Санкт-Петербургский политехнический университет Петра Великого

Институт машиностроения, материалов и транспорта

Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Тема: реализация алгоритмов сортировок Bubble Sort, Heap Sort, Insertion Sort, Merge Sort, Quick Sort на языке C++

Студент гр.3331506/00401 Пузиков А.Н.

Преподаватель Ананьевский М. С.

Санкт-Петербург

2023

Сортировка – это процесс расстановки элементов «в некотором порядке». Элементы размещаются так, чтобы, во-первых, вычисления, требующие определенного порядка расположения данных, могли выполняться эффективно, во-вторых, результаты имели осмысленный вид, и в-третьих, последующие процессы имели бы пригодные исходные данные.

Принцип сортировки

1. Quick sort

1) Выбирается опорный элемент из массива. От выбора опорного элемента не зависит корректность сортировки, но может сильно зависеть его скорость;

2) Сравниваются все элементы массива и переставляются так, чтобы весь массив был разбит на три части: меньшие, чем опорный элемент (“слева”), равные ему и большие (“справа”). Также иногда делят не на три части, а на две: меньшие, чем опорный элемент и большие или равные, так как это упрощает алгоритм разделения;

3) Пока длина отрезка больше единицы, повторяется рекурсивно вызов для каждого из отрезков.

Принцип работы алгоритма показан на рисунке 1.

Средняя сложность алгоритма $O(n \log n)$, а худшая $O(n^2)$.

