Министерство образования и науки РФ
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Отчёт по дисциплине «Алгоритмы и структуры данных»

Реализация алгоритмов сортировки

Работу выполнил:

студент группы 3530904/00006 Смирнов Е. А.

Работу принял:

преподаватель Павлов Е. А

Санкт-Петербург
2021

Постановка задачи

1. Реализация указанных сортировок

    1.1 Сортировка по методу вставок с двоичным поиском места вставки

    1.2 Сортировка Шелла с выбором длин промежутков, предложенных Шеллом, Хиббардом, Седжвиком. Сравнение результатов.

    1.3 Сортировка поразрядная

2. Описание проведенного тестирования

    2.1 Выполнить тесты для лучшего, худшего и среднего случаев

    2.2 Сравнить результаты экспериментальной оценки временной сложности с теоретическими

Реализация

1. Сортировка вставками с бинарным поиском

```
void binaryInsertion(int inputArray[], int size)
{
	int left = 0;  // индекс первого элемента отсортированного массива
	int right = 0; //  индекс последнего элемента отсортированного массива
	int temp;

	for (int i = 1; i < size; i++)
	{
		if (inputArray[i] < inputArray[i-1])
		{
		 temp = inputArray[i];
			left = 0;
			right = i - 1;
			while (left <= right) // бинарный поиск
			{
				int mid = (left + right) / 2;
				if (inputArray[mid] < temp)
				{left = mid + 1;}
				else
				{right = mid - 1;}
			}
			for (int j = i - 1; j >= left; j--)// сдвиг вправо
			{inputArray[j + 1] = inputArray[j];}
			inputArray[left] = temp;
		}
	}
}
```

2. Поразрядная сортировка

```
void radix(int inputArray[], int size, int d, int outArray[])
{
	int temp = 0;
	const int divider = 10;
	int integer = 1;

	for (int k = 0; k < d; k++) // цикл по разрядам
	{
		int count[10]{ 0,0,0,0,0,0,0,0,0,0 };

		// количесво элементов от 0 до 9
		for (int i = 0; i < size; i++)
			count[(inputArray[i] / integer) % divider]++;

		// количество чисел меньших i-индекса
		for (int i = 1; i < 10; i++)
			count[i] = count[i] + count[i - 1];

		// заполнение выходного массива
		for (int i = size - 1; i >= 0; i--)
		{
			temp = (inputArray[i] / integer) % divider;
			count[temp]--;
			outArray[count[temp]] = inputArray[i];
		}

		integer *= 10;
		std::swap(inputArray, outArray);
	}
}
```

3. Сортировка Шелла

3.1 с шагом Шелла

```
void shellShell(int inputArray[], int size)
{
  int step = size;
  for (step = step / 2; step > 0; step = step / 2)
    {
    for (int i = step; i < size; i++)
      {
       for (int j=i-step; j>=0 && inputArray[j]>inputArray[j+step]; j-=step)

       {
          std::swap(inputArray[j], inputArray[j + step]);
       }
     }
   }
 }
```

3.2 с шагом Хиббарда

```
void shellHibbard(int inputArray[], int size)
{
int I = 1;
while (pow(2, I) - 1 <= size)  // O(logN)
{
  I++;
}
while (I > 0)
 {
  int step = pow(2, I) - 1;
  for (int i = step; i < size; i++)
    {
    for (int j = i - step; j >= 0 && inputArray[j] > inputArray[j + step]; j -= step)
       {
         std::swap(inputArray[j], inputArray[j + step]);
       }
    }
    I--;
 }
}
```

3.3 с шагом Сэджвика

```
void shellSedgwick(int inputArray[], int size)
{
	double p1 = 1;
	double p2 = 1;
	double p3 = 1;

	int inc[20];// массив, в который заносятся шаги


	int s = -1;
	do {
		// заполняем по формуле Роберта Седжвика
		if (++s % 2) {
			inc[s] = 8 * p1 - 6 * p2 + 1;
		}
		else {
			inc[s] = 9 * p1 - 9 * p3 + 1;
			p2 *= 2;
			p3 *= 2;
		}
		p1 *= 2;
	} while (3 * inc[s] < size);
	s--;
	int step = 0;

	while (s >= 0) {
		//извлекаем из массива очередную инкременту
		step = inc[s--];
		// сортировка вставками с инкрементами inc
		for (int i = step; i < size; i++)
		{
			// сдвигаем элементы до тех пор, пока не дойдем до конца или не
упорядочим в нужном порядке
			for (int j = i - step; j >= 0 && inputArray[j] > inputArray[j +
1]; j -= step)
			{
				std::swap(inputArray[j], inputArray[j + step]);
			}
		}
	}
}
```

