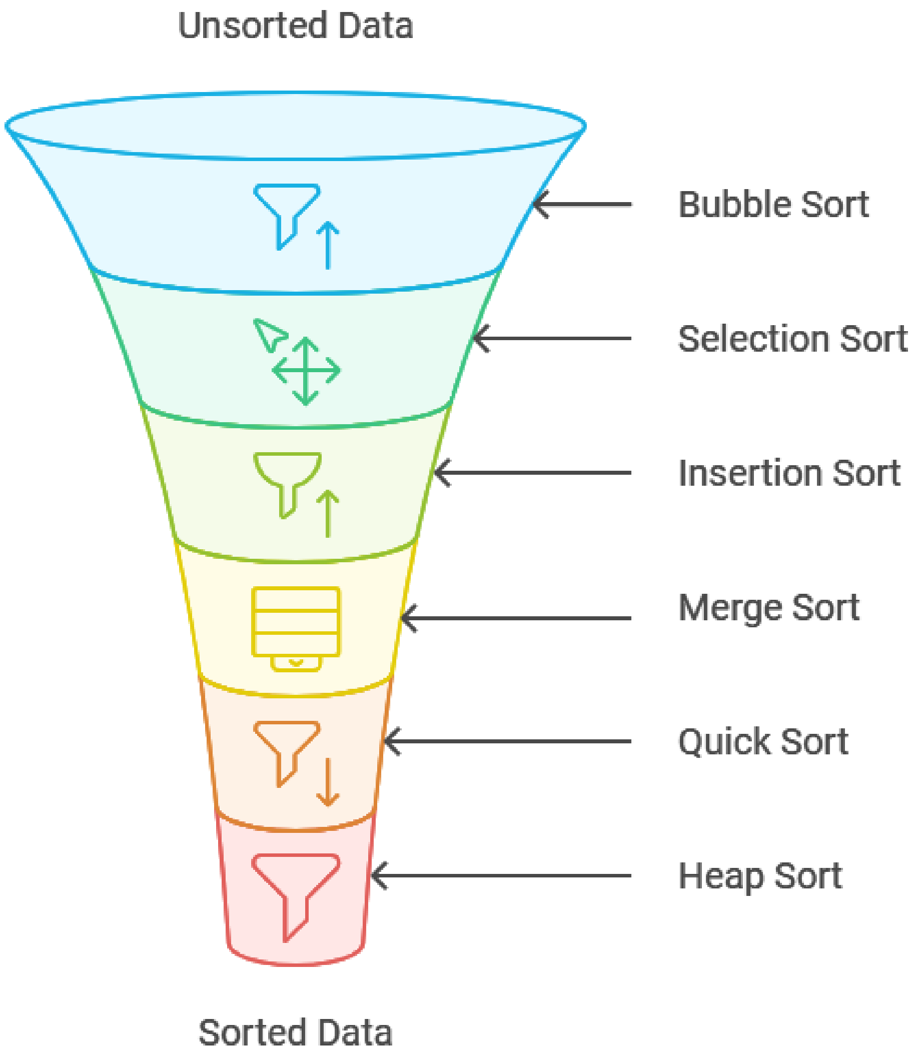
Module 5

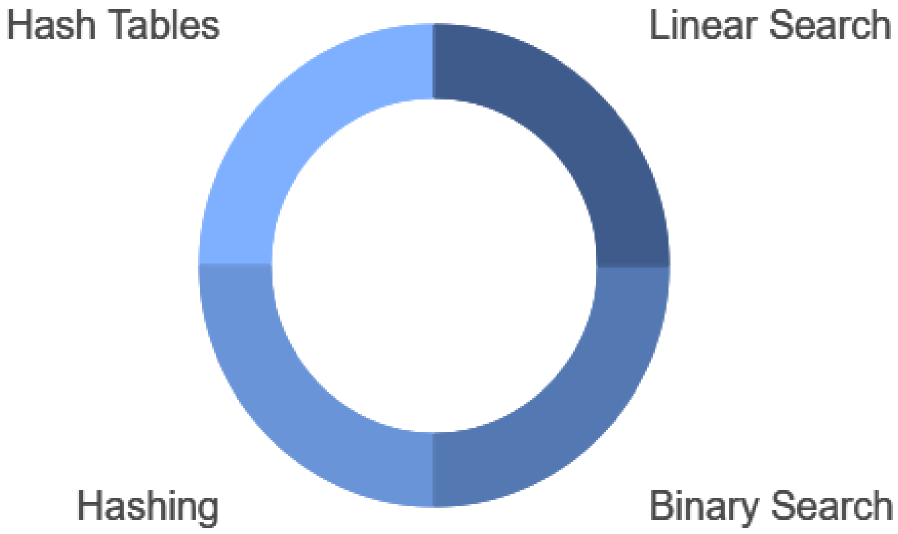
Searching and Sorting Algorithm

Content

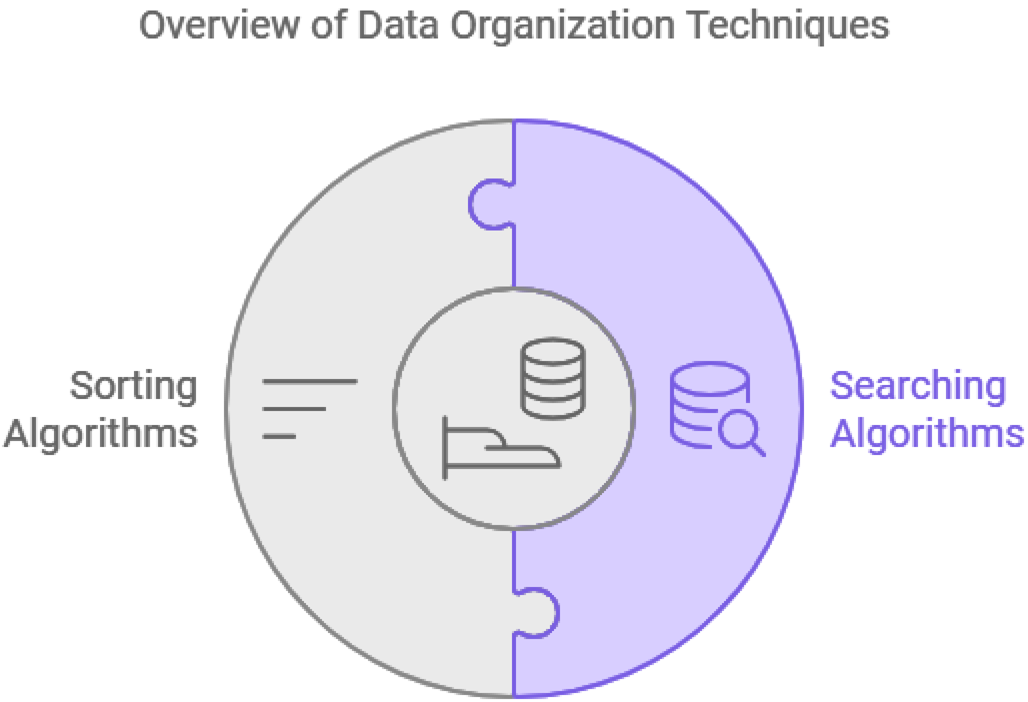
Part 1: Sorting Algorithm



Part 2: Searching Algorithms

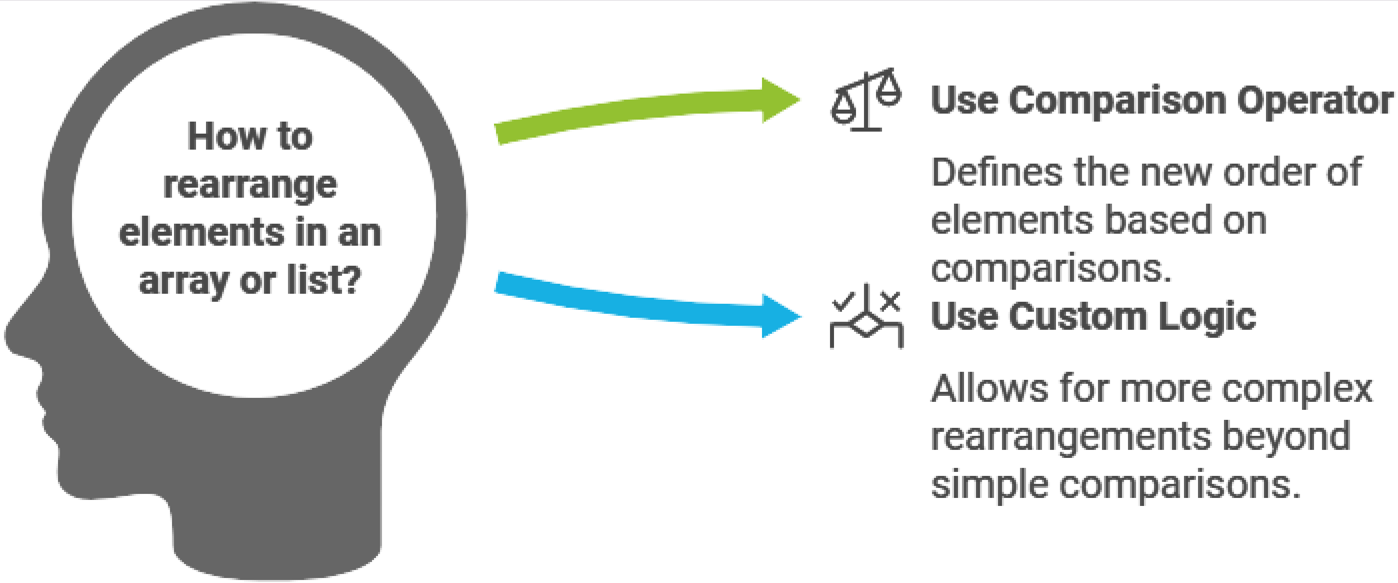


Objective


Overview of Data Organization Techniques
Sorting Algorithms
Searching Algorithms

Part 1: Sorting Algorithm

I. Sorting Definition



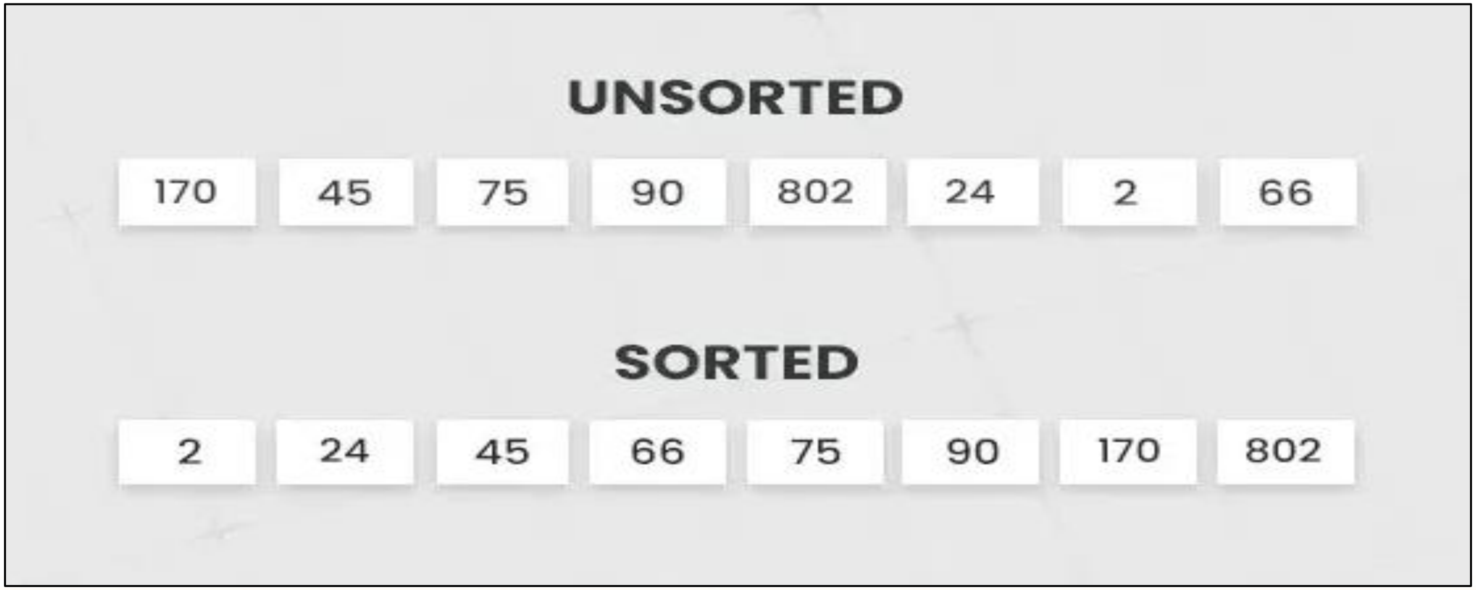
How to rearrange elements in an array or list?

Use Comparison Operator

Defines the new order of elements based on comparisons.

Use Custom Logic

Allows for more complex rearrangements beyond simple comparisons.



