Карманная сортировка — алгоритм  [сортировки](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8) , основанный на предположении о равномерном распределении входных данных.

Для карманной сортировки нужно разбить элементы массива входных данных на  ~~n   блоков (карманов, корзин). Далее каждый из таких блоков сортируется либо другой сортировкой, либо рекурсивно тем же методом разбиения. После сортировок внутри каждых блоков данные записываются в массив в порядке разбиения на блоки. При этом нужно учитывать, что данная сортировка работает только в том случае, если разбиение на блоки производится таким образом, чтобы элементы каждого следующего блока были больше предыдущего.~~

Оценим сложность алгоритма блочной сортировки для случая, при котором в качестве алгоритма сортировки блоков (next-sort из псевдокода) используется сортировка вставками.

Для оценки сложности алгоритма введём случайную величину ni, обозначающую количество элементов, которые попадут в карман B[i]. Время работы сортировки вставками равно  O(n^2), тогда время карманной сортировки будет равна O(n).

Преимущества: относится к классу быстрых алгоритмов с линейным временем исполнения O(N) (на удачных входных данных).

Недостатки: сильно деградирует при большом количестве мало отличных элементов, или же на неудачной функции получения номера корзины по содержимому элемента.

static void BucketSort(int[] a)

{

// Примем, что количество корзин равно количеству элементов в массиве-источнике.

// Тогда:

// массив корзин

List<int>[] aux = new List<int>[a.Length];

// каждую корзину проинициализировать

for (int i = 0; i < aux.Length; ++i)

aux[i] = new List<int>();

// найти диапазон значений в массиве-источнике

int minValue = a[0];

int maxValue = a[0];

for (int i = 1; i < a.Length; ++i)

{

if (a[i] < minValue)

minValue = a[i];

else if (a[i] > maxValue)

maxValue = a[i];

}

// эта величина будет использоваться a.Length раз, поэтому имеет смысл её сохранить.

double numRange = maxValue - minValue;

for (int i = 0; i < a.Length; ++i)

{

// вычисление индекса корзины

int bcktIdx = (int)Math.Floor((a[i] - minValue) / numRange \* (aux.Length - 1));

// добавление элемента в соответствующую корзину

aux[bcktIdx].Add(a[i]);

}

// сортировка корзин. Здесь я, для упрощения себе писанины, использую библиотечную сортировку

for (int i = 0; i < aux.Length; ++i)

aux[i].Sort();

// собираем отсортированные элементы обратно в массив-источник

int idx = 0;

for (int i = 0; i < aux.Length; ++i)

{

for (int j = 0; j < aux[i].Count; ++j)

a[idx++] = aux[i][j];

}

}