ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное образовательное бюджетное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет телекоммуникаций
им. проф. М. А. Бонч-Бруевича»

Отчет по лабораторной работе №9

«Сравнительный анализ алгоритмов сортировки»

по дисциплине

«Алгоритмические основы программной инженерии»

Выполнил ст. группы_____

ФИО_____

2017г, СПб

1. **Реализация алгоритма сортировки «Пузырьком»:**

```
void BubbleSort(int *m, int dlina) {
    int buff, j;
    for (int i = 0; i < dlina; i++) {
           for (j = i; j > 0; j--) {
                  if (m[j] < m[j - 1]) {
                         buff = m[j - 1];
                         m[j - 1] = m[j];
                         m[j] = buff;
                  }
           }
    }
}
```

Сложность сортировки	$O(n^2)$
Лучшее время	$O(n^2)$
Среднее время	$O(n^2)$
Худшее время	$O(n^2)$

Полученные данные

Последовательность. длина / степень отсортированности	Полученное время (или количество перестановок)
20000 / 70%	0.218 сек
20000 / 50%	0.260 сек
8000 / 70%	0.034 сек
8000 / 30%	0.047 сек
1000 / 50%	0.61 млсек
1000 / 30%	0.727 млсек

Выводы по сортировке «Пузырьком»:

Данная сортировка очень медленно работает (квадратичная сложность т. е. для сортировки массива в N элементов операций надо сделать N * N), что видно по полученным данным, но она очень проста с точки зрения понимания и реализации. Время сортировки слабо зависит от степени отсортированности массива. Я считаю, что кроме учебных целей сортировку «пузырьком» применять негде.

2. Реализация алгоритма сортировки «Выбором»:

```
void ChoiceSort(int *m, int dlina) {
    int buff, min;
    for (int i = 0; i < dlina; i++) {
           min = i;
           for (int j = i + 1; j < dlina; j++)
                  if (m[j] < m[min])
                         min = j;
           buff = m[i];
           m[i] = m[min];
           m[min] = buff;
    }
   }
```

Сложность сортировки	$O(n^2)$
Лучшее время	$O(n^2)$
Среднее время	$O(n^2)$
Худшее время	$O(n^2)$

Полученные данные

Последовательность. длина / степень отсортированности	Полученное время (или количество перестановок)
20000 / 70%	0.779 сек
20000 / 50%	0.777 сек
8000 / 70%	0.124 сек
8000 / 30%	0.124 сек
1000 / 50%	1.918 млсек
1000 / 30%	1.933 млсек

Выводы по сортировке «Выбором»:

Сортировка «выбором» - самая медленная из всех представленных здесь сортировок. Она очень простая, но медленно работает (сложность и время сортировки n^2). Время сортировки не зависит от степени отсортированности массива (будь хоть полностью не отсортированный массив, время сортировки будет такой же).

3. **Реализация алгоритма сортировки «Вставкой»:**

```
void insertSort(int *m, int dlina) {
    int buff, j;
    for (int i = 0; i < dlina; i++) {
           buff = m[i];
           for (j = i; j > 0; j--) {
                  if (buff < m[j - 1])
                         m[j] = m[j - 1];
                  else
                         break;
           }
           m[j] = buff;
    }
    }
```

Сложность сортировки	$O(n^2)$
Лучшее время	O(n)
Среднее время	$O(n^2)$
Худшее время	$O(n^2)$

Полученные данные

Последовательность. длина / степень отсортированности	Полученное время (или количество перестановок)
20000 / 70%	0.018 сек
20000 / 50%	0.05 сек
8000 / 70%	2.838 млсек
8000 / 30%	17.643 млсек
1000 / 50%	0.126 млсек
1000 / 30%	0.243 млсек

Выводы по сортировке «Вставкой»:

Неоднозначные получаются выводы по сортировке «вставкой». Не смотря на такую же сложность сортировки, как и у «выбором» и у «пузырьковой», она сортирует в разы быстрее. При частичной отсортированности массива сортировка проходит быстрее. Если немного улучшить данную сортировку, то она будет работать намного быстрее (сортировка «Шелла» - одна из модификаций сортировки «вставкой»). Она простая для реализации на языках программирования.

4. **Реализация алгоритма сортировки «Шелла»:**

```
void ShellSort(int *m, int len) {
    int j, tmp;
    for (int k = len / 2; k > 0; k /= 2)
           for (int i = k; i < len; i++) {
                  tmp = m[i];
                  for (j = i; j >= k; j -= k) {
                         if (tmp < m[j - k])
                                m[j] = m[j - k];
                         else
                                break;
                  }
                  m[j] = tmp;
           }
    }
```

Сложность сортировки	O(n log2 n)
Лучшее время	$O(n \log^2 n)$
Среднее время	зависит от выбранных шагов
Худшее время	$O(n^2)$

Полученные данные

Последовательность. длина / степень отсортированности	Полученное время (или количество перестановок)
20000 / 70%	0.787 млсек
20000 / 50%	0.785 млсек
8000 / 70%	0.252 млсек
8000 / 30%	0.291 млсек
1000 / 50%	0.033 млсек
1000 / 30%	0.028 млсек

Выводы по сортировке «Шелла»:

Сортировка «Шелла» - одна из самых быстрых сортировок, известных на данный момент. Сортирует массив из двадцати тысяч элементов менее, чем за миллисекунду. Однако, по полученным данным, сложно сказать, зависит ли время сортировки от степени отсортированности массива. Так же, можно настраивать сортировку под свои нужны, увеличивая или уменьшая значение «k» (при сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии «k»).