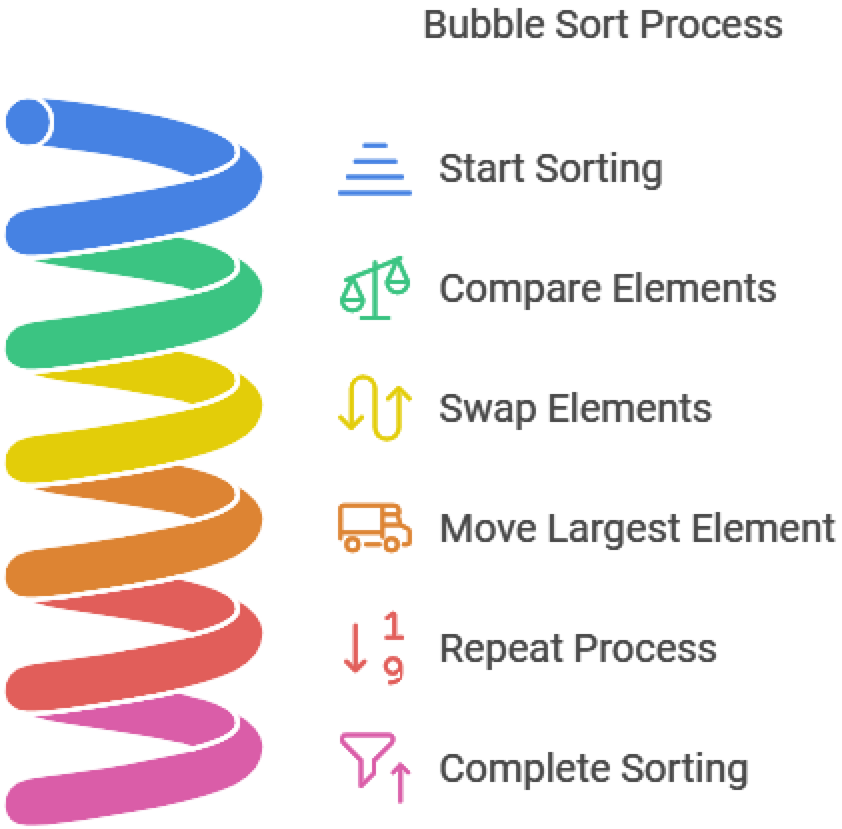
UNSORTED

170	45	75	90	802	24	2	66

SORTED

2	24	45	66	75	90	170	802

II. Bubble Sort



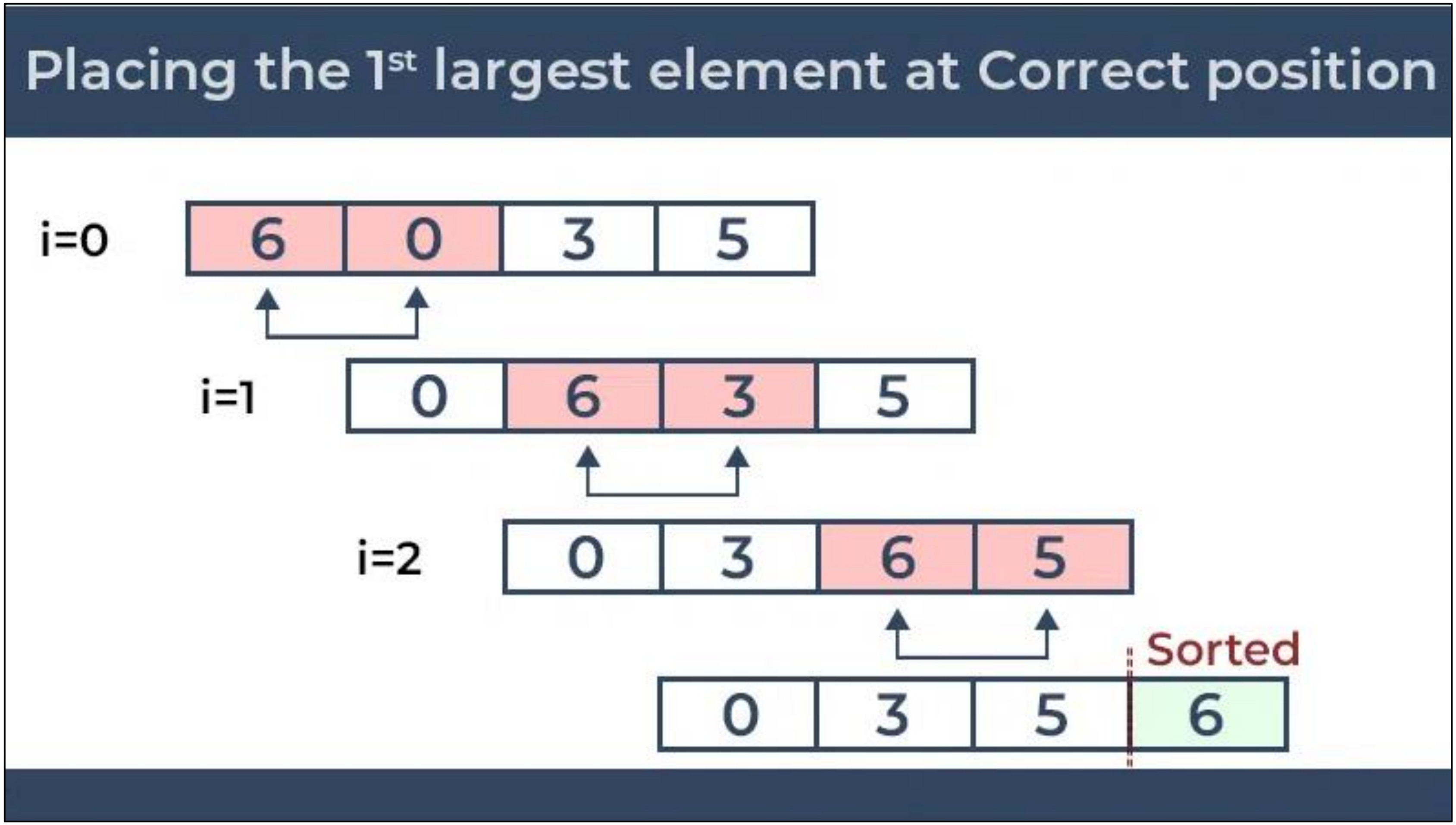
Bubble Sort Process

- Start Sorting
- Compare Elements
- Swap Elements
- Move Largest Element
- Repeat Process
- Complete Sorting

II. Bubble Sort

Input: arr[] = {6, 0, 3, 5}

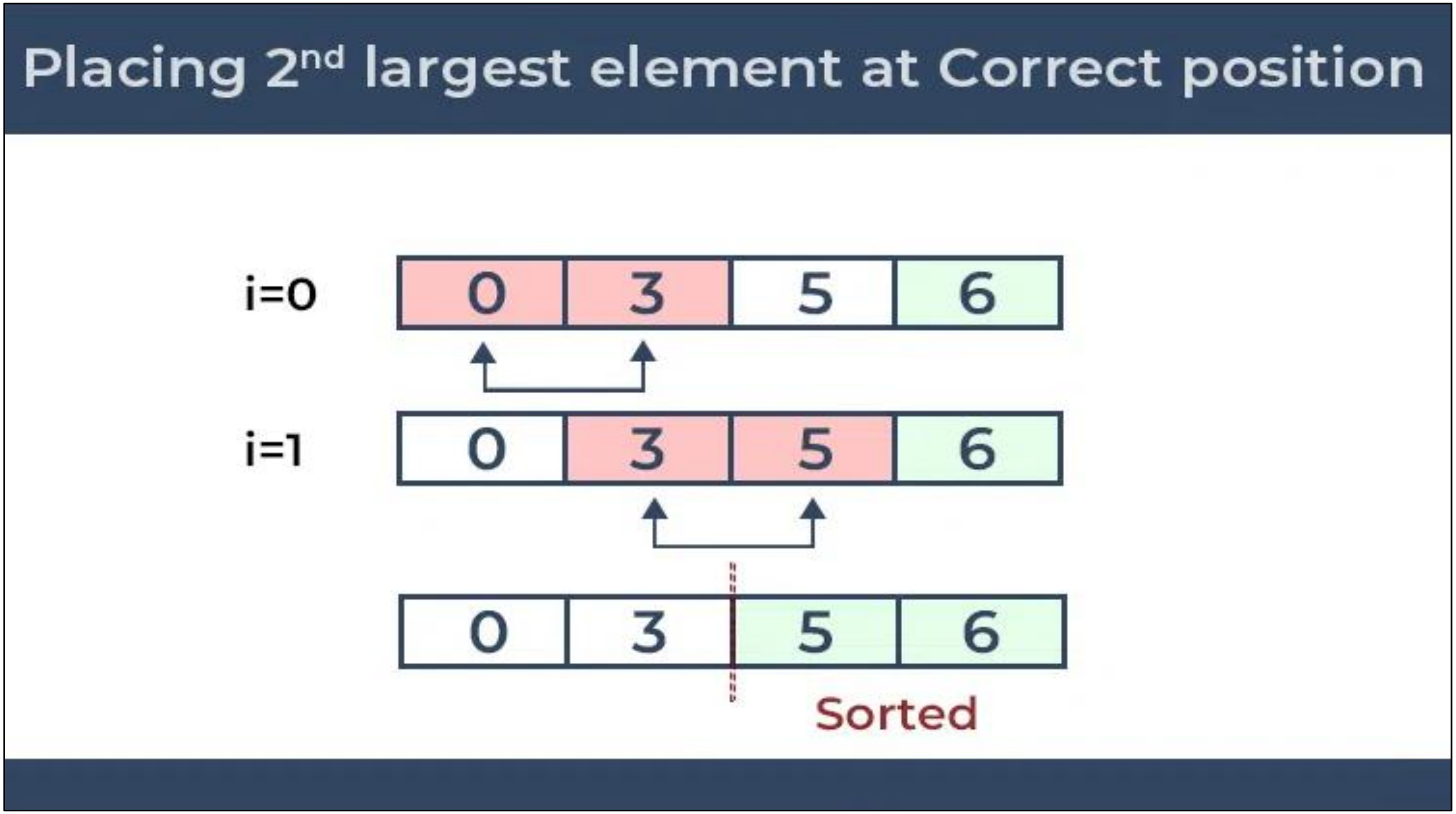
First Pass: The largest element is placed in its correct position, the end of the array.



Placing the 1st largest element at Correct position

II. Bubble Sort

Second Pass: Place the second largest element at correct position



Placing 2nd largest element at Correct position

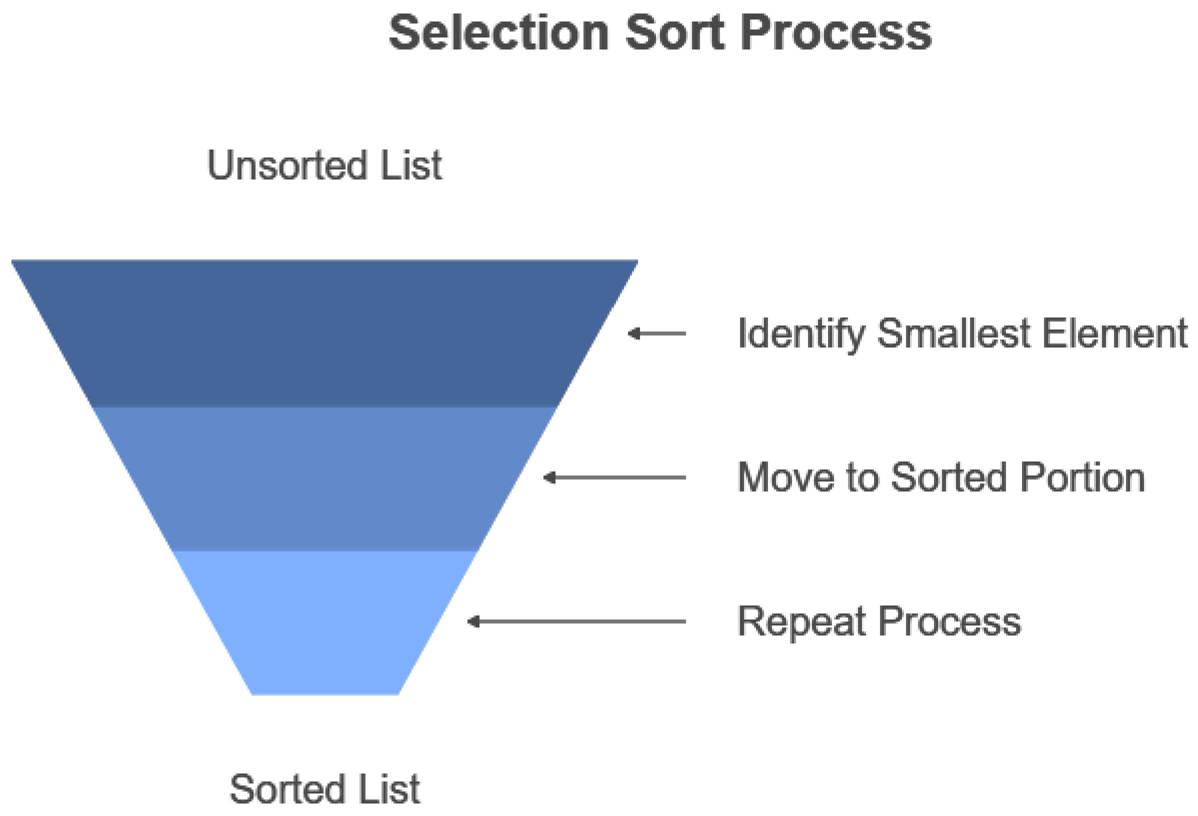
II. Bubble Sort

```
using System;  // C# implementation of Bubble sort
class GFG { // An optimized version of Bubble Sort
  static void bubbleSort(int[] arr, int n)
  {   int i, j, temp;
    bool swapped;
    for (i = 0; i < n - 1; i++) {
      swapped = false;
      for (j = 0; j < n - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
          // Swap arr[j] and arr[j+1]
          temp = arr[j];
          arr[j] = arr[j + 1];
          arr[j + 1] = temp;
          swapped = true;
          }
  }
```

```
// If no two elements were swapped, then break
      if (swapped == false)  break;       }
  } // Function to print an array
static void printArray(int[] arr, int size)
  {  for (int i = 0; i < size; i++)
      Console.Write(arr[i] + " ");
    Console.WriteLine();
  }
  public static void Main() // Driver method
  {   int[] arr = { 64, 34, 25, 12, 22, 11, 90 };
    int n = arr.Length;
    bubbleSort(arr, n);
    Console.WriteLine("Sorted array:");
    printArray(arr, n);    }
}
```

