Searching & Sorting

Computer Science Principles

Learning objectives

1: The student can use computing tools and techniques to create artifacts

3: The student can use computing tools and techniques for creative expression.

16: The student can express an algorithm in a language.

21: The student can evaluate a program for correctness.

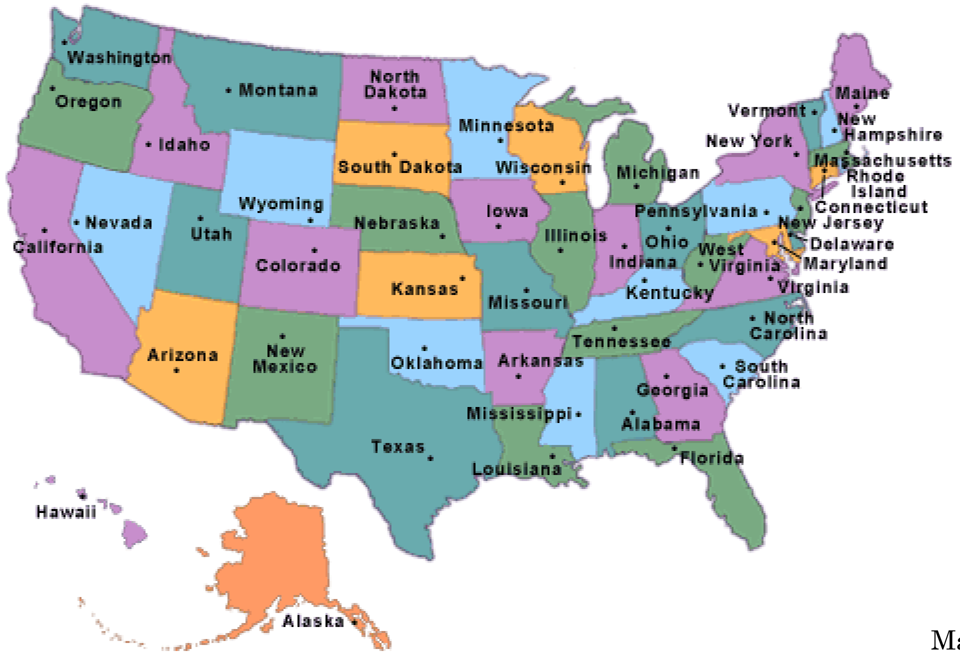
22: The student can develop a correct program.

23: The student can employ appropriate mathematical and logical concepts in programming.

More Algorithms

The state guessing game

- The Rules
  - I will choose a state
  - I will answer questions in such a way that my answers to your questions are true/false (yes/no)
  - You may have up to 5 guesses to guess the state.



Map from www.infoplease.com

Searches

- The strategy of trying to divide choices in half, then again and again until the answer is found is called a Binary Search.
  - Remember NoseGuy and the Guessing Game?

- We are going to look at strategies (or Algorithms) for searching for the "correct" answer.

Searching vs. Sorting

- When you search a list, it is to determine if a specified element is present.
  - There are two primary searching algorithms
    1. Linear Search
    2. Binary Search

- Sorting is done to order the values in the list based upon some key value.
  - There are three primary sorting algorithms
    1. Selection Sort
    2. Insertion Sort
    3. Merge Sort

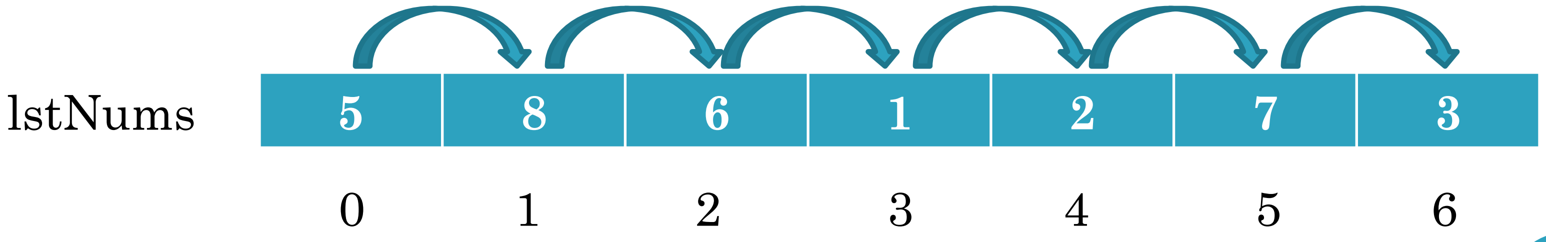
Searching Algorithms

Where is that element?