5. **Реализация алгоритма сортировки «Пирамидой»:**

```
void siftDown(int *numbers, int root, int bottom)
{
    int maxChild;
    int done = 0;
    while ((root * 2 <= bottom) && (!done))
    {
           if (root * 2 == bottom)
                  maxChild = root * 2;
           else if (numbers[root * 2] > numbers[root * 2 + 1])
                  maxChild = root * 2;
           else
                  maxChild = root * 2 + 1;
           if (numbers[root] < numbers[maxChild])
           {
                  int temp = numbers[root];
                  numbers[root] = numbers[maxChild];
                  numbers[maxChild] = temp;
                  root = maxChild;
           }
           else
                  done = 1;
    }
} //для пирамидальной сортировки (построение дерева)
void heapSort(int *numbers, int array_size)
{
    for (int i = (array_size / 2) - 1; i >= 0; i--)
           siftDown(numbers, i, array_size);
    for (int i = array_size - 1; i >= 1; i--)
    {
           int temp = numbers[0];
           numbers[0] = numbers[i];
           numbers[i] = temp;
           siftDown(numbers, 0, i - 1);
    }
   }
```

Сложность сортировки	O(n log n)
Лучшее время	O(n log n)
Среднее время	O(n log n)
Худшее время	O(n log n)

Полученные данные

Последовательность. длина / степень отсортированности	Полученное время (или количество перестановок)
20000 / 70%	1.527 млсек
20000 / 50%	1.682 млсек
8000 / 70%	0.533 млсек
8000 / 30%	0.622 млсек
1000 / 50%	0.066 млсек
1000 / 30%	0.068 млсек

Выводы по сортировке «Пирамидой»:

«Пирамидальная» сортировка - одна из самых быстрых сортировок, известных на данный момент, однако, она сложна, как и для понимания, так и для реализации на языках программирования. Можно заметить, что время сортировки зависит от времени отсортированности массива. Если сравнить сортировку «Пирамидой» с сортировкой «Шелла», что получается, что код «Пирамиды» в 3 раза длиннее, сортирует так же в 3 раза дольше. Я считаю, что не стоит пользоваться данной сортировкой в своих проектах.

6. **Реализация алгоритма сортировки «Быстрая»:**

```
void q_sort(int *arr, int j) {
    int size = j;
    int k = 0;
    int c = arr[(int)(size / 2)];
    int tmp;
    do {
           while (arr[k] < c)k++;
           while (arr[j] > c)j--;
           if (k <= j) {
                  tmp = arr[k];
                  arr[k] = arr[j];
                  arr[j] = tmp;
                  k++;
                  j--;
           }
    } while (k <= j);
    if (j > 0) q_sort(arr, j);
    if (size > k) q_sort(&arr[k], size - k);
   }
```

Сложность сортировки	O(n log n)
Лучшее время	O(n) или O(n log n)
Среднее время	O(n log n)
Худшее время	$O(n^2)$

Полученные данные

Последовательность. длина / степень отсортированности	Полученное время (или количество перестановок)
20000 / 70%	0.318 млсек
20000 / 50%	0.329 млсек
8000 / 70%	0.126 млсек
8000 / 30%	0.131 млсек
1000 / 50%	0.017 млсек
1000 / 30%	0.017 млсек

Выводы по сортировке «Быстрая»:

Самая быстрая сортировка из всех представленных. Сортирует массив почти мгновенно. Время сортировки коротких массивов приближается к микросекунде. Она проста и в реализации, и в понимании ее работы. Зависимость времени сортировки от степени отсортированности – минимальная.

ОБЩИЕ ВЫВОДЫ:

Сравним все шесть сортировок: «Быстрая», «Шелла», «Пирамидой», «Вставкой», «Пузырьком» и «Выбором».

Самой быстрой сортировкой оказалась: «Быстрая». Она быстрее всех сортирует массивы, как и короткие, так и длинные.

Самой медленной сортировкой оказалась: сортировки «Выбором». Массив из двадцати тысяч элементов сортирует за 0.78 секунду. Это очень долго.

Объемы памяти, затраченные для реализации сортировок:

Быстрая: 353 байт.

Пирамида: 850 байт.

Пузырьком: 223 байта.

Выбором: 236 байт.

Вставкой: 230 байт.

Самой ресурсоёмкой сортировкой оказалась «Пирамида». Меньше всего памяти для реализации на языке C++ понадобилось «Пузырьковой» сортировке. Однако, если размер памяти сильно ограничен, рекомендую использовать сортировку «Шелла». Если её правильно настроить под свои нужды, то она будет мгновенно отсортировывать массив.

Но, по моему мнению, «золотой серединой» оказалась «Быстрая» сортировка. Маленький объем памяти, затрачиваемый для её реализации, малое время сортировки, малая сложность сортировки (зависит от ситуации).