Государственный Университет Молдовы

Факультет Математики и Информатики

Департамент Информатики

Лабораторная работа №2

“Алгоритмы и структуры данных”

Тема:”Алгоритмы сортировки”

Проверил: Угнуряну Валерий

Выполнил: Чобану Артём

Группа: i1902

Кишинев 2021

**Задание:**  
Реализовать следующие алгоритмы сортировки:

1. Пузырьковая сортировка
2. Сортировка вставками
3. Сортировка выбором
4. Быстрая сортировка
5. Сортировка Шелла
6. Пирамидальная
7. Сортировка слиянием

https://github.com/ArtiomCiobanu/Algorithms\_Labs

**Реализация алгоритмов (сортировка по цене):**

1. Пузырьковая сортировка (bubble sort)

for (int f = 0; f < aircrafts.Length - 1; f++)  
{  
 for (int i = 0; i < aircrafts.Length - 1; i++)  
 {  
 if (aircrafts[i].Price > aircrafts[i + 1].Price)  
 {  
 var swapper = aircrafts[i];  
 aircrafts[i] = aircrafts[i + 1];  
 aircrafts[i + 1] = swapper;  
 }  
 }  
}

**Описание алгоритма**: все элементы сравниваются попарно и меняются местами если их порядок неверный.

Сложность алгоритма составляет O(n2) т.к. нужно провести n2 итераций.

1. Сортировка вставками (insertion)

for (int i = 1; i < aircrafts.Length - 1; i++)  
{  
 for (int j = i + 1; j > 0; j--)  
 {  
 if (aircrafts[j].Price < aircrafts[j - 1].Price)  
 {  
 var c = aircrafts[j];  
 aircrafts[j] = aircrafts[j - 1];  
 aircrafts[j - 1] = c;  
 }  
 }  
}

**Описание алгоритма**: каждый элемент просматривается и помещается на подходящее место среди уже отсортированных элементов. Просмотр начинается со второго элемента, так как первый элемент сравнивать не с чем.

Сложность алгоритма составляет O(n2) т.к. нужно провести n2 итераций.

1. Сортировка выбором (selection)

for (int i = 0; i < aircrafts.Length; i++)  
{  
 var minIndex = i;  
 for (int j = i + 1; j < aircrafts.Length; j++)  
 {  
 if (aircrafts[j].Price < aircrafts[i].Price)  
 {  
 minIndex = j;  
 }  
 }  
  
 var c = aircrafts[minIndex];  
 aircrafts[minIndex] = aircrafts[i];  
 aircrafts[i] = c;  
}

**Описание алгоритма:**

1. Находится минимальный элемент в текущем неотсортированном массиве.
2. Найденное значение обменивается со значением на первой неотсортированной позиции
3. Алгоритм повторяется вплоть до последнего элемента

Сложность алгоритма составляет O(n2) т.к. нужно провести n2 итераций.

1. Быстрая сортировка (quicksort)

int lastIndex = aircrafts.Length;  
var areas = new List<Tuple<int, int>>  
{  
 new(0, lastIndex)  
};  
while (areas.Any())  
{  
 var (left, right) = areas.First();  
 var pivotIndex = (right + left) / 2;  
 var pivot = aircrafts[pivotIndex];  
 for (int i = pivotIndex - 1; i >= left; i--)  
 {  
 if (aircrafts[i].Price > pivot.Price)  
 {  
 var current = aircrafts[i];  
 //Сдвиг всего влево  
 for (int j = i; j < right - 1; j++)  
 {  
 aircrafts[j] = aircrafts[j + 1];  
 }  
 aircrafts[right - 1] = current;  
 pivotIndex--;  
 }  
 }  
 for (int i = pivotIndex + 1; i < right; i++)  
 {  
 if (aircrafts[i].Price < pivot.Price)  
 {  
 var current = aircrafts[i];  
  
 for (int j = i; j >= left + 1; j--)  
 {  
 aircrafts[j] = aircrafts[j - 1];  
 }  
 aircrafts[left] = current;  
 pivotIndex++;  
 }  
 }  
 areas.RemoveAt(0);  
 if (right - pivotIndex > 1)  
 {  
 var area2 = new Tuple<int, int>(pivotIndex, right);  
 areas.Insert(0, area2);  
 }  
 if (pivotIndex - left > 1)  
 {  
 var area1 = new Tuple<int, int>(left, pivotIndex);  
 areas.Insert(0, area1);  
 }  
}

**Описание алгоритма:**

1. Выбирается опорный элемент
2. Все меньшие элементы перебрасываются в левую часть
3. Все большие – в правую часть (если необходимо отсортировать по убыванию, то наоборот)

Алгоритм повторяется для каждой области, на которую разделился массив благодаря опорному элементу, и до тех пор, пока размер области не достигнет одного элемента.

