Laboratory Assignment 1

You can use “random” library. Please do not use any other libraries. All unspecified library lines will be suppressed and if your code fails, your submission will be graded “0” points

Problem 1 – Generate an array of random numbers [1 pts]

Write a function that takes an integer n as an input and returns a one dimensional list with random integer entries in [0:1000]. You can use random library.

- Function parameter names do not matter, but the function should expect 1 parameter.
- Function won’t be tested for exceptional cases.
- input :
    - n ∈ ℤ and n ∈ [1:10^4]
- output : x = { x : x is a list and x[i] ∈ [0:1000]}

```
>>> rl = problem1(3)
>>> print(rl)
[960, 53, 573]
```

Problem 2 – Generate a matrix of random numbers [1 pts]

Write a function that takes two integers n,m as inputs and returns a list of lists with random integer entries in [0:1000]. You can use random library.

- Function parameter names do not matter, but the function should expect 2 parameters.
- First parameter should determine the rows and second parameter should determine the columns of the matrix.
- Function won’t be tested for exceptional cases.
- inputs :
    - n,m ∈ ℤ and {n,m} ∈ [1:10^4]
- output : x = { x : x is a list and x[i,j] ∈ [0:1000]}

```
>>> rl = problem2(3,2)
>>> print(rl)
[[153, 693], [411, 483], [71, 99]]
```

Problem 3 – Straightforward 1D peak finder [3 pts]

Write a function that takes a list and an integer as inputs and finds a peak using the following algorithm, then returns the peak value and location as a tuple and returns the indices of the tested elements.

- Function parameter names do not matter, but the function should expect 2 parameters.

- First parameter should consists of a list with random entries, second parameter determines the starting index.
- Default value of the second parameter should be half of the length of the input list.
- **Be careful about the order of the steps!** Outputs will be evaluated also considering the steps of the algorithm.
- `inputs :`
    - `rl = { rl : rl is a list}`
    - `n ∈ ℤ, n ∈ [0:len(rl)-1], and default value of n = round(len(rl)/2)`
- `outputs:`
    - `x = { x : x is a tuple, x[0]: peak value, x[1]: peak index}`
    - `y = { y : y is a list and holds the tested indices of rl}`

Peak Problem:

For a given index `i`,

- `rl[i]` is a peak if `rl[i]≥rl[i+1]` and `rl[i] ≥ rl[i-1]`.
- If the index is `i=0` then, `rl[0]` is a peak if `rl[i]≥rl[i+1]`.
- If the index is `i=len(rl)-1` then, `rl[i]` is a peak if `rl[i]≥rl[i-1]`.

Algorithm:

```
Step 0    : Start with index n. If n is not specified, then update n = round(len(rl)/2).
Step 1    : Check rl[n]. If rl[n] is a peak, if true then return, found a peak!
Step 2    : Else
   Step 2.1 : If rl[n]≥rl[n+1] or n=len(rl)-1; update n=n-1 and repeat Steps 1 and 2.
   Step 2.2 : Else; update n=n+1 and repeat Steps 1 and 2.
Step 3    : Peak could not be found! (Is this possible?)
```

```
>>> rl = [250, 853, 376, 897, 704, 468, 281, 937, 396, 390]
>>> valloc,loclist=problem3(rl)
>>> print(valloc)
(897, 3)
>>> print(loclist)
[5, 4, 3]

>>> rl = [921, 343, 634, 507, 937, 613, 263, 866, 49, 780]
>>> valloc,loclist=problem3(rl,1)
>>> print(valloc)
(634, 2)
>>> print(loclist)
[1, 2]

>>> rl = [0,1,2,3,4,5,6,7,8,9]
>>> valloc,loclist=problem3(rl,0)
>>> print(valloc)
(9, 9)
>>> print(loclist)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Problem 4 – Divide and conquer based 1D peak finder `[5 pts]`

Write a function that takes a list and an integer as inputs and finds a peak using the divide and conquer approach, then returns the peak value and location as a tuple and returns the indices of the tested elements.

- Function parameter names do not matter, but the function should expect 2 parameters.
- First parameter should consist of a list with random entries, second parameter determines the starting index.
- Default value of the second parameter should be half of the length of the input list.
- **Peak Problem** definition is same as in **Problem 3**.
- **Be careful about the order of the steps!** Outputs will be evaluated also considering the steps of the algorithm.
- If you decide to follow a recursive coding style, then do not forget to update indices to the global ones, then return them.
- `inputs :`
  - `rl = { rl : rl is a list}`
  - `n ∈ ℤ, n ∈ [0:len(rl)-1], and default value of n = round(len(rl)/2)`
- `outputs:`
  - `x = { x : x is a tuple, x[0]: peak value, x[1]: peak index}`
  - `y = { y : y is a list and holds the tested indices of rl}`

Algorithm:

Step 0 : Start with index `n`. If `n` is not specified, then update `n = round(len(rl)/2)`.
Step 1 : Check `rl[n]`. If `rl[n]` is a peak, if true then return, found a peak!
Step 2 : Else
  Step 2.1 : If `rl[n]<rl[n-1]` and `n≠0`; repeat Steps 1 and 2 for `rl[0:n-1]` and update `n=round((n-1)/2)` .
  Step 2.2 : If `rl[n]<rl[n+1]` and `n≠len(rl)-1`; repeat Steps 1 and 2 for `rl[n+1:]` and update `n=round((n+len(rl))/2)`.
Step 3: Peak could not be found! (Is this possible?)

