Slot 01 - Sorting/Searching Algorithms

Presenter:

Dr. LE Thanh Tung

Content

1. Reviews
2. Searching Algorithms
3. Sorting Algorithms

Reviews

Exercise 1: Write a program to find the biggest prime in a list of integer numbers

Return 0, if there is no prime number in the list.

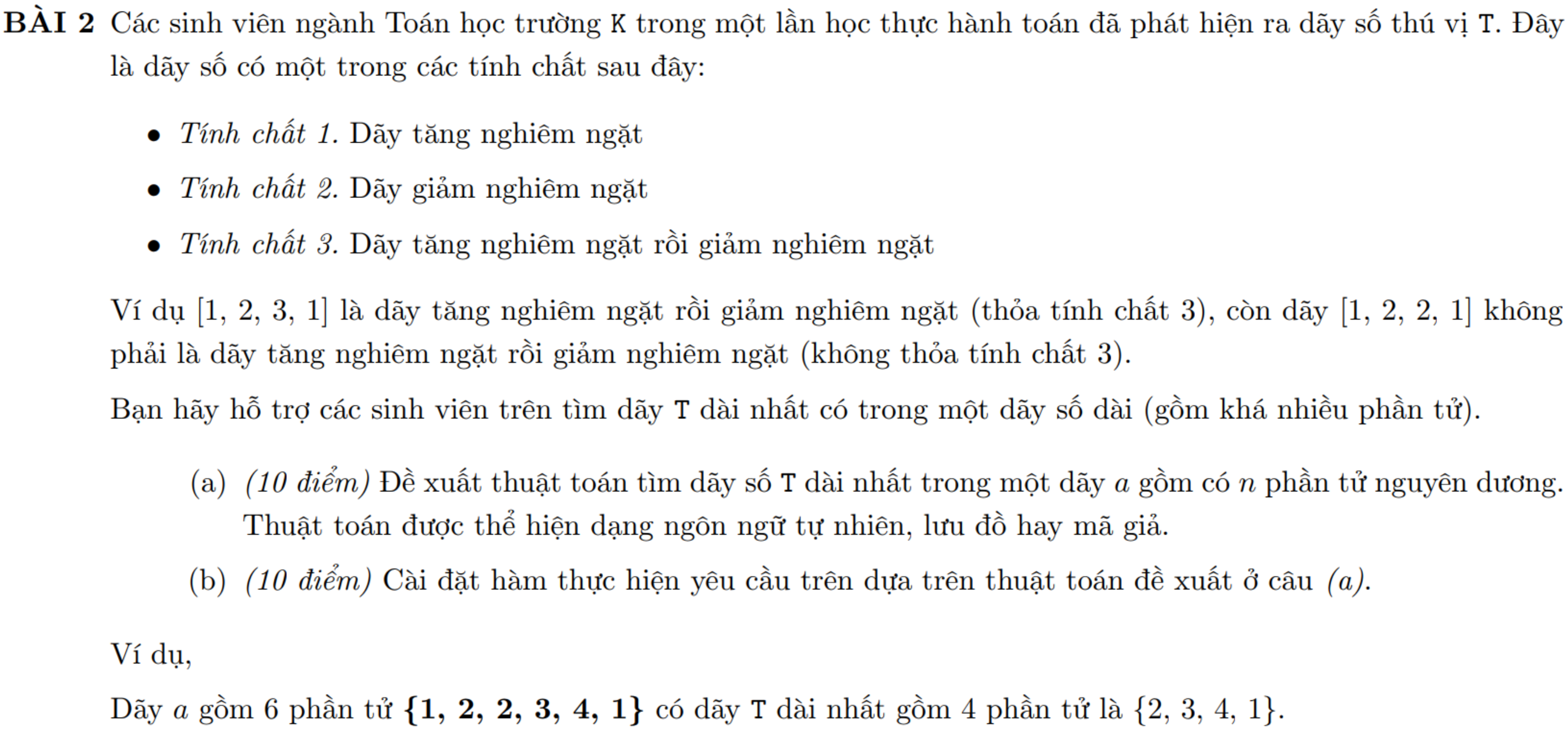
Exercise 2: Write a program to find the longest symmetric sub-array in a list of integer numbers

Known that a symmetric array A with n elements satisfies the following characteristic:

$$A_i = A_{n-i-1}$$

Where i and (n-i-1) is the index of elements in A starting from 0

Exercise 3: Problem 2 in the Final Exam of CSC10001



BÀI 2 Các sinh viên ngành Toán học trường K trong một lần học thực hành toán đã phát hiện ra dãy số thú vị T. Đây là dãy số có một trong các tính chất sau đây:

- *Tính chất 1.* Dãy tăng nghiêm ngặt
- *Tính chất 2.* Dãy giảm nghiêm ngặt
- *Tính chất 3.* Dãy tăng nghiêm ngặt rồi giảm nghiêm ngặt

Ví dụ [1, 2, 3, 1] là dãy tăng nghiêm ngặt rồi giảm nghiêm ngặt (thỏa tính chất 3), còn dãy [1, 2, 2, 1] không phải là dãy tăng nghiêm ngặt rồi giảm nghiêm ngặt (không thỏa tính chất 3).

Bạn hãy hỗ trợ các sinh viên trên tìm dãy T dài nhất có trong một dãy số dài (gồm khá nhiều phần tử).

(a) *(10 điểm)* Đề xuất thuật toán tìm dãy số T dài nhất trong một dãy a gồm có n phần tử nguyên dương. Thuật toán được thể hiện dạng ngôn ngữ tự nhiên, lưu đồ hay mã giả.

(b) *(10 điểm)* Cài đặt hàm thực hiện yêu cầu trên dựa trên thuật toán đề xuất ở câu *(a)*.

Ví dụ,

Dãy a gồm 6 phần tử $\mathbf{\{1,\ 2,\ 2,\ 3,\ 4,\ 1\}}$ có dãy T dài nhất gồm 4 phần tử là $\{2,\ 3,\ 4,\ 1\}$.

Searching

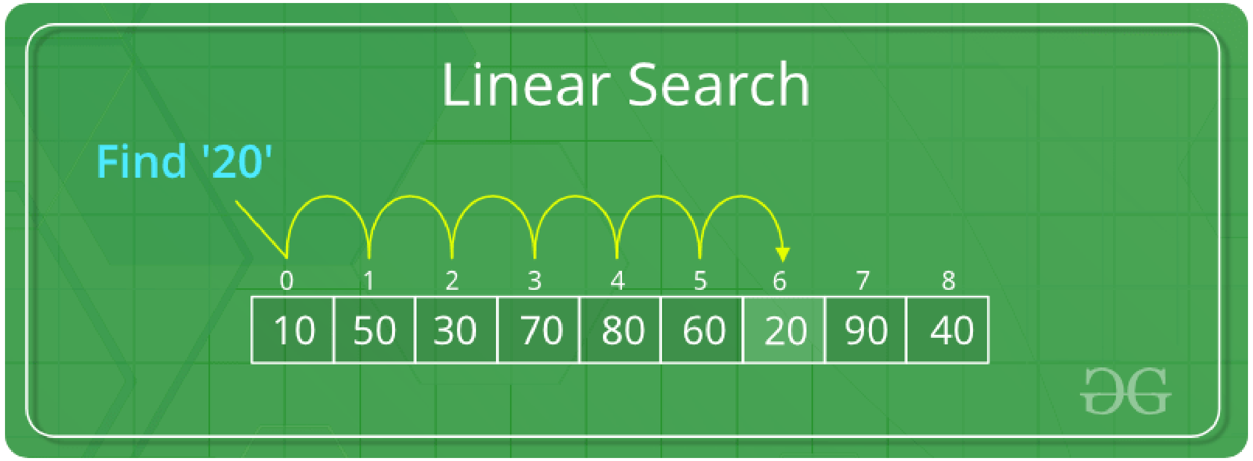
Searching algorithms are generally classified into two categories:

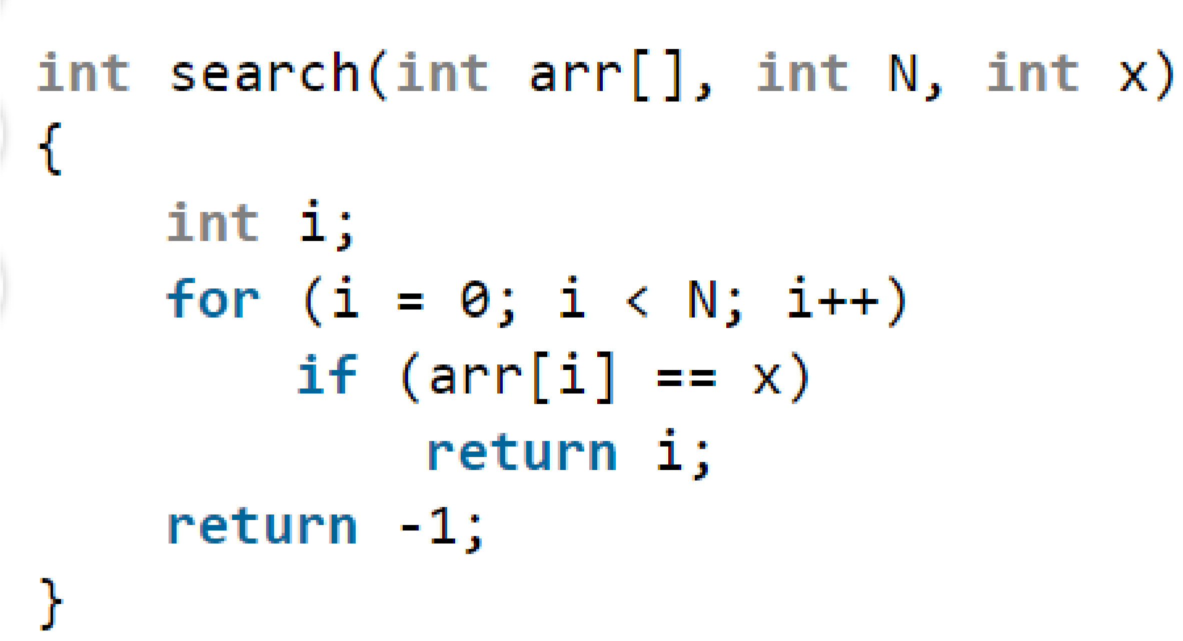
- **Sequential Search**: In this, the list or array is traversed sequentially and every element is checked

- **Interval Search**: These algorithms are specifically designed for searching in sorted data-structures

Searching

Linear Search:

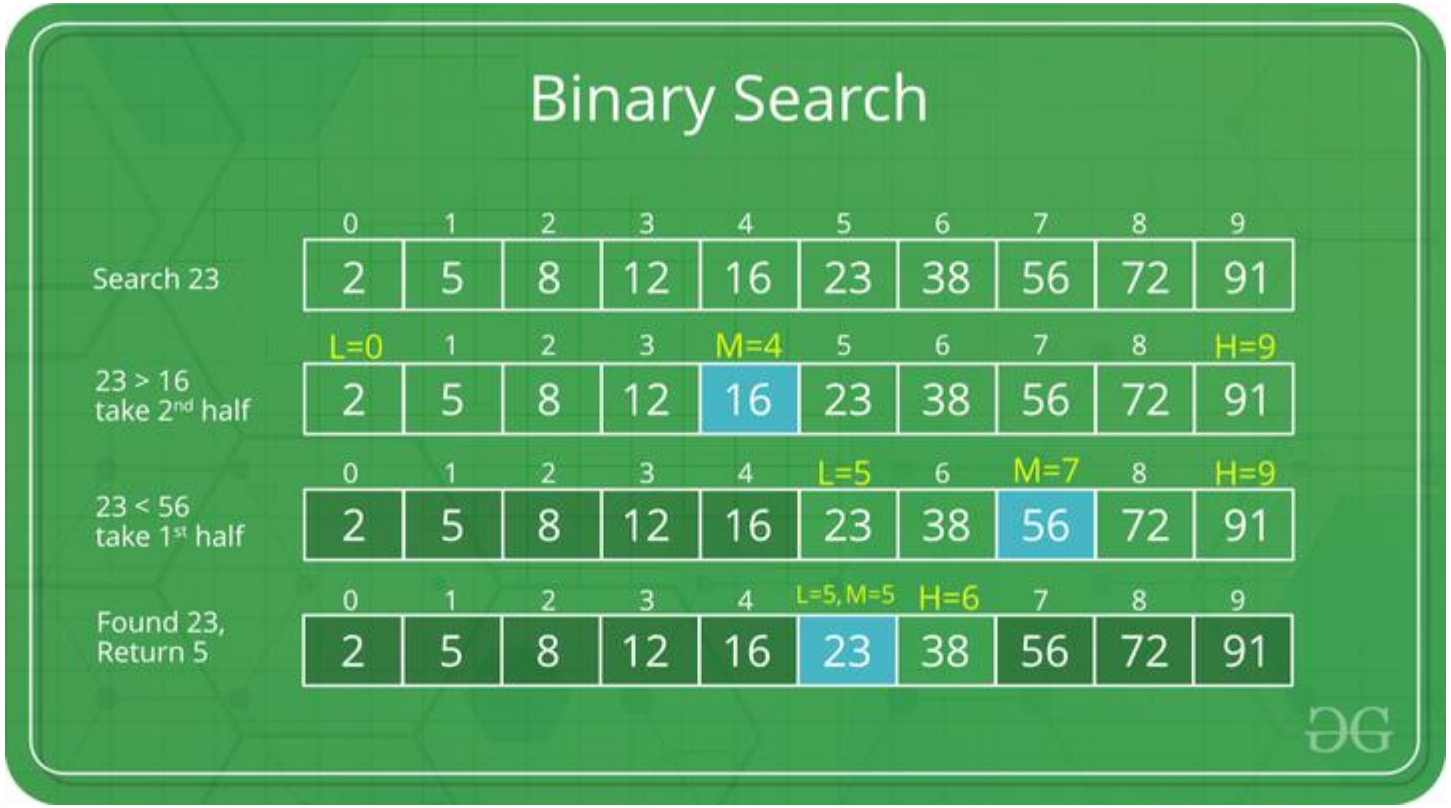




```
int search(int arr[], int N, int x)
{
    int i;
    for (i = 0; i < N; i++)
        if (arr[i] == x)
            return i;
    return -1;
}
```

