EECS 1012 Programming Assignment/Project 1
Due: July 10 (Sunday) 23:00 pm

This assignment/project can be done by yourself, or, with a team of two. Only one member of each team submits the answers.
Note that, however, **you can neither share your solution with other teams nor get help from them**. Please be aware that your code will be checked by both a plagiarism detection tool and TAs to make sure your solution to this problem is unique.

This programming assignment is intended to polish your ability for the following skills:

- Ability to develop algorithms using flowchart
- Ability to use control structures such as branches and loops
- Ability to implement algorithms using JavaScript

You have 5 algorithms to develop (4 problems + one variations).

Problem 1

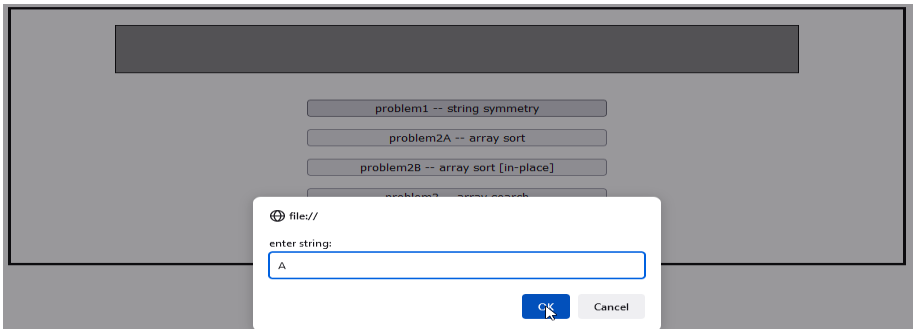
In this problem, develop an algorithm that, given an input string, determines if the string is symmetric. A string is symmetric if it reads the same backward as forward, e.g., "*a*" "*aa*", "*dad*", "*madam*", "*that is a si taht*". On the other hand, "*ab*" "*random*" or "*Dad*" "*this is that*" are not symmetric.
There are several ways to check symmetry. One approach, for instance, is to create a reverse string and then check if they are the same. Here we refrain from using this approach as it requires extra storage for the reverse string. The algorithm here operates on the input string only. The main idea is to check the characters in the string and see if all the corresponding pair of characters (who are they?) match.

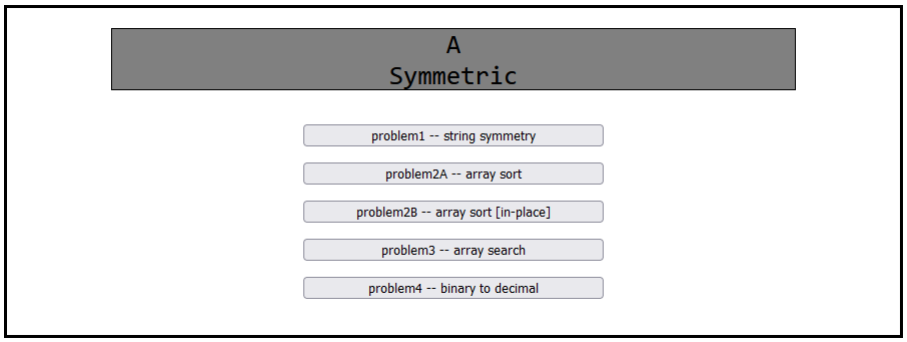
- Develop flowchart for this algorithm
  - Give pre-condition and post-condition for your algorithm.
  - Hint: you will need a loop for checking the characters. You may need two indexes, for the pair of characters to be compared. (What are the initial values of the two indexes and how they proceed?)
  - You can use *atr.length* to get the number of characters in the string.
  - For accessing characters in string, you can use array notation *str[i]*
  - For characters comparisons, just use =, same as comparing numerical values
  - The algorithm should stop the loop immediately/early when a mismatch is found. But you cannot say, "go out of loop". Use the approach mentioned in class and lab to stop the loop earlier. The loop should have only one exit point.
  - The algorithm proceeds until a mismatch is found, or, all the corresponding characters are compared (how to determine?) and no mismatch is found. In the former case the algorithm outputs "Not symmetric". In the latter case the algorithm outputs "Symmetric".
  - Save the flowchart as **img_symmetry.jpg**