```
>>> rl = [293, 170, 207, 323, 963, 445, 351, 921, 165, 759]
>>> valloc,loclist=problem4(rl)
>>> print(valloc)
(963, 4)
>>> print(loclist)
[5, 2, 4]

>>> rl = [0,1,2,3,4,5,6,7,8,9]
>>> valloc,loclist=problem4(rl)
>>> print(valloc)
(9, 9)
>>> print(loclist)
[5, 8, 9]

>>> rl = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
>>> valloc,loclist=problem4(rl)
>>> print(valloc)
```

```
(16, 16)
>>> print(loclist)
[8, 12, 14, 16]
```

Problem 5 – Straightforward 2D peak finder `[4 pts]`

Write a function that takes a list and two integers as inputs and finds a peak using the following algorithm, then returns the peak value and location as a tuple and returns the indices (as list) of the tested elements.

- Function parameter names do not matter, but the function should expect 3 parameters.
- First parameter should consist of a list of lists with random entries, second and third parameters determine the starting indices.
- Default value of the second and third parameters should be half of the length of rows and columns of the input list, respectively.
- **Be careful about the order of the steps!** Outputs will be evaluated also considering the steps of the algorithm.
- For the updates give priority to `[i-1,j]`, then `[i,j-1]`, then `[i+1,j]`, and then `[i,j+1]`.
- Similarly, for the boundary elements keep this priority, if possible:
  - if `i=0`, `j=0`, and `rl[0,0]` is not a peak; update `i`, if `rl[i+1,j]≥rl[i,j]`, if not, update `j`.
- `inputs :`
  - `rl = { rl : rl is a list of lists}`
  - `{n,m} ∈ ℤ, n ∈ [0:len(rl)-1], m ∈ [0:len(rl[0])-1], and default values of n = round(len(rl)/2) and m = round(len(rl[0])/2)`
- `outputs:`
  - `x = { x : x is a tuple, x[0]: peak value, x[1]: peak indices [i,j]}`
  - `y = { y : y is a list and holds the tested indices of rl}`

Peak Problem for 2D:

For a given index `i,j`,

- `rl[i,j]` is a peak if `rl[i,j]≥rl[i+1,j]`, `rl[i,j]≥rl[i,j+1]`, `rl[i,j]≥rl[i-1,j]`, and `rl[i,j] ≥ rl[i,j-1]`.
- Exceptional conditions should be satisfied row and column wise as in 1D peak finding problem. For example,
  - if `i=0` then, `rl[0,j]` is a peak if `rl[i,j]≥rl[i+1,j]`, `rl[i,j]≥rl[i,j+1]`, and `rl[i,j] ≥ rl[i,j-1]` or
  - if `i=0` and `j=0`, then, `rl[0,0]` is a peak if `rl[i,j]≥rl[i+1,j]` and `rl[i,j]≥rl[i,j+1]`

Algorithm:

```
Step 0   : Start with index n,m. If n and m are not specified, then update n=round(len(rl)/2)
           and m=round(len(rl[0])/2).
Step 1   : Check rl[n,m]. If rl[n,m] is a peak, if true then return, found a peak!
Step 2   : Else
  Step 2.1 : If rl[n-1,m]≥rl[n,m] and n≠0; update n=n-1 and
             repeat Steps 1 and 2.
```

Step 2.2 : If `rl[n,m-1]≥rl[n,m]` and `m≠0`; update `m=m-1` and repeat Steps 1 and 2.
Step 2.3 : If `rl[n+1,m]≥rl[n,m]` and `n≠len(rl)-1`; update `n=n+1` and repeat Steps 1 and 2.
Step 2.4 : If `rl[n,m+1]≥rl[n,m]` and `m≠len(rl[0])-1`; update `m=m+1` and repeat Steps 1 and 2.
Step 3 : Peak could not be found! (Is this possible?)

