Задание

Реализовать программу сортировки на двоичном дереве. В качестве исходных данных использовать случайные числа и слова текстового файла. Измерить «грязное» время выполнения программы и трудоемкость по основным операциям для различных значений

Сущность и особенности алгоритма

Двоичное дерево

Нумерация «слева-направо» дает нам основу для создания другого вида деревьев, включение в которые производится не по логическому номеру, а с учетом естественного упорядочения значений. Такое дерево называется **деревом двоичного поиска** или просто **двоичным деревом.** Основное свойство, рекурсивно соблюдаемое для всех поддеревьев, звучит так: в левом поддереве каждой вершины содержатся значения, меньшие чем в корневой, а в правом – большие. Отсюда следуют все прочие свойства такового дерева:

- обход двоичного дерева «слева-направо» соответствует порядку возрастания значений, хранящихся в нем;
- поиск заданного значения в двоичном дереве использует линейно рекурсивный (или циклический) алгоритм. В каждой вершине производится сравнение с искомым значением (ключом). Если содержимое вершины совпадает и искомым, то вершина найдена. Если искомое значение меньше, то оно ищется в левом поддереве, иначе – в правом. Данный алгоритм очень сильно напоминает нам алгоритм **двоичного поиска в массиве** . Там одно сравнение выбирает половину интервала, здесь – поддерево;
- включение нового значения в двоичного дерево также базируется на ветвлении: при сравнении включаемого значения со значением в текущей вершине выбирается правая или левая ветвь до тех пор, пока не встретится свободная.

Оценка трудоемкости

Все процедуры идентичны– дереву с обходом «справа-налево». Но по своей природе двоичное дерево соответствует идее **структурной упорядоченности данных.** Эта упорядоченность присутствует с самом определении дерева. Поэтому простое включение последовательности значение в двоичное дерево уже является их сортировкой: для этого нужно просто сделать обход полученного дерева. Что же касается трудоемкости такого процесса, то она зависит от вида полученного дерева: если оно сбалансировано, то получаем $\mathbf{T=N\log_2 N,}$ если вырождается в линейный список, то $\mathbf{T=N^2/2.}$

Вид полученного двоичного дерева в каждом случае будет различен. Это зависит от последовательности значений включаемых данных. Если в них будет цепочка одинаковых или возрастающих значений, то они создадут линейную цепочку правых потомков, которая уменьшает сбалансированность. Если данные исходно упорядочены, то при включении их в двоичное дерево последнее вырождается в правосторонний список.

Особенности сортируемых данных. Датчик случайных чисел имеет диапазон в Visual Studio 6.0 0…32000, поэтому в массиве будет реальный коэффициент повторений для случайных данных **N/(32000*K),** причем с ростом N количество повторяющихся значений будет увеличиваться.

Случайные неповторяющиеся данные. Формула трудоемкости имеет вид **Тсрав=3N $\log_2$N**

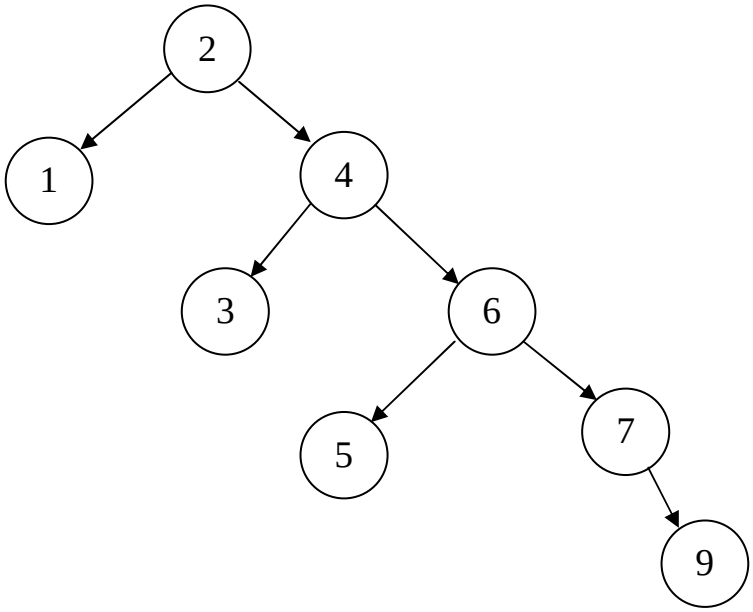
Случайные повторяющиеся данные. При наличии повторений трудоемкость будет увеличиваться . Если все значения будут одинаковы, то данные вырождаются в линейный список (**Тсрав=N^2/2**).