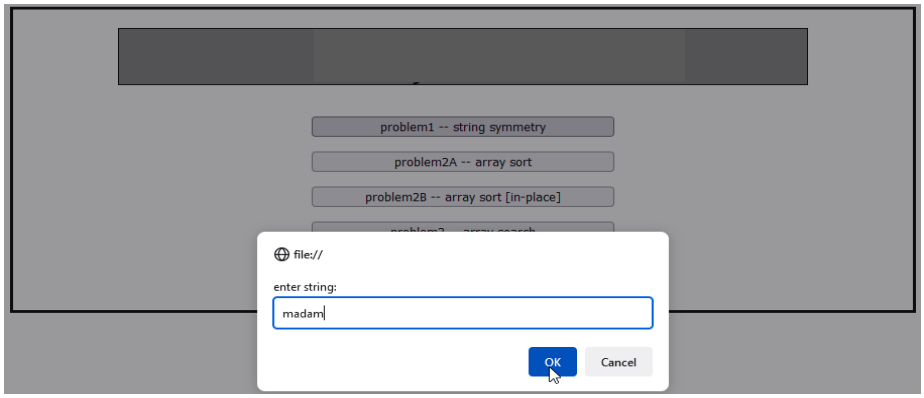
- Implement the algorithm in JavaScript
  - Complete the function *problem1()* in JS file, and attach the onclick event to the corresponding button in html.
  - *prompt* to ask users for the input string
  - Use *prompt* to ask users for the input string
  - For accessing characters in string, use array notation *str[i]*, no need to use other functions
  - For characters comparisons, just use ==, same as comparing numerical values. == operators on two characters will compare if they are the same.

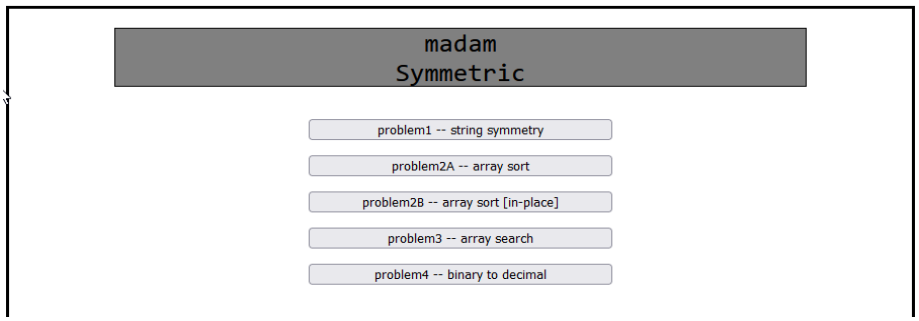
- You should stop the loop immediately when a mismatch is found. But you cannot use *break*. Use the approach mentioned in class and used in lab to stop the loop earlier. The loop should only have one exit point.

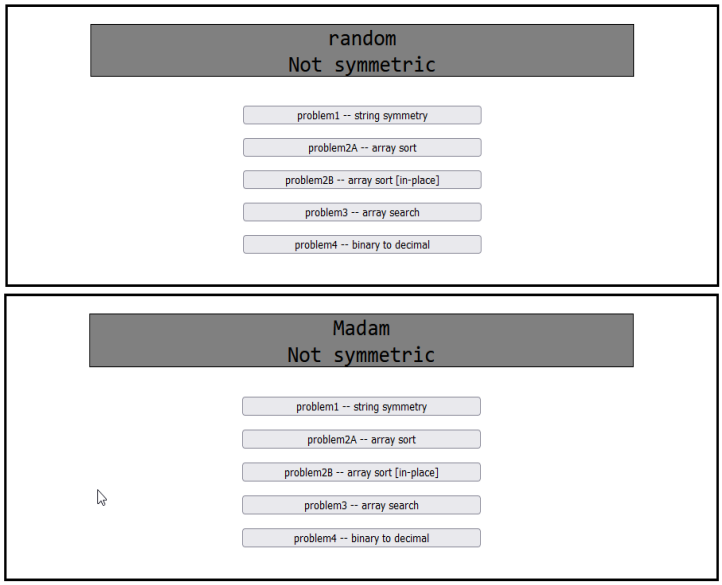
Sample output











Problem 2

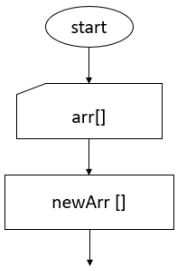
In this problem, we want to sort an input array.

Problem 2A (easier)

In this problem, develop an algorithm that, given an unsorted array of integers, creates a new array which is the sorted version of the given array.