```
>>> rl = [[485, 972, 829], [522, 32, 696], [651, 965, 476], [389, 724, 185]]
>>> valloc,loclist=problem5(rl)
>>> print(valloc)
(972, [0, 1])
>>> print(loclist)
[[2, 2], [1, 2], [0, 2], [0, 1]]

>>> rl = [[861, 93, 242, 477, 324], [936, 198, 177, 914, 944], [315, 211, 479,
717, 348], [544, 712, 667, 967, 119], [816, 920, 155, 419, 57]]
>>> valloc,loclist=problem5(rl,0,0)
>>> print(valloc)
(936, [1, 0])
>>> print(loclist)
[[0, 0], [1, 0]]

>>> valloc,loclist=problem5(rl,4,4)
>>> print(valloc)
(944, [1, 4])
>>> print(loclist)
[[4, 4], [3, 4], [2, 4], [1, 4]]
```

Problem 6 – Divide and conquer based 2D peak finder `[6 pts]`

Write a function that takes a list and two integers as inputs and finds a peak using the divide and conquer approach, then returns the peak value and location as a tuple and returns the indices (as list) of the tested elements.

- Function parameter names do not matter, but the function should expect 3 parameters.
- First parameter should consist of a list of lists with random entries, second and third parameters determine the starting indices.
- Default value of the second and third parameters should be half of the length of rows and columns of the input list, respectively.
- **Peak Problem** definition is same as in **Problem 5**.
- **Be careful about the order of the steps!** Outputs will be evaluated also considering the steps of the algorithm.
- If you decide to follow a **recursive** coding style, then **do not forget to update** then return the global indices.
- Tested elements for finding the global maximum of a column should **not included** to the list of indices of the tested elements, only the indices of the global maximum should be **included**.

- inputs :
  - rl = { rl : rl is a list of lists}
  - {n,m} ∈ ℤ, n ∈ [0:len(rl)-1], m ∈ [0:len(rl[0])-1], and default values of n = round(len(rl)/2) and m = round(len(rl[0])/2)
- outputs:
  - x = { x : x is a tuple, x[0]: peak value, x[1]: peak indices [i,j]}
  - y = { y : y is a list and holds the tested indices of rl}

Algorithm:

```
Step 0    : Start with index n,m. If n and m are not specified, then update n=round(len(rl)/2)
            and m=round(len(rl[0])/2).
Step 1    : Check rl[n,m]. If rl[n,m] is a peak, if true then return, found a peak!
Step 2    : Else
   Step 2.1 : Find the global maximum on column m and update n.
   Step 2.2 : Check rl[n,m]. If rl[n,m] is a peak, if true then return.
   Step 2.3 : Else
       Step 2.3.1 : If rl[n,m-1]>rl[n,m] and m≠0; repeat Step 2 for rl[:,0:m-1]
                    and update m=round((m-1)/2).
       Step 2.3.2 : If rl[n,m+1]>rl[n,m] and m≠ len(rl[0])-1; repeat Step 2 for rl[:,m+1:]
                    and update m=round((m+len(rl[0]))/2).
Step 3    : Peak could not be found! (Is this possible?)
```

```
>>> rl = [[718, 239, 177, 770], [777, 663, 630, 300], [595, 726, 6, 73], [253,
105, 879, 863], [189, 862, 277, 682]]
>>> valloc,loclist=problem6(rl)
>>> print(valloc)
(879, [3, 2])
>>> print(loclist)
[[2, 2], [3, 2]]

>>> rl = [[399, 922, 327, 971], [767, 854, 256, 106], [339, 25, 74, 183], [613,
746, 126, 114], [558, 368, 880, 822]]
>>> valloc,loclist=problem6(rl,0,0)
>>> print(valloc)
(880, [4, 2])
>>> print(loclist)
[[0, 0], [1, 0], [1, 2], [4, 2]]
```

Explanation: Starts with the 399 at [0, 0], finds the global maximum at column m=0, at n=1, i.e. 767 at [1, 0]. Then updates m = round((m+len(rl[0]))/2) = round((0+4)/2) = 2, since m=0. Repeats the procedure for 256 at [1,2] and finds the global maximum at column m=2, at n=4, i.e. 880 at [4, 2]. Checks the peak conditions and voilà!