Прямо упорядоченные данные. Будет происходить вырождение в линейный список и трудоемкость будет равна **$T=N^2/2$.**

Обратно упорядоченные данные. Также как и с прямо упорядоченными данными**.**

Пример работы алгоритма на числовых данных

2 4 3 1 6 7 5 9



Изменения, вносимые в программу для оценки трудоемкости

Для оценки трудоемкости программы на числовых данных внесены изменения:

- добавлены счетчики операций сравнения, обменов и рекурсивных вызовов;
- создается структура данных
- вызывается функция clock() для измерения времени выполнения программы;

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

 int cnt1=0;       // сравнений
 int cnt2=0;       // ОБМЕНЫ
 int cnt3=0;       // вызовы
//-----------------------------------------------------85-02.cpp
// Двоичное дерево
// Обход дерева слева - направо (левое - текущая - правое)
struct btree{
        int cnt;            // Количество вершин в дереве
        int s;
        btree *l,*r;        // Левый и правый потомки
        };

// Создание терминальной вершины
btree *create(int ss){
        btree *q=new btree;
        q->l = q->r = NULL;
        q->cnt = 1;
        q->s = ss;
        return q; }


// Включение с сохранием порядка
void insert(btree *&p, int ss){
        cnt3++;
        if (p == NULL) { p=create(ss); return; }
        p->cnt++;
        cnt1++;
        //mif (ss==p->s) break;
        if (ss < p->s)
                insert(p->l,ss);
        else
                insert(p->r,ss);
        }

// Обход дерева
void scan(btree *p, int level, int &ln){
        if (p==NULL) return;
        scan(p->l, level+1,ln);
        printf("l=%d n=%d cnt=%d :%d\n", level, ln, p->cnt, p->s);
        ln++;
        scan(p->r, level+1,ln);
        }
```

```
void destroy(btree *p){
	if (p==NULL) return;
	destroy(p->l);
	destroy(p->r);
	delete p;
}


void main(){
int i,N,K=100;
int NN[]={50000,70000,90000,100000,150000,200000,250000,300000,0};
printf("N	time	cmp	calls\n");
for (int kk=0;NN[kk]!=0;kk++){
	btree *ph=NULL;
	srand(time(NULL));
	N=NN[kk];
	cnt1=cnt2=cnt3=0;
	long t=clock();
	for (i=0;i<N;i++) {
		insert(ph,rand()%(N/K));
// insert(ph,N);
// insert(ph,N-i);
		}

	//int mm=0;
	//scan(ph,0,mm);
	printf("%-10d%-10d%-10d%-10d\n",N,clock()-t,cnt1,cnt3);
	destroy(ph);
	}
}
```

В предыдущий вариант программы внесены следующие изменения:

- структура целых заменен на структуру указателей (int на char*);
- операция сравнения вида ss < p->s заменена на вызов функции сравнения strcmp(ss,p->s) ;

```
struct btree{
	int cnt;		// Количество вершин в дереве
	char *s;
	btree *l,*r;		// Левый и правый потомки
	};



// Включение с сохранием порядка
void insert(btree *&p, char *ss){
	cnt3++;
	if (p == NULL) { p=create(ss); return; }
	p->cnt++;
	cnt1++;
	//mif (ss==p->s) break;
	if (strcmp(ss,p->s)<0)
		insert(p->l,ss);
	else
		insert(p->r,ss);
	}

// Обход дерева
void scan(btree *p, int level, int &ln){
	if (p==NULL) return;
	scan(p->l, level+1,ln);
```