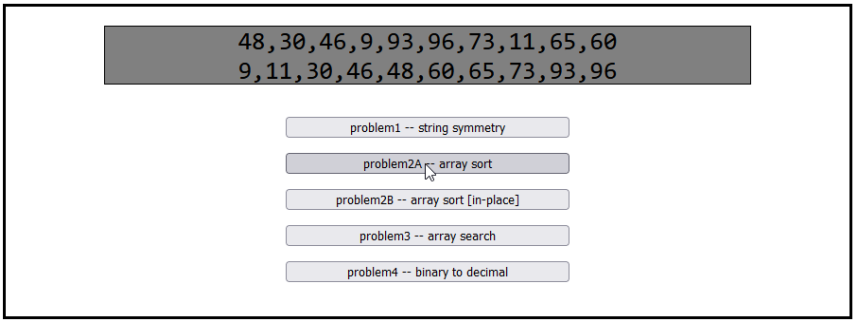
Here is the main idea of the algorithm: first scan through the original input array, finding the smallest element, and then insert this element into the beginning (index 0) of the new array, and remove this smallest element from the input array. Next, scan the (shortened) input array again, finding the smallest element, and then insert it into the next available position of the new array and remove it from the input array. Repeat the process until the original array becomes size 0 (this is also the time when the new array contains all the values in the original input array).

- Develop flowchart for this algorithm
  - Give pre-condition and post-condition for your algorithm.
  - Hint: you will need nested loops for doing the work in one main algorithm. You can also write a sub-program to do some work. In this case the main algorithm has one loop.
  - Hint: to find the smallest element in an array, define a variable, say, *minIndex*, which is initially 0, then if an element at index *k* of the array is smaller than the element at index *minIndex*, update *minIndex* to be *k*.
  - For removing an element at index *j* from array *arr*, you can just say "remove element at index j from arr", or "remove arr[j] from arr"
  - Start your flowchart as shown in the following picture, which first get an input array *arr*, and then create new array *newArr*.
  - Save the flowchart as **img_sortA.jpg**



- Implement the algorithm using JavaScript
  - Complete function *problem2A()* in JS file, and attach the onclick event to the corresponding button in html. This function first generates a random array, and at the end output both the original and the sorted arrays.
  - Your implementation should match your flowchart. E.g., if your flowchart contains only one main algorithm, then you should implement the whole algorithm in the function. If you use sub-algorithm in flowchart, then you should implement a sub-function.
  - For removing an element from array, turn to w3school for help. Hint, function splice() is one of the choices. To use this, you need to keep track of the index of the element to be removed.

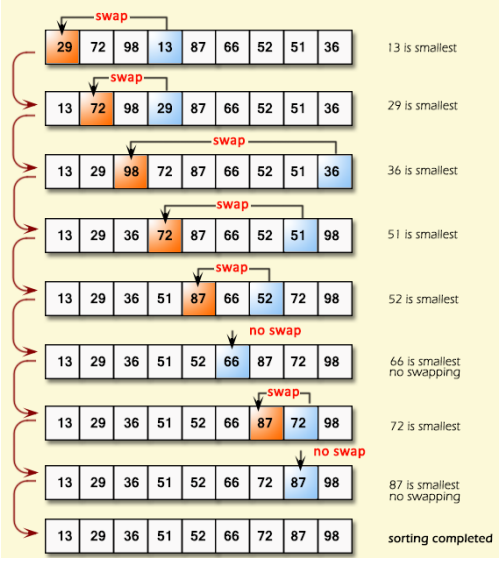
Sample output (each run will generate a different input array)



Problem 2B

In this problem we want to do an "in-place" sorting, that is, given an unsorted array, sort the array directly, by swapping elements on the array, without using and returning another array.

Here is the main idea: first scan through the original input array from index 0 to the end, finding the smallest element. This element should be the first element of the sorted array. To make it the first element of the array, the algorithm swaps the smallest element with the element at index 0. Now, subarray at index 0 is considered the sorted part of the array, and the rest is unsorted. Next, scan through the unsorted part of the array, from index 1 to the end, finding the smallest element in the range. Then make it the 2nd element of the array by swapping it with the element at index 1. Now subarray at index 0 and index 1 are sorted, and the rest are unsorted. Next, scan through the unsorted part of the array from index 2 to the end, finding the smallest element and put it as the 3rd element of the array. Repeat the process until no more unsorted subarray exists. Following figure shows the steps of sorting an array.