Searching

Binary Search: Write a function to illustrate the binary search



Binary Search

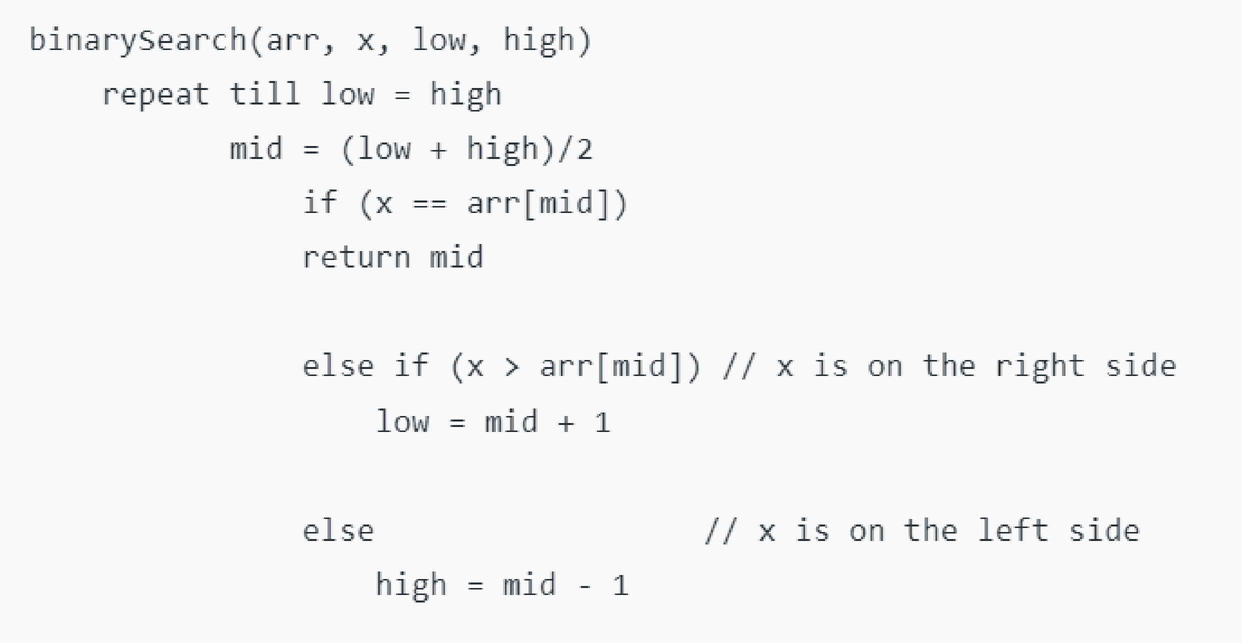
	0	1	2	3	4	5	6	7	8	9
Search 23	2	5	8	12	16	23	38	56	72	91

	L=0	1	2	3	M=4	5	6	7	8	H=9
23 > 16 take 2nd half	2	5	8	12	16	23	38	56	72	91

	0	1	2	3	4	L=5	6	M=7	8	H=9
23 < 56 take 1st half	2	5	8	12	16	23	38	56	72	91

	0	1	2	3	4	L=5, M=5	H=6	7	8	9
Found 23, Return 5	2	5	8	12	16	23	38	56	72	91

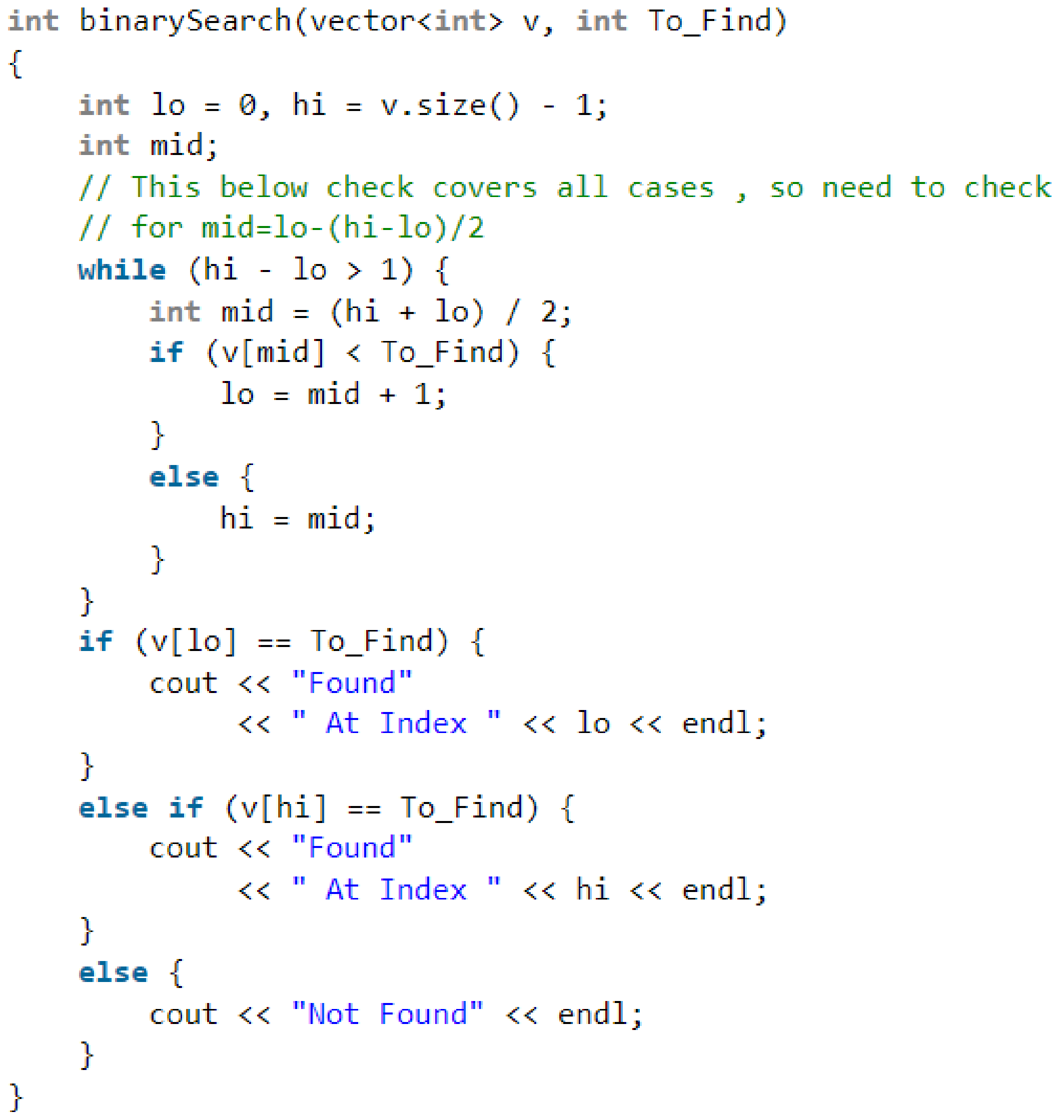
Iterative Approach to Binary Search



```
binarySearch(arr, x, low, high)
    repeat till low = high
           mid = (low + high)/2
               if (x == arr[mid])
               return mid

               else if (x > arr[mid]) // x is on the right side
                   low = mid + 1

               else                  // x is on the left side
                   high = mid - 1
```



```cpp
int binarySearch(vector<int> v, int To_Find)
{
    int lo = 0, hi = v.size() - 1;
    int mid;
    // This below check covers all cases , so need to check
    // for mid=lo-(hi-lo)/2
    while (hi - lo > 1) {
        int mid = (hi + lo) / 2;
        if (v[mid] < To_Find) {
            lo = mid + 1;
        }
        else {
            hi = mid;
        }
    }
    if (v[lo] == To_Find) {
        cout << "Found"
             << " At Index " << lo << endl;
    }
    else if (v[hi] == To_Find) {
        cout << "Found"
             << " At Index " << hi << endl;
    }
    else {
        cout << "Not Found" << endl;
    }
}
```

Searching

- **Question 1**: What are the advantages of binary search against linear search?

