**4 kyu**

**Simple Fun #159: Middle Permutation**

1893993% of 293119of 1,178[myjinxin2015](https://www.codewars.com/users/myjinxin2015)

C#

* [TRAIN AGAIN](https://www.codewars.com/kata/simple-fun-number-159-middle-permutation/train/csharp)
* [NEXT KATA](https://www.codewars.com/trainer/csharp)

Details

[Solutions](https://www.codewars.com/kata/simple-fun-number-159-middle-permutation/solutions/csharp)

[Forks (2)](https://www.codewars.com/kata/simple-fun-number-159-middle-permutation/forks/csharp)

[Discourse (123)](https://www.codewars.com/kata/simple-fun-number-159-middle-permutation/discuss/csharp)

* Add to Collection
* |
* Share this kata:

**Task**

You are given a string s. Every letter in s appears once.

Consider all strings formed by rearranging the letters in s. After ordering these strings in dictionary order, return the middle term. (If the sequence has a even length n, define its middle term to be the (n/2)th term.)

**Example**

For s = "abc", the result should be "bac".

The permutations in order are:

"abc", "acb", "bac", "bca", "cab", "cba"

So, The middle term is "bac".

**Input/Output**

* [input] string s

unique letters (2 <= length <= 26)

* [output] a string

middle permutation.

<https://www.codewars.com/kata/simple-fun-number-159-middle-permutation/csharp>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Program

{

//public static bool NextPermutation(char[] array)

//{

// // Find non-increasing suffix

// int i = array.Length - 1;

// while (i > 0 && array[i - 1] >= array[i])

// i--;

// if (i <= 0)

// return false;

// // Find successor to pivot

// int j = array.Length - 1;

// while (array[j] <= array[i - 1])

// j--;

// char temp = array[i - 1];

// array[i - 1] = array[j];

// array[j] = temp;

// j = array.Length - 1;

// while (i < j)

// {

// temp = array[i];

// array[i] = array[j];

// array[j] = temp;

// i++;

// j--;

// }

// return true;

//}

//static int factorial(int n)

//{

// int prod = 1;

// for (int i = 2; i <= n; i++)

// {

// prod \*= i;

// }

// return prod;

//}

//public static string MiddlePermutation(string s)

//{

// if (s.Length <= 1) return s;

// int total = factorial(s.Length);

// List<string> perms = new List<string>();

// //coding and coding..

// char[] ch = s.ToCharArray();

// Array.Sort(ch);

// int cont = 1;

// while (NextPermutation(ch))

// {

// if (cont >= total / 2)

// {

// break;

// }

// cont++;

// perms.Add(new string(ch));

// }

// //for (int i = 0; i < perms.Count; i++)

// //{

// // Console.WriteLine(i + " " + perms[i]);

// //}

// //Console.WriteLine();

// return perms.Last();

//}

public static string MiddlePermutationEficiente(string s)

{

if (s.Length <= 1) return s;

char[] ch = s.ToCharArray();

Array.Sort(ch);

Array.Reverse(ch);

string rev = new string(ch);

string copia = rev.ToString();

string resto = "";

if (rev.Length % 2 == 0)

{

rev = rev.Remove(rev.Length / 2, 1);

resto= copia.Substring(copia.Length / 2, 1);

}

else

{

rev = rev.Remove(rev.Length / 2, 2);

resto = copia.Substring(copia.Length / 2, 2);

}

//Console.WriteLine(new string(ch));

return resto + rev;

}

static void Main(string[] args)

{

//string s = "abcdxgz";

string s = "123456";

string[] arr =

{ "11",

"123",

"1234",

"12345",

"123456",

"1234567",

"12345678",

"123456789",

"0123456789"

};

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

{

Console.WriteLine(MiddlePermutation(arr[i]) + " " + MiddlePermutationEficiente(arr[i]));

}

Console.ReadLine();

}

}

}