lab2

October 9, 2023

1 ECS529U Algorithms and Data Structures

2 Lab sheet 2 (deadline: week 3)

This second lab gets you to work array algorithms. For each of the questions, write down and run at least 3 tests, trying to capture corner cases (e.g. empty arrays). In the `lab2-tests.ipynb` file, we have already provided you with tests for the first two Questions.

Marks (max 5): Questions 1-3: 1 each | Questions 4-7: 0.5 each

2.1 Question 1

Write a Python function:

`def squareArray(A)`

which takes an array of integers as input and squares every integer in the array.

For example, if it takes the array `[5,12,31,7,25]` as input, it will change the array to `[25, 144, 961, 49, 625]`.

[ ]:
```
# some tests, but you can add your own ones as well
def squareArray(array):
    for i in range(len(array)):
        array[i] = array[i]**2

tests = (([5,12,31,7,25]),
         ([-5,12,-31,7,25]),
         ([-5,12,0,7,25]),
         ([0]),
         ([]))

# should print: [25, 144, 961, 49, 625],[25, 144, 961, 49, 625],[25, 144, 0,␣
 ↪49, 625],[0],[]
for A in tests:
    squareArray(A)
    print(A)
```

```
[25, 144, 961, 49, 625]
[25, 144, 961, 49, 625]
[25, 144, 0, 49, 625]
```

```
[0]
[]
```

2.2 Question 2

Write a Python function:

`def squareArray2(A)`

which performs the same task as Question 1, but does it by creating and returning a new array with the squared values rather than changing the array passed to it.

For example, if it takes the array `[5,12,31,7,25]` as input, it will return the array `[25, 144, 961, 49, 625]` but the initial array will remain `[5,12,31,7,25]`.

[ ]:
```
# some tests, but you can add your own ones as well
tests = (([5,12,31,7,25]),
         ([-5,12,-31,7,25]),
         ([-5,12,0,7,25]),
         ([0]),
         ([]))

def squareArray2(array):
    new_array = [value**2 for value in array]
    return new_array

# should print: [25, 144, 961, 49, 625] [5, 12, 31, 7, 25]
# [25, 144, 961, 49, 625] [-5, 12, -31, 7, 25]
# [25, 144, 0, 49, 625] [-5, 12, 0, 7, 25]
# [0] [0]
# [] []
for A in tests:
    print(squareArray2(A),A)
```

```
[25, 144, 961, 49, 625] [5, 12, 31, 7, 25]
[25, 144, 961, 49, 625] [-5, 12, -31, 7, 25]
[25, 144, 0, 49, 625] [-5, 12, 0, 7, 25]
[0] [0]
[] []
```

2.3 Question 3

Write a Python function:

`def minWithPos(A)`

which takes an array of integers and returns the smallest integer in the array along with its position, as a pair. For example, if it takes the array `[10,5,12,31,7,25]` as input it will return the pair `(5,1)`. If `A` is empty, the function should return `None` (which corresponds to the null value in Python).

Note: The specification does not mention which position to return in case the biggest element occurs in more than one positions, so the function can return any of those positions.

[ ]:
```
# some tests, but you can add your own ones as well
# should print: (5, 1),(-31, 2),(-25, 4),(12, 0),None,(-5, 0)

def minWithPos(array):
    if not array:
        return None

    min_val = float('inf')
    min_pos = None

    for index, val in enumerate(array):
        if val < min_val:
            min_val, min_pos = val, index

    return min_val, min_pos

tests = ([10,5,12,31,7,25],[-5,12,-31,7,25],[-5,-12,31,-7,-25],[12,13],[],[-5])

for A in tests:
    print(minWithPos(A))
```

```
(5, 1)
(-31, 2)
(-25, 4)
(12, 0)
None
(-5, 0)
```

2.4 Question 4

Write a function

`def minWithPoss(A)`

which takes an array of integers and a pair containing the smallest integer in the array and an array with all its positions. The array of positions should sorted. For example, if it takes the array `[10,5,12,31,7,25]` as input, it will return `(5,[1])`, and if it takes the array `[10,5,12,31,7,25,5,18,7,5]` it will return `(5,[1,6,9])`. If it takes `[]` as it argument, it should return `None`.

Hint: the array of indices that you need to return can be of arbitrary size – e.g. in the examples above it is `[1]`, or `[1,6,9]` or `[]` – so you might want to build it on the fly. You can use the function `append` that we saw in the lecture Exercises of week 1, which takes an array and an integer and creates new array extending the old one with that integer.

