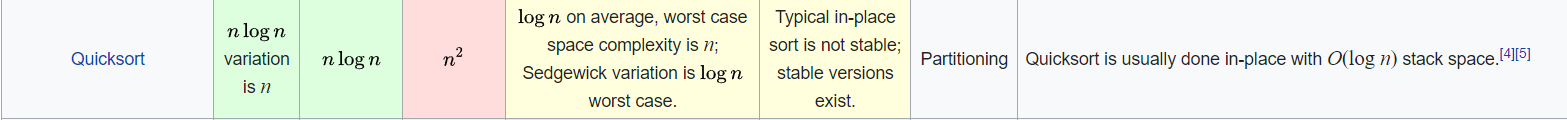
Quick Sort



using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Quick\_Sort

{

class Program

{

private static void Quick\_Sort(int[] arr, int left, int right)

{

if (left < right)

{

int pivot = Partition(arr, left, right);

if (pivot > 1) {

Quick\_Sort(arr, left, pivot - 1);

}

if (pivot + 1 < right) {

Quick\_Sort(arr, pivot + 1, right);

}

}

}

private static int Partition(int[] arr, int left, int right)

{

int pivot = arr[left];

while (true)

{

while (arr[left] < pivot)

{

left++;

}

while (arr[right] > pivot)

{

right--;

}

if (left < right)

{

if (arr[left] == arr[right]) return right;

int temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

}

else

{

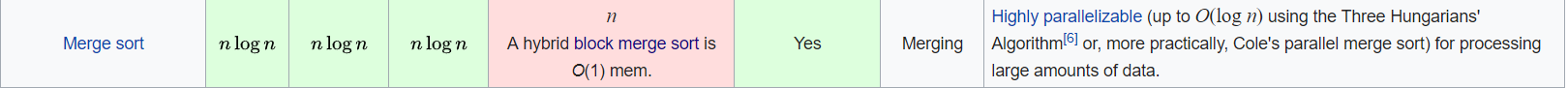
return right;

}

}

}

Merge Sort



private void Merge(int[] input, int left, int middle, int right)

{

    int[] leftArray = new int[middle - left + 1];

    int[] rightArray = new int[right - middle];

    Array.Copy(input, left, leftArray, 0, middle - left + 1);

    Array.Copy(input, middle + 1, rightArray, 0, right - middle);

    int i = 0;

    int j = 0;

    for (int k = left; k < right + 1; k++)

    {

        if (i == leftArray.Length)

        {

            input[k] = rightArray[j];

            j++;

        }

        else if (j == rightArray.Length)

        {

            input[k] = leftArray[i];

            i++;

        }

        else if (leftArray[i] <= rightArray[j])

        {

            input[k] = leftArray[i];

            i++;

        }

        else

        {

            input[k] = rightArray[j];

            j++;

        }

    }

}

private void MergeSort(int[] input, int left, int right)

{

    if (left < right)

    {

        int middle = (left + right) / 2;

        MergeSort(input, left, middle);

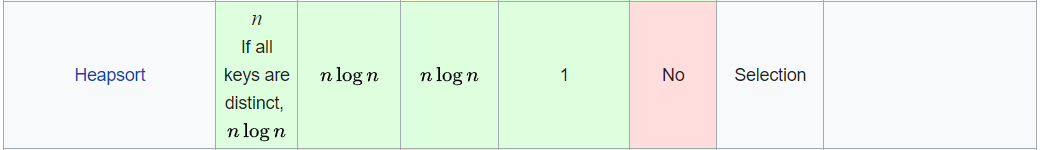
        MergeSort(input, middle + 1, right);

        Merge(input, left, middle, right);

    }

}

Heap Sort



using System;

namespace HeapSortDemo {

public class example {

static void heapSort(int[] arr, int n) {

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (int i = n-1; i>=0; i--) {

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

}

static void heapify(int[] arr, int n, int i) {

int largest = i;

int left = 2\*i + 1;

int right = 2\*i + 2;

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

if (largest != i) {

int swap = arr[i];

arr[i] = arr[largest];

arr[largest] = swap;

heapify(arr, n, largest);

}

}

public static void Main() {

int[] arr = {55, 25, 89, 34, 12, 19, 78, 95, 1, 100};

int n = 10, i;

Console.WriteLine("Heap Sort");

Console.Write("Initial array is: ");

for (i = 0; i < n; i++) {

Console.Write(arr[i] + " ");

}

heapSort(arr, 10);

Console.Write("\nSorted Array is: ");

for (i = 0; i < n; i++) {

Console.Write(arr[i] + " ");

}

}

}

}

Insertion Sort



public int[] InsertionSort(int[] inputArray)

{

for (int i = 0; i < inputArray.Length - 1; i++)

{

for (int j = i + 1; j > 0; j--)

{

if (inputArray[j - 1] > inputArray[j])

{

int temp = inputArray[j - 1];

inputArray[j - 1] = inputArray[j];

inputArray[j] = temp;

}

}

}

return inputArray;

}

Bubble Sort



Public int[] BubbleSort(int[]a)

{

for (int p = 0; p <= a.Length - 2; p++)

{

for (int i = 0; i <= a.Length - 2; i++)

{

if (a[i] > a[i + 1])

{

t = a[i + 1];

a[i + 1] = a[i];

a[i] = t;

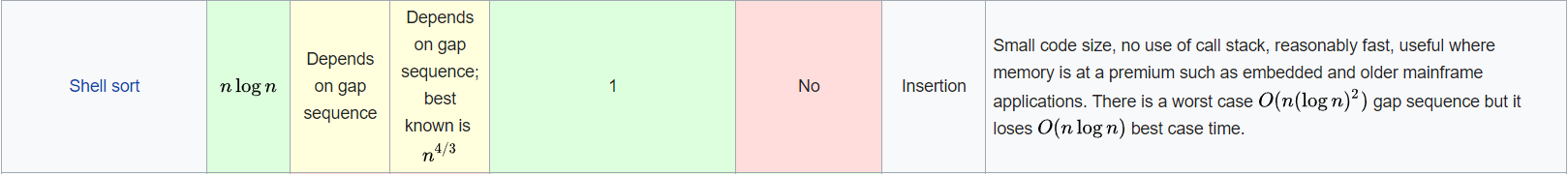
}

}

}

}

Shell Sort



public void ShellSort (int[] data, int[] intervals)

{

int i, j, k, m;

int N = data.Length;

// The intervals for the shell sort must be sorted, ascending

for (k=intervals.Length-1; k>=0; k--) {

int interval = intervals [k];

for (m=0; m<interval; m++) {

for (j=m+interval; j<N; j+=interval) {

for (i=j; i>=interval && data[i]<data[i-interval]; i-=interval) {

exchange (data, i, i - interval);

}

}

}

}

}