- **Question 2**: Why do we need binary search whereas it requires a sorted series of numbers?

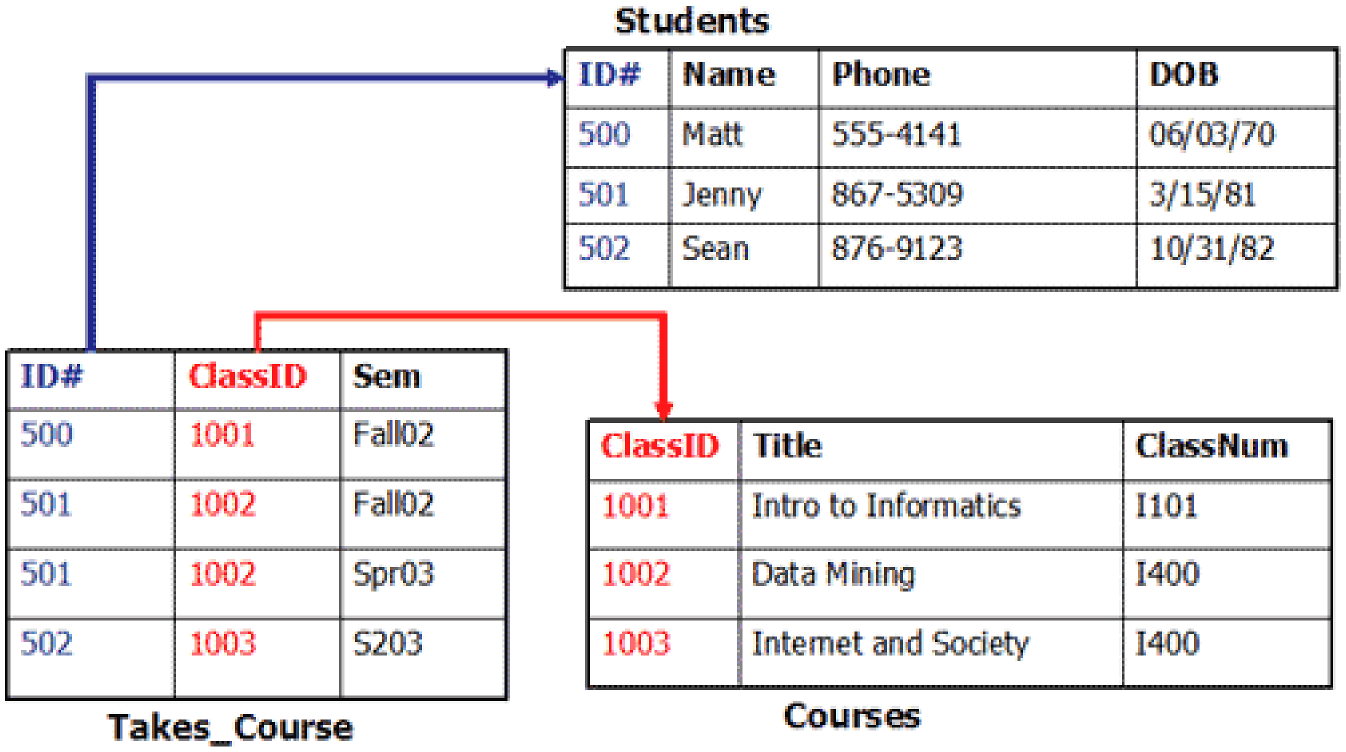
Searching

- **Question 1**: What are the advantages of binary search against linear search?

	Linear Search	Binary Search
Pre-condition	Random order	sorted
Speed	Low	Fast
Manner	Sequential	Divide-and-conquer
Dimensions	Single/Multidimensional	Single
Size	Preferred for small size	Preferred for large size
Time complexity	O(n)	$O(\log_n)$

Searching

- **Question 2**: Why do we need binary search whereas it requires a sorted series of numbers?



Students

ID#	Name	Phone	DOB
500	Matt	555-4141	06/03/70
501	Jenny	867-5309	3/15/81
502	Sean	876-9123	10/31/82

ID#	ClassID	Sem
500	1001	Fall02
501	1002	Fall02
501	1002	Spr03
502	1003	S203

Takes_Course

ClassID	Title	ClassNum
1001	Intro to Informatics	I101
1002	Data Mining	I400
1003	Internet and Society	I400

Courses

Sorting

- In this section, we will introduce 3 basic sorting algorithms:
  - Selection sort
  - Bubble sort
  - Insertion sort

Sorting

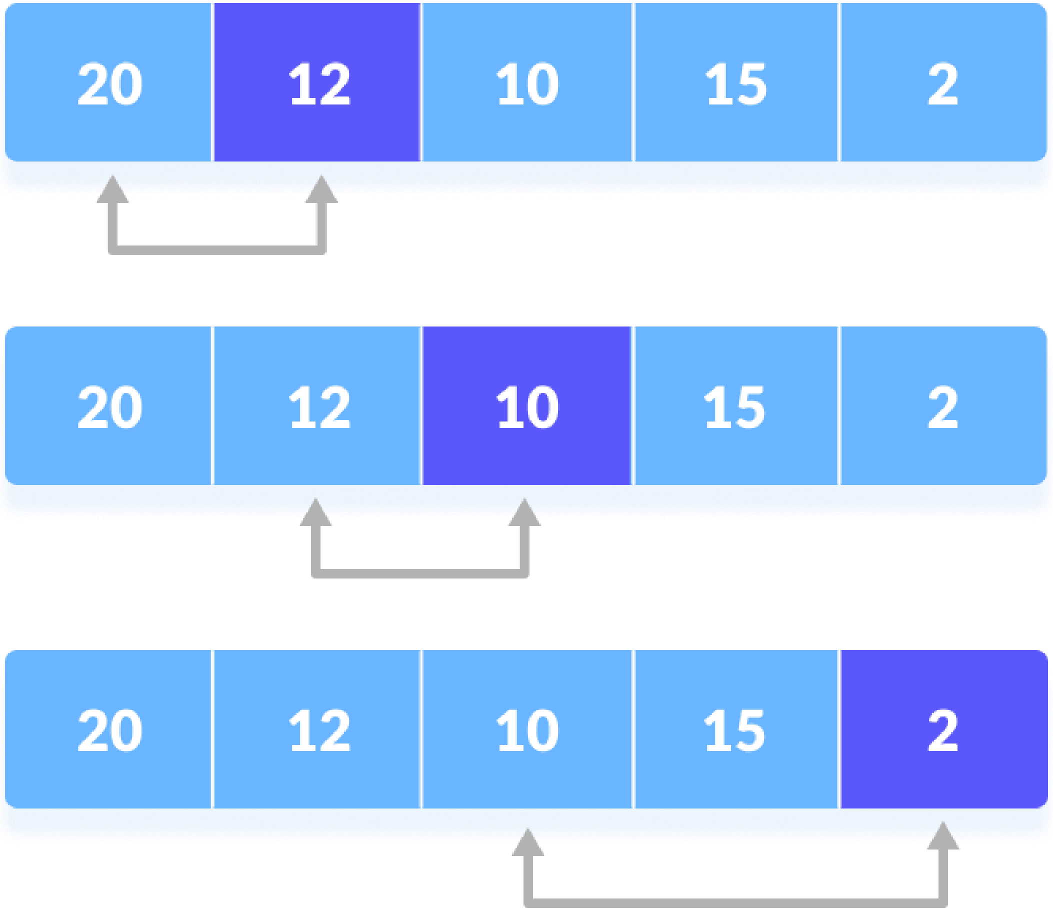
- In this section, we will introduce 4 basic sorting algorithms:
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Interchange sort