Output: Sorted array: 11 12 22 25 34 64 90

III. Selection Sort



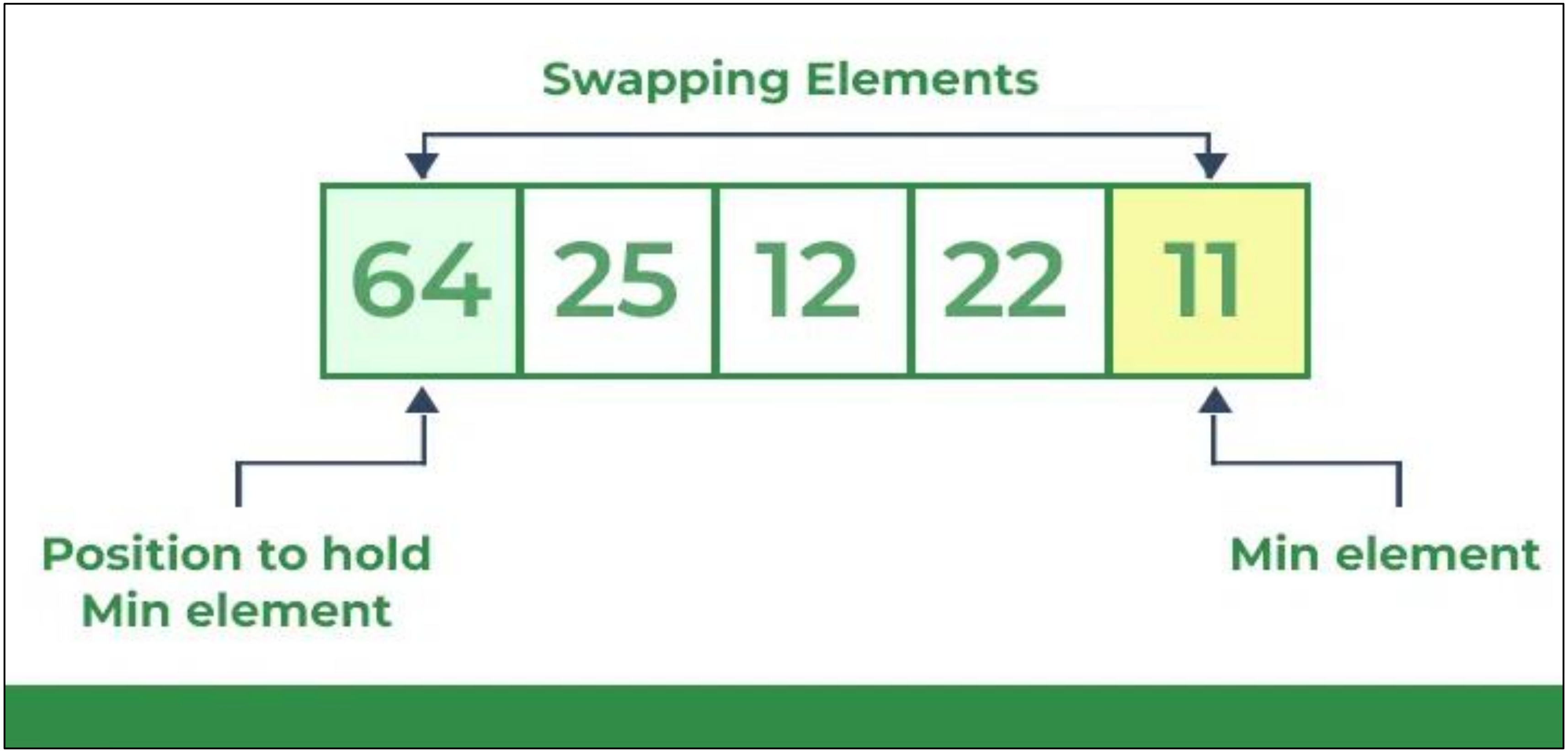
How does Selection Sort Algorithm work?

- Lets consider the following array as an example: arr[] = {64, 25, 12, 22, 11}

III. Selection Sort

First pass:

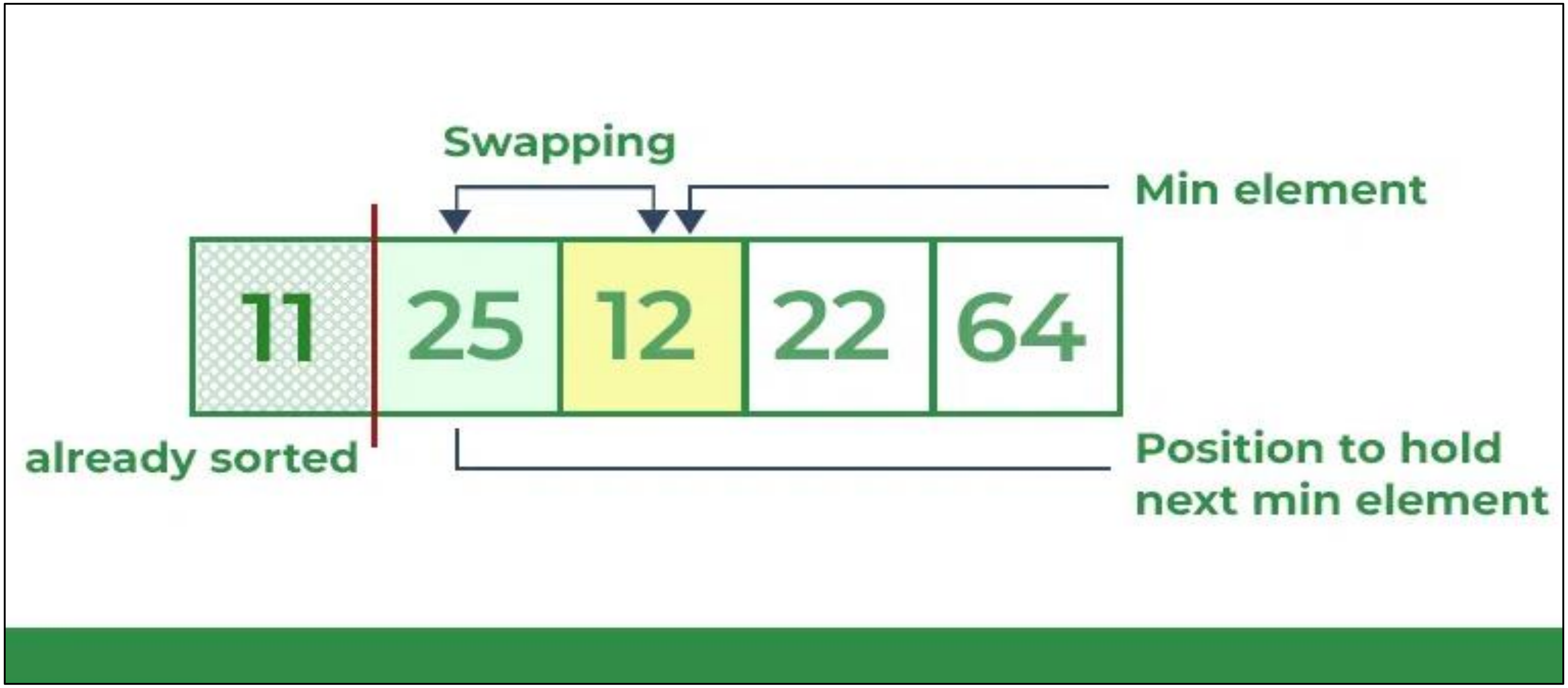
The first position where 64 is stored presently, after traversing whole array it is clear that 11 is the lowest value.Thus, replace 64 with 11.



III. Selection Sort

Second Pass:

- The second position, 25 is present, again traverse the rest of the array in a sequential manner. After traversing, we found that 12 is the second lowest value



III. Selection Sort

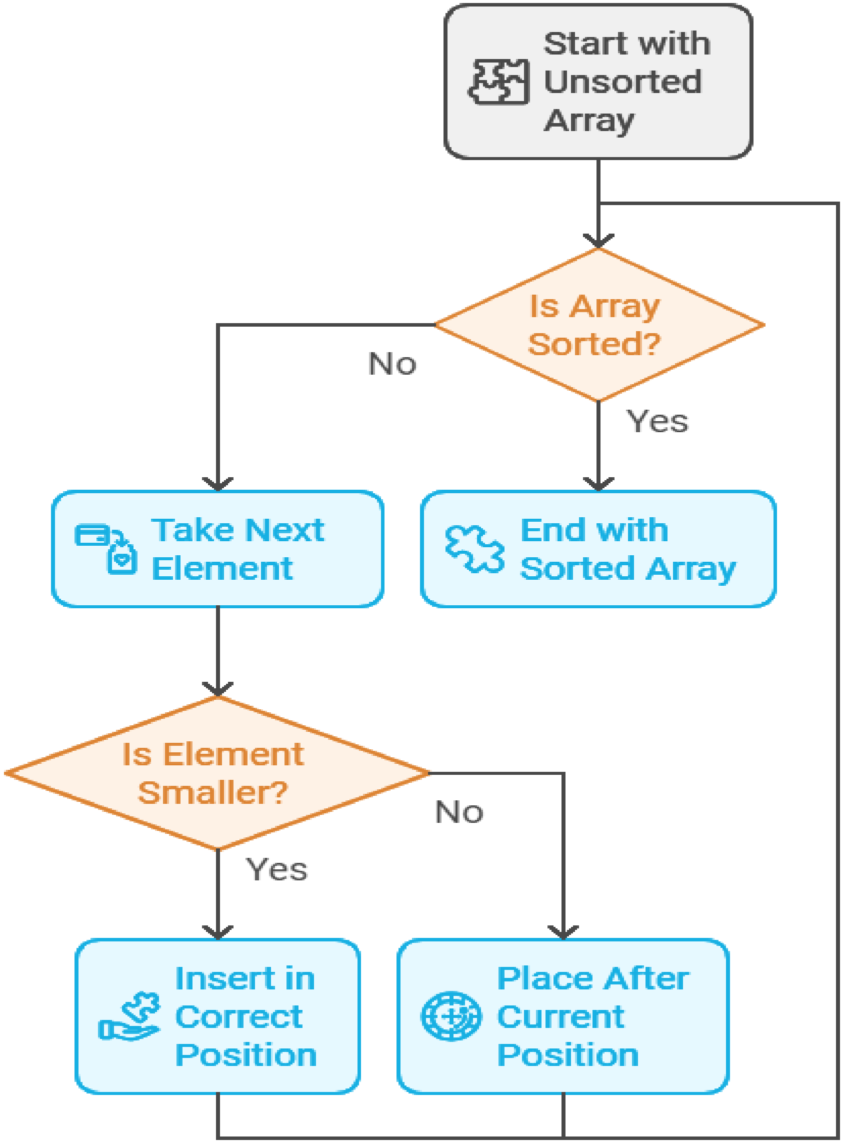
```
using System; // C# program for implementation of
Selection Sort
class GFG
{     static void sort(int []arr)
   {   int n = arr.Length;
       // One by one move boundary of unsorted array
       for (int i = 0; i < n - 1; i++)
       {  // Find the min element in unsorted array
          int min_idx = i;
          for (int j = i + 1; j < n; j++)
             if (arr[j] < arr[min_idx])
                min_idx = j;
         // Swap the found min elem with the first elem
          int temp = arr[min_idx];
          arr[min_idx] = arr[i];
          arr[i] = temp;       }
   }
```

```
static void printArray(int []arr) // Prints the array
   {   int n = arr.Length;
       for (int i=0; i<n; ++i)
          Console.Write(arr[i]+" ");
       Console.WriteLine();
   }
   public static void Main() // Driver code
   {   int []arr = {64,25,12,22,11};
       sort(arr);
       Console.WriteLine("Sorted array");
       printArray(arr);    }
}
```

Output:

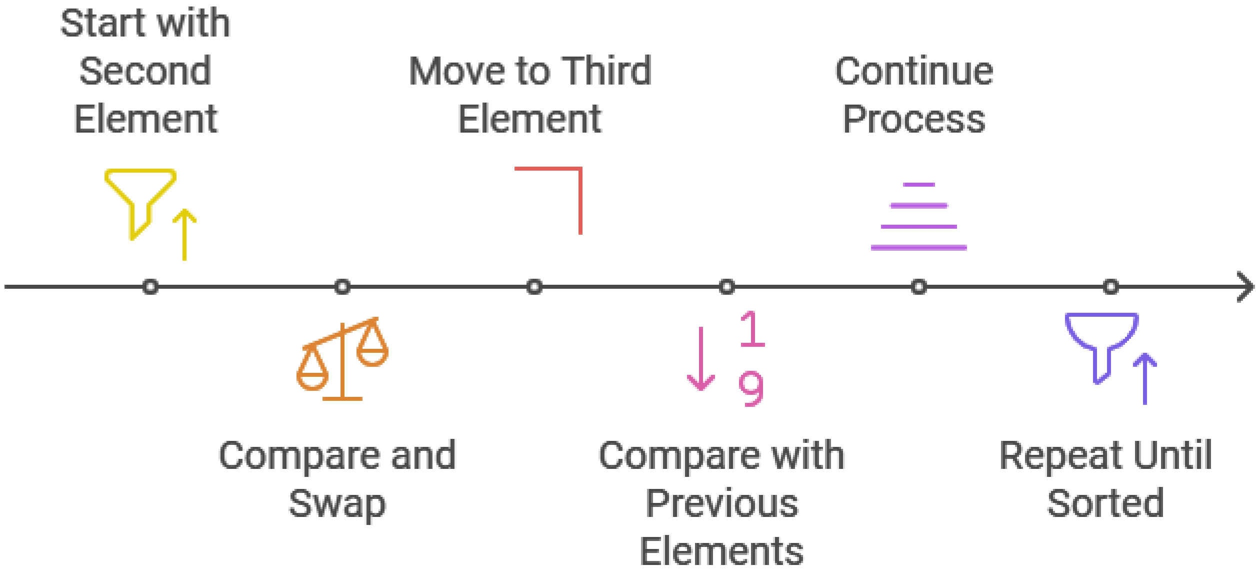
```
Sorted array:   11 12 22 25 64
```

IV. Insertion Sort



IV. Insertion Sort

Insertion Sort Process



Start with Second Element

Compare and Swap

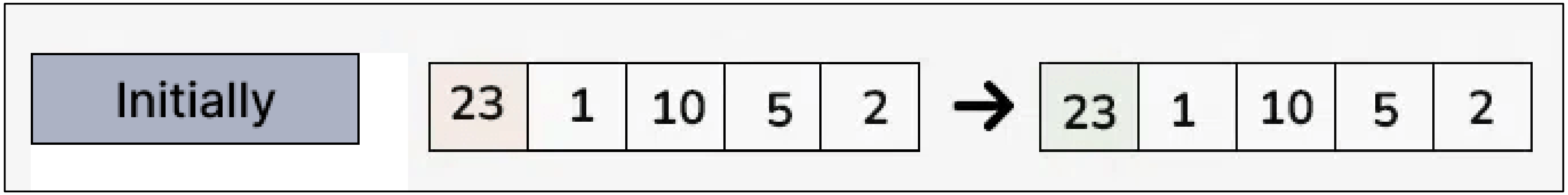
Move to Third Element

Compare with Previous Elements

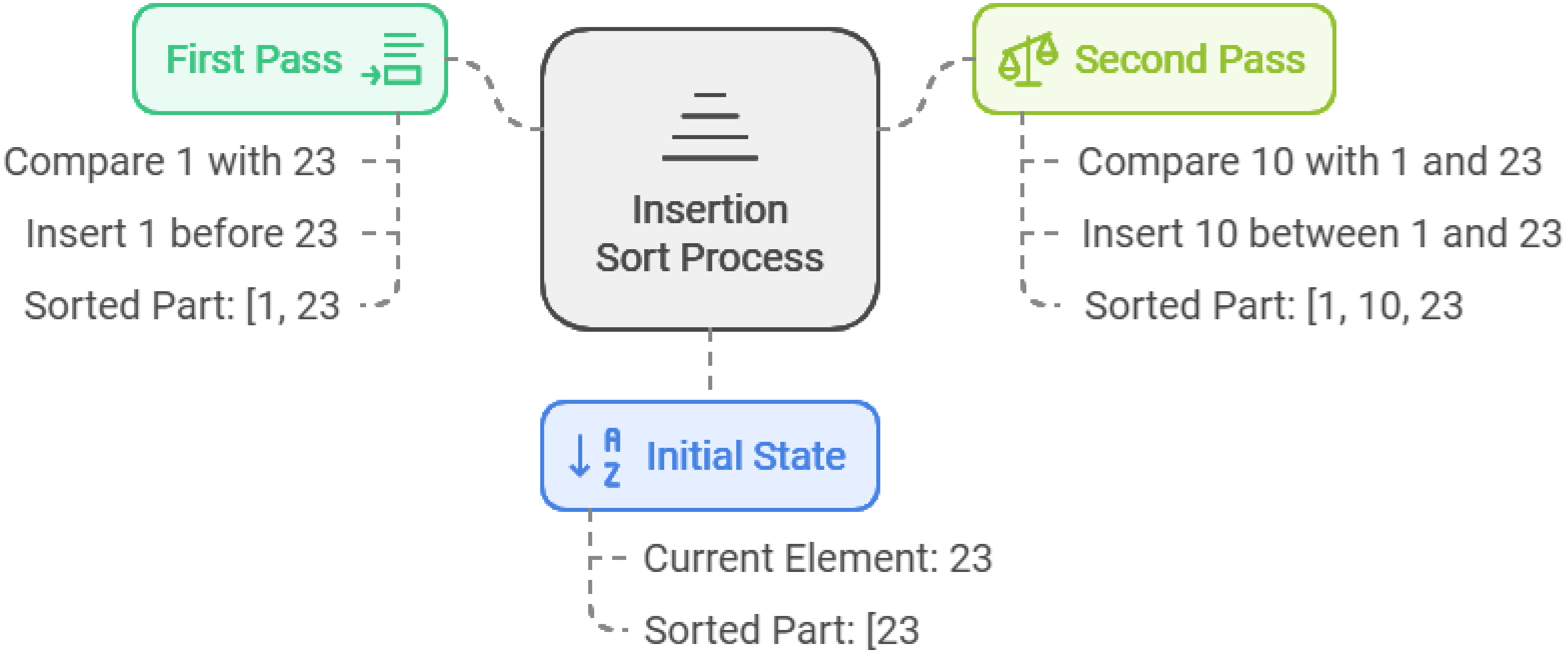
Continue Process

Repeat Until Sorted

IV. Insertion Sort



Initially	23	1	10	5	2	→	23	1	10	5	2



Insertion Sort Process

First Pass

- Compare 1 with 23
- Insert 1 before 23
- Sorted Part: [1, 23

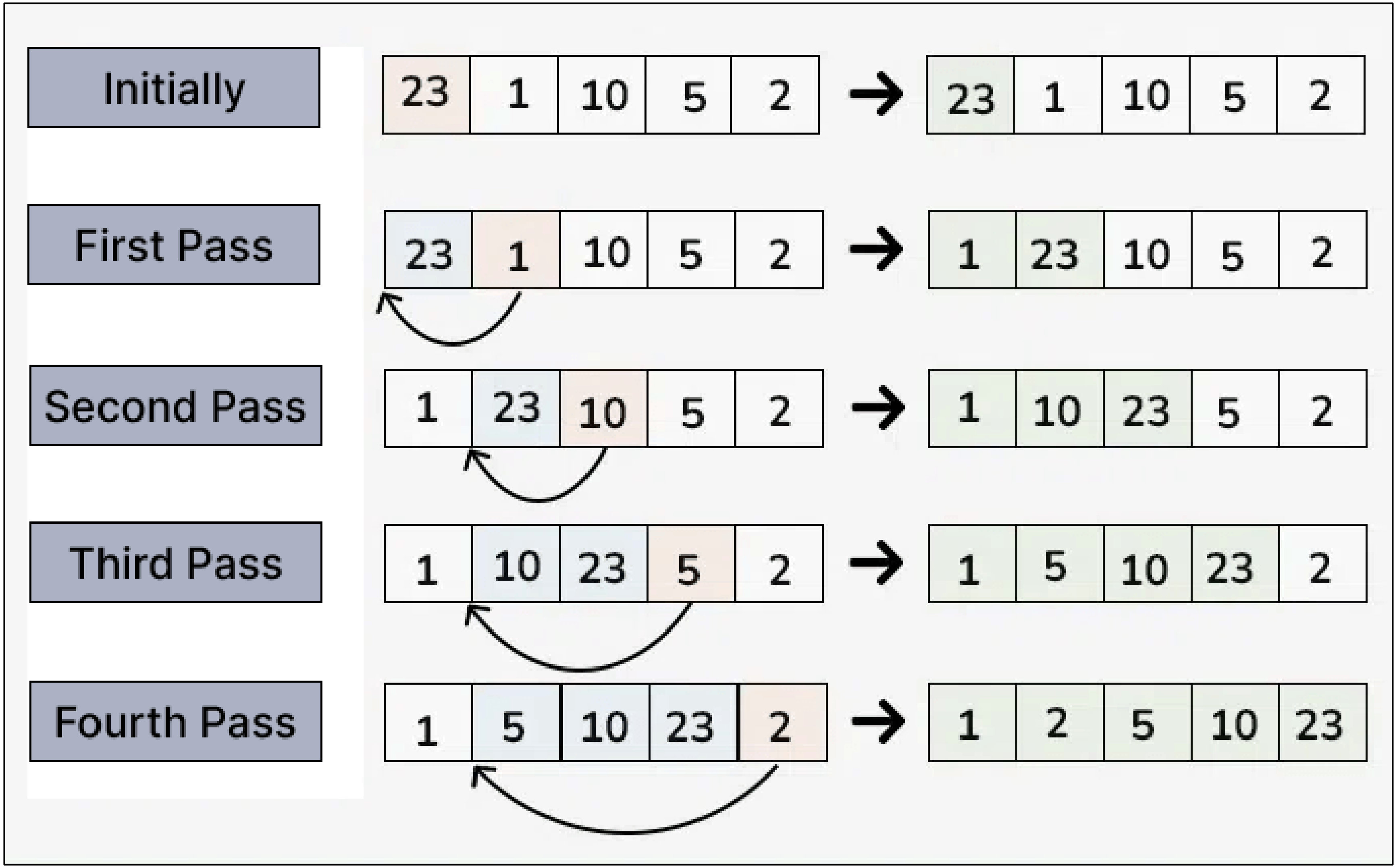
Second Pass

- Compare 10 with 1 and 23
- Insert 10 between 1 and 23
- Sorted Part: [1, 10, 23

Initial State

- Current Element: 23
- Sorted Part: [23

IV. Insertion Sort



Initially | 23 | 1 | 10 | 5 | 2 → 23 | 1 | 10 | 5 | 2

First Pass | 23 | 1 | 10 | 5 | 2 → 1 | 23 | 10 | 5 | 2

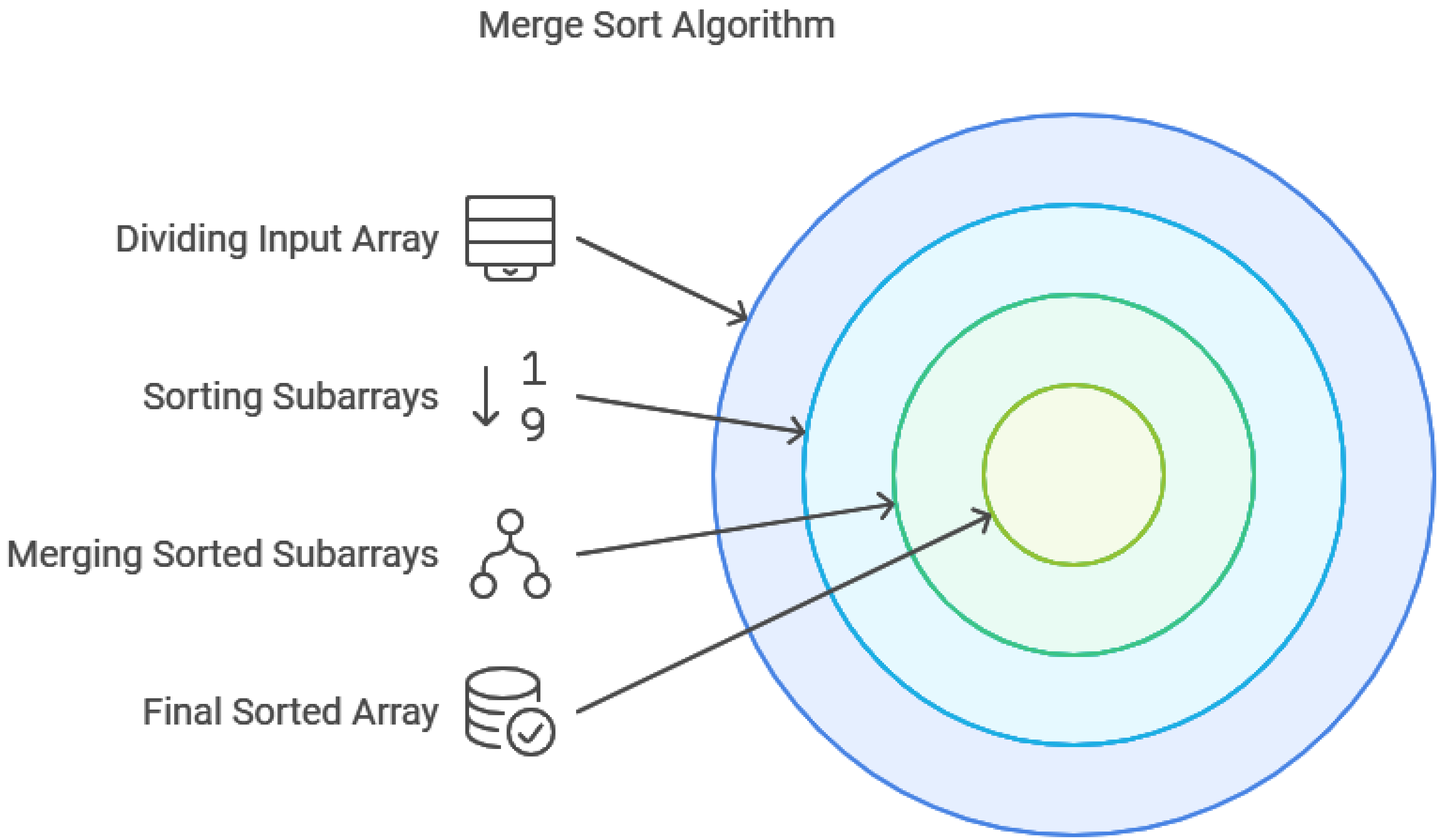
Second Pass | 1 | 23 | 10 | 5 | 2 → 1 | 10 | 23 | 5 | 2

Third Pass | 1 | 10 | 23 | 5 | 2 → 1 | 5 | 10 | 23 | 2

Fourth Pass | 1 | 5 | 10 | 23 | 2 → 1 | 2 | 5 | 10 | 23

Output: Number of passes: 4 5 6 11 12 13

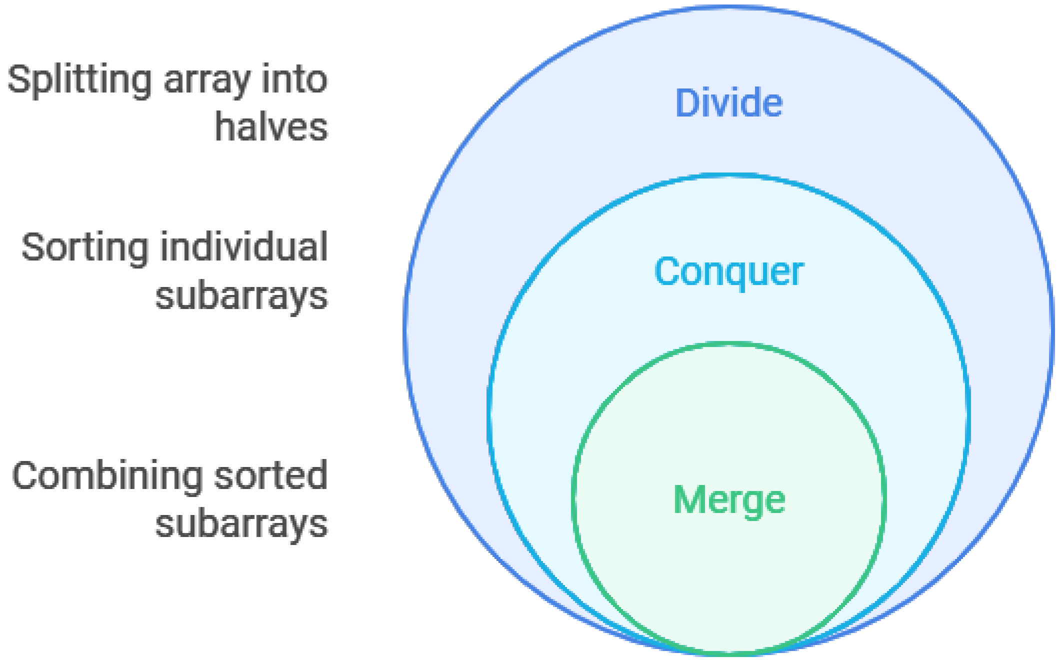
V. Merge Sort


Merge Sort Algorithm
Dividing Input Array
Sorting Subarrays
1
9
Merging Sorted Subarrays
Final Sorted Array