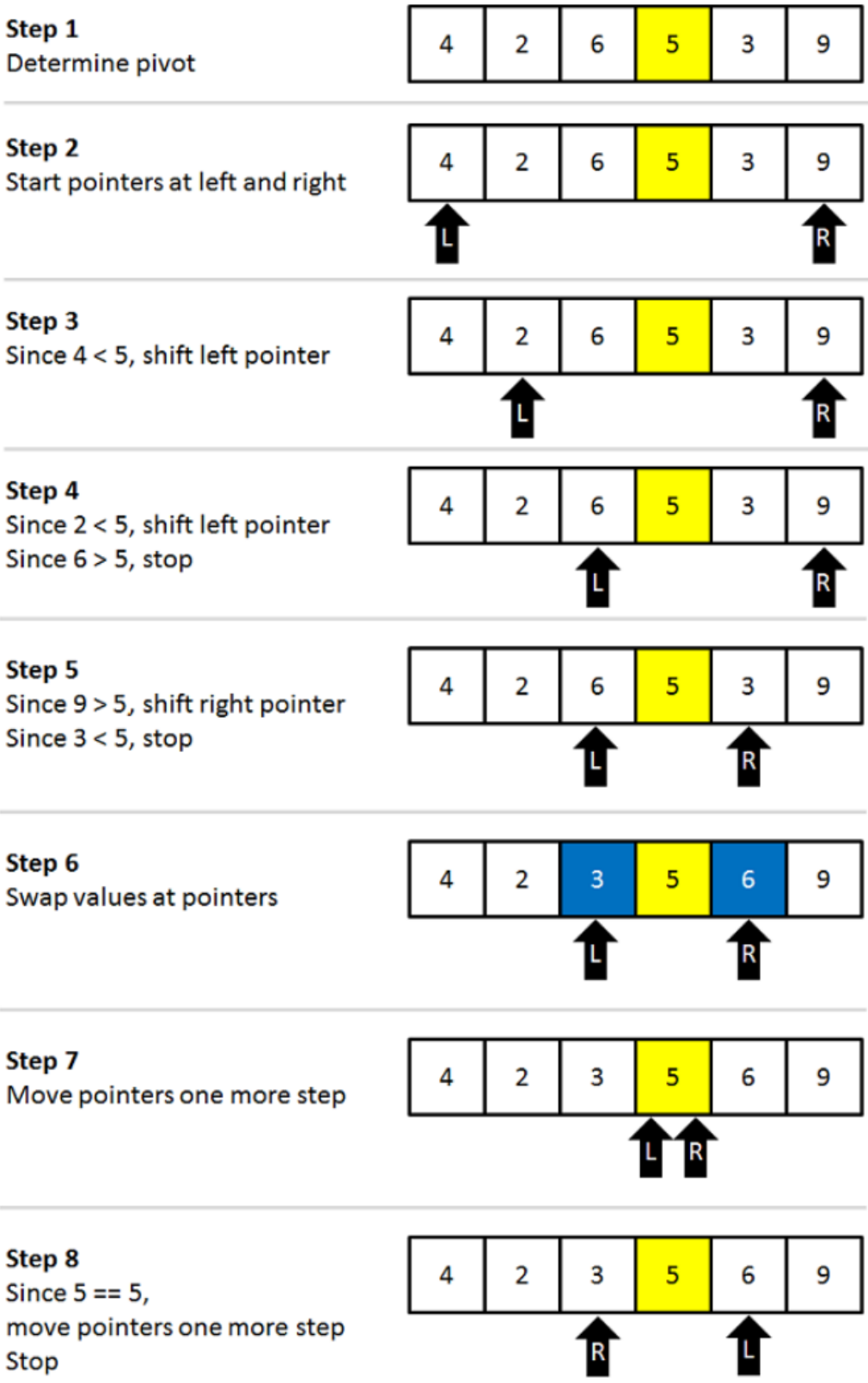

Рисунок 1 - Принцип работы алгоритма Quick sort.

2. Heap sort

Heap sort, также известная как пирамидальная сортировка/сортировка кучей, использует в принципе своей работы бинарно сортирующее дерево, изображенное на рисунке 2. У такого дерева должны быть выполненные некоторые условия:

1) Каждый лист дерева имеет глубину d или $d-1$, где d – максимальная глубина дерева;

2) Значение в любой вершине не меньше (или не больше для другого случая) значений ее потомков.

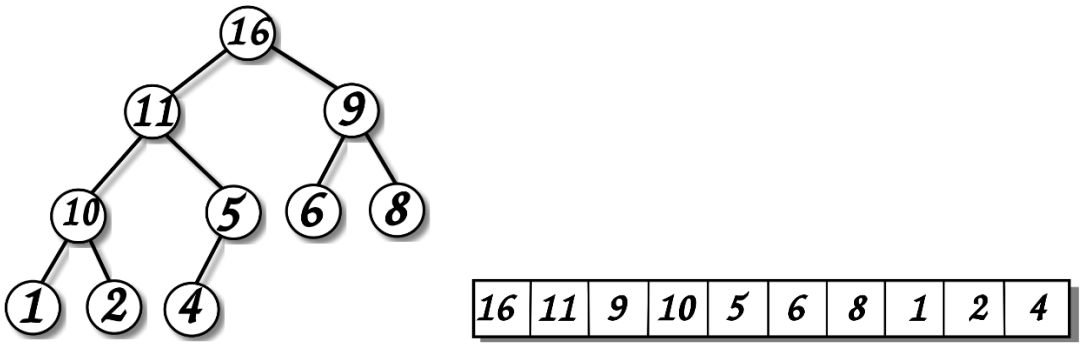

Рисунок 2 - Бинарное сортирующее дерево

После постройки дерева, чтобы получить отсортированный массив, удаляем элементы из корня по одному и перестраиваем дерево, повторяя процесс, пока в дереве не останется всего один элемент.

Средняя сложность алгоритма $O(n\log n)$, она не зависит от входного массива и всегда такова.

3. Insertion sort

Insertion sort, также известная как сортировка вставками, работает на данном принципе:

1) Массив делится на две части, отсортированную (“левая”) и нет (“правая”);

2) Берется один элемент из неотсортированной части массива и вставляется в сортированную, в соответствии с необходимым местоположением;

3) Процесс повторяется, до окончания неотсортированной части массива.

Принцип работы сортировки вставками показан на рисунке 3.

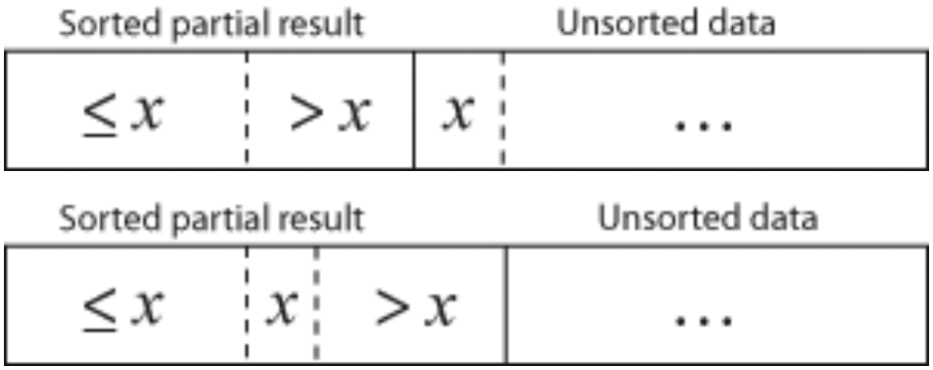

Рисунок 3 - Принцип работы сортировки вставками

Средняя сложность алгоритма $O(n^2)$, такая же, как и худшая.