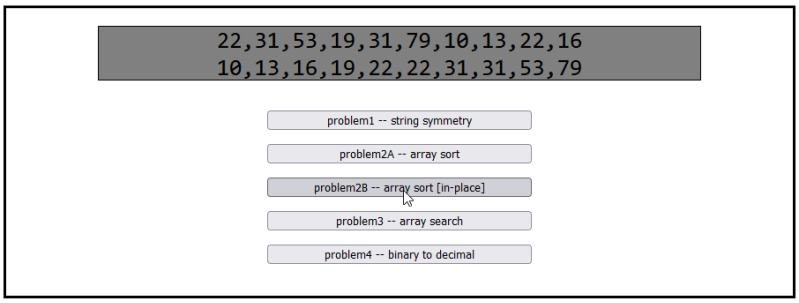


In each iteration, subarray to the left of the orange element is sorted. From orange element to the end is the unsorted subarray

- Develop flowchart for this in-place sorting algorithm,
  - Give pre-condition and post-condition for your algorithm.
  - Hint: you will need nested loops for doing the work in one algorithm.
  - For swapping two elements, create a variable *temp* to hold one of the values temporarily.
  - Save the flowchart as **img_sortB.jpg**

- Implement the algorithm using JavaScript
  - Complete function *problem2B()* in JS file, attach the onclick event to the corresponding html button
  - Your implementation should match your flowchart

Sample output (each run will generate a different input array)



Problem 3

In this problem, develop an algorithm to search a key in a sorted array.

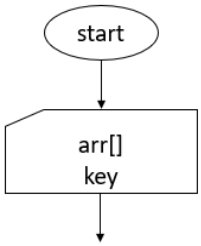
Given an input array of integers and a search key, one general approach to searching the key in the array is to scan the array, comparing each element against the search key, until one element matches the key, or no match is found till the end of the array. In the latter case the algorithm concludes that the key is not in the array.

When the input array is sorted, we can search in a more efficient way. Assume the array is sorted in ascending order. The idea is, we retrieve the middle value from the array and compare it against the search key. If the middle value and the search key match, we've found it and the algorithm stops. If not so and the search key is smaller than the middle value, we search the 1st (left) half of the array as the key could not be in the 2nd (right) half (convince yourself this is true!), otherwise -- the search key is larger than the middle value -- we search the 2nd half of the array as the key could not be in the 1st half of the array.

In the half sub-array, we repeat the above steps, retrieving the middle key of the sub-array, comparing it against the search key. If they match, we've found it and stop. If they are not same, then go to either first half or second half of the subarray based on the comparison result.

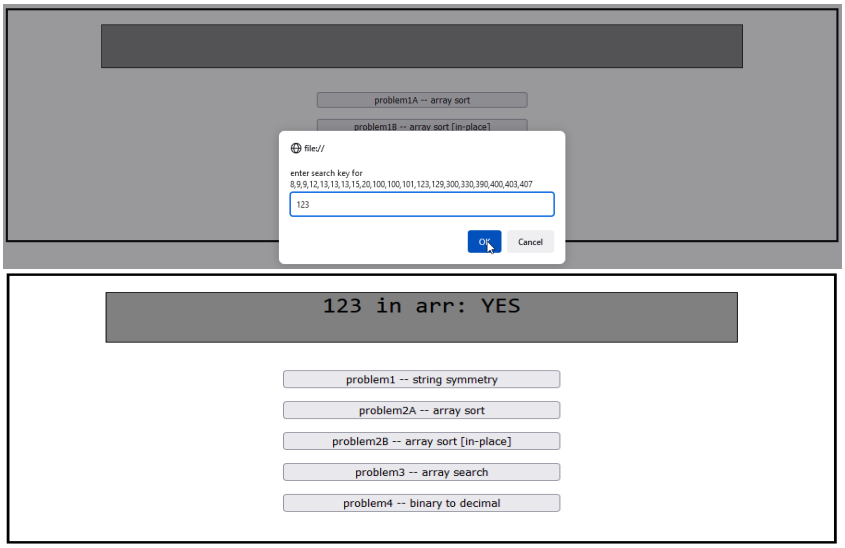
The search stops either when a match is found, or there is no subarray to search. In the latter case the algorithm concludes that the key is not in the array. Following figure shows the steps of searching 38 in a sorted array.