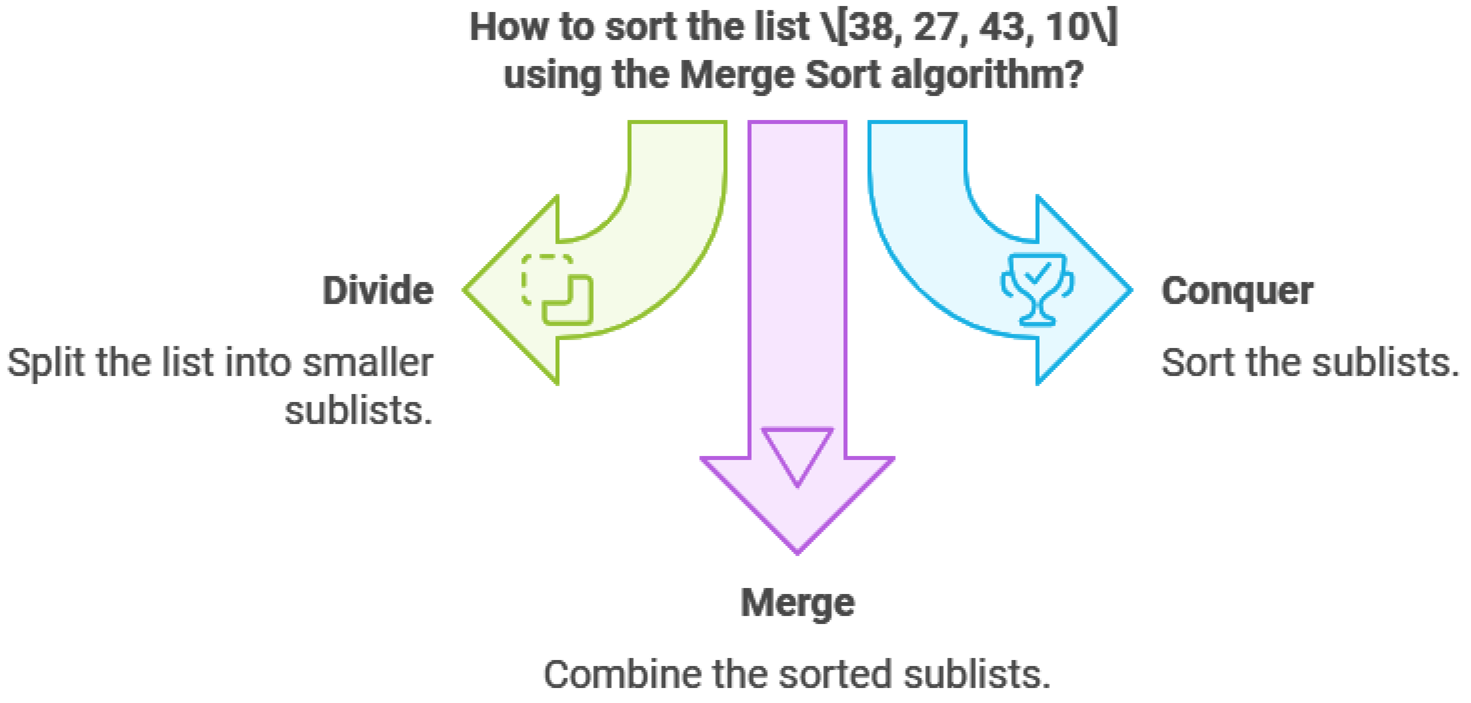
V. Merge Sort

How does Merge Sort work?

Merge Sort Algorithm


Splitting array into halves
Divide
Sorting individual subarrays
Conquer
Combining sorted subarrays
Merge

V. Merge Sort



How to sort the list \[38, 27, 43, 10\] using the Merge Sort algorithm?

Divide

Split the list into smaller sublists.

Conquer

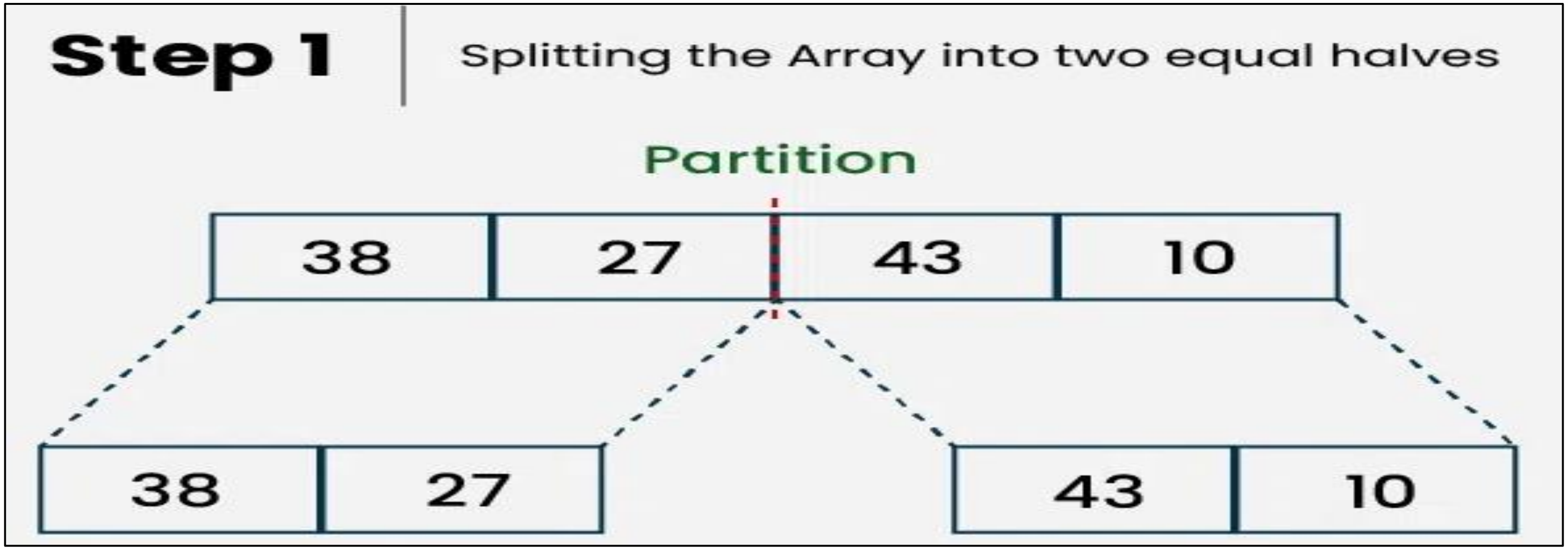
Sort the sublists.

Merge

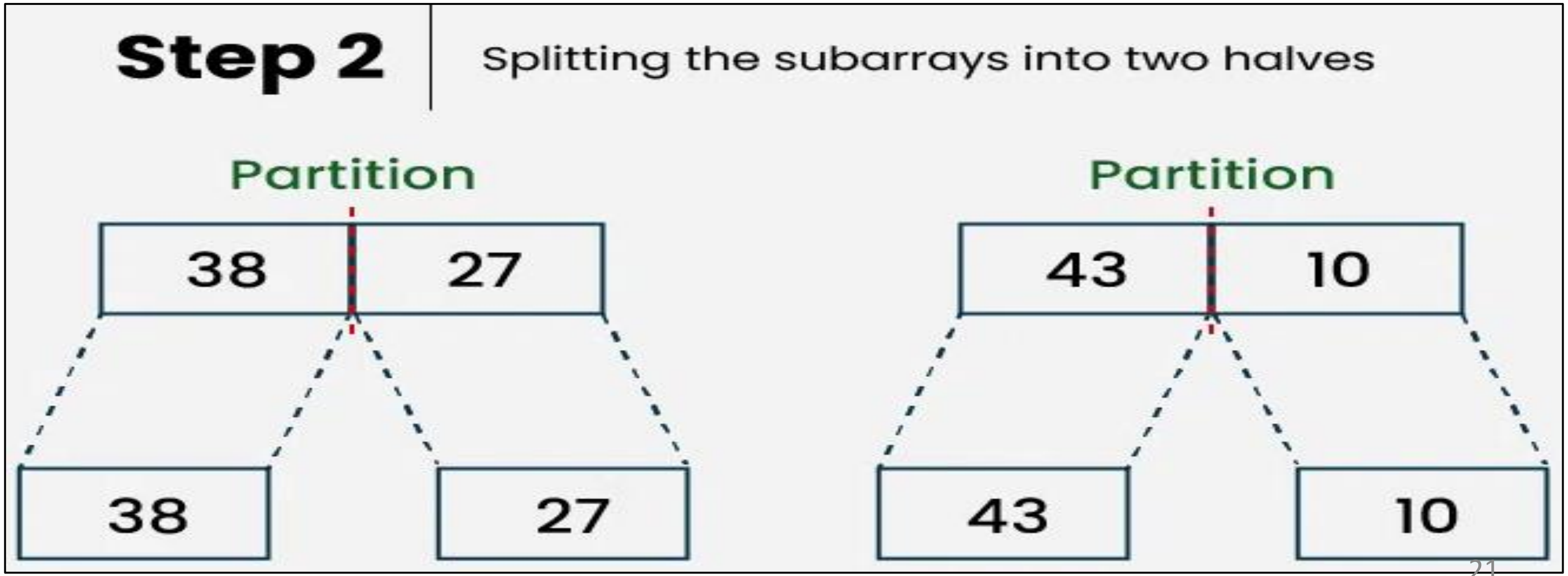
Combine the sorted sublists.

Therefore, the sorted list is [10, 27, 38, 43]

V. Merge Sort



Step 1 Splitting the Array into two equal halves



Step 2 Splitting the subarrays into two halves

V. Merge Sort

```
using System; // C# program for Merge Sort
class GfG {// Merges two subarrays of []arr. First subarray is arr[l..m]
   // Second subarray is arr[m+1..r]
   static void merge(int[] arr, int l, int m, int r)
   {   // Find sizes of two subarrays to be merged
      int n1 = m - l + 1;    int n2 = r - m;
      // Create temp arrays
      int[] L = new int[n1];  int[] R = new int[n2];  int i, j;
      // Copy data to temp arrays
      for (i = 0; i < n1; ++i)          L[i] = arr[l + i];
      for (j = 0; j < n2; ++j)          R[j] = arr[m + 1 + j];
      // Merge the temp arrays Initial indexes of first and second subarrays
      i = 0; j = 0; int k = l; // Initial index of merged subarray array
while (i < n1 && j < n2) {   if (L[i] <= R[j]) { arr[k] = L[i];  i++;
         }else { arr[k] = R[j]; j++; } k++; }// Copy remaining elements of L[] if any
```

Output: Given array is   12 11 13 5 6 7, Sorted array is 5 6 7 11 12 13

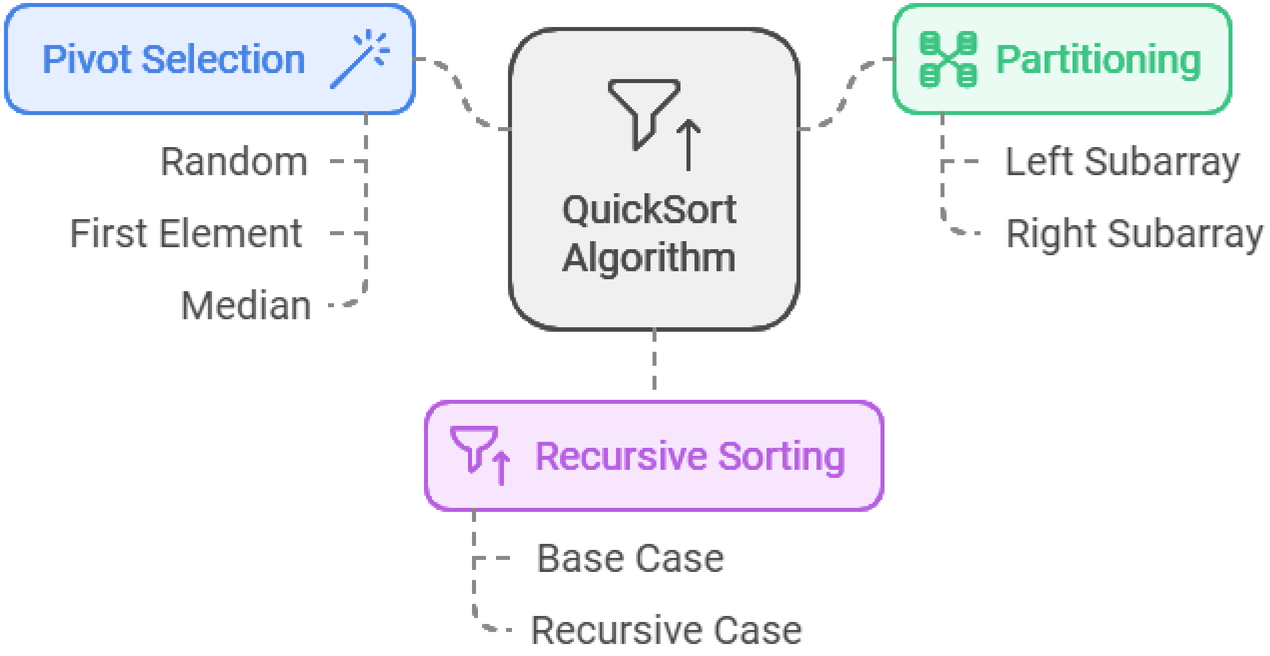
V. Merge Sort

```
  while (i < n1) {  arr[k] = L[i];  i++; k++;     }
     // Copy remaining elements of R[] if any
     while (j < n2) { arr[k] = R[j];  j++; k++;        }
  } // Main function that sorts arr[l..r] using merge()
  static void mergeSort(int[] arr, int l, int r)
  {   if (l < r) { // Find the middle point
        int m = l + (r - l) / 2;
        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        // Merge the sorted halves
        merge(arr, l, m, r);  }   }
// A utility function to print array of size n
  static void printArray(int[] arr)
```

```
{   int n = arr.Length;
for (int i = 0; i < n; ++i)
         Console.Write(arr[i] + " ");
      Console.WriteLine();
   } // Driver code
   public static void Main(String[] args)
   {   int[] arr = { 12, 11, 13, 5, 6, 7 };
      Console.WriteLine("Given array is");
      printArray(arr);
      mergeSort(arr, 0, arr.Length - 1);
      Console.WriteLine("\nSorted array is");
      printArray(arr);
   }
}
```

