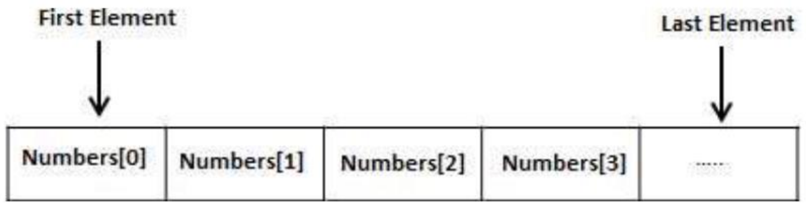
Advanced Algorithm

Before start any advanced algorithm we need to mansion a new concept of representing data that help
us when we try to solve advanced problems

An array is a data structure, which can store a fixed-size collection of elements of the same data type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number1, number2, ..., number100, you just declare one array variable **number** of integer type and use number1[0], number1[1], and ..., number1[99] to represent individual variables. Here, 0, 1, 2, ...99 are **index** associated with **var** variable and they are being used to represent individual elements available in the array.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Introduction to Searching Algorithms

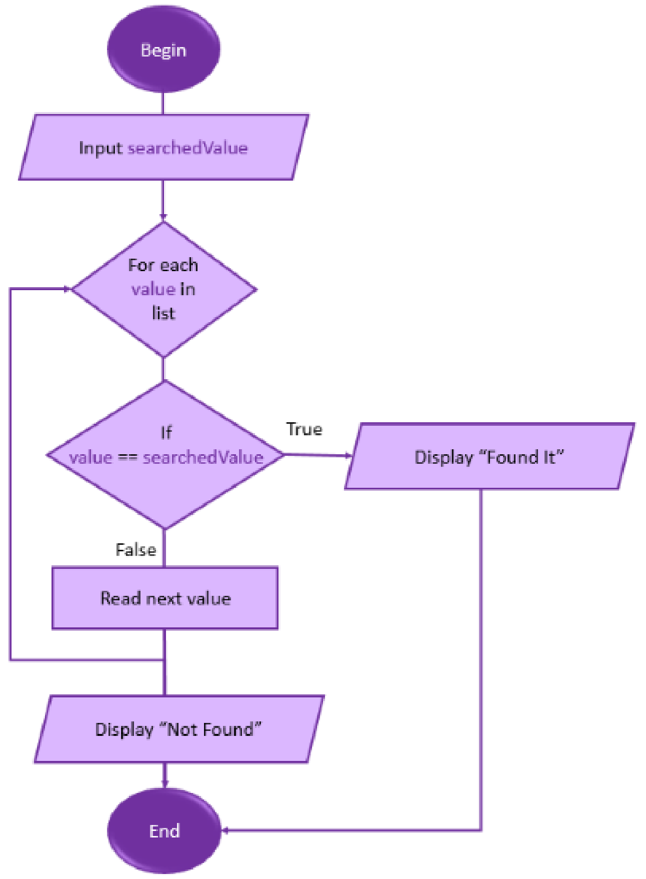
Finding a specific member of an **array** means **searching** the **array** until the member is found. ... It compares each element with the value being searched for, and stops when either the value is found or the end of the **array** is encountered.

Linear Search
Binary Search
Jump Search
Interpolation Search
Exponential Search

Linear Search

A simple algorithm is to do a linear search:

- Start from the leftmost element of data and one by one compare the target value with each element of data
- If target value matches with an element, return the position or "found".
- If x doesn't match with any of elements, return -1 or "not found".



```
for each item in the list
    if match item == value
      pos=  item's location
      found = true
      exit loop
    end if
  end for
If found==true
     item found at position pos
Else
     item not found
endif
```