4. Bubble sort

Bubble sort (пузырьковая сортировка) – метод сортировки массива, путем последовательного сравнения и обмена соседних элементов, если последующий меньше предыдущего. Принцип его работы можно описать так:

1) Проходим массив от первого элемента до последнего, сравнивая их друг с другом, и если предыдущий элемент больше последующего, меняя их;

2) Повторяем этот процесс столько раз, сколько элементов в массиве.

Пример работы алгоритма показан на рисунке 4.

Данный алгоритм плох в реальном мире и используется как демонстрация для обучения.

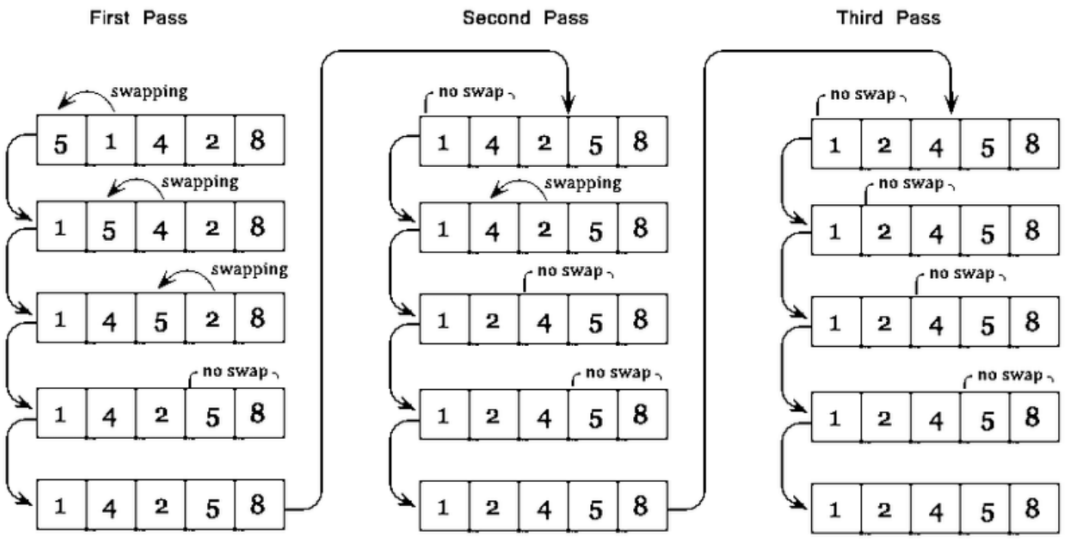

Рисунок 4 - Принцип работы Bubble sort

Средняя сложность алгоритма $O(n^2)$, такая же, как и худшая.

5. Merge sort

Сортировка слиянием также следует стратегии «разделяй и властвуй». Разделяем исходный массив на два равных подмассива. Повторяем сортировку слиянием для этих двух подмассивов и объединяем обратно.

Цикл деления повторяется, пока не останется по одному элементу в массиве. Затем объединяем, пока не образуем полный список.

Алгоритм сортировки состоит из четырех этапов:

1. Найти середину массива.
2. Сортировать массив от начала до середины.
3. Сортировать массив от середины до конца.
4. Объединить массив.

Принцип работы алгоритма представлен на рисунке 5.