Selection Sort

- Main idea: repeatedly doing the following procedure:
  - finding the minimum element (considering ascending order) from unsorted part
  - putting it at the beginning

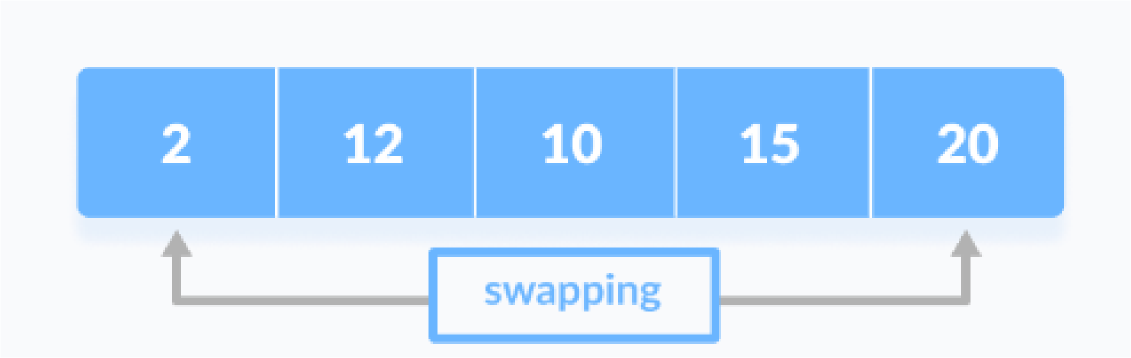
Selection Sort

- Step 1: find the minimum value of the list

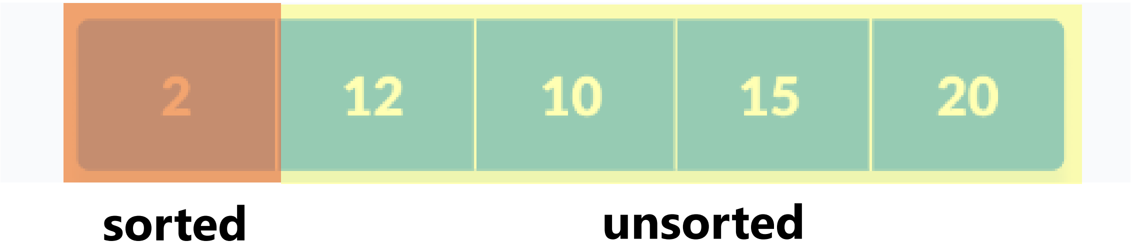


Selection Sort

- Step 2: min val is placed in the front of the unsorted list



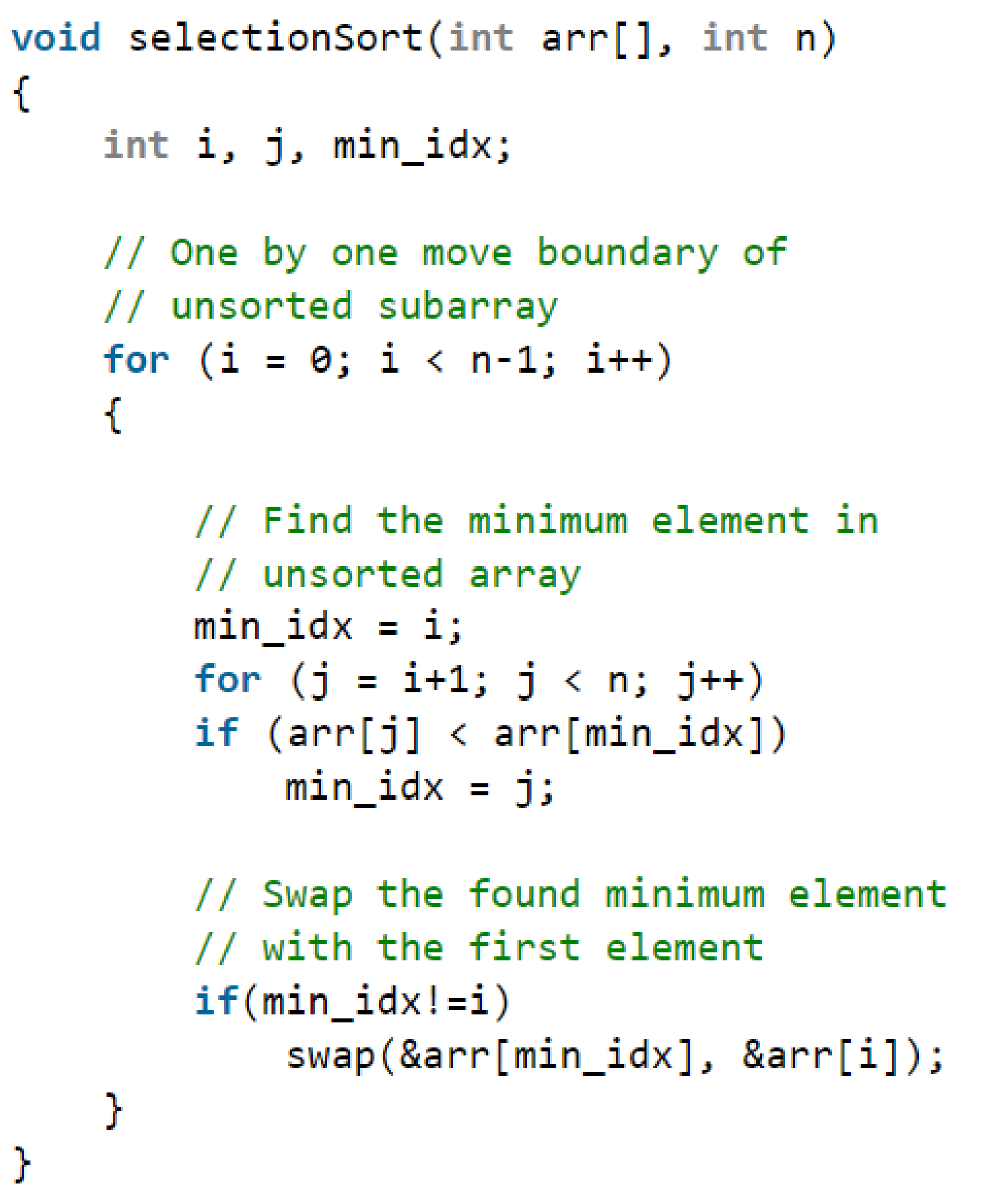
- Step 3: repeatedly step 1-2 for the unsorted parts