Binary Search

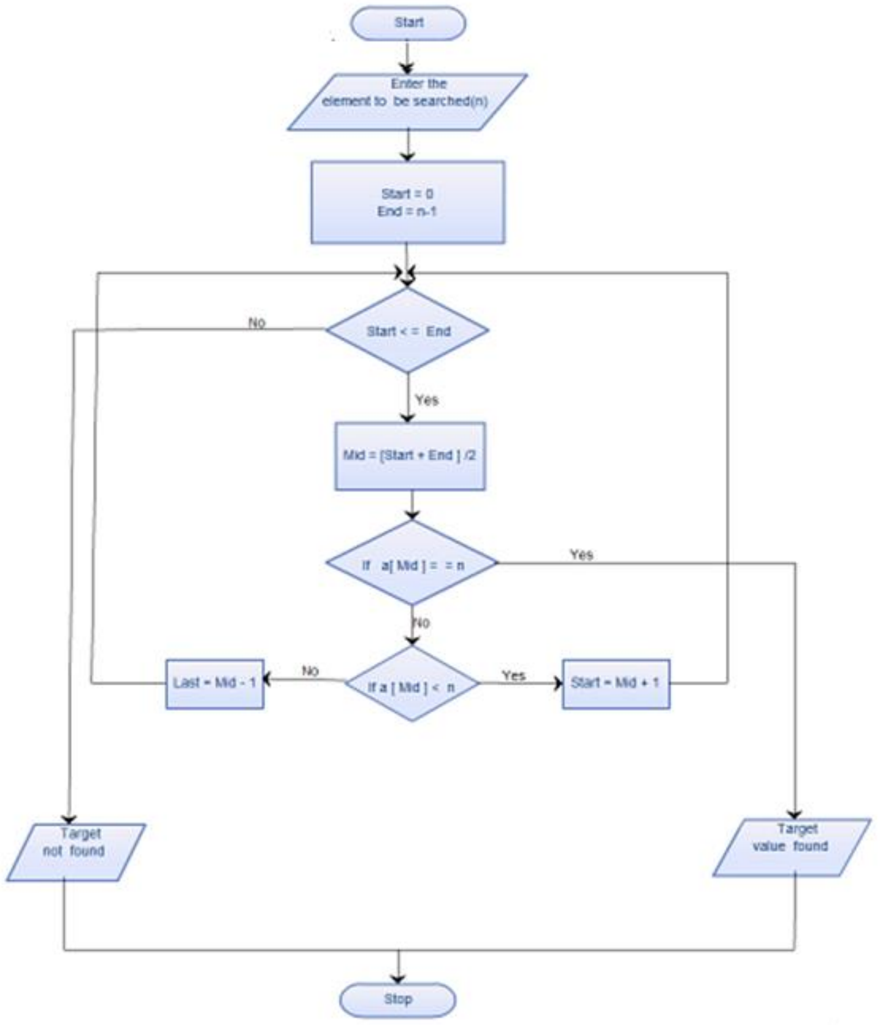
Binary search is a fast search algorithm with run-time complexity of O(log n).

This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.

Begin with an interval covering the whole data.

If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.

Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.



```
Set lower = 1
Set upper = n

while x not found
  if upper < lower
    EXIT loop : x does not exists.
  set mid = (lower + upper ) / 2

  if A[mid] < x
    set lower = mid + 1

  if A[mid] > x
    set upper = mid - 1

  if A[mid] = x
    EXIT: x found at location mid
end while
```

Introduction to Sorting Algorithms

A **sorted array** is an **array** data structure in which each element is **sorted** in numerical, alphabetical, or some other order, and placed at equally spaced addresses in computer memory. It is typically used in computer science to implement static lookup tables to hold multiple values which have the same data type.

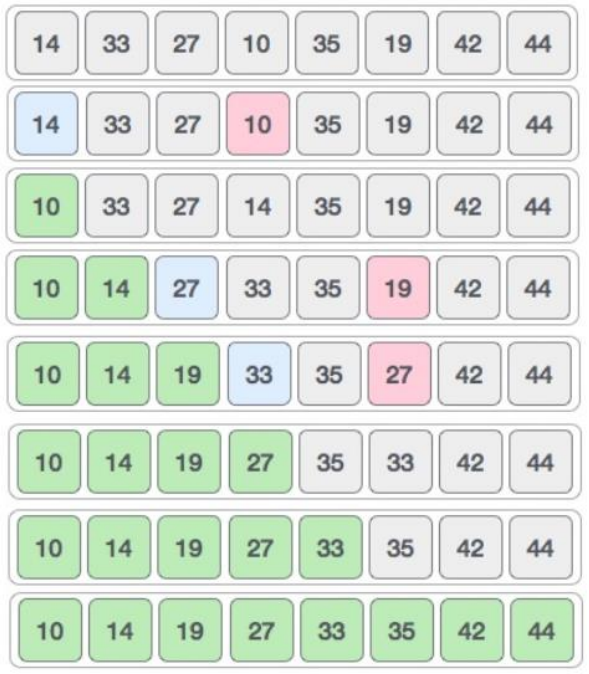
Selection Sort.
Bubble Sort.
Insertion Sort.
Merge Sort.
Quick Sort.
Heap Sort.

