

1. ArrayUtility class

1. Implementation of the basic operations of the ArrayUtility class

Implement in Java the class `ArrayUtility` which offers basic operations over mono-dimensional and bi-dimensional arrays. ALL methods MUST be implemented as class methods (i.e., static methods). The signature of the methods the `ArrayUtility` class must contain are the following:

- `public static int findMax(int[] A, int i, int j):` return the maximum element of the array `A` between position i and j .
- `public static int findMaxPos(int[] A, int i, int j):` return the position of the maximum element of the array `A` between position i and j .
- `public static int findMin(int[] A, int i, int j):` return the minimum element of the array `A` between position i and j .
- `public static int findMinPos(int[] A, int i, int j):` return the position of the minimum element of the array `A` between position i and j .
- `public static void swap(int[] A, int i, int j):` swap the elements in position i and j in the array `A`.
- `public static void shiftRight(int[] A, int i, int j):` shift on the right all the elements of the array `A` starting from position i and until position j (i.e., move the element in position k to position $k + 1$ for all $i \leq k < j$).
- `public static void shiftLeft(int[] A, int i, int j):` shift on the left all the elements of the array `A` from position j until position i (i.e., move the element in position k to position $k - 1$ for all $i < k \leq j$).

- `public static int[] createRandomArray(int size, int min, int max):` create and return an array of size `size` of random elements with values between `min` and `max` (use the `Math.random()` method of java).
- `public static int[][] createRandomMatrix(int rows, int cols, int min, int max):` create and return an bi-dimensional array with `rows` rows and `cols` columns of random elements with values between `min` and `max` (use the `Math.random()` method of java).
- `public static int[] copyArray(int[] A):` return an array that is the copy of `A`.
- `public static int[][] copyMatrix(int[][] A):` return a bi-dimensional array that is the copy of `A`.
- `public static int findElement(int[] A, int n):` return the position of the number `n` in the array `A` (it returns `-1` if `n` is not present in `A`).
- `public static int binarySearch(int[] A, int n):` return the position of the number `n` in the array `A` (it returns `-1` if `n` is not present in `A`). The array `A` is sorted and the search is implemented using the binary search algorithm.

2. Running Time Comparison - Maxsort

Add to your class `ArrayUtility` a static method implementing the algorithm *Maxsort*, that takes an unsorted array of integer numbers as input and sorts it in descending order, by doing the following:

First, it searches in the whole array for the biggest element. It then puts this element to the beginning of the array. Then, it searches the whole array excluding the first element for the biggest value, and puts it to the second position, and so on.

Implement the algorithm according to two different strategy:

- by using the method `shiftRight(int[] A, int i, int j):` if the maximum element is found in position `j` and should be put in position `i` then: (i) `A` is shifted on the right starting from position `i` but taking care of remembering the element in position `j` that will be overwrote; (ii) copy the saved element in position `i`.

- by using the method `swap(int[] A, int i, int j)`: if the maximum element is found in position i and should be put in position j then use `swap` to exchange the element in position i with the element in position j .

Then:

1. write a main class that creates random arrays of size $n = 10, 100, 1000$, etc.
2. For each array created it orders it using the two implementations of *Maxsort* and measures the running times. To measure the running time use the java method `System.nanoTime()` in the following way:

```
long startTime = System.nanoTime();  
... the code being measured ...  
long estimatedTime = System.nanoTime() - startTime;
```