Selection Sort

- Implementation

  in C++



```cpp
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of
    // unsorted subarray
    for (i = 0; i < n-1; i++)
    {

        // Find the minimum element in
        // unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element
        // with the first element
        if(min_idx!=i)
            swap(&arr[min_idx], &arr[i]);
    }
}
```

Selection Sort

- Exercise 1: Write a function to sort the integer's array such that all prime numbers are put into the left hand-side and their relative positions are remained.
- For example, with the following list of integers

  4 2 7 3 1 11 5 9

- We have the expected array after sorting as:

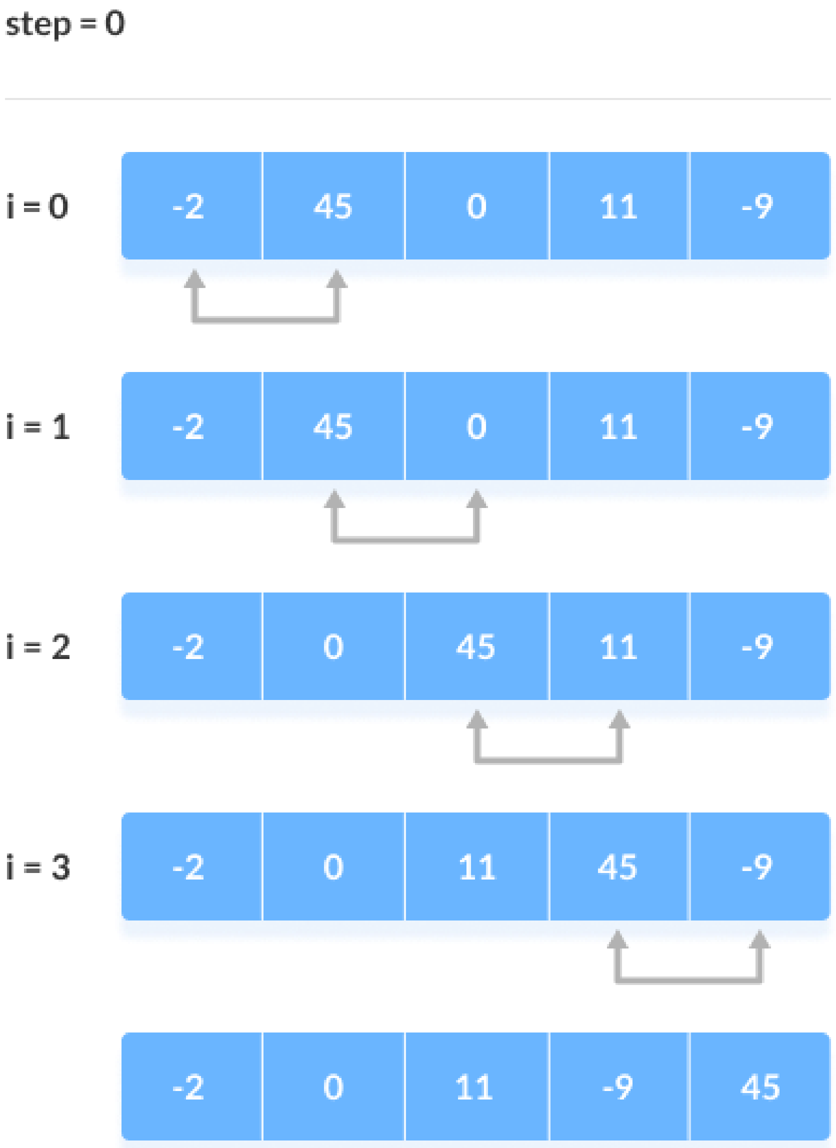
  7 3 11 5 1 4 2 9

Bubble Sort

- Main idea: compares two adjacent elements and swaps them until they are in the intended order

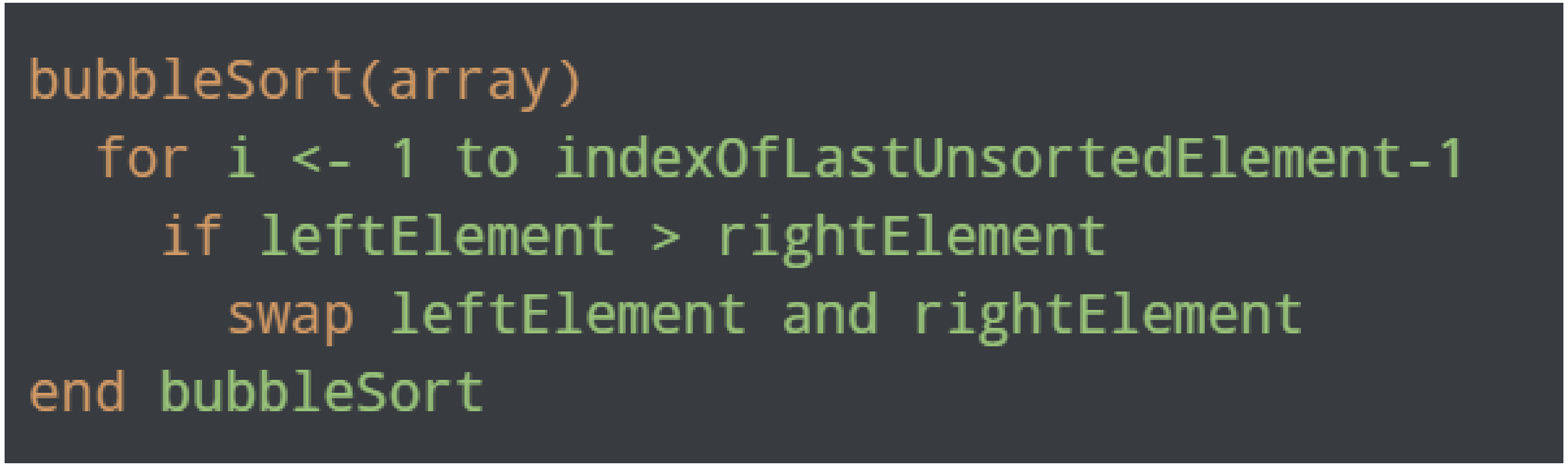
Bubble Sort

- Step 1: Compare and Swap
  - If ith and (i+1)th elements are in the incorrect positions, swap them