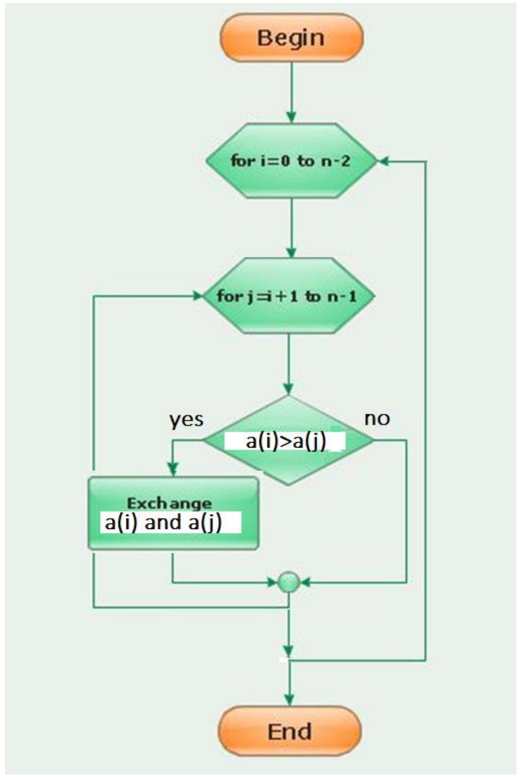
Selection Sort

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array.

This process continues moving unsorted array boundary by one element to the right.

This algorithm is not suitable for large data sets as its average and worst case complexities are of $O(n^2)$, where n is the number of items.



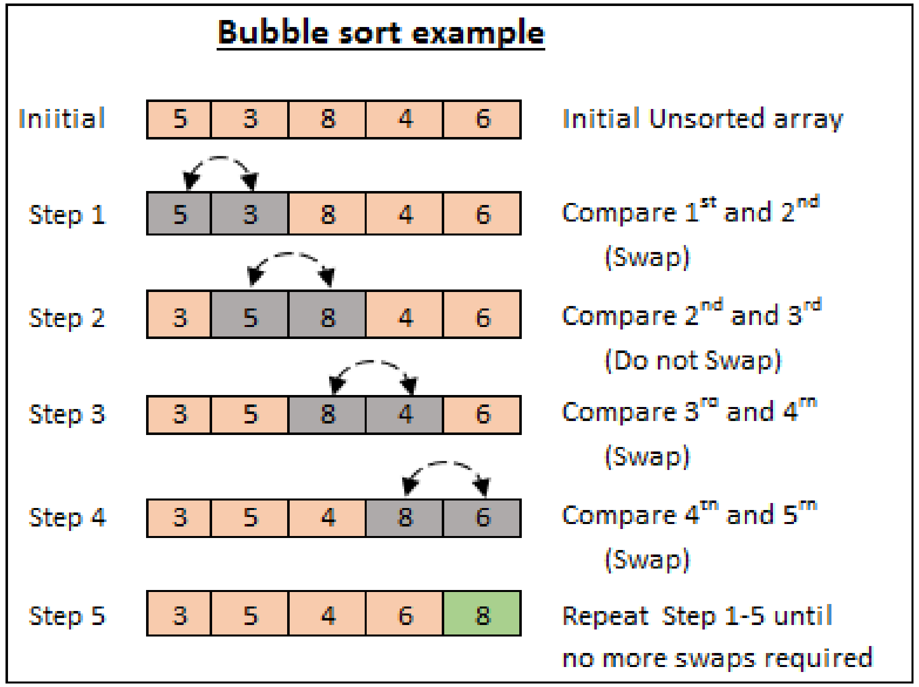


1.Find the smallest card. Swap it with the first card.
2.Find the second-smallest card. Swap it with the second card.
3.Find the third-smallest card. Swap it with the third card.
4.Repeat finding the next-smallest card, and swapping it into the correct position until the array is sorted.

