Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Отчет по заданию №1

«Методы сортировки»

Вариант 2 / 3 / 1 / 5

Выполнил:
студент 119 группы
Дроздов Н.А.

Преподаватель:
Сковорода Н.А.s

Москва
2021

Содержание

Постановка задачи

Требуется реализовать два метода сортировки массивов и провести их сравнение между собой, а также проверить теоретические оценки их сложности на практике.

- Методы, которые нужно сравнить - сортировка методом "пузырька"и пирамидальная сортировка;
- Тип сортируемых данных – long long int (64-битные целые числа);
- Массив нужно отсортировать по неубыванию модулей чисел.

Способ сравнения: реализация обеих методов на языке программирования C, подсчет числа сравнений и обменов элементов в каждом методе на случайно сгенерированных массивах длины 10, 100, 1000, 10000.

Результаты экспериментов

Приведем теоретические оценки числа сравнений и перемещений для рассматриваемых сортировок, а так же их вычислительной сложности. Для сортировки "пузырьком" число сравнений для массива длины n всегда фиксированное: $(n^2 - n)/2$; число обменов элементов может колебаться от 0 в лучшем случае до $3(n^2 - n)/2$ в худшем случае [2]. Асимптотическая сложность пузырьковой сортировки – $O(n^2)$.

В пирамидальной сортировке количество сравнений и обменов за одну процедуру "просеивания" оценивается в $O(\log n)$ [1]. Всего таких вызовов $O(n)$, поэтому сложность сортировки пирамидой - $O(n \log n)$.

В таблицах ниже приведены результаты запуска соответствующих сортировок на массивах разной длины и структуры (1 и 2 номер массива - расстановка элементов в массиве случайна, 3 - элементы уже упорядочены, 4 - элементы упорядочены в обратном порядке).

n	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	45	45	45	45	45
	Перемещения	24	21	0	45	23
100	Сравнения	4950	4950	4950	4950	4950
	Перемещения	2187	2496	0	4950	2408
1000	Сравнения	499500	499500	499500	499500	499500
	Перемещения	249894	252096	0	499500	250373
10000	Сравнения	49995000	49995000	49995000	49995000	49995000
	Перемещения	24988337	24581501	0	49995000	24891210

Таблица 1: Результаты работы сортировки "пузырьком"

n	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	60	64	70	52	62
	Перемещения	25	27	30	21	26
100	Сравнения	1248	1308	1380	1132	1267
	Перемещения	574	604	640	516	584
1000	Сравнения	19136	19278	20416	17632	19116
	Перемещения	9068	9139	9708	8316	9058
10000	Сравнения	258344	258458	273912	243392	258527
	Перемещения	124172	124229	131956	116696	124314

Таблица 2: Результаты работы пирамидальной сортировки

По результатам работы обеих сортировок видно значительное превосходство сортировки пирамидой над пузырьковой (что, впрочем, было очевидно по их асимптотике). Суммарное значение обменов и сравнений в пирамидальной сортировке росло в 21.03, 15.22 и 13.48 раз при переходе от размера массива n к размеру $10n$. Среднее значение такого роста - 16.57 раз, что вполне соответствует заявленной ранее сложности ($5 \log_2 10 \approx 16.6$). Аналогичное значение у сортировки пузырьком – 92.3 раза, что так же соответствует квадратичному росту числа операций.

Структура программы и спецификация функций

Функция, генерирующая массив нужного формата:

```
long long int *mas_gen(int n, int type){
  int i;
  long long int *a = (long long int *)malloc(n * sizeof(long long int));

  if(type < 3){
    srand(time(NULL));
    for(i = 0; i < n; i++){
      a[i] = (long long int) (rand() + 1) * (rand() + 1) * (rand() + 1);
      a[i] = a[i] * (rand() % 2 ? 1 : -1);
    }
  }
  else if(type == 3){
    for(i = 0; i < n; i++){
      a[i] = i + 1;
      a[i] = a[i] * (rand() % 2 ? 1 : -1);
    }
  }
  else{
    for(i = 0; i < n; i++){
      a[i] = n - i;
      a[i] = a[i] * (rand() % 2 ? 1 : -1);
    }
  }

  return a;
}
```

Функция - компаратор:

```
int cmp(long long int a, long long int b){
  a = a > 0 ? a : -a;
  b = b > 0 ? b : -b;
  return a > b;
}
```

Функция, осуществляющая сортировку пузырьком и вывод информации о обменах и сравнениях:

```
void Bubble_Sort(long long int *a, int n){
  int swaps = 0;
  int compares = 0;
  swaps = compares = 0;

  for(int i = n - 1; i > 0; i--){
    for(int j = 0; j < i; j++){
      compares++;
      if(cmp(a[j], a[j + 1])){
        swap(&a[j], &a[j + 1]);
```

```
        swaps++;
      }
    }
  }

  printf("%s %d %s\n", "---- BUBBLE SORT WITH N = ", n, " ----");
  printf("%s %d %c %s %d\n", "Swaps: ", swaps, ' ', "Compares: ",
     compares);
}
```