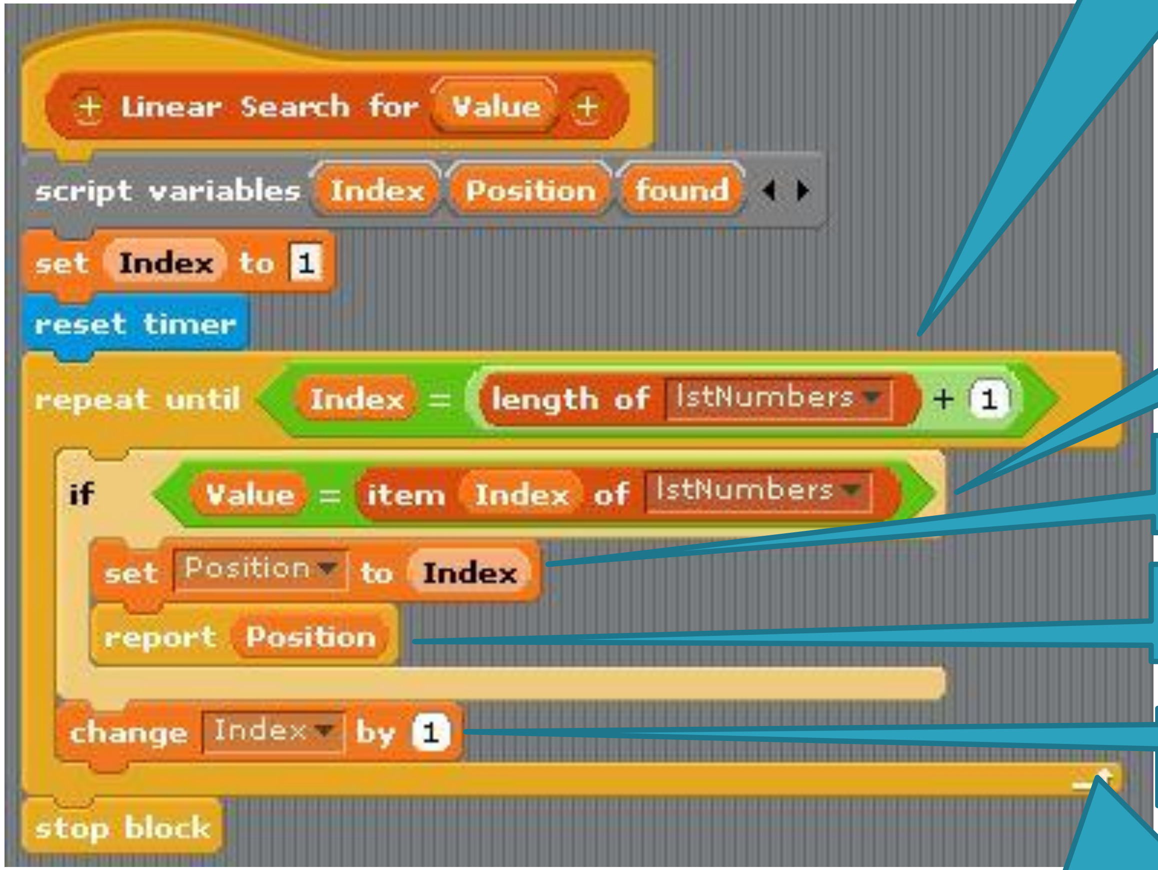
LINEAR SEARCH

- A linear search algorithm searches the list in a sequential manner.

- The algorithm moves through the list, comparing the key value with the values of the elements. If it does not find the key value, it simply moves to the next element.