Bubble Sort

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of ***adjacent*** elements is compared and the elements are swapped if they are not in order.

This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n^2) where **n** is the number of items.


Bubble sort example
Iniitial 5 3 8 4 6 Initial Unsorted array
Step 1 5 3 8 4 6 Compare 1st and 2nd (Swap)
Step 2 3 5 8 4 6 Compare 2nd and 3rd (Do not Swap)
Step 3 3 5 8 4 6 Compare 3rd and 4th (Swap)
Step 4 3 5 4 8 6 Compare 4th and 5th (Swap)
Step 5 3 5 4 6 8 Repeat Step 1-5 until no more swaps required

Initial Unsorted array

Compare 1st and 2nd
(Swap)

Compare 2nd and 3rd
(Do not Swap)

Compare 3rd and 4th
(Swap)

Compare 4th and 5th
(Swap)

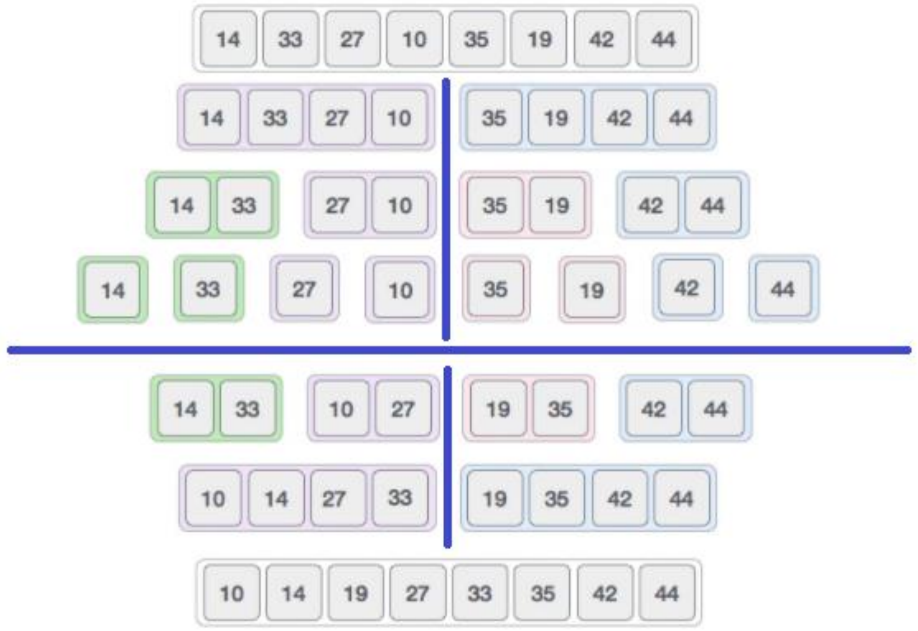
Repeat Step 1-5 until
no more swaps required

Merage sort

Merge algorithms are a family of algorithms that take multiple sorted lists as input and produce a single list as output, containing all the elements of the inputs lists in sorted order.

The ***MergeSort*** function repeatedly divides the array into two halves until we reach a stage where we try to perform MergeSort on a subarray of size 1.

After that, the merge function comes into play and combines the sorted arrays into larger arrays until the whole array is merged.



Step 1 − if it is only one element in the list it is already sorted, return.
Step 2 − divide the list recursively into two halves until it can no more be divided.
Step 3 − merge the smaller lists into new list in sorted order.

Asymptotic Analysis
big (O , θ, Ω) notation

In computer science, ***Asymptotic Analysis*** (big O notation) is used to classify algorithms according to how their run ***time*** (time Complexity) or ***space*** (Space Complexity) requirements grow as the input size grows.

Time complexity

	Best cases (Ω)	Average cases (θ)	Worst cases (O)
Sequential search	**O(1)**	**O((n)/2)**	O(n)
Binary search	O(1)	O(log(n))	O(log(n))
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Merge Sort	O(n log(n))	O(n log(n))	O(n log(n))