Функция, просеивающая элемент с индексом i в пирамиде:

```
void sift(long long int *a, int i, int n, int *swaps, int *compares){
  long long int parent_ind, max_son_ind, elem;
  parent_ind = i; max_son_ind = 2 * i + 1, elem = a[i];
  (*compares) += 2;

  if(max_son_ind < n - 1 && !cmp(a[max_son_ind], a[max_son_ind + 1])){
    max_son_ind++;
  }
  while(max_son_ind < n && !cmp(elem, a[max_son_ind])){
    a[parent_ind] = a[max_son_ind];
    a[max_son_ind] = elem;
    parent_ind = max_son_ind;
    max_son_ind = 2 * max_son_ind + 1;
    if(max_son_ind < n - 1 && !cmp(a[max_son_ind], a[max_son_ind + 1])){
      max_son_ind++;
    }
    (*swaps)++;
    (*compares) += 2;
  }
}
```

Функция, осуществляющая пирамидальную сортировку и подсчет и вывод соответствующей информации:

```
void Heap_Sort(long long int *a, int n){
  int i, swaps = 0, compares = 0;

  for(i = n / 2 - 1; i >= 0; i--){
    sift(a, i, n, &swaps, &compares);
  }
  for(i = n - 1; i > 0; i--){
    swaps++;
    swap(&a[0], &a[i]);
    sift(a, 0, i, &swaps, &compares);
  }

  printf("%s %d %s\n", "---- HEAP SORT WITH N = ", n, " ----");
  printf("%s %d %c %s %d\n", "Swaps: ", swaps, ' ', "Compares: ",
     compares);
}
```

Функция main:

```
int main(void){
  int i, size = start_size; // #define start_size 10
  long long int *a;

  for(int i = 0; i < 4; i++){
    a = mas_gen(size, curr_seq_type);
    Bubble_Sort(a, size);
    a = mas_gen(size, curr_seq_type);
    Heap_Sort(a, size);
    size *= 10;
    printf("%c", '\n');
  }

  free(a);
  return 0;
}
```

Отладка программы, тестирование функций

Отладка методов сортировки производилась путем предварительного вывода отсортированного массива. В функцию main перед циклом был вставлен следующий код:

```
a = mas_gen(size, curr_seq_type); // #define curr_seq_type 1
for(i = 0; i < size, i++){        // (fully randomized array)
  printf("%lld ", a[i]);
}
Heap_Sort(a, size);               // or Bubble_Sort
for(i = 0; i < size, i++){
  printf("%lld ", a[i]);
}
return 0;
```

В таблице ниже приведены результаты тестирования функций сортировки.

Сортировка	Вывод до сортировки	Вывод после сортировки
HeapSort	-4464 -3582 10436 -10828 -6602 9471 9144 3100 -1927 4960	-1927 3100 -3582 -4464 4960 -6602 9144 9471 10436 -10828
HeapSort	5872 2560 431 -8807 -12442 11260 -2776 334 -6622 -7568 3233 3918 3204 -8047 -13933 13956 -4047 6522 -2331 -9785	334 431 -2331 2560 -2776 3204 3233 3918 -4047 5872 6522 -6622 -7568 -8047 -8807 -9785 11260 -12442 -13933 13956
HeapSort	-16404 8068 -19677 1848 26855 -2849 -11380 22509 -16567 21063 -281 4101 1425 -1838 -14386 -21590 -23813 9897 19738 -12500 27304 20977 20900 27315 2414	-281 1425 -1838 1848 2414 -2849 4101 8068 9897 -11380 -12500 -14386 -16404 -16567 -19677 19738 20900 20977 21063 -21590 22509 -23813 26855 27304 27315
BubbleSort	-17112 -17969 -3052 -21890 5390 -2176 -1380 -21312	-1380 -2176 -3052 5390 -17112 -17969 -21312 -21890
BubbleSort	17 16 -15 -14 13 -12 -11 -10 -9 -8 7 6 5 4 3 2 1	1 2 3 4 5 6 7 -8 -9 -10 -11 -12 13 -14 -15 16 17
BubbleSort	22571 -818 4197 3882 28254 -14886 22872 20256 27336 -24271 16823 20009 9576 11815 -15163 16584 21457 -20312 -4977 -13810 -31146 -18500 -3130 29586 -10543	-818 -3130 3882 4197 -4977 9576 -10543 11815 -13810 -14886 -15163 16584 16823 -18500 20009 20256 -20312 21457 22571 22872 -24271 27336 28254 29586 -31146

Видно, что функции сортировки корректно работают с введенными массивами.

Анализ допущенных ошибок

Ошибки, допущенные в процессе написания программы:

- Ошибки при реализации алгоритмов сортировки;
- Ошибки при работе с памятью (выход за границу массива).

Все ошибки были устранены при отладке программы. Причиной ошибок послужила невнимательность.

Список литературы

[1] Кормен Т., Лейзерсон Ч., Ривест Р, Штайн К. Алгоритмы: построение и анализ. Второе издание. — М.:«Вильямс», 2005.

[2] Вирт Н. Алгоритмы и структуры данных. — М.: Мир, 1989.