[ ]:
```
def append(A,k):
    B = [None for _ in range(len(A)+1)]
```

```
    for i in range(len(A)): B[i]=A[i]
    B[len(A)]=k
    return B

def minWithPoss(array):
    if not array:
        return None

    min_val = float('inf')
    positions = []

    for index, val in enumerate(array):
        if val < min_val:
            positions = []
            positions = append(positions, index)
            min_val = val
        elif val == min_val:
            positions = append(positions, index)

    return min_val, positions

# some tests, you can add your own ones as well
# should print (5, [1, 6, 9]),(-31, [3, 5]),None,(1, [0]),(1, [0, 1]),(0, [4])
tests =␣
 ↪([10,5,12,31,7,25,5,18,7,5],[-5,12,-5,-31,7,-31,25],[],[1],[1,1],[1,1,1,1,0])
for A in tests:
    print(minWithPoss(A))
```

```
(5, [1, 6, 9])
(-31, [3, 5])
None
(1, [0])
(1, [0, 1])
(0, [4])
```

2.5 Question 5

Write a Python function:

`def occurInBoth(A, B)`

which takes two arrays of integers, each of which has no repeating elements, and returns a new array containing all integers that occur in both arrays. For example, if it takes the arrays `[5,12,31,7,25]` and `[4,12,8,7,42,31]` it will return `[12,31,7]` (the order of elements in the return array does not matter).

Hint: you will probably find useful to use the function `isIn` that we saw in the lecture Exercises of week 1, which checks whether a given element appears inside a given array.

[ ]:
```
def occurInBoth(A, B):
    common_elements = []
    for element in A:
        if element in B:
            common_elements.append(element)
    return common_elements

# some tests, but you can add your own ones as well
tests = (([5,12,31,7,25],[4,12,8,7,42,31]),
         ([5,31,7,12,25],[]),
         ([],[4,12,8,7,42,31]),
         ([4,12,8,7,42,31],[5,12,7,31,7,25]))

for (A,B) in tests:
    print(occurInBoth(A,B))
```

```
[12, 31, 7]
[]
[]
[12, 7, 31]
```

2.6 Question 6

Write a Python function:

`def occurInBothReps(A, B)`

which takes two arrays of integers, which may have repeating elements, and returns a new array containing all integers that occur in both arrays. For example, if it takes the arrays `[5,7,12,31,7,25]` and `[4,12,8,12,7,42,31]` it will return `[7,12,31]` (i.e. the repeated elements are only counted once).

[ ]:
```
# some tests, but you can add your own ones as well
tests = (([5,7,12,31,7,25],[4,12,8,12,7,42,31]),
         ([5,31,7,12,25],[4,12,8,7,42,31]),
         ([5,31,7,12,25],[]),
         ([],[4,12,8,7,42,31]))

def occurInBothReps(A, B):
    common_elements = []

    for element in A:
        if element in B and element not in common_elements:
            common_elements.append(element)

    return common_elements

for (A,B) in tests:
    print(occurInBothReps(A,B))
```

```
[7, 12, 31]
[31, 7, 12]
[]
[]
```

2.7 Question 7

Consider how a different algorithm could be used for Question 6 if it were known that both arrays passed to the method were sorted (in increasing order). Write a function:

`def occurInBothSorted(A,B)`

which has the same specification as `occurInBothReps` but where `A` and `B` are sorted. Your function should examine each element of `A` and `B` only once.

Your solution should have linear time complexity, which you should be able to calculate (covered in Week 2). Solutions with higher complexity marked with max 0.2.

Hint: You should avoid using the function `append` that we saw in the lectures, as it has linear time complexity so repeated applications thereof will lead to quadratic time overall. Instead, you could sacrifice space and start with a large enough array where you store the common elements, and then only return (a copy of) the part of the array that you have used.

[ ]:
```python
# some tests, but you can add your own ones as well
tests = (([1,3,5,5,17,17,23,24,31,56,56],[3,5,10,23,23,37,42,56]),
         ([1,3,5,5,17,17,23,24,31,56,56],[4]),
         ([4],[4,7,8,31,42]),
         ([],[4,7,8,31,42]),
         ([4],[]))

def occurInBothSorted(A, B):
    max_common_elements = min(len(A), len(B))
    common_elements = [0] * max_common_elements
    i, j, count = 0, 0, 0

    while i < len(A) and j < len(B):
        if A[i] == B[j]:
            common_elements[count] = A[i]
            count, i, j = count+1, i+1, j+1
        elif A[i] < B[j]:
            i += 1
        else:
            j += 1

    return common_elements[:count]

for (A,B) in tests:
    print(occurInBothSorted(A,B))
```

```
[3, 5, 23, 56]
```

[]
[4]
[]
[]