1 Linear Interpolation

In the course we have discussed the implementation of a pump class providing a relationship between manometric pressure and flow rate of a pump using its characteristic curve. Provided was a class with the following public interface

```
class Pumpe{
    public Pumpe (double[] H, double[] Q)
    public double GetH()
    public double GetQ()
    public double SetH(double H);
    public double SetQ(double Q);
}
```

In addition, the class contains the following private fields:

```
    private double h, q;
    private double[] ha, hq;
```

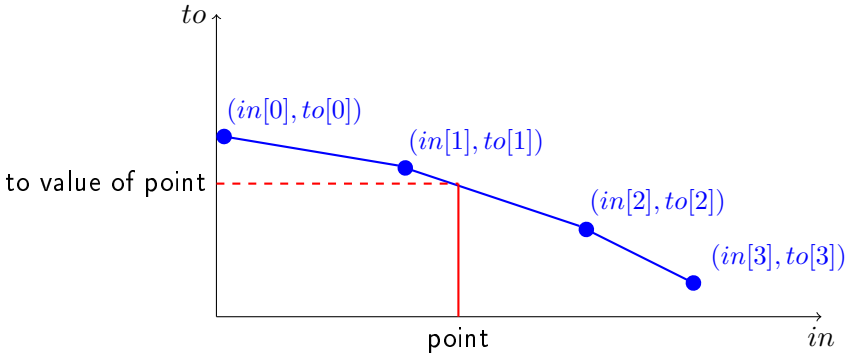
In order to accomodate the charcteristic curve we postulated a method

```
static double Interpolate(double point, double[] in, double[] to)
```

in the course.

Note that since the method is static, it cannot access the private elements of a class object directly, but requires them to be passed as arguments. `Interpolate` finds the interval in `in` containing `point` and returns the value linearly interpolated between the corresponding indices of `to`. Implement the function such that

- if `point` is smaller than all values of `in`, it returns the value of `to[k]` where `k` is the index of the smallest element of `in`,
- if `point` is greater than all values of `in`, it returns the value of `to[k]` where `k` is the index of the greatest element of `in`,
- otherwise it returns the to-value corresponding to $point$ when linearly interpolating `to` for indices of the two neighboring points of `in`



Download a skeleton for this assignment from http://lec.inf.ethz.ch/baug/informatik2/2015/ex/ex04/Pumpe/Main.java

Validate your solution here: https://judge.inf.ethz.ch/team/websubmit.php?cid=60&problem=IB1511

Hints

Given are n points on the characteristic curve by two arrays $X \in \mathbb{R}^n$ (called `in` above) and $Y \in \mathbb{R}^n$ (called `to` above). With this notation, the linear interpolation problem from above has two aspects:

(a) For a given point $x \in \mathbb{R}$, **find the indices l and r of the two points X_l and Y_r** that are closest and left and right from x, i.e. find the greatest X_l with $X_l \leq x$ and the smallest X_r with $X_r \geq x$. If x is smaller than all x_i $(0 \leq i < n)$, then x_l is undefined. If x is greater than all x_i $(0 \leq i < n)$ then x_r is undefined. Note that we did not postulate a particular order for the entries in X or Y.

(b) Given two indices l and r and x with $x_l \leq x \leq x_r$, compute the **local linear interpolation**, i.e. provide the corresponding point $y = y_l + \frac{x-x_l}{x_r-x_l} \cdot (y_r - y_l)$.

Knowing this separation of concerns, you can obviously split the solution of this assignment by solving (a) and (b) separately. Moreover, we suggest that you **implement and test** your solution of these subtasks in separation from anything else. For example write a little class that inputs x, X and Y and apply the algorithm to your testdata. Only when that works and you are confident you did not miss important cases, integrate it back into the context of this exercise.

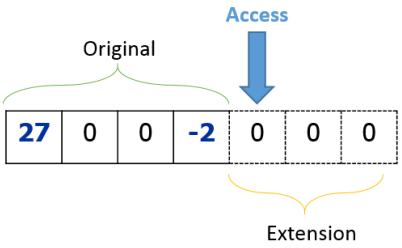
2 Selfgrowing Array

By now you've already gained some experience with arrays: once the size of an array is fixed, it cannot be changed. Any attempt to write to an array index bigger then the size of the allocated array will cause a java exception. (Error and termination of the program if not handled).

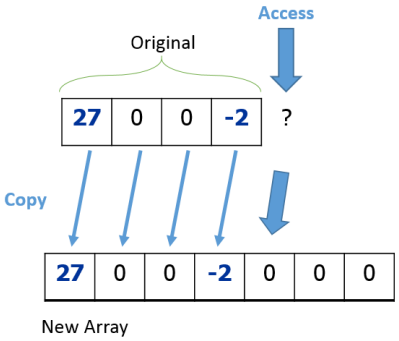
The aim of this assignment is to implement an array of int-numbers, which implements adaptive growth. The public interface for the growing array is as follows:

```
class GrowingArray {
    // constructor
    public GrowingArray()

    // pre: any index ind >= 0
    // post: return value at position ind
    // If no value has been set to ind previously, return 0
    public int get(int ind)

    // pre: any index ind >= 0, any value val
    // post: store value val at position ind
    public void set(int ind, int val)

    // post: returns size of currently allocated storage
    public int size()
}
```

By giving the user access via this object's interface (instead of direct access to an array) it is possible to hide the actual implementationas there are many different possibilities to implement such a dynamic array. For this exercise, it should be implemented as indicated by the following illustration:



Impression from the outside



Realisation behind the scenes

Next power of two

Growing the array exactly to the size required to host a newly requested element can result in a huge number of allocations when sequentially traversing and resizing the array. In order to avoid this, the array should be grown to the next size which is a power of two. So e.g. if the user requests Element 931 we grow the array to size 1024 which corresponds to 2^{10}.

Your first task is to implement the subroutine that returns the next power of two given any input number. Download a skeleton for this assignment from http://lec.inf.ethz.ch/baug/informatik2/2015/ex/ex04/GrowingArray/Main.java

Resize

Finally your task is to implement the routine

```
private void resize(int size)
```

which is called within set. As motivated earlier it is supposed to

- create a new array using the NextPowerOfTwo function,
- copy the old (smaller) array into the new array,
- replace the internal storage (called "data" in the skeleton code) with the new storage.

Validate your solution here: https://judge.inf.ethz.ch/team/websubmit.php?cid=60&problem=IB1512