```
            char cc[80];
            CharToOem(p->s,cc);
            printf("l=%d n=%d cnt=%d :%s\n", level, ln, p->cnt, cc);
            ln++;
            scan(p->r, level+1,ln);
            }

void main(){
int i,N,K=1;
int NN[]={10000,50000,70000,90000,100000,150000,200000,250000,300000,0};
printf("N       time      cmp       calls\n");
for (int kk=0;NN[kk]!=0;kk++){
            btree *ph=NULL;
            srand(time(NULL));
            N=NN[kk];
            cnt1=cnt2=cnt3=0;
            long t=clock();
            ifstream F("text.txt");
            for (i=0;i<N;i++){
                        if (!F.good()) break;
                        char c[80];
                        F >> c;
                        insert(ph,strdup(c));
                        }
            N=i;
            int mm;
            //scan(ph,0,mm);
            printf("%-10d%-10d%-10d%-10d\n",N,clock()-t,cnt1,cnt3);
            destroy(ph);        }
}
```

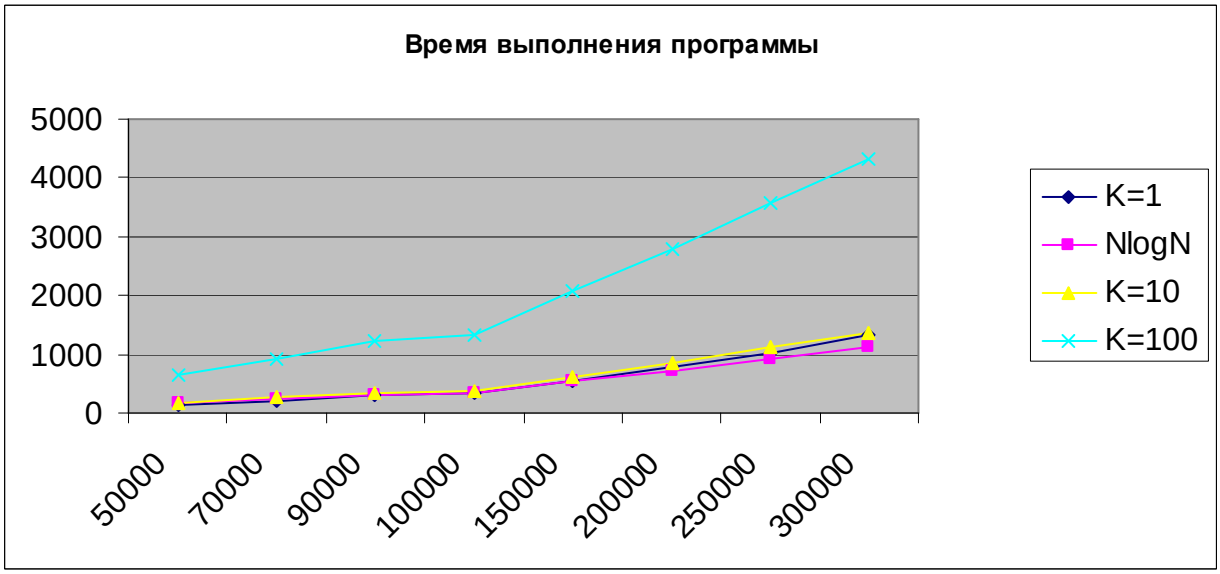
Результаты измерений и их анализ

Числовые данные

«Грязное» время выполнения программы

N	K=1	теория	отклонение	коэф по N1	K=10	K=100
50000	140	160	-12%	0,0002047004	170	630
70000	210	231	-9%		260	922
90000	291	303	-4%		331	1211
100000	340	340	0%		380	1341
150000	541	528	2%		611	2073
200000	771	721	7%		851	2804
250000	1021	918	11%		1132	3574
300000	1332	1117	19%		1362	4326

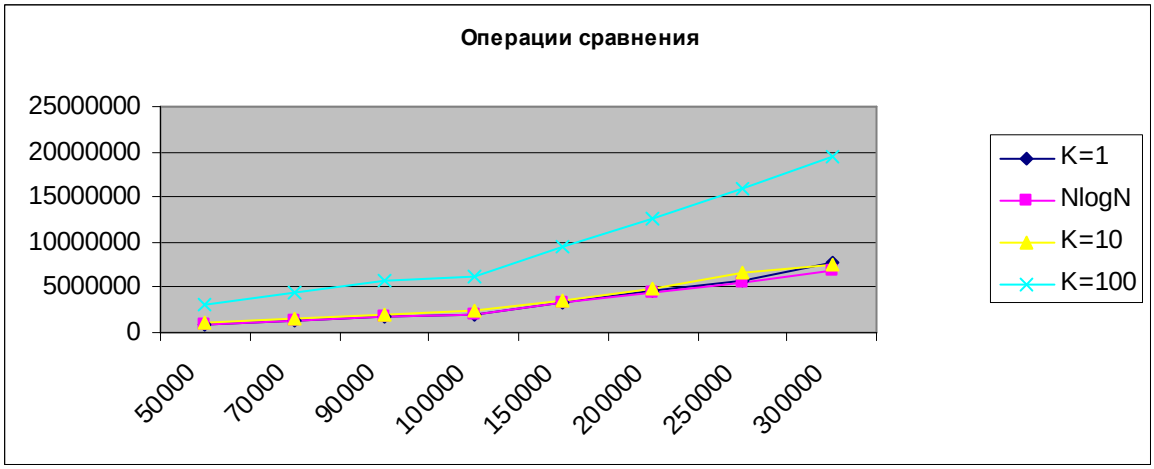