Output: Given array is 12 11 13 5 6 7, Sorted array is 5 6 7 11 12 13

VI. Quick Sort



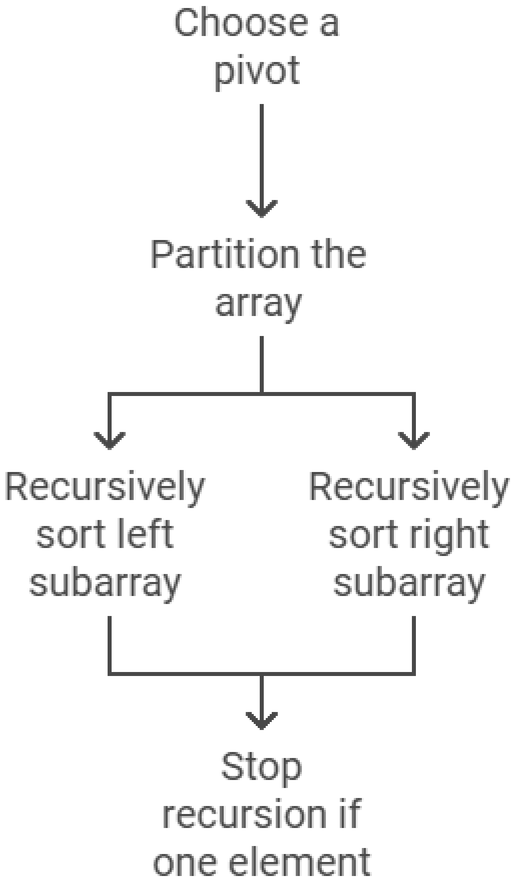
QuickSort Algorithm

Pivot Selection
- Random
- First Element
- Median

Partitioning
- Left Subarray
- Right Subarray

Recursive Sorting
- Base Case
- Recursive Case

VI. Quick Sort


Choose a pivot

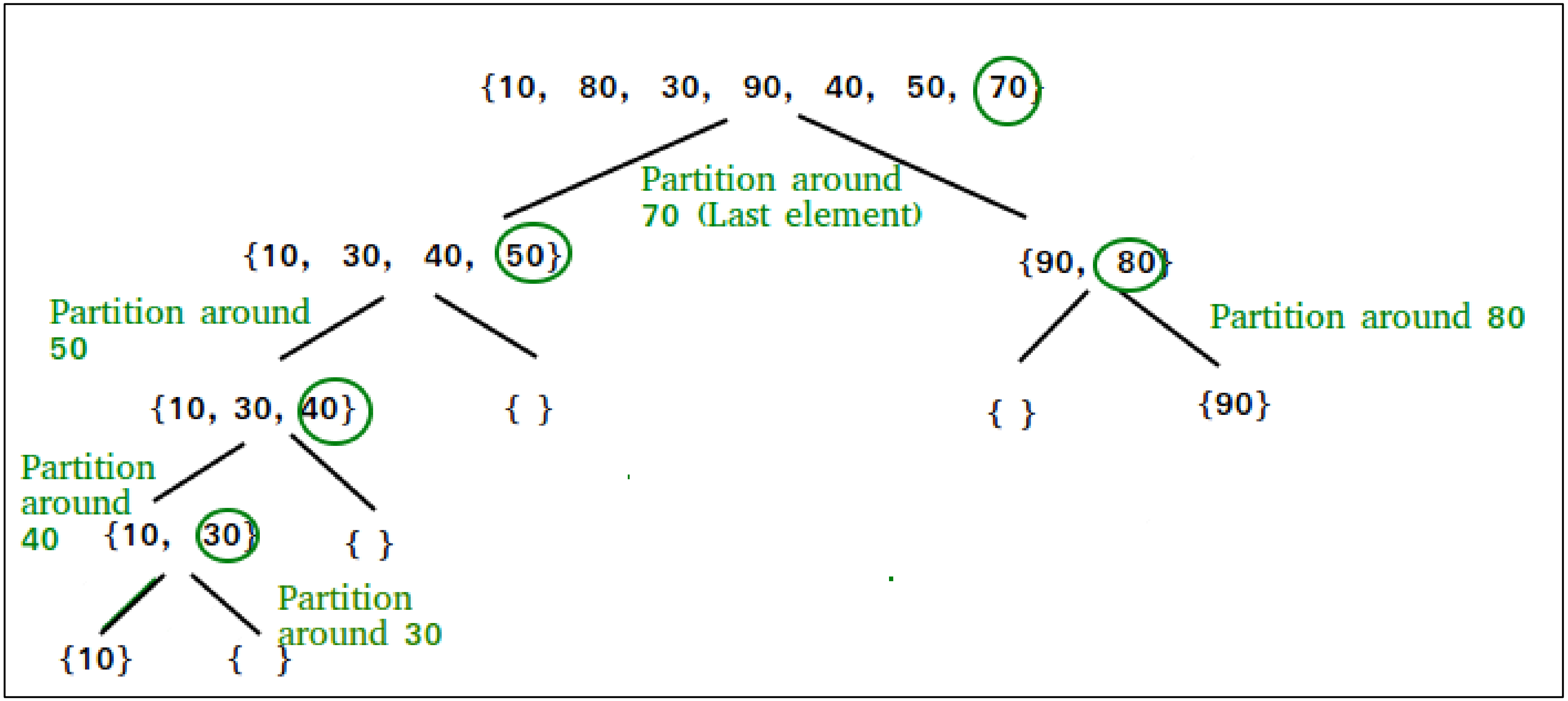
Partition the array

Recursively sort left subarray

Recursively sort right subarray

Stop recursion if one element

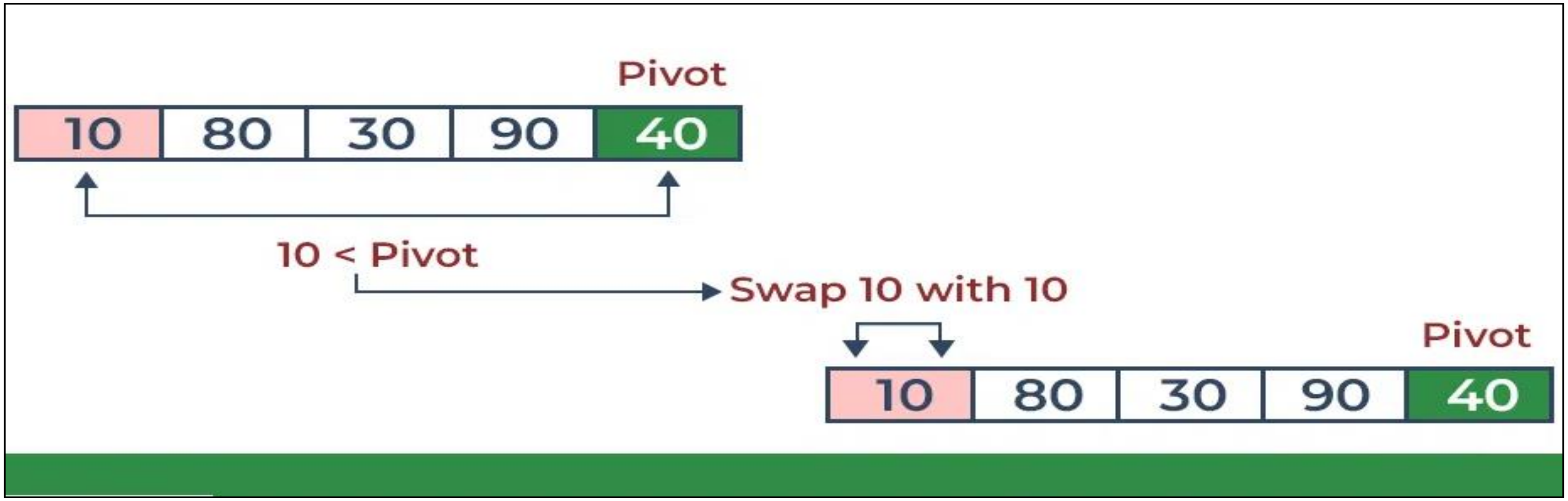
VI. Quick Sort



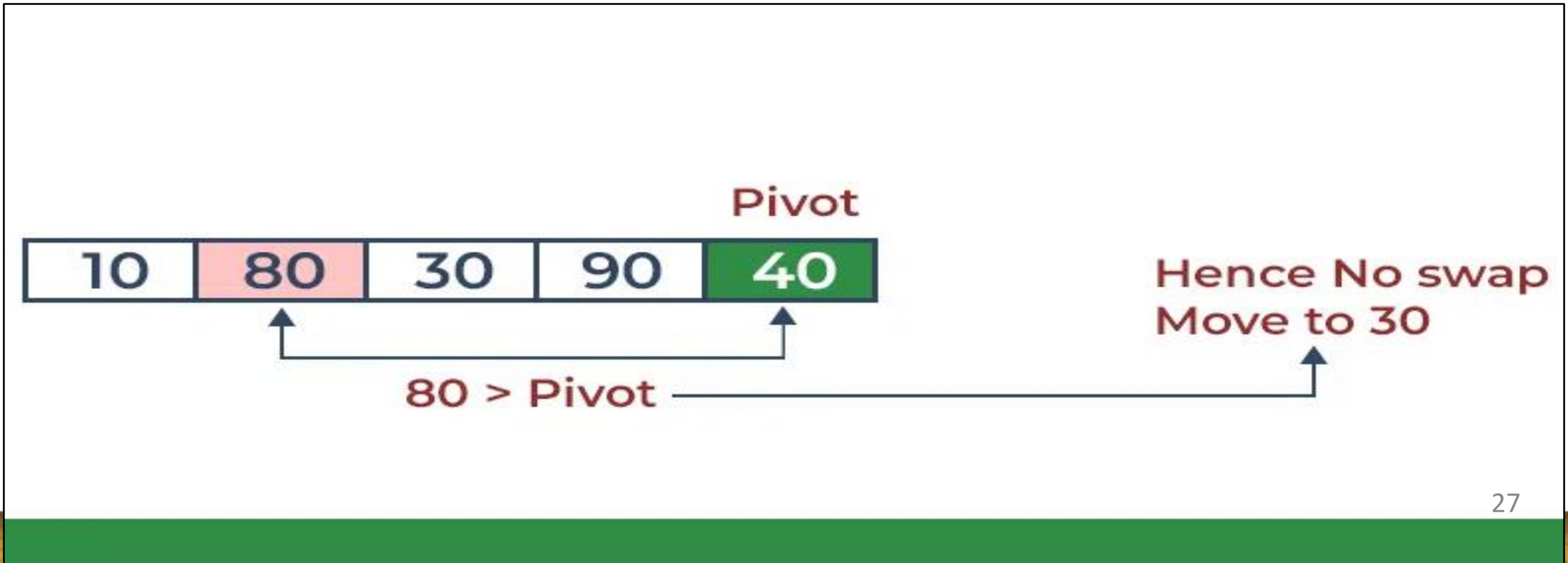
Consider: arr[] = {10, 80, 30, 90, 40}.

Compare 10 with the pivot and as it is less than pivot arrange it accordingly.