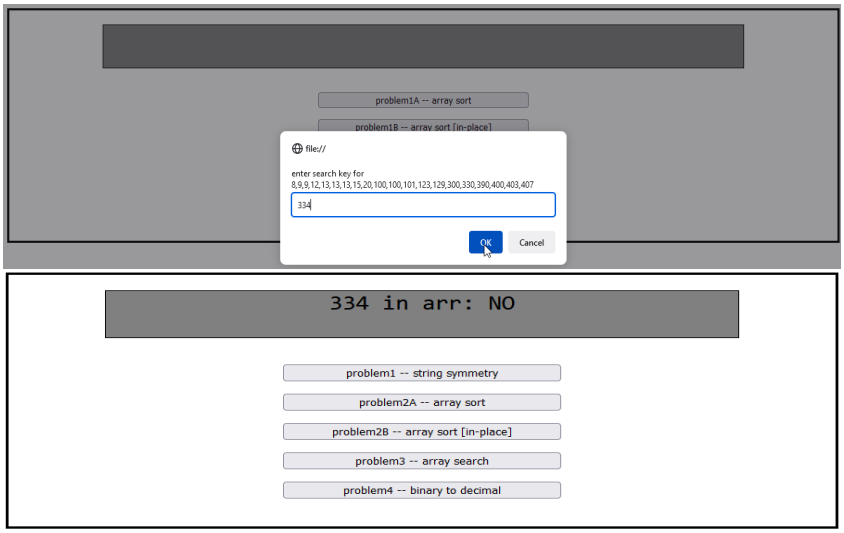
- Develop flowchart for this algorithm
  - Give pre-condition and post-condition for your algorithm.
  - The algorithm takes as input a search key and an array, output “key in arr: YES" or “key in arr: NO"
  - Hint: use loop and two variables, one to store the starting index of the current search subarray and another to store the end of the current search subarray. These variables are often called *start*, *end*, *low*, *high*, etc. Let’s call it L and H. In each iteration we search array from index L to H. Initially L is 0 and H is the last index of the original array. In each iteration of the loop, we get middle index M and middle value. If the search key is smaller than the middle values, search 1st half of the array by setting H to be M-1 (L does not change), otherwise, search the 2nd half by setting L to be M+1 (H does not change). Convince yourself this works correctly.
  - If the search (sub)array is of even size, the middle index can be either the end of the 1st half or the beginning of the 2nd half (so to be precise, the “middle” element should be called “middle-most” element).
  - The loop should stop immediately when the key is found.
  - What is the terminating condition of the search loop?
  - Must use loops. You should not use recursions. (We will learn recursion later).
  - Start your flowchart as shown in the following figure. The algorithm takes as input an array *arr* and a search key *key*
  - Save the flowchart as **img_search.jpg**



- Implement the algorithm using JavaScript
  - Complete the function *problem3()*, and attach the onclick event to the corresponding button in html.
  - Your implementation should match your flowchart.

Sample output





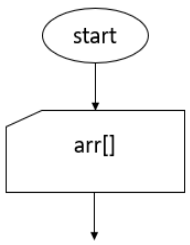
Problem 4

In this problem, develop an algorithm to convert a binary representation to its decimal value. That is, convert base 2 representation to base 10 representation.

Given an input array of 0 and 1s, treated as binary representation of a number, calculate the decimal value of the number. For example, if the array is filled with 1 1 0 1, the algorithm should output 13. If the array is filled with 0 0 1 1, then the output should be 3.

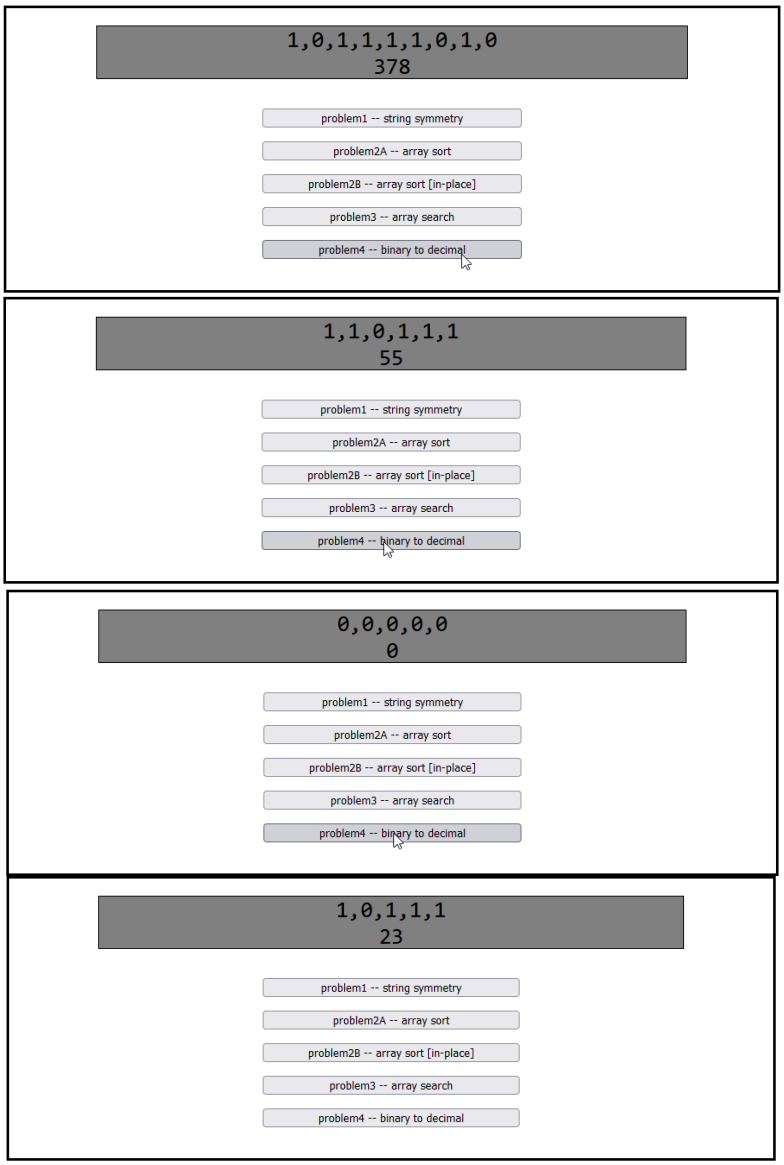
To understand the conversion, you can look at the provided document "*Number system 1012 22s.pdf*", or search online for relevant information. In general, if a binary number has *n* digits as $d_1d_2d_3\ldots d_{n-1}d_n$, it can be converted to decimal by formula $d_1*2^{n-1} + d_2*2^{n-2} + d_3*2^{n-3} + \ldots d_{n-1}*2^1 + d_n*2^0$. As another example, if the array is filled with 0 0 1 0 0 1 1 1 then the program should output 39 because $0*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 39$.