Bubble Sort

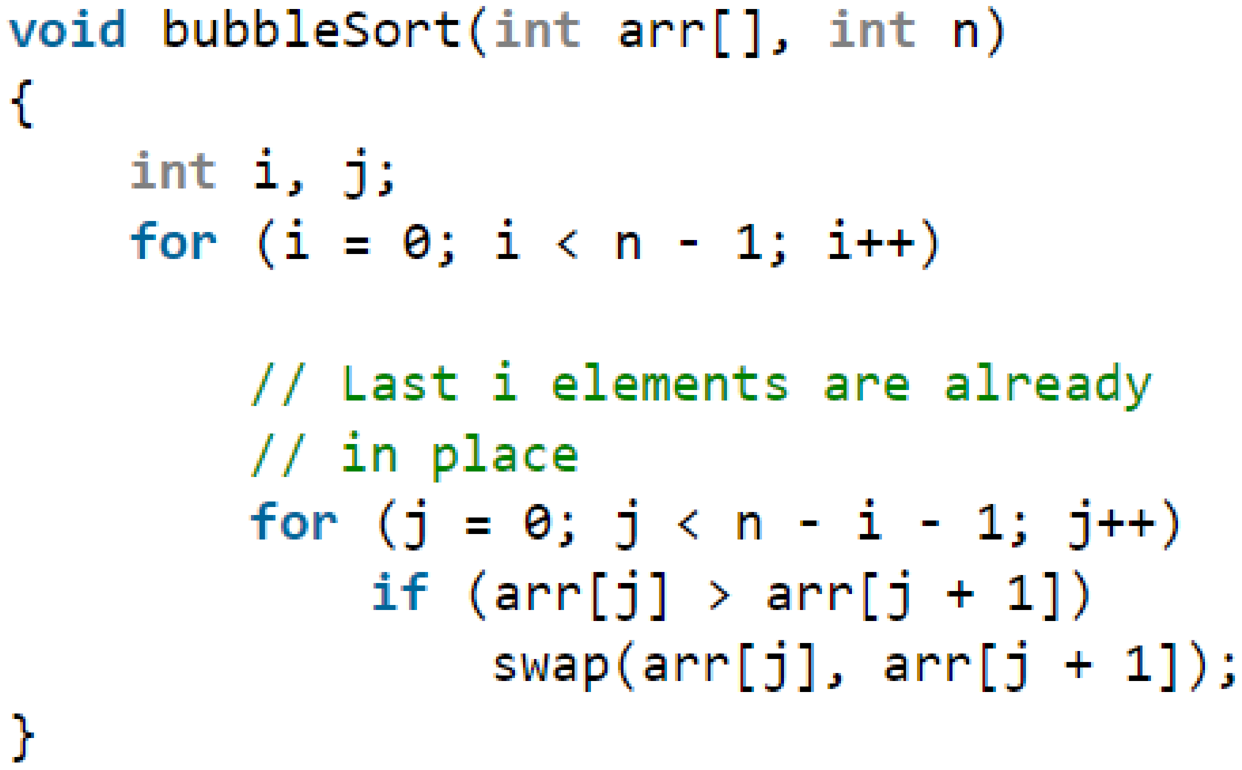
- Step 2: Remaining Iteration
  - Repeat Step 1
  - Until sorted list or len(list) – 1 times



```
bubbleSort(array)
  for i <- 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap leftElement and rightElement
end bubbleSort
```

Bubble Sort

- Implementation in C++



```
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)

        // Last i elements are already
        // in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}
```

Bubble Sort

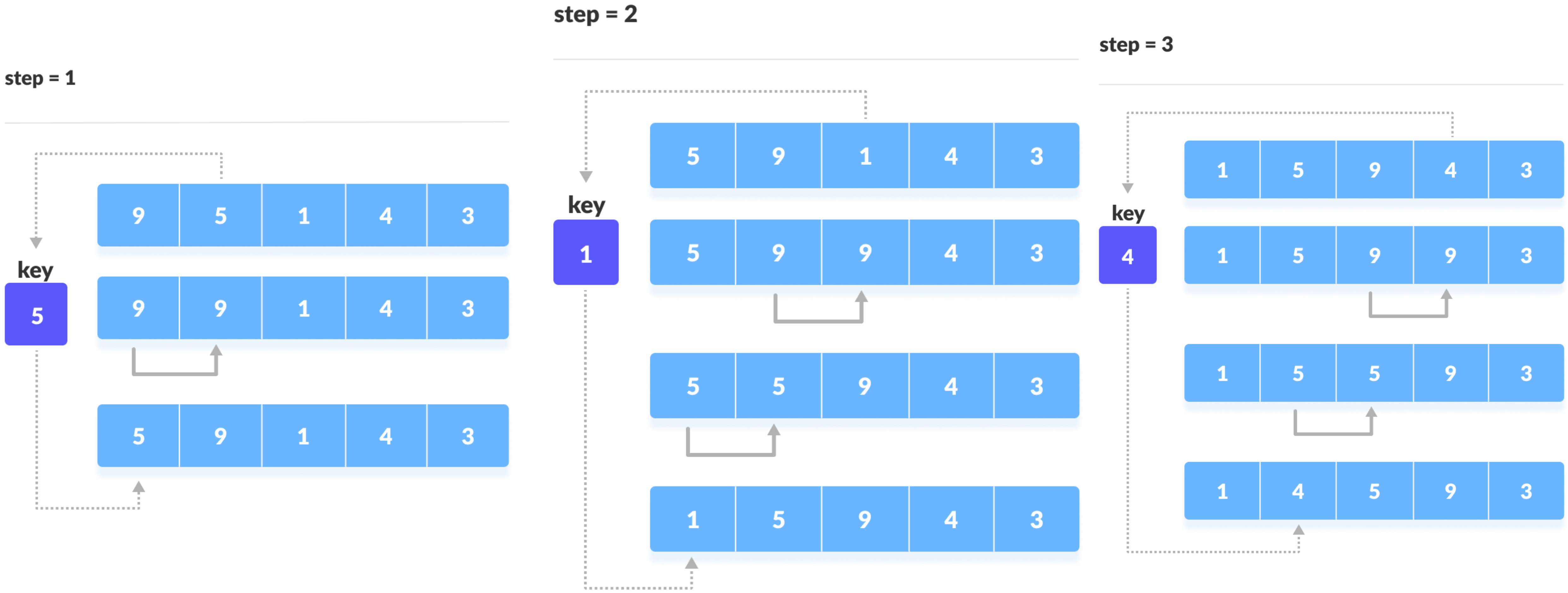
- Exercise 2: Write a program to sort the list of integers such that all prime numbers are sorted in ascending and the others are relatively remained

  e.g: the unsorted list: [4, 5, 1, 7, 9, 3, 0, 2]

  the sorted list: [4, 1, 3, 5, 7, 9, 0, 2]