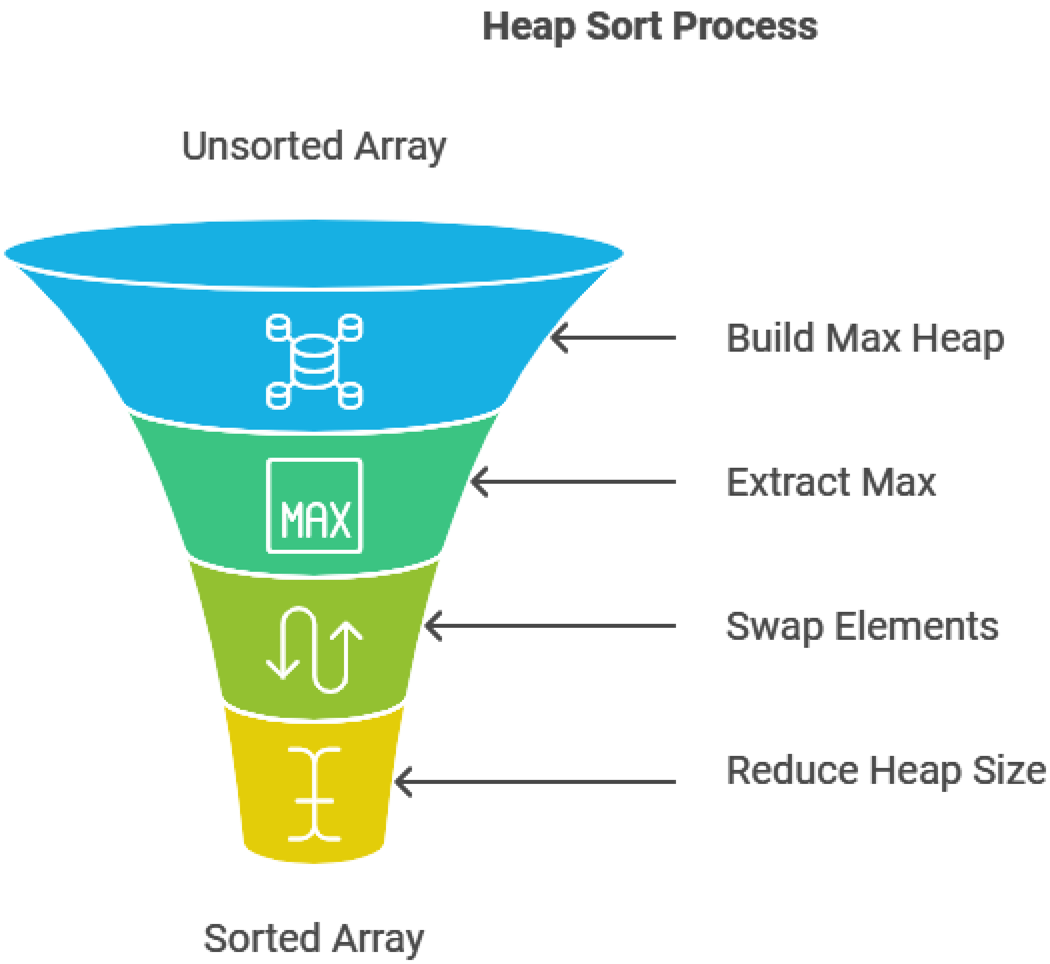
VI. Quick Sort



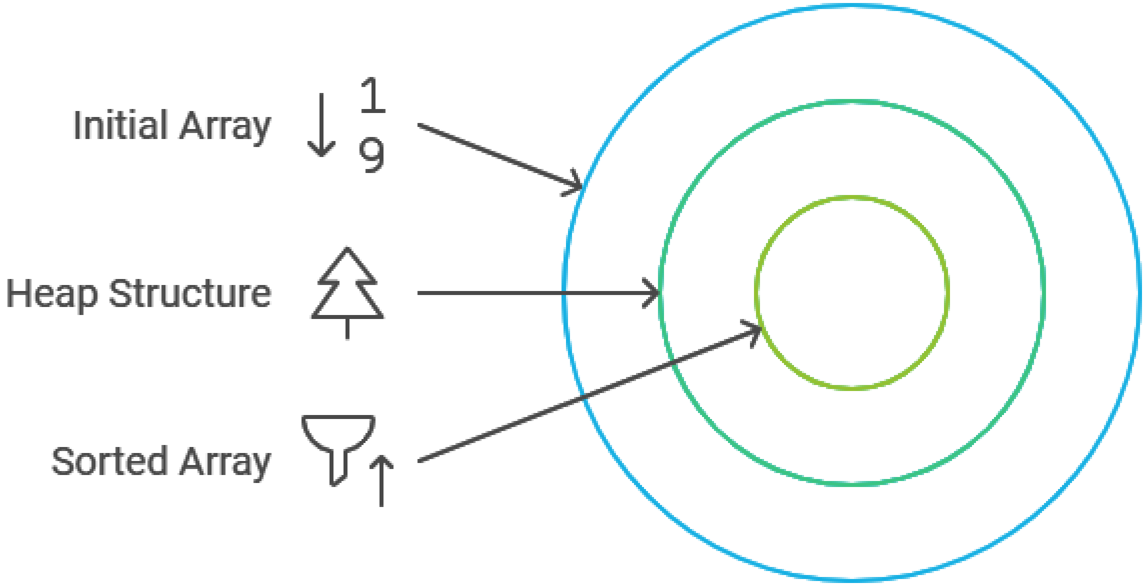
Compare 80 with the pivot. It is greater than pivot.



VII. Heap Sort

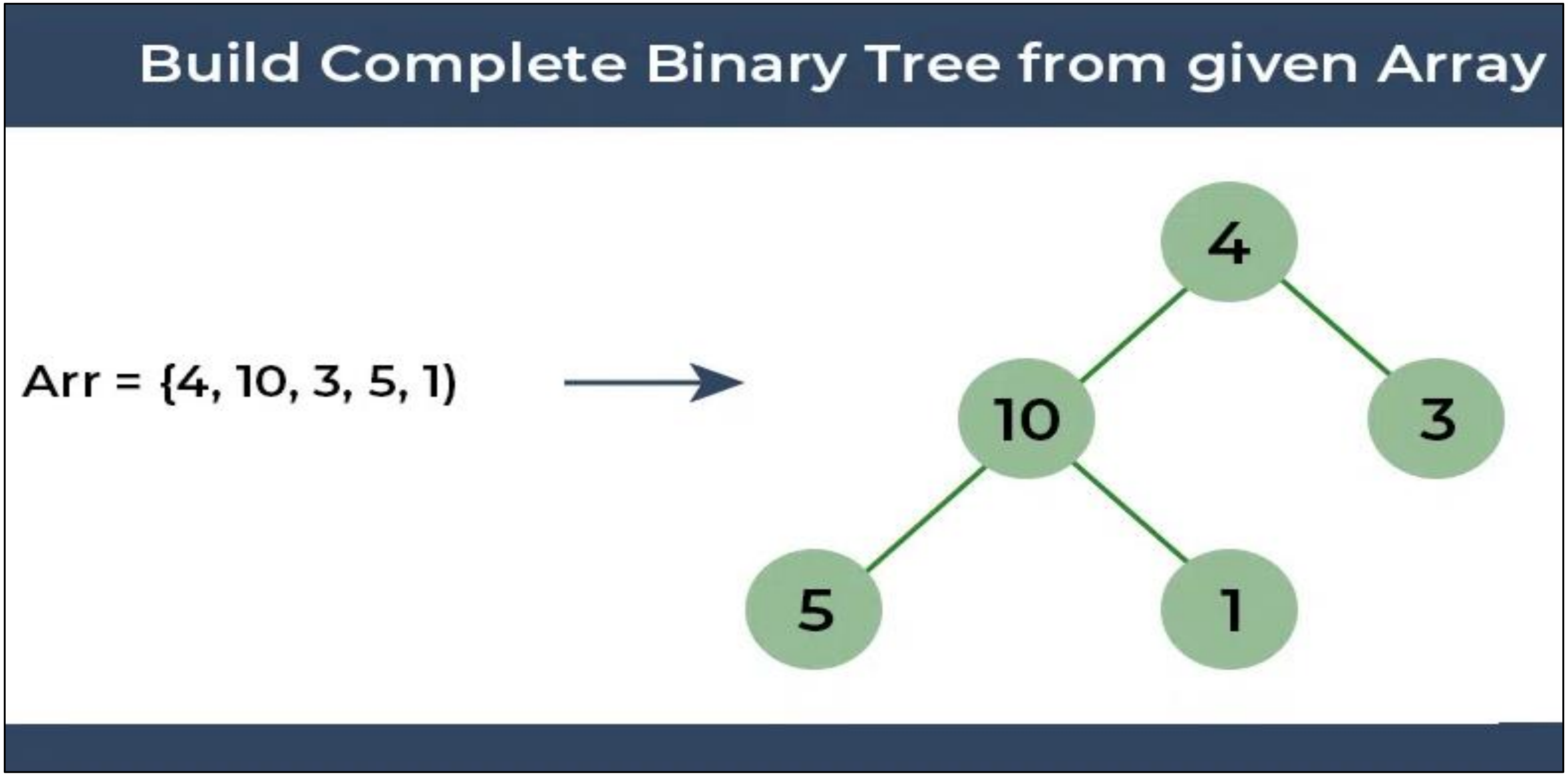


VII. Heap Sort



- Consider the array: arr[] = {4, 10, 3, 5, 1}.
- To understand heap sort more clearly, let’s take an unsorted array and try to sort it using heap sort.
- Consider the given array as Complete Binary Tree (For every index i, the left child is at index 2*i + 1 and right at 2*i + 2.

VII. Heap Sort



Transform the array into max heap:

- To transform a heap into a max-heap, the parent node should always be greater than or equal to the child nodes
- Now, 4 as a parent is smaller than the child 5, thus swap both of these again and the resulted heap and array should be like this:

VII. Heap Sort

```
using System; // C# program for implementation of Heap Sort
public class HeapSort {
   public void sort(int[] arr)
   {    int N = arr.Length;   // Build heap (rearrange array)
      for (int i = N / 2 - 1; i >= 0; i--)
         heapify(arr, N, i);
     for (int i = N - 1; i > 0; i--) {// One by one extract an element from heap
         int temp = arr[0]; // Move current root to end
         arr[0] = arr[i];  arr[i] = temp;
         // call max heapify on the reduced heap
         heapify(arr, i, 0);       }
    }
```

```
    for (int i = N - 1; i > 0; i--) {// One by one extract an element from heap

         int temp = arr[0]; // Move current root to end

         arr[0] = arr[i];  arr[i] = temp;

         // call max heapify on the reduced heap

         heapify(arr, i, 0);       }

    }
```

VII. Heap Sort

```
// To heapify a subtree rooted with node i
which is an index in arr[]. n is size of heap
  void heapify(int[] arr, int N, int i)
  {   int largest = i; // Initialize largest as root
      int l = 2 * i + 1; // left = 2*i + 1
      int r = 2 * i + 2; // right = 2*i + 2
      // If left child is larger than root
      if (l < N && arr[l] > arr[largest])  largest = l;
      // If right child is larger than largest so far
      if (r < N && arr[r] > arr[largest])
          largest = r;  // If largest is not root
      if (largest != i) { int swap = arr[i];
          arr[i] = arr[largest];  arr[largest] = swap;
    // Recursively heapify the affected subtree
          heapify(arr, N, largest);       }
  }
```

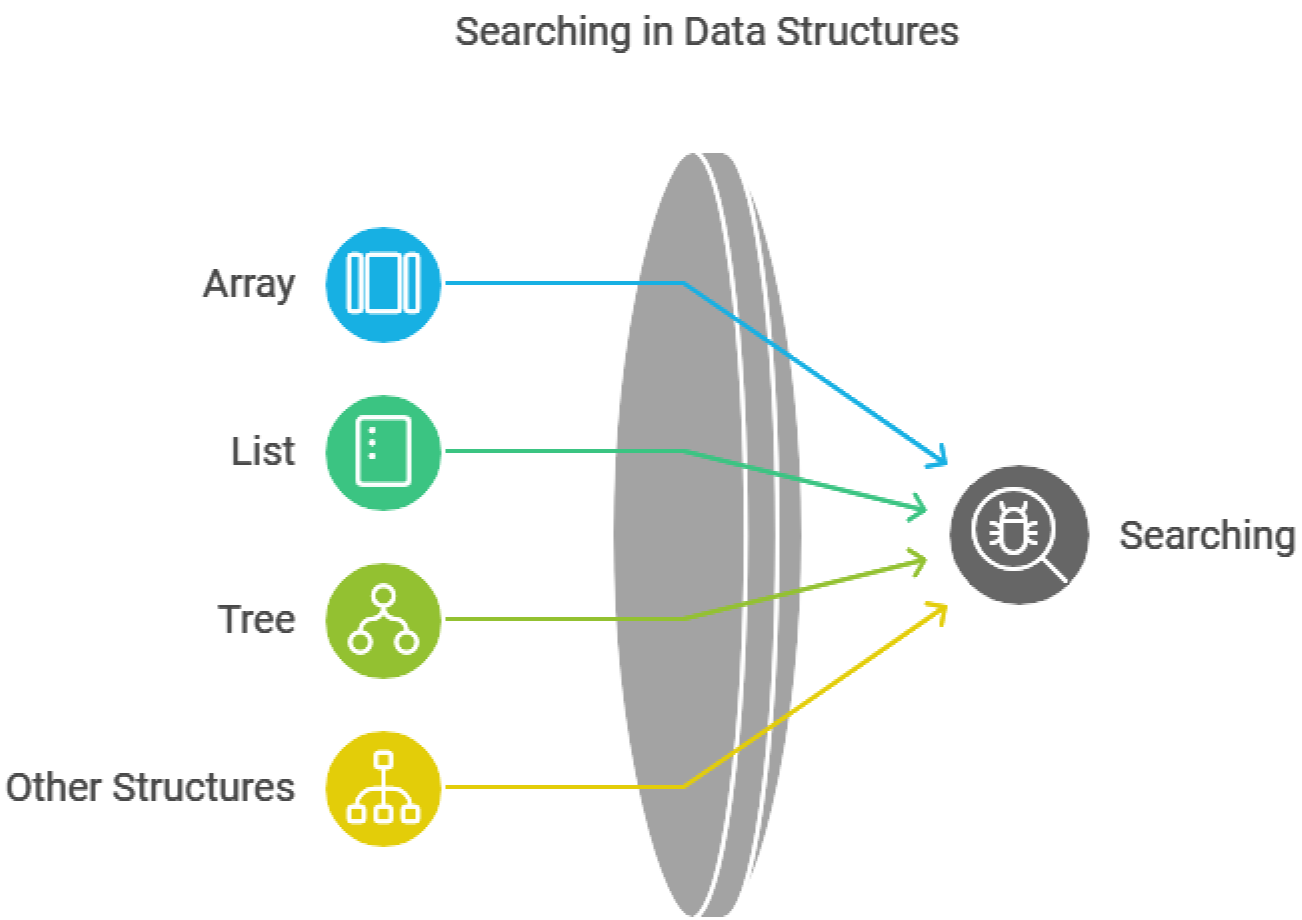
```
/* A utility function to print array of size n */
  static void printArray(int[] arr)
  {   int N = arr.Length;
      for (int i = 0; i < N; ++i)
          Console.Write(arr[i] + " ");
      Console.Read();
  }
  public static void Main()    // Driver's code
  {   int[] arr = { 12, 11, 13, 5, 6, 7 };
      int N = arr.Length;
      // Function call
      HeapSort ob = new HeapSort();
      ob.sort(arr);
      Console.WriteLine("Sorted array is");
      printArray(arr);   }
}
```