N	возрастание	убывание
3000	579	552
5000	2330	2199
7000	5219	4959
9000	8888	8539



Операции сравнения

N	K=1	матем	отклонение	коэф по N1	K=10	K=100
50000	962194	971794	-1%	1,245	1059059	3055369
70000	1389769	1402820	-1%		1596326	4399200
90000	1838311	1844256	0%		2023514	5723928
100000	2068099	2068099	0%		2350080	6238386
150000	3424825	3211401	7%		3632751	9471054
200000	4620273	4385222	5%		4778368	12696894
250000	5776393	5581737	3%		6538024	16017413
300000	7734457	6796337	14%		7551293	19546918
		1,25NlogN				

N	возрастание	убывание
3000	4498500	4498500
5000	12497500	12497500
7000	24496500	24496500
9000	40495500	40495500
	N*(N-1)/2	



Рекурсивные вызовы

N	K=1	мат	откл	коэф по N1	K=10	K=100
50000	1012194	1018783	-1%	1,3053	1109059	3105369
70000	1459769	1470651	-1%		1666326	4469200
90000	1928311	1933432	0%		2113514	5813928
100000	2168099	2168099	0%		2450080	6338386
150000	3574825	3366683	6%		3782751	9621054
200000	4820273	4597263	5%		4978368	12896894
250000	6026393	5851634	3%		6788024	16267413
300000	8034457	7124965	13%		7851293	19846918
		1,3NlogN				

N	возрастание	убывание
3000	4501500	4501500
5000	12502500	12502500
7000	24503500	24503500
9000	40504500	40504500
	(N^2)/2	

Анализ результатов:

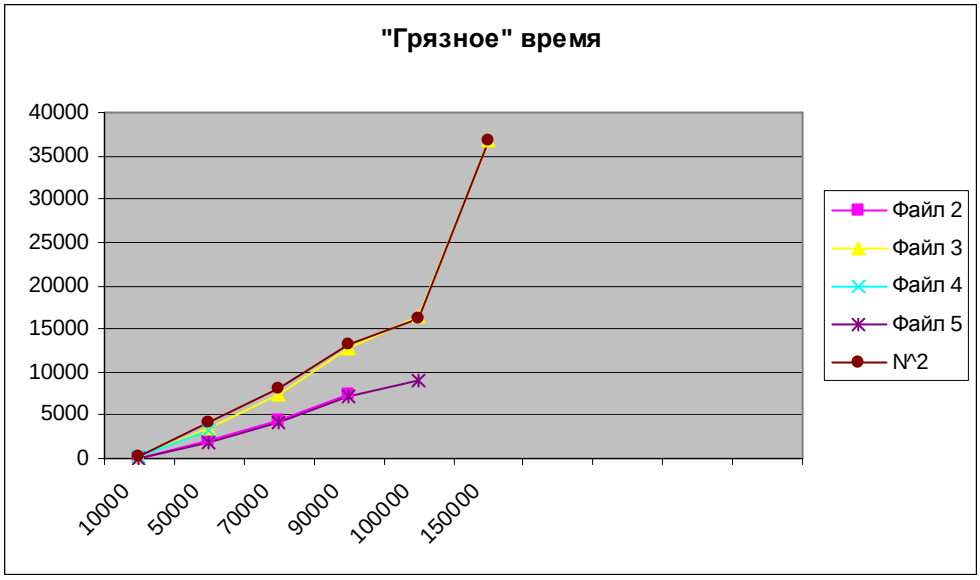