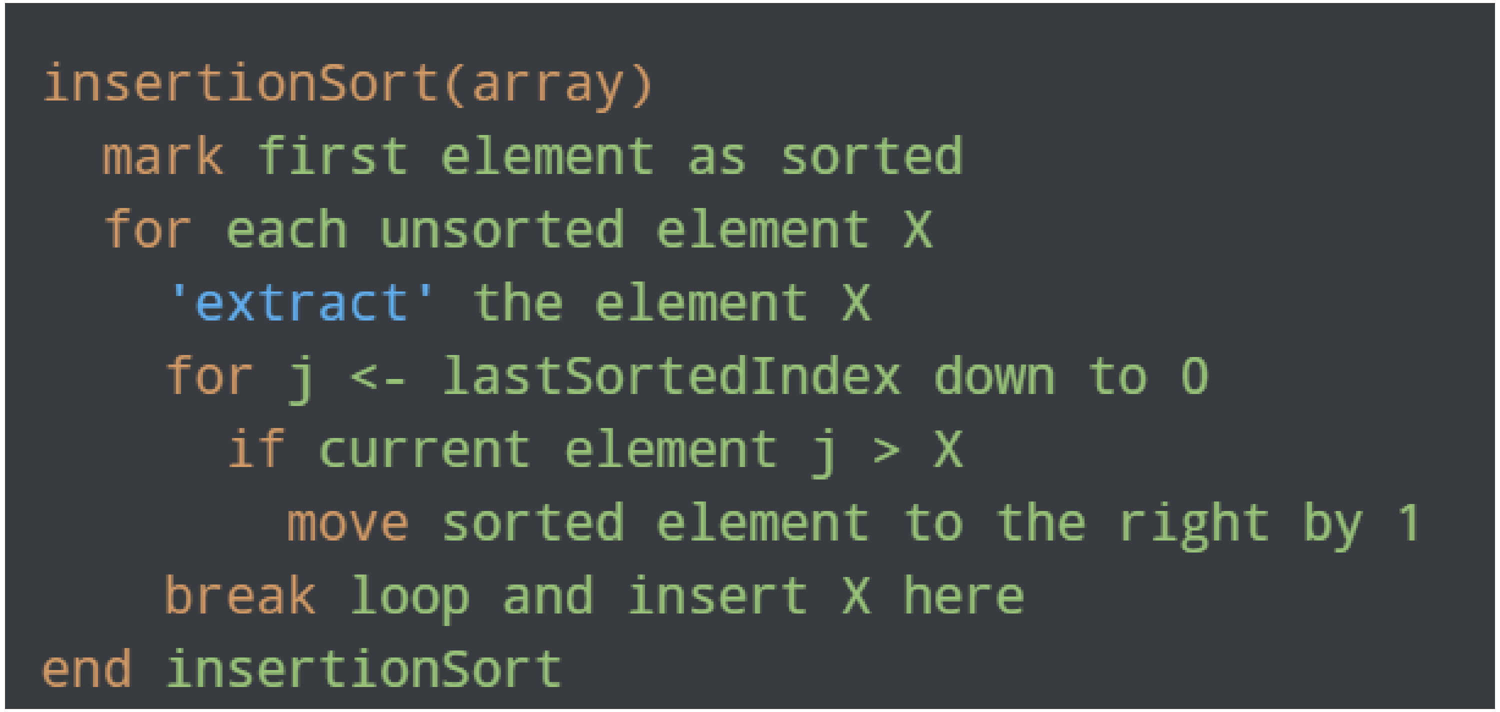
Insertion Sort

- Main idea: places an unsorted element at its suitable place in each iteration



Insertion Sort

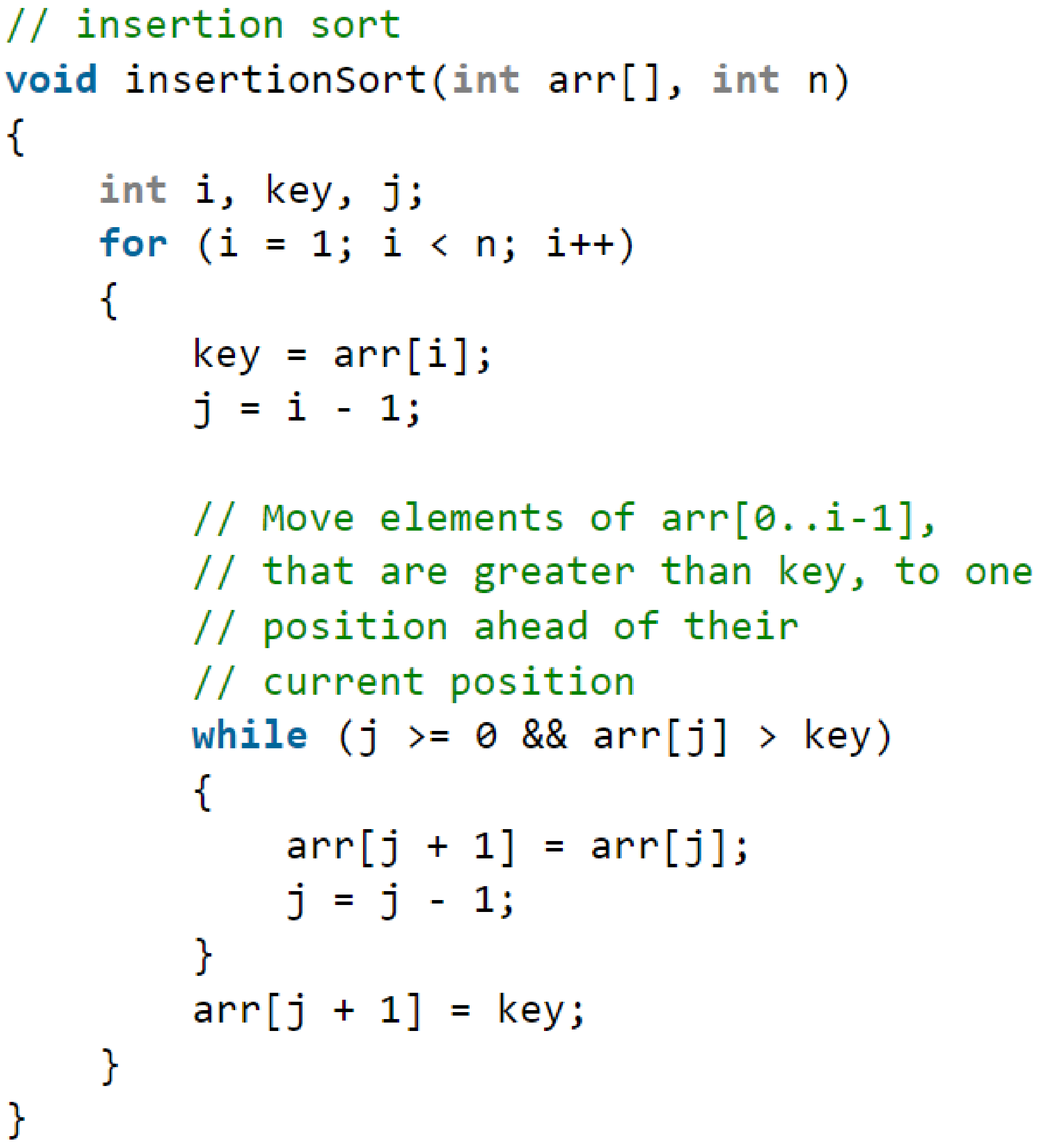
- Pseudo Code



```
insertionSort(array)
  mark first element as sorted
  for each unsorted element X
    'extract' the element X
    for j <- lastSortedIndex down to 0
      if current element j > X
        move sorted element to the right by 1
    break loop and insert X here
end insertionSort
```

Insertion Sort

- Implementation in C++



```
// insertion sort
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        // Move elements of arr[0..i-1],
        // that are greater than key, to one
        // position ahead of their
        // current position
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Exercise

- Develop an efficient in-place algorithm that partitions an array a in even and odd numbers. The algorithm must terminate with a containing all its even elements preceding all its odd elements.
- In addition, even elements are in ascending order and odd elements are in descending order

THANK YOU

for YOUR ATTENTION