1. Implement Selection sort.
2. Implement Quicksort.

3. Create a function that takes an array and returns a string.
    - **0** Repeat the actual decrypted value (using like this : 014 to repeat 14 times). *Warning: When you start a repeat you can't stop it to add other numbers.*
    - **1, 2, 3, 4** = g, o, l, e
    - **5** Corresponding to up case of letter before this.
    - **6** Add a point to the end.
    - **7** Change case of the first letter.
    - **8** Reverse the string.
    - **9** Clear the actual decrypted string.

```
numToGoogle(["12213467"]) → "Google."

numToGoogle(["12213467", "321"]) → "Google.log"

numToGoogle(["12213467", "321", "122906"]) → "Google.log"

numToGoogle(["15", "2502", "15", 345]) → "GOOGLE"

numToGoogle(["15", "2502", "15", 35, 45]) → "GOOGLE"

numToGoogle([15, 202, 1, 3, 4]) → "Google"
```

4. Build a function that creates histograms. Every bar needs to be on a new line and its length corresponds to the numbers in the array passed as an argument. The second argument of the function represents the character to be used for the *bar*.

```
histogram(arr, char) → str

histogram([1, 3, 4], "#") → "#\n###\n####"

#
###
####

histogram([6, 2, 15, 3], "=") → "======\n==\n==============\n==="

======
==
```

```
===============
===

histogram([1, 10], "+") ➞ "+\n++++++++++"

+
++++++++++
```

5. Binary search is the fastest method to search a sorted array.

   Imagine you are an undercover agent and you're at a cocktail party hosted by the bad guys. You sneak into the drug lord's office looking for an important document. Fortunately, the drug lord has organized his files by number from lowest to highest. You know the number of the file you are looking for is 3412. Since there are thousands of files and you have very limited time you wouldn't start at 0 and check each one. This would be the iterative approach. Instead, you would look halfway through the stack and see what number it is, if it's less than 3412 you know that everything below it won't contain the file for so you discard the bottom pile and search the top. Each time looking in the middle and discarding the half that won't contain the number you are looking for. This is a binary search.

   You Your challenge is to write a function that searches a sorted array and returns the index of the number you are searching for.
   “BUT WAIT!!!” you say.
   "Why don't I just used `findIndex`, or `indexOf` to get the index?" you ask.
   Because both of those are iterative. They have a time complexity of O(n), in our spy scenario that is too long, they will find you searching the files and you'll go swimming with the fishes. The binary search has a time complexity of O(logN). Meaning it is substantially faster.”

   Arguments

   - `Array`: The array over which you searching.
   - `Number`: The number whose index you want to return.
   - Returns `Index` || `-1`: If the number is found the index is returned, otherwise it returns `-1`.

```
binary([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],2)

binary([11, 12, 21, 23, 40, 40, 42, 43, 54, 57, 92], 40)
```