Тестирование

```
void testBinaryInsertion()
{
	using namespace std;

	cout << "\nСортировка вставками с двоичным поиском\n";

	cout << "\nЛучший случай: ";
	int bestArray[SIZE] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
	binaryInsertion(bestArray, SIZE);
	print(bestArray, SIZE);

	cout << "\nСредний случай: ";
	int midArray[SIZE] = {1, 2, 3, 5, 7, 4, 9, 10, 8, 6};
	binaryInsertion(midArray, SIZE);
	print(midArray, SIZE);

	cout << "\nХудший случай: ";
	int worstArray[SIZE] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
	binaryInsertion(worstArray, SIZE);
	print(worstArray, SIZE);

	cout << endl;
}


void testRadix()
{
	using namespace std;
	cout << "\nПоразрядная сортировка\n";

	cout << "\nЛучший случай: ";
	int outbest[SIZE]{0,0,0,0,0,0,0,0,0,0};
	int bestArray[SIZE] = { 100, 102, 113, 214, 225, 336, 347, 458, 459, 569 };
	radix(bestArray, SIZE, 3, outbest);
	print(outbest, SIZE);

	cout << "\nСредний случай: ";
	int outmid[SIZE]{ 0,0,0,0,0,0,0,0,0,0 };
	int midArray[SIZE] = { 101, 254, 336, 328, 439, 212, 223, 335, 447, 459 };
	radix(midArray, SIZE, 3, outmid);
	print(outmid, SIZE);

	cout << "\nХудший случай: ";
	int outworst[SIZE]{ 0,0,0,0,0,0,0,0,0,0 };
	int worstArray[SIZE] = { 989, 878, 767, 656, 545, 434, 323, 212, 101, 100 };
	radix(worstArray, SIZE, 3, outworst);
	print(outworst, SIZE);

	cout << endl;
}
```

```
void testShell()
{
	using namespace std;
	cout << "\n此E刀\n";

	int shell[SIZE] = { 900, 800, 700, 600, 500, 400, 300, 200, 100, 10 };
	shellShell(shell, SIZE);
	print(shell, SIZE);

	int shell1[SIZE] = { 989, 878, 767, 656, 545, 434, 323, 212, 101, 100 };
	shellHibbard(shell1, SIZE);
	print(shell1, SIZE);

	int shell2[SIZE] = { 989, 878, 767, 656, 545, 434, 323, 212, 101, 100 };
	shellSedgwick(shell2, SIZE);
	print(shell2, SIZE);
}
```

Вычисление сложности Сортировка простыми вставками с бинарным поиском:

Линия сравнения	**Худший случай**	**Средний случай**	**Лучший случай**
Внешний цикл	N-1	N-1	N-1
Бинарный поиск	O(logN)	O(logN)	O(1)
Сдвиг (число обменов)	O(N*(N+1))/2 = O(N^2/2)	O(N*N(+1)/2*2 = O(N^2/4)	O(1)
Итого:	**O(N^2/2)**	**O(N^2/4)**	**O(N)**

1. Присвоение temp, left, right … = O(константы) – не влияет на вычисление сложности алгоритма.
2. For() – внешний цикл, в любом случае будет выполнено n-1 сравнений, поэтому его сложность O(N);
3. При лучшем случае произойдет только N-1 сравнений, в других случаях будет выполняться бинарный поиск, сложность которого O(logN).
4. Сами вставки (сдвиг элементов и вставка i-ого элемента) выполняется за O(N(N-1)/2) в худшем случае.

Алгоритм работает за O(N+k) где k — число обменов элементов входного массива.

Табличная сложность алгоритмов :

Алгоритм	Худший случай	Средний случай	Лучший случай
Вставками с бинарным поиском	O(N^2/2)	O(N^2/4)	O(N)
Поразрядная сортировка	O(N*k), k – число разрядов	O(N*k)	O(N*k)

Для поразрядной сортировки нужна доп. Память O(N).

Сортировка Шелла с выбором длины промежутка предложенные:

Линия сравнения	Худший случай	Средний случай	Лучший случай
Шеллом	O(N^2)	O(N*logN)	O(N*logN)
Хиббардом	O(N^1.5)	O(N*logN)	O(N*logN)
Сэджвиком	O(N^4/3)	O(N^7/6)	O(N*logN)

Экспериментальная сложность алгоритмов:

Линия сравнения	Худший случай	Средний случай	Лучший случай
Вставками с бинарным поиском	O(N^2/2)	O(N^2/4)	O(N)
Поразрядная сортировка	O(N*k), k – число разрядов	O(N*k)	O(N*k)
Шеллом	O(N*logN – (N-1)N/N*N)= O(NlogN)=**O(N^2)**	O(N*logN – (N-1)N/N)= **O(NlogN)**	O(N*logN)
Хиббардом	O(N*logN – (N-1)N/N*N)= O(NlogN)=**O(N^2)**	O(N*logN – (1-N)N/N)= **O(NlogN)**	O(N*logN)
Сэджвиком	O(N^4/3)	O(N^7/6)	O(N*logN)