1. Количество операций сравнения соответствует формуле **Тсрав=3,8Nlog$_2$N** с отклонением до 14% (в теории **Тсрав=1,25Nlog$_2$N**), что несколько больше за счет повторяющихся значений. При этом с увеличением N повторения возрастают и трудоемкость сильнее отличается от теоретической.
2. При увеличении числа повторений (**K=10,100**) трудоемкость по операциям сравнения увеличивается. При возрастании и убывании данных она, возрастает до теоретической **Тсрав=N^(N-1)/2.**
3. Трудоемкость рекурсивных соответствует формуле **Трек=1,8NlogN**
4. - при возрастающих и убывающих данных трудоемкость практически равна теоретической **Трек=N^2**
5. при повторении **K=10,100** получаем большое количество рекурсивных вызовов.
6. «грязное» время изменяется линейно - логарифмически с отклонением 19% и чувствительно к большому количеству повторов и упорядоченным данным.

Текстовые файлы

Производилось измерение производительности программы для 4 различных тестовых файлов. Оценка вида зависимости производилась по самому длинному файлу (3)

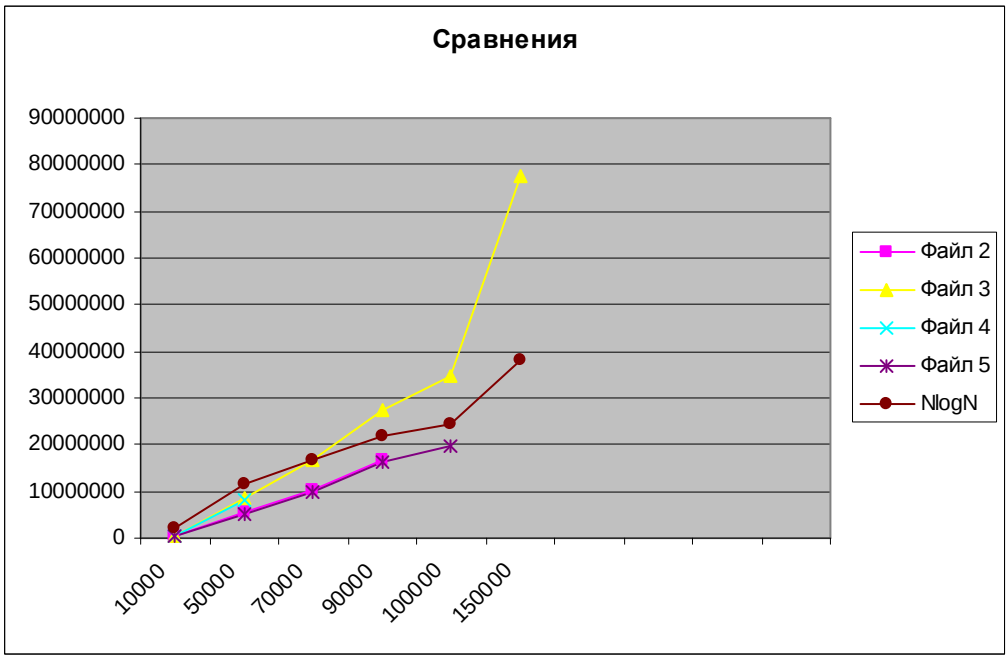
«Грязное» время работы программы

	Файл 2	Файл 3	Файл 4	Файл 5				
10000	100	160	130	100	163	-2%	Степень	
50000	2122	3354	3154	1952	4074	-18%		**2**
70000	4416	7379		4215	7985	-8%		
90000	7419	12606		7219	13199	-4%		
100000		16350		9071	16296	0%		
150000		36665			36665	0%	0,0000016296	



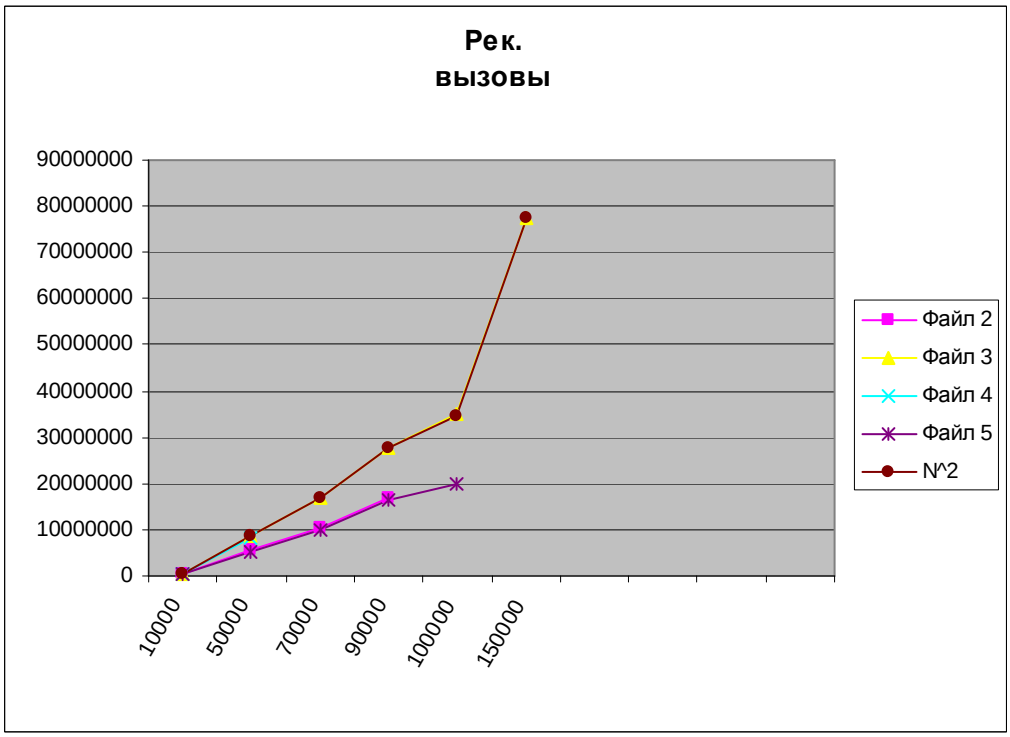
Операций сравнения

	Файл 2	Файл 3	Файл 4	Файл 5	NlogN	Отклонение	
10000	347603	573179	499780	324940	1970138	-71%	