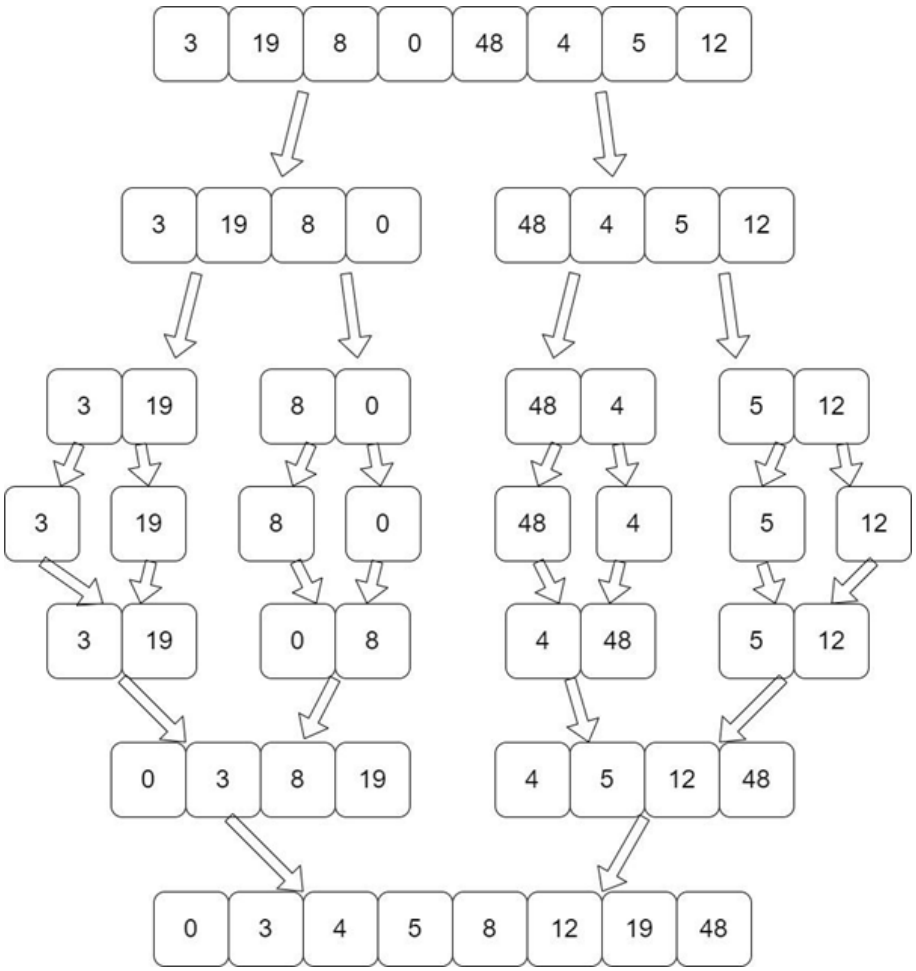

Рисунок 5 - Принцип работы Merge sort

Сложность в любом случае: *O(n*logn).*

Заключение

В работе были разработаны алгоритмы сортировки Bubble Sort, Heap Sort, Insertion Sort, Merge Sort, Quick Sort и рассмотрены их особенности. Код представлен в приложении №1.

Среди алгоритмов, применяемых для обработки небольших по размеру массивов, предпочтение следует отдать методу сортировки вставками, а при работе с большими массивами – алгоритму быстрой сортировки.

Список литературы

1. https://ru.wikipedia.org/wiki/Сортировка_пузырьком
2. https://ru.wikipedia.org/wiki/Пирамидальная_сортировка
3. https://ru.wikipedia.org/wiki/Сортировка_вставками
4. https://ru.wikipedia.org/wiki/Сортировка_слиянием
5. https://ru.wikipedia.org/wiki/Быстрая_сортировка
6. «Сортировка и системы сортировки»/ Г. Лорин: пер. с англ. Р. Л. Смелянского М.: Наука. Главная редакция физико-математической литературы, 1983 384 с
7. «Алгоритмы сортировки. Анализ, реализация, применение: учебное пособие» / Д.В. Шагбазян, А.А. Штанюк, Е.В. Малкина. – Нижний Новгород: Нижнегородский университет, 2019.

Приложение №1

Quick sort

```
#include <iostream>
#include <vector>
#define vl std::vector<int>
using std::cout;

int partition(vl &vector, int start, int pivot)
{
  int I = start;
  while(I < pivot)
  {
    if(vector[i] > vector[pivot] && I == pivot-1)
    {
      std::swap(vector[i], vector[pivot]);
      pivot--;
    }

    else if(vector[i] > vector[pivot])
    {
      std::swap(vector[pivot - 1], vector[pivot]);
      std::swap(vector[i], vector[pivot]);
      pivot--;
    }

    else i++;
  }
  return pivot;
}

void 9uicksort(vl &vector, int start, int end)
{
  if(start < end)
  {
    int pivot = partition(vector, start, end);

    9uicksort(vector, start, pivot - 1);
    9uicksort(vector, pivot + 1, end);
  }
}

int main()
{
  vl vector = {5, 2, 12, 7, 4, 24, 8, 15, 32, 6};

  9uicksort(vector, 0, vector.size());

  cout<<"Sorted array\n";
  for (int I : vector)
  {
```

```
    cout<<i<<" ";
  }

  return 0;
}
```