- Develop flowchart for this algorithm
  - Give pre-condition and post-condition for your algorithm.
  - The algorithm takes as input an array, filled with 0s and 1s, outputs the decimal value represented by the 0s and 1s.
  - You will need a loop to traverse the array. The array can be traversed from left to right or, from right to left. Here you should traverse from right to left.
  - You will need to calculate the power of the base, i.e., 2^0 2^1 2^2. Define a sub-algorithm named *power (base, n)* which calculates the *base* to the power of *n*, i.e., $base^n$. Call this sub-algorithm in the main algorithm. In the main algorithm, if need to calculate 2 to power of *n*, don't write 2^n, instead, call function *power* to calculate.
  - Implement sub-algorithm *power* using loop.
  - Start your flowchart as shown in the following figure. The algorithm takes as input an array *arr*
  - Save the flowchart as **img_binary.jpg**



- Implement the algorithm using JavaScript (Your implementation should match your flowchart)
  - Complete the function *problem4()*, and attach the onclick event to the corresponding button in html. This function first generates an array of random size, filled with randomly generated 0s and 1s.
  - Define a sub-algorithm *power(base, n)* that calculates $base^n$. Call this function in main when you need to calculate the power of base.

Sample output (each run will generate a different input array)



You may want to use an online binary to decimal conversion calculator to check the correctness of your output. One such a calculator can be found at http://www.cleavebooks.co.uk/scol/calnumba.htm
Another can be found at https://www.rapidtables.com/convert/number/base-converter.html

---

SUBMISSION

Submit the following files:

`team.txt` (only need to submit this file if you work in a team of two. List names, student numbers of both the team members)

```
img_symmetry.jpg, img_sortA.jpg, img_sortB.jpg, img_search.jpg,  img_binary.jpg
A1.html
A1.css
A1.js
```

Compress the files (.zip or .tar or .gz), and then submit the (single) compressed file on eClass.

- **Note that if you work on a team, only on team member submit (include the team.txt file in the submission). Another member does not need to submit anything.**

Develop algorithms to determine the lexicographical (alphabetical) order of two input strings. If a string *str1* lexicographically precedes another string *str2*, we can say *str1 < str2*, and it means *str1* should appear earlier in dictionary than *str2*. If *str1* lexicographically follows *str2*, we can say *str1 > str2*, and it means *str1* should appear later in dictionary.
In general, the algorithm compares the two strings character by character, comparing the alphabetical order of the corresponding characters, based on the ASCII values of the characters (for example, character 'a' has value 97 and 'd' has value 100. 'A' has value 65 and 'D' has value '68'). If the two corresponding characters are the same, continue with the next corresponding characters. The comparison stops when corresponding characters are different, or, end of the string is reached.