Output: Sorted array is 5 6 7 11 12 13

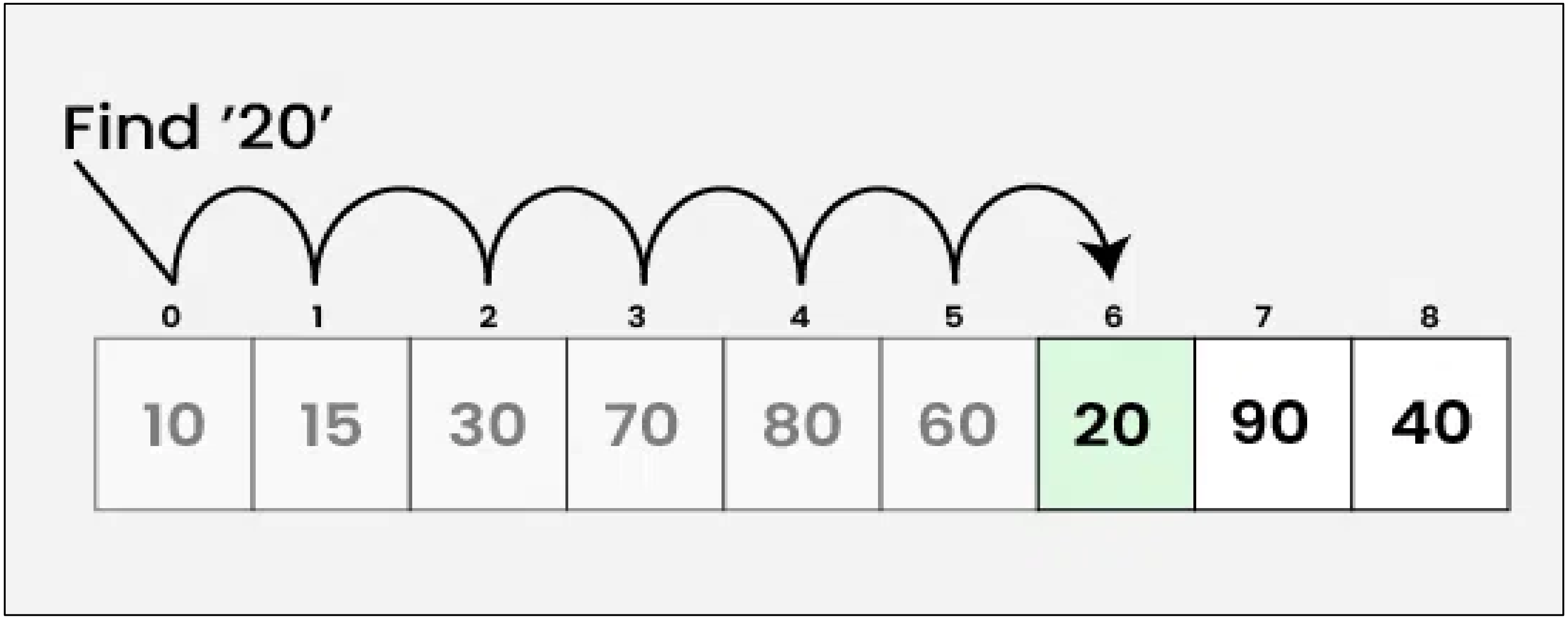
Part 2: Searching Algorithm

I. Searching Definition

I. Searching Definition

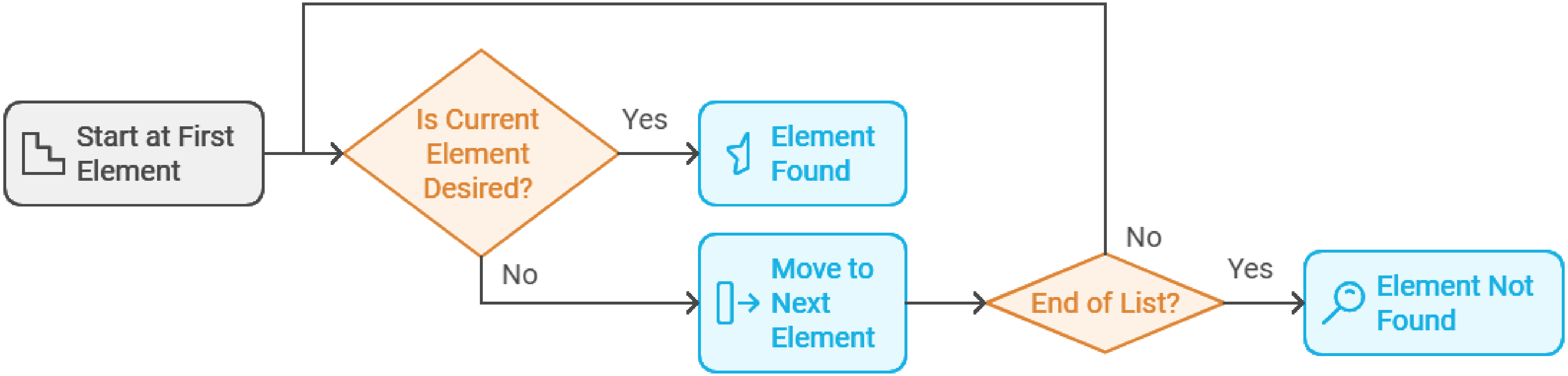


II. Linear Search



Find ’20’

0	1	2	3	4	5	6	7	8
10	15	30	70	80	60	20	90	40



Start at First Element

Is Current Element Desired?

Yes

Element Found

No

Move to Next Element

End of List?

No

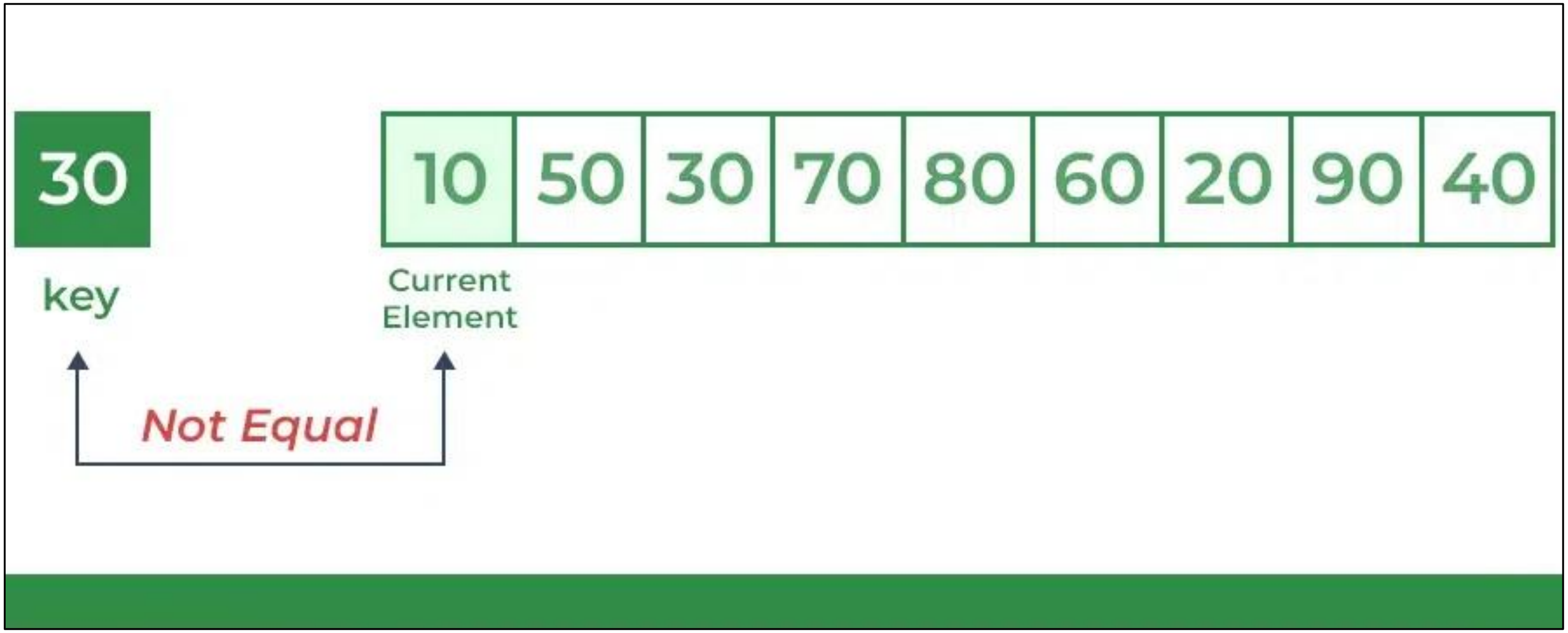
Yes

Element Not Found

II. Linear Search

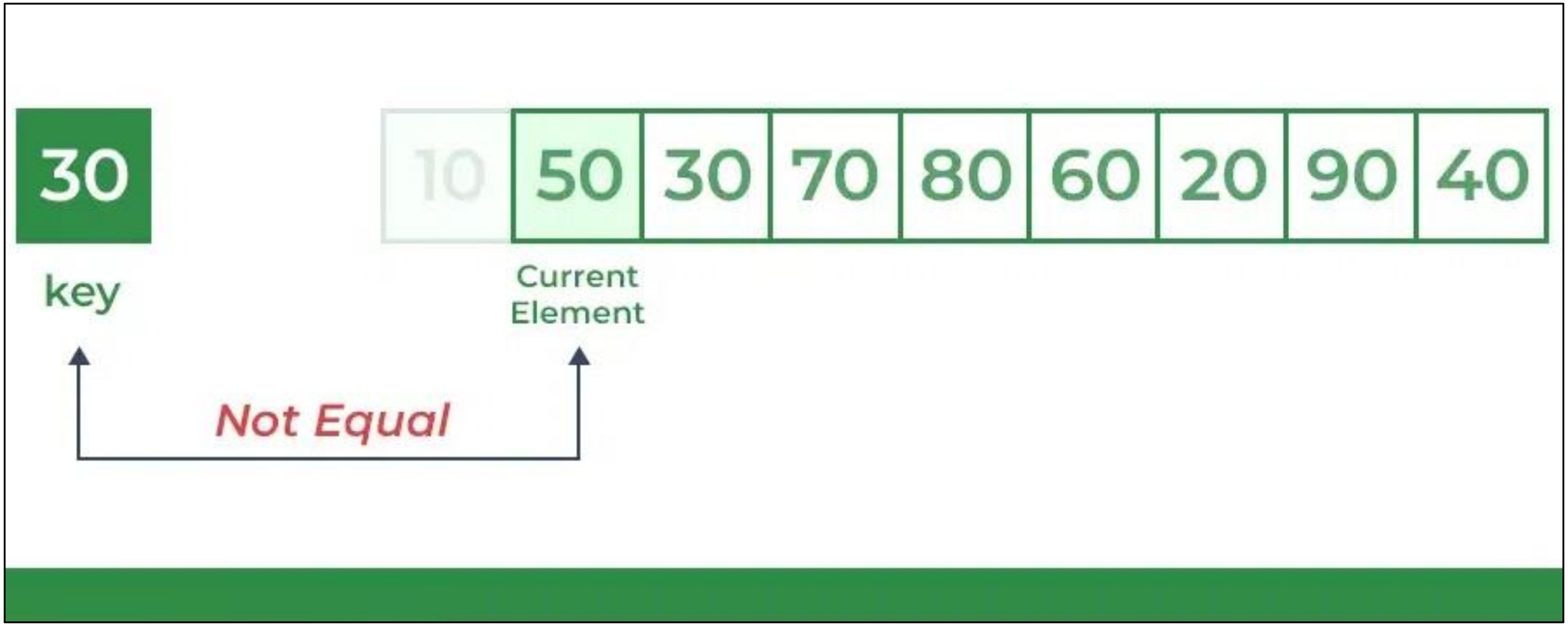
Consider the array arr[] = {10, 50, 30, 70, 80, 20, 90, 40} and key = 30

Step 1: Start from the first element (index 0) and compare key with each element (arr[i]).



II. Linear Search

- Comparing key with next element arr[1]. Since not equal, the iterator moves to the next element as a potential match.

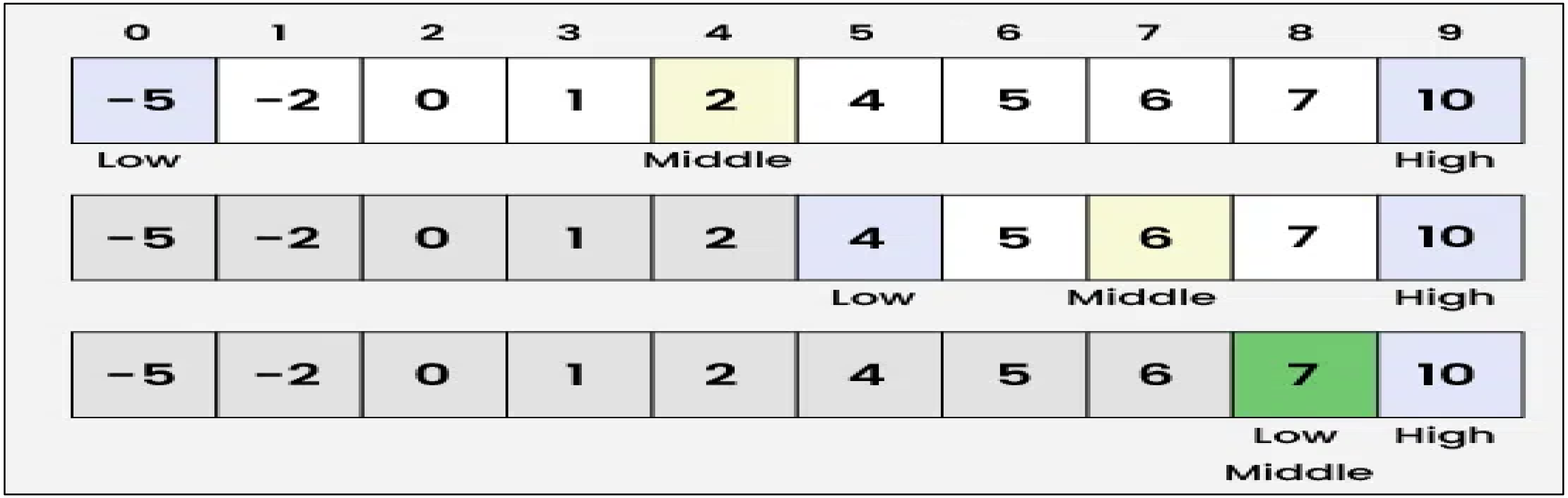


II. Linear Search