Heap sort

```
#include <iostream>
#include <vector>

using namespace std;

// Процедура для преобразования в двоичную кучу поддерева с
корневым узлом i, что является
// индексом в vector1[]. n - размер кучи

void heapify(std::vector<int> &vector, int n, int i)
{
  int largest = i;
// Инициализируем наибольший элемент как корень
  int l = 2*i + 1; // левый = 2*i + 1
  int r = 2*i + 2; // правый = 2*i + 2

  // Если левый дочерний элемент больше корня
  if (l < n && vector[l] > vector[largest])
    largest = l;

  // Если правый дочерний элемент больше, чем самый большой
элемент на данный момент
  if (r < n && vector[r] > vector[largest])
    largest = r;

  // Если самый большой элемент не корень
  if (largest != i)
  {
    swap(vector[i], vector[largest]);

// Рекурсивно преобразуем в двоичную кучу затронутое поддерево
    heapify(vector, n, largest);
  }
}

// Основная функция, выполняющая пирамидальную сортировку
void heapSort(std::vector<int> &vector, int n)
{
  // Построение кучи (перегруппируем массив)
  for (int i = n / 2 - 1; i >= 0; i--)
    heapify(vector, n, i);

  // Один за другим извлекаем элементы из кучи
```

```
  for (int i=n-1; i>=0; i--)
  {
    // Перемещаем текущий корень в конец
    swap(vector[0], vector[i]);

    // вызываем процедуру heapify на уменьшенной куче
    heapify(vector, i, 0);
  }
}

/* Вспомогательная функция для вывода на экран массива размера
n*/
void printArray(std::vector<int>& vector, int n)
{
  for (int i=0; i<n; ++i)
    cout << vector[i] << " ";
  cout << "\n";
}

// Управляющая программа
int main()
{
  std::vector<int> vector1 = {5, 2, 12, 7, 4, 24, 8, 15, 32, 6};
  int n = vector1.size();

  heapSort(vector1, n);

  cout << "Sorted array is \n";
  printArray(vector1, n);
}
```

Insertion sort

```
1. #include <iostream>
   #include <vector>

   using namespace std;


   int main()
   {
     std::vector<int> vector = {5, 2, 12, 7, 4, 24, 8, 15, 32,
   6};
     for (int i = 1; i < vector.size(); i++)
       for (int j = i; j > 0 && vector[j - 1] > vector[j]; j--)//
   пока j>0 и элемент j-1 > j, x-массив int
       {
         std::swap(vector[j - 1], vector[j]);         // меняем
   местами элементы j и j-1
       }
     cout << "Sorted array is \n";
```

```
    for (int i : vector)
      cout << i << " ";
    cout << "\n";
  }
```

Bubble sort

```
#include <iostream>;
#include <vector>;

int main()
{

  std::vector<int> vector{5, 2, 12, 7, 4, 24, 8, 15, 32, 6};

  for (int i = 0; i < 10; i++)
  {
    for (int j = 0; j < 9; j++)
    {
      if (vector[j] > vector[j + 1])
      {
        int b = vector[j]; // создали дополнительную переменную
        vector[j] = vector[j + 1]; // меняем местами
        vector[j + 1] = b; // значения элементов
      }
    }
  }

  for (int i = 0; i < 10; i++)
  {
    std::cout << vector[i] << " "; // выводим элементы массива
  }

  return 0;
}
```

Merge sort

```
#include <iostream>
#include <vector>

#define vl std::vector<int>
using std::cout;

void merge(vl &vector, int start, int end, int mid);

void mergeSort(vl &vector, int start, int end)
{
  int mid;
```

```
  if (start < end)
  {

    mid = (start + end) / 2;
    mergeSort(vector, start, mid);
    mergeSort(vector, mid + 1, end);
    merge(vector, start, end, mid);
  }
}

void merge(vl &vector, int start, int end, int mid)
{
  vl mergedList;
  mergedList.resize(vector.size());
  int i, j, k;
  i = start;
  k = start;
  j = mid + 1;

  while (i <= mid && j <= end)
  {
    if (vector[i] < vector[j])
    {
      mergedList[k] = vector[i];
      k++;
      i++;
    } else
    {
      mergedList[k] = vector[j];
      k++;
      j++;
    }
  }

  while (i <= mid)
  {
    mergedList[k] = vector[i];
    k++;
    i++;
  }

  while (j <= end)
  {
    mergedList[k] = vector[j];
    k++;
    j++;
  }

  for (i = start; i < k; i++)
  {
    vector[i] = mergedList[i];
  }
}
```

```
int main()
{
  vl vector = {5, 2, 12, 7, 4, 24, 8, 15, 32, 6};
  mergeSort(vector, 0, vector.size() - 1);
  cout << "Sorted array\n";
  for (int i: vector)
  {
    cout << i << " ";
  }
}
```