Problem 3A (easier)
In this version, assume the two input strings are of the same length.
The function starts by comparing the first character of each string. If they are equal, it continues with the following pairs until the characters differ, or, reaches the ends of the stings. If the first character that does not match has a lower alphabetical order in *str1* than that in *str2* (for example, 'a' in *str1* has lower ASCII value than 'd' in *str2*), then *str1* is deemed lexicographically precedes *str2*. The algorithm stops immediately and outputs " *str1 < str2*". If the first character that does not match has a higher alphabetical order in *str1* than that in *str2* (for example, 'd' in *str1* has larger ASCII value than 'A' in *str2*), then *str1* lexicographically follows *str2* and the algorithm outputs "*str1 > str2*". If no mismatch is detected until the end string, the two strings have the same content and the algorithm outputs "*str1 == str2*".
As an example, given first input string "*jklc*" and second input string "*jkqc*", comparison starts off by comparing the first characters from the two strings. i.e '*j*' from "*jkl*" and '*j*' from "*jkm*". Since they are equal, the next two characters are compared, i.e *'k'* from "*jkl*" and *'k'* from "*jkm*". Since they are also equal, the next two characters are compared i.e, *'l'* from "*jkl*" and '*q*' from "*jkm*". Since ASCII value of '*l*' is smaller than that of '*q*', we conclude that the first string lexicographically precedes the second string and the algorithm will output "*jklc < jiqc*". Note that the algorithm stops immediately and will not compare the rest of characters.

More examples (convince yourself about the lexicographic orders):

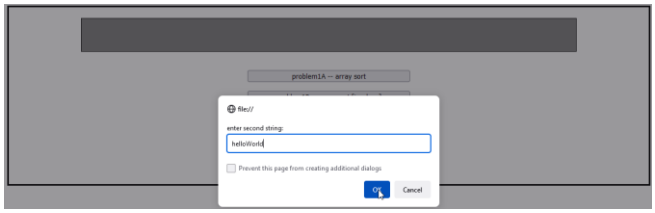
First input string	Second input string	Expected output
"abc"	"abe"	"abc < abe"
"cue"	"bat"	"cue > bat"
"apple"	"beast"	"apple < beast"
"exit"	"exam"	"exit > exam"
"exam"	"exam"	"exam" == "exam"
"exam"	"Exam"	"exam > Exam"

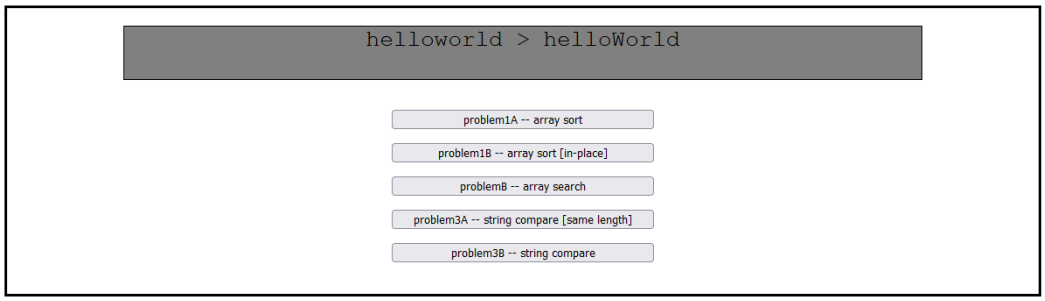
- Develop flowchart for this algorithm,
  - For accessing characters in string, you can use array notation *str[i]*
  - For characters comparisons, just use < > ==, as numerical values

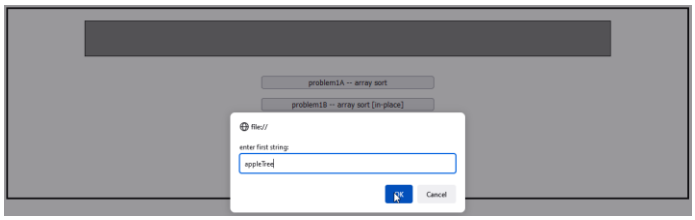
- Should stop the searching loop immediately when a mismatch is found. But you cannot say, "go out of loop". Use the approach mentioned in class and lab to stop the loop earlier. The loop should have only one exit point.
- Save the flowchart as **img_03A.jpg**

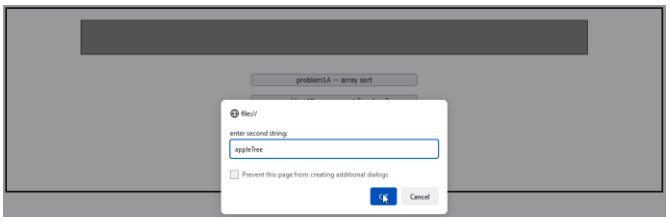
- Implement the algorithm using JavaScript
  - Use *prompt* to ask users for two input strings
  - For accessing characters in string, use array notation *str[i]*, no need to use other functions
  - For characters comparisons, just use < > ==, as numerical values. < > == operators on two characters will compare the characters' ASCII values automatically.
  - You should stop loop immediately when a mismatch is found. But you cannot use *break*. Use the approach mentioned in class and used in lab to stop the loop earlier. The loop should only have one exit point.