50000	5586639	8532111	7974826	5169877	11572025	-26%	
70000	10381099	16704647		9900035	16704647	0%	
90000	16746274	27591507		16139450	21961220	26%	
100000		34853205		19819354	24626727	42%	
150000		77368320			38241056	102%	14,8267670966



Рекурсивные вызовы

	Файл 2	Файл 3	Файл 4	Файл 5		Отклонение	
10000	357603	583179	509780	334940	344526	69%	Степень
50000	5636639	8582111	8024826	5219877	8613147	0%	**2**
70000	10451099	16774647		9970035	16881767	-1%	
90000	16836274	27681507		16229450	27906595	-1%	
100000		34953205		19919354	34452587	1%	
150000		77518320			77518320	0%	0,0034452587



Анализ результатов:

1. Операция сравнения довольно не точно соответствуют зависимости вида **$N\log_2 N$ – $T_{срав}=14{,}8N\log_2 N$** с отклонением 102% на больших размерностях так как в тексте возможны повторения что заметно ухудшает работу данного алгоритма.
2. Количество рекурсивных вызовов довольно точно соответствуют зависимости вида **N^2 – $T_{рек}=0.003$ N^2** с отклонением **1%** на больших размерностях
3. На маленьких размерностях до **69%** имеются достаточно большие отклонения, что соответствует неустановившемуся процессу и влиянию случайных факторов.
4. «Грязное» время работы программы соответствует зависимости вида **$N\log_2 N$,**.

Выводы

1. На числовых данных и текстовых файлах в целом сортировка имеет линейно-логарифмическую трудоемкость и сильно чувствительна к данным.