Exercise Session n. 4 (13 March 2024)

Algorithms and Data Structures

Clone the following repository: **https://github.com/jindosanda/algo_tests** or run `git pull` if you already clone it, and place your programs inside the exercises folder under the appropriate session folder (session_4/exercises/session4.py)
Read the README files to run the tests!

Let's practice with some divide-and-conquer algorithms.

Finding the Peak *in Less-than-Linear Time* (session4.py)

Write a function `find_peak(A)` that, given a "peak" sequence A, finds the maximal value in A in (n) time—meaning, strictly better than linear time. A "peak" sequence consists of an increasing sequence A_i attached to a decreasing sequence A_{in}, for some index i. These two sequences share exactly one element, namely A_i.

Examples

```
>>> find_peak([1,2,3])
  3
  >>> find_peak([1,2,7,8,9])
  9
  >>> find_peak([9,6,5])
  9
  >>> find_peak([1,2,1,-3])
  2
  >>> find_peak([1,2,1,0,-3])
  2
  >>> find_peak([1,2,3,2,1])
  3
  >>> find_peak([1,2,3,4])
  4
  >>> find_peak([1,2])
  2
  >>> find_peak([4,7,4])
  7
```

```
>>> find_peak([7])
7
>>> find_peak([7,4])
7
>>> find_peak([4,7])
7
```

Describe and Improve… (session4.py)

Consider the following function `algo_x(n)` that takes a non-negative integer n. Explain what `algo_x(n)` does. Do not just paraphrase the code. Explain at a high-level *what* the function does, not *how* it does it. Also, analyze the complexity of `algo_x(n)` as a function of n (worst case).

```
def algo_x(n):
        assert n >= 0
        m = 0
        while m*m <= n:
            m = m + 1
        return m - 1
```

Write a function `better_algo_x(n)` that is functionally identical to `algo_x(n)` with a strictly better time complexity. `better_algo_x(n)` may only use the basic arithmetic operations of addition, subtraction, multiplication and division (integer). In particular, `better_algo_x(n)` may not use any other function, not even from the standard libraries of Python.

Needs more practice? Try to solve the following exercises:

Minimum and Maximum (session4.py)

Write a function `min(A)` that, given an array of numbers, A, returns element a_i such that this element has the smallest value of all elements in the array. Correspondingly, write a function `max(A)` that finds the maximum element of an array.

Examples

```
>>> min([1,2,3])
1
>>> min([1, 10, 20, 30, -1, 40, 50])
-1
>>> max([1, 2, 3])
3
>>> max([1, 10, 20, 30, -1, 40, 50])
50
```

Palindromic String (session4.py)

Write a function `palindrome(s)` which takes a string `s` as input and returns `True` if the string is a palindrome, otherwise `False`. A palindrome is a word, phrase, number or any sequence of characters which yields the same sequence when read in a reversed manner. Examples of palindromic words: `racecar`, `level`, `rotator`.

Examples

```
>>> palindrome("abba")
True
>>> palindrome("ciao!")
False
```

Longest Palindromic Substring (session4.py)

Write a function `lps(s)` that given a string `s` returns the longest substring in `s` which is a palindrome.

Examples

```
>>> lps("babad")
"bab" or ("aba")
>>> lps("cbbd")
"bb"
>>> lps("racecarlevel")
"racecar"
```

Maximal Difference (session4.py)

Write a function `md(A)` that returns the maximal difference between any two elements of the given sequence `A`.
*Hint:*Try to find a way to use previously defined functions.

Examples

```
>>> md([2, 1, 5, 9, 4, 10, 8])
9
>>> md([1])
0
>>> md([1, 1, 1])
0
>>> md([10, -3, 4, 11, 0, 9])
14
```

Partition Even-Odd (session4.py)

Write a function `partition_even_odd(A)` that takes an array `A` of integers and sorts the elements of `A` so that all the even elements precede all the odd elements. The function must sort `A` in-place. This means that it must operate directly on `A` just by swapping elements, without creating an additional array.

Examples

```
>>> A = [-1,1,7,5,-2,1,2,7,7,5,5,1,1,4,1]
>>> partition_even_odd(A)
>>> print(A)
[2, 4, -2, 1, 7, 5, 1, 7, 7, 5, 5, 1, 1, 1, -1]
```

If your solution is not in `O(n)` complexity, can you find a way to achieve this?

Prime Factors (session4.py)

Write a function `prime_factorize(n)` that takes a positive integer `n`, and returns a string with its prime factorization using the exponent (E) notation. The symbol "E" is not displayed if it equals to 1.

Examples

```
>>> prime_factorize(2312)
2E3 17E2
>>> prime_factorize(10242311)
19 701 769
>>> prime_factorize(1)
""
```