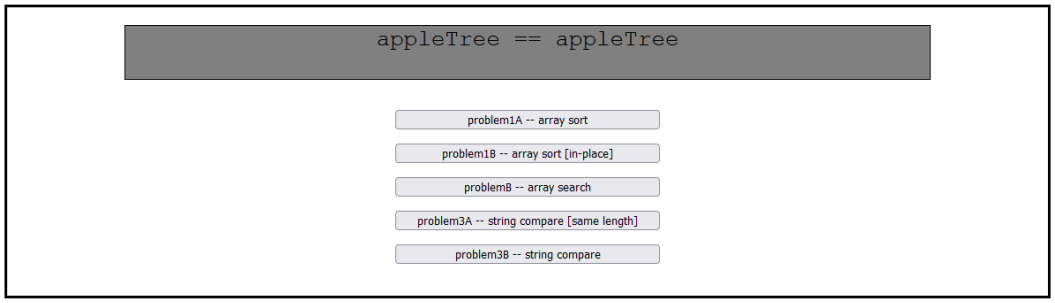
Sample input/output











Problem 3B (harder)

In this version, assume the two input strings can be of same or different lengths. For example, "*apple*" vs. "*ave*", "*base*" vs. "*basement*".

Same as above, this function starts by comparing the first character of each string. If they are equal, it continues with the following pairs until the characters differ, or, we reach the end of one of the stings (or reach both ends if the strings have the same length). If no mismatch is detected when one string is exhausted, for example, "*base*" vs. "*basement*", then the shorter string is deemed lexicographically preceding the longer one, i.e., "*base*" < "*basement*", which means "*base*" should appear earlier in dictionary than "*basement*" (make sense, right)

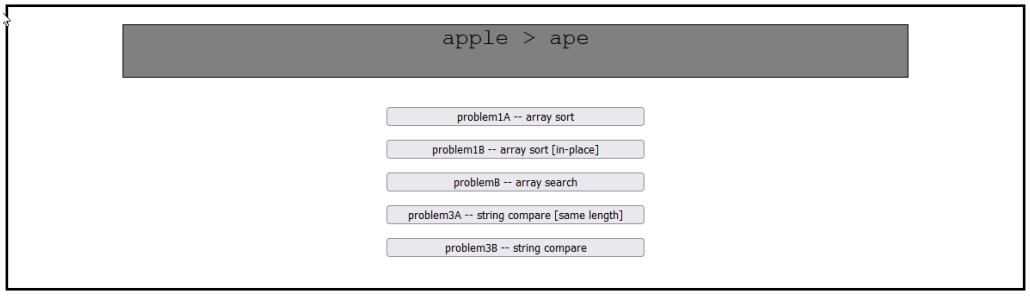
More examples,

First input string	Second input string	Expected output
"abacus"	"boat"	"abacus < boat"
"abacus"	"Boat"	"abacus > Boat"
"building"	"beat"	"building > beat"
"apples"	"abut"	"apple > abut"
"exam"	"examines"	"exam < examines"
"apples"	"apple"	"apples > apple"
"apple"	"apple"	"apple == apple"

- Develop flowchart for this algorithm,
  - Hint: first find the shorter length, and loop through only that range.
  - For accessing characters in string, you can use array notation *str[i]*
  - Should stop loop immediately when a mismatch is found. But you cannot say, "go out of loop". Use the approach mentioned in class and lab to stop a loop earlier. The loop should only have one exit point.
  - Save the flowchart as **img_03B.jpg**

- Implement this using JavaScript
  - Use *prompt* to ask users for two strings
  - For characters comparisons, just use < > ==, as numerical values
  - You should stop loop immediately when a mismatch is found. But cannot use *break*. Use the approach mentioned in class and use in labs to stop a loop earlier. The loop should only have one exit point.

Sample input /output



SUBMISSION

Submit the following files:

`team.txt` (only need to submit this file if you work in a team of two. List names, student numbers of both the team members)

```
img_{01A,01B,02,03A,03B}.jpg
A1.html
A1.css
A1.js
```

Compress the files (.zip or .tar or .gz), and then submit the (single) compressed file on eClass.

- **Note that if you work on a team, only on team member submit. another member does not need to submit anything.**