Средняя сложность алгоритма составляет О(nlogn).

Алгоритм очень нестабилен, и деградирует до O(n2) при неудачном выборе опорного элемента (максимум или минимум).

1. Сортировка Шелла (Shell sort)

for (int gap = aircrafts.Length / 2; gap > 0; gap--)  
{  
 for (int i = 0; i < aircrafts.Length - gap; i++)  
 {  
 var current = aircrafts[i];  
 var other = aircrafts[i + gap];  
  
 if (current.Price > other.Price)  
 {  
 aircrafts[i + gap] = current;  
 aircrafts[i] = other;  
 }  
 }  
}

Описание алгоритма:

1. Выбирается интервал, который изначально в 2 раза меньше длины
2. Элементы на растоянии этого интервала сравниваются и меняются если порядок неверный
3. Интервал уменьшается на 1

Алгоритм повторяется до того, как интервал не будет иметь длину 1.

Средняя сложность: O(n5/4)

Алгоритм очень нестабилен.

1. Пирамидальная сортировка (heapsort)

FixHeap(aircrafts, aircrafts.Length);  
  
for (int i = aircrafts.Length - 1; i > 0; i--)  
{  
 var c = aircrafts[0];  
 aircrafts[0] = aircrafts[i];  
 aircrafts[i] = c;  
  
 FixHeap(aircrafts, i - 1);  
}

Метод, престраивающий “кучу”:

public static void FixHeap(Aircraft[] aircrafts,

int lastHeapFixingIndex)  
{  
 int middleIndex = lastHeapFixingIndex / 2;  
 for (int i = middleIndex - 1; i >= 0; i--)  
 {  
 int firstIndex = 2 \* i + 1;  
 int secondIndex = 2 \* i + 2;  
 if (firstIndex < aircrafts.Length &&

aircrafts[i].Price < aircrafts[firstIndex].Price)  
 {  
 var c = aircrafts[i];  
 aircrafts[i] = aircrafts[firstIndex];  
 aircrafts[firstIndex] = c;  
 }  
 if (secondIndex < aircrafts.Length &&

aircrafts[i].Price < aircrafts[secondIndex].Price)  
 {  
 var c = aircrafts[i];  
 aircrafts[i] = aircrafts[secondIndex];  
 aircrafts[secondIndex] = c;  
 }  
 }  
}

Описание алгоритма:

1. Элементы выстраиваются в сортирующее дерево, в котором каждый элемент не меньше элементов на позициях 2i+1 и 2i+2, где i – индекс текущего элемента.
2. Элемент в корне переносится в начало отсортированного массива.

Сложность: O(nlogn) в худшем случае

1. Сортировка слиянием (mergesort)

int g = 1;  
while (g < aircrafts.Length)  
{  
 g \*= 2;  
  
 if (g > aircrafts.Length)  
 {  
 g = aircrafts.Length;  
 }  
  
 for (int i = 0; i <= aircrafts.Length - g; i += g)  
 {  
 int difference = g / 2;  
  
 var sorted = new List<Aircraft>();  
 for (int j = i; j < i + difference; j++)  
 {  
 var first = aircrafts[j];  
 var second = aircrafts[j + difference];  
  
 sorted.InsertBeforeHighest(first);  
 sorted.InsertBeforeHighest(second);  
 }  
  
 for (int j = 0; j < sorted.Count; j++)  
 {  
 aircrafts[i + j] = sorted[j];  
 }  
 }  
}

**Описание алгоритма:**

1. Массив делится пополам
2. Каждая половина сортируется алгоритмом Шелла. Один элемент считается отсортированным.

Алгоритм очень стабилен

Сложность: O(nlogn)