LINEAR SEARCH



```
+ Linear Search for Value +
script variables Index Position found ◂ ▸
set Index to 1
reset timer
repeat until < Index = ( length of lstNumbers + 1 ) >
    if < Value = item Index of lstNumbers >
        set Position to Index
        report Position
    change Index by 1
stop block
```

Move through the list

Check to see if the value is equal to the value in the "Index" position of the list

If Yes, then set Position to the Index number

Report Position

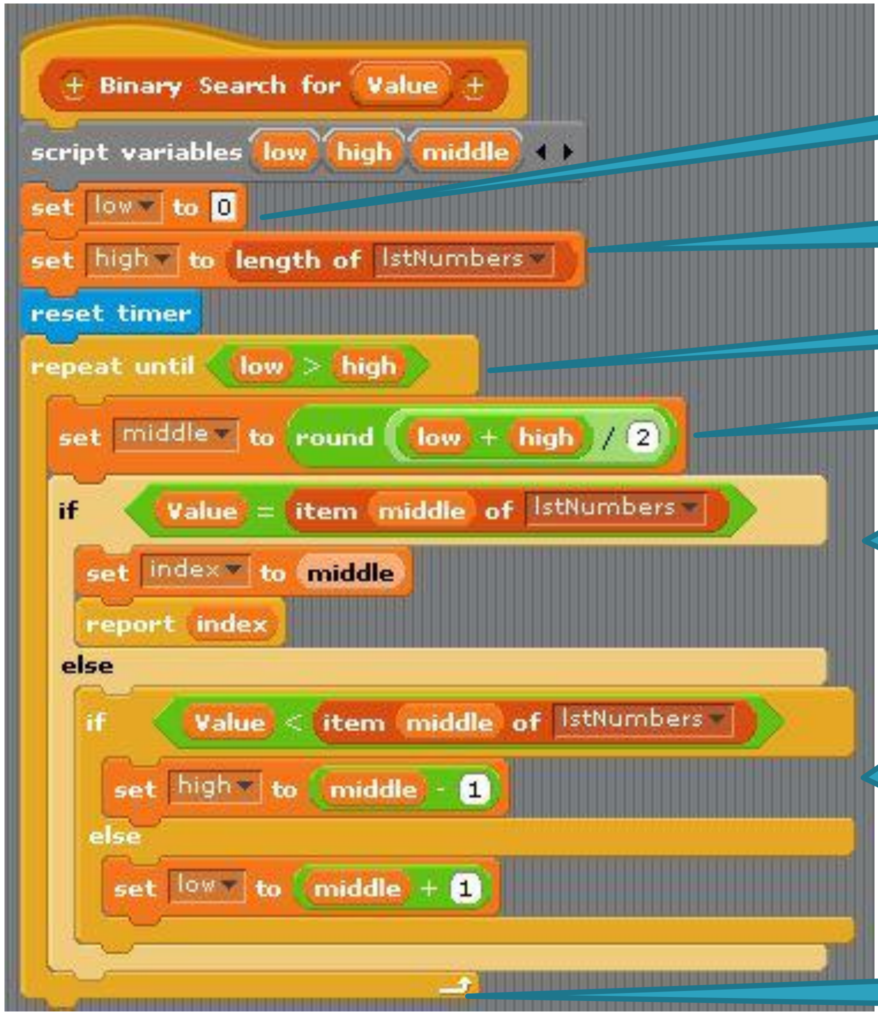
Add 1 to my Index variable

Loop again to heck the next value in the loop at the new Index position for my answer

Binary Search

- The binary search algorithm is more efficient than the linear search algorithm.
- Binary search requires that the array be <u>sorted first.</u>
- The algorithm splits the array and checks the middle value.
  - If it is not found it compares the values.
  - If the search value is higher than the middle value, the algorithm moves to the upper half (now a subarray). (Lower – it moves to the lower half.
  - It splits the subarray in half, checks the middle for a match.
  - If not found, it checks to see if it is higher/lower and moves to appropriate subarray.
  - This continues until it has no more values or finds a match.

Binary Search



```
+ Binary Search for (Value) +
script variables (low) (high) (middle)
set low to 0
set high to length of lstNumbers
reset timer
repeat until <(low) > (high)>
    set middle to round ((low) + (high)) / 2
    if <(Value) = item (middle) of lstNumbers>
        set index to (middle)
        report (index)
    else
        if <(Value) < item (middle) of lstNumbers>
            set high to (middle) - 1
        else
            set low to (middle) + 1
```

Set the low to 0 to start

Set high to the length of the list for the max index

When low is greater than high, I can stop

Calculate the middle index value

Check to see if the value is located at the middle position in the list
If yes, then set the index to the middle value and report it.

If No, then if the target value is less than the value in the middle of the list set “high” to middle – 1
otherwise set “low” to middle +1

Repeat the process

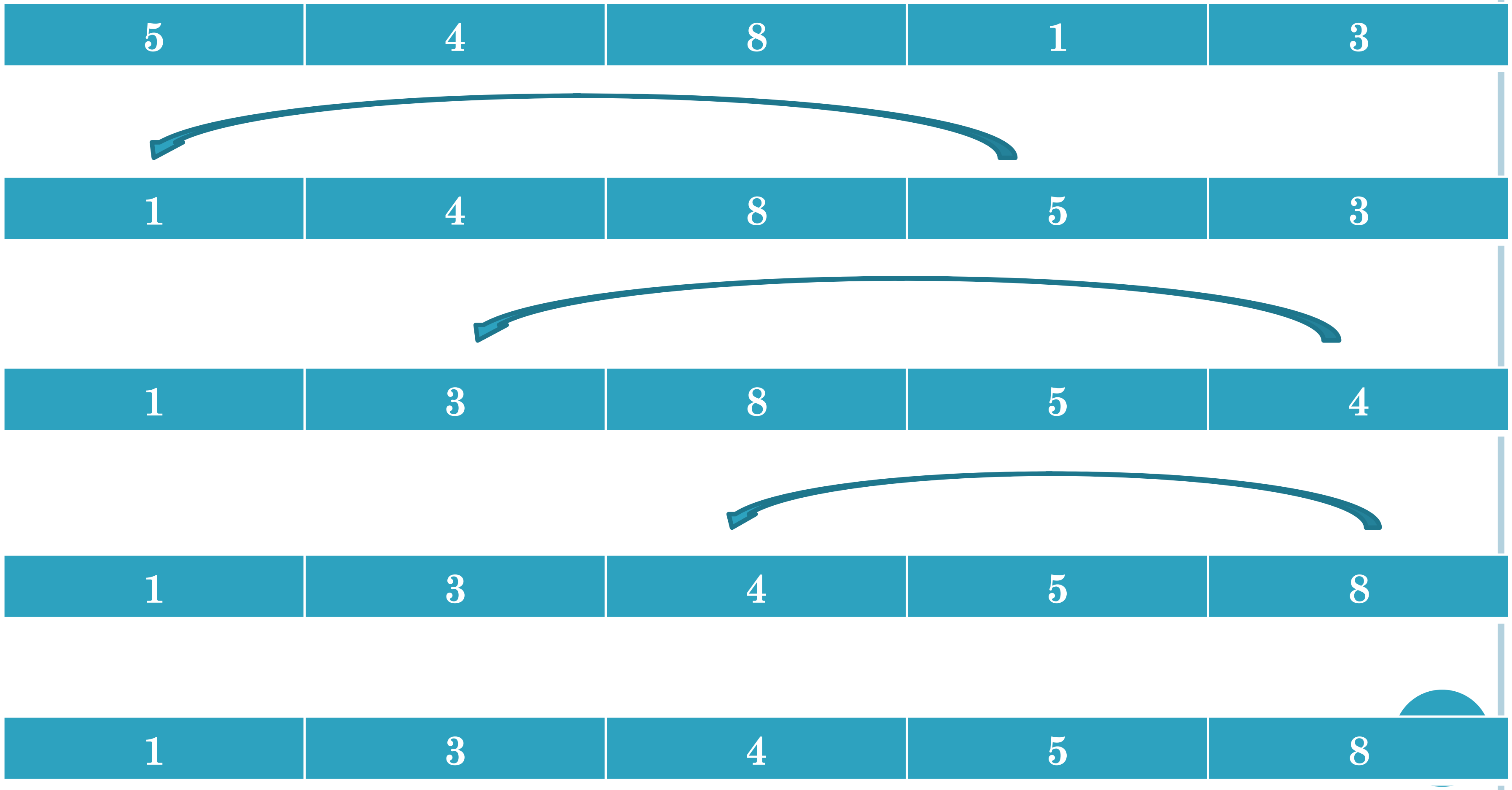
Sorting Algorithms

C# Programming

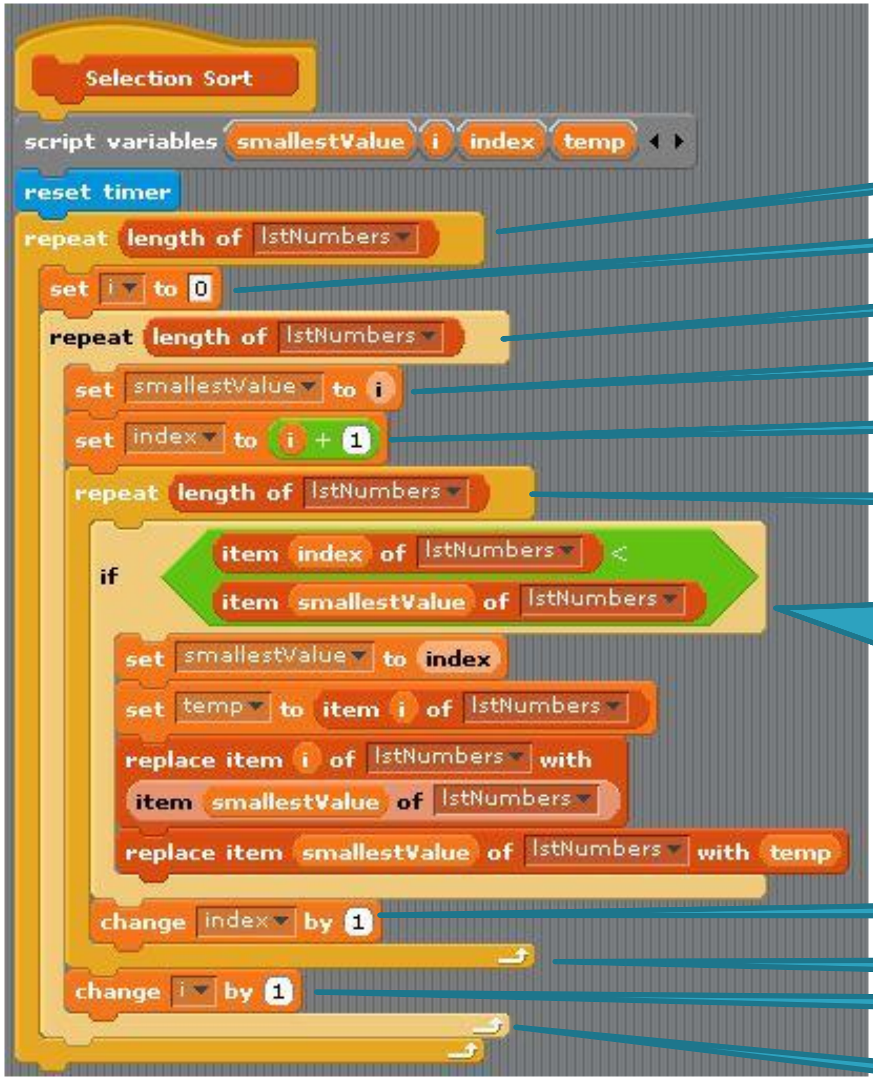
Selection Sort

- This is a simple sorting algorithm.
- It moves through the array looking for the lowest value, then moves it to the front.
- The second pass through the array, it looks for the second lowest and moves it to the second position.
- It keeps passing through the array until the last iteration.

Selection Sort



Selection Sort



```
Selection Sort
script variables smallestValue i index temp
reset timer
repeat length of lstNumbers
    set i to 0
    repeat length of lstNumbers
        set smallestValue to i
        set index to i + 1
        repeat length of lstNumbers
            if item index of lstNumbers < item smallestValue of lstNumbers
                set smallestValue to index
                set temp to item i of lstNumbers
                replace item i of lstNumbers with
                item smallestValue of lstNumbers
                replace item smallestValue of lstNumbers with temp
            change index by 1
        change i by 1
```

Loop through the List

Set i to 0

Loop through the List

Sets the smallestValue to i

Sets Index to i + 1

Loop through the List

Checks to see if the value at the Index position is smaller than the value at the “smallestValue” position.
If yes, smallestValue is reset to index, temp is set to the value at i position of the list (to hold it) and the values are swapped.

Index is changed by 1

The inner loop is repeated

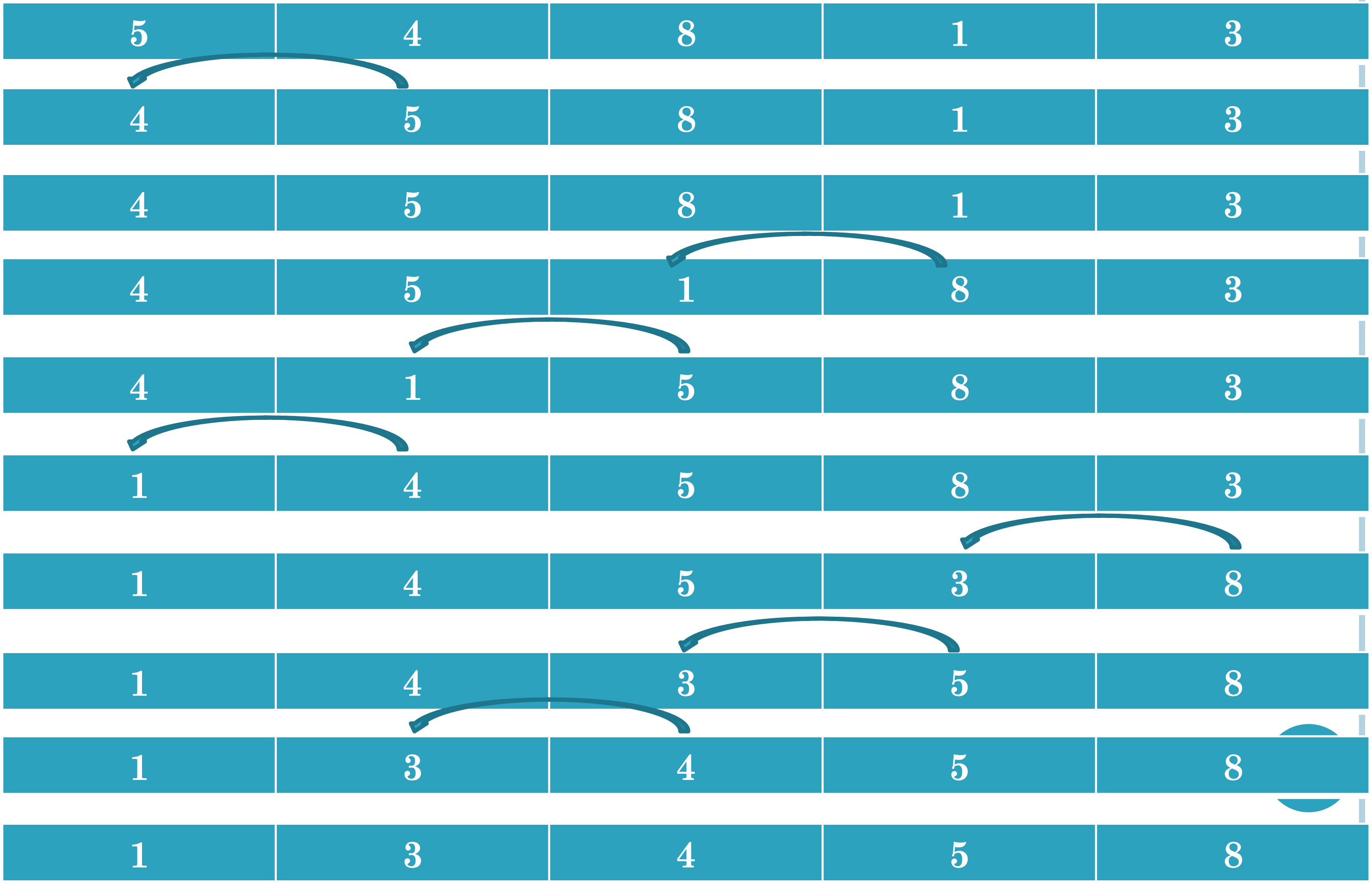
i is changed by 1

The next loop is repeated

Insertion Sort

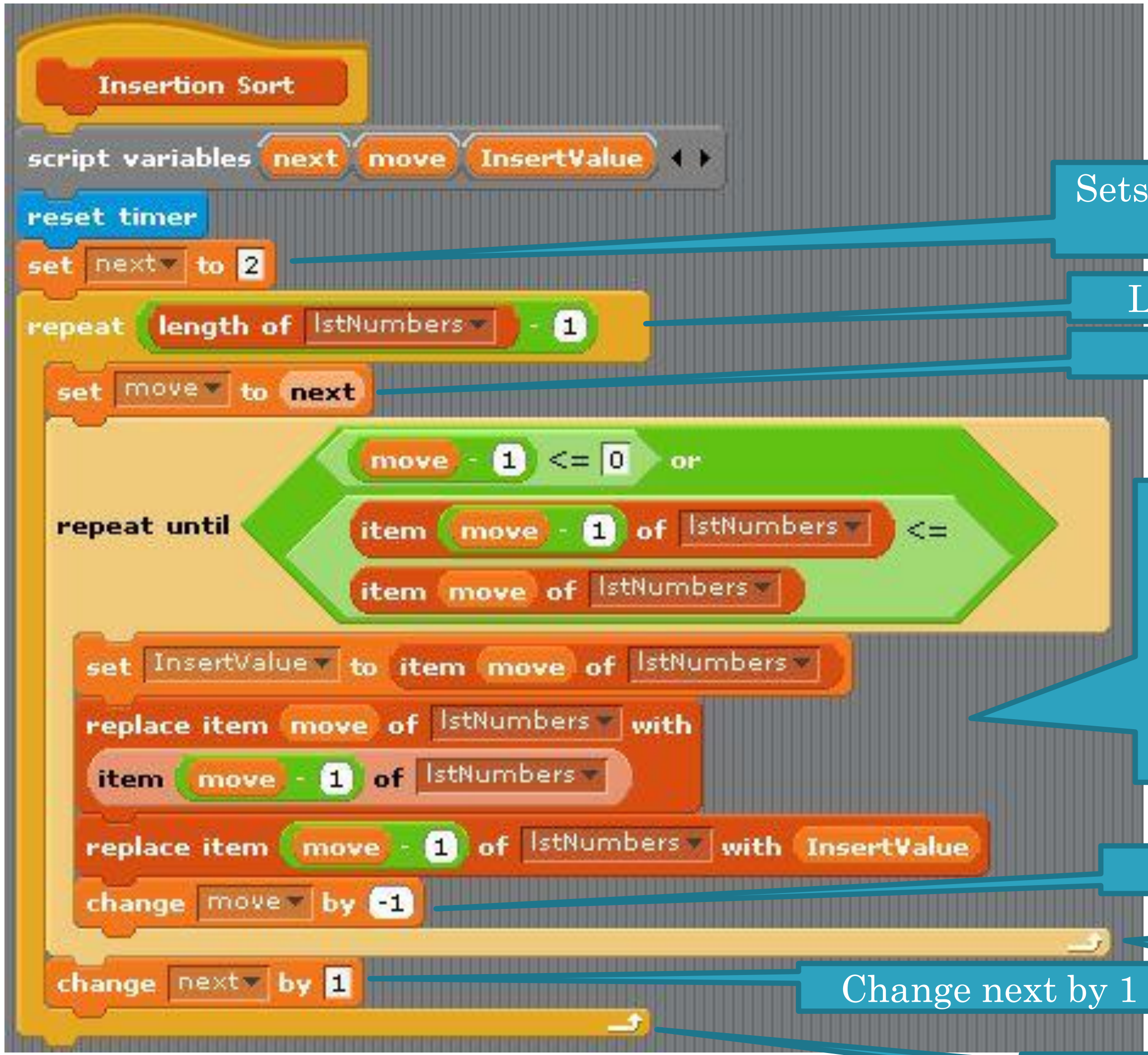
- Another simple sorting algorithm.
- 1st Iteration – Compares element 1 & 2 – Swaps if element 1 > element 2.
- 2nd Iteration – Looks at element 3 – Inserts it in position given element 1 & 2.
- It keeps comparing and inserting until the end.

Insertion Sort



5	4	8	1	3
4	5	8	1	3
4	5	8	1	3
4	5	1	8	3
4	1	5	8	3
1	4	5	8	3
1	4	5	3	8
1	4	3	5	8
1	3	4	5	8
1	3	4	5	8

Insertion Sort



```
Insertion Sort
script variables next move InsertValue
reset timer
set next to 2
repeat (length of lstNumbers - 1)
    set move to next
    repeat until <<(move - 1) <= 0> or <(item (move - 1) of lstNumbers) <= (item move of lstNumbers)>>
        set InsertValue to item move of lstNumbers
        replace item move of lstNumbers with
            item (move - 1) of lstNumbers
        replace item (move - 1) of lstNumbers with InsertValue
        change move by -1
    change next by 1
```

Sets a variable to 2 for the second index position in the List

Loops through the List

Sets move to next

Repeats while move -1 is greater than 0 or the value at move -1 is greater than the value at move.
Hold the value to insert.
Swap the values.

Change next by -1

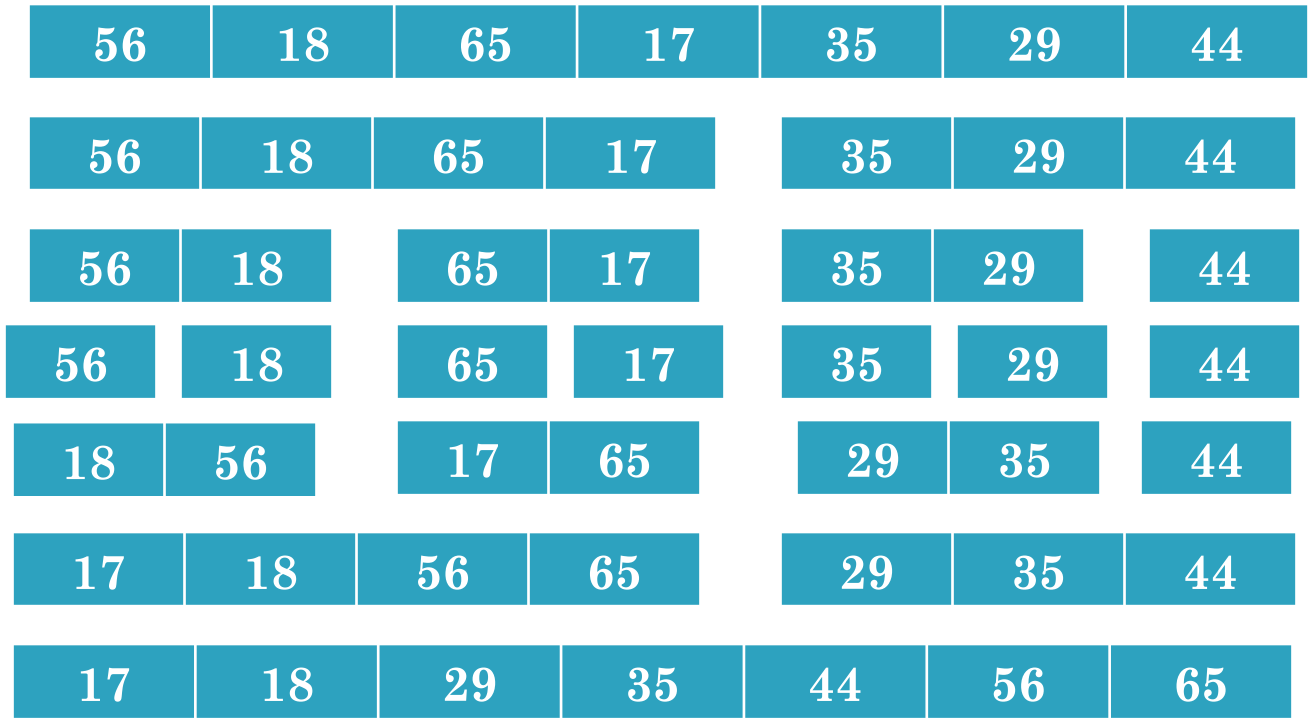
Repeat based upon conditions

Change next by 1

Loop until at end of list

Merge Sort

- The merge sort algorithm sorts by splitting the array into two subgroups, sorting the subgroups, then merging them back together sorted.



Measuring Efficiency

- How efficiency an algorithm is can be measured using Big-O notation. (Also called Landau's symbol)
- It tells you how fast a function grows or declines.
- Big-O notation measures the worst-case runtime for an algorithm.

Efficiency of searching

- Linear Search
  - Worst Case time O(N) - It goes through entire list
- Binary Search
  - Worst Case time O(Log N) – The number of times N can be divided in half before there is nothing left
  - This is better than the linear search.
- Selection Sort & Insertion Sort
  - Worst Case time O(N^2)
- Merge Sort
  - Worst Case time O(N log N)
    - This is a little better than the selection and insertion sorts.