001 до 1,01 для особого случая.

Список источников.

Структуры данных и алгоритмы. Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. – М.: Издательский дом «Вильямс», 2016

Алгоритмы: построение и анализ. Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М.: МЦНМО, 2013.

Седжвик, Роберт, Уэйн, Кевин. Алгоритмы на Java, 4-е изд. : Пер. с англ. – М. ООО “И.Д. Вильямс”, 2013. – 848 с. : ил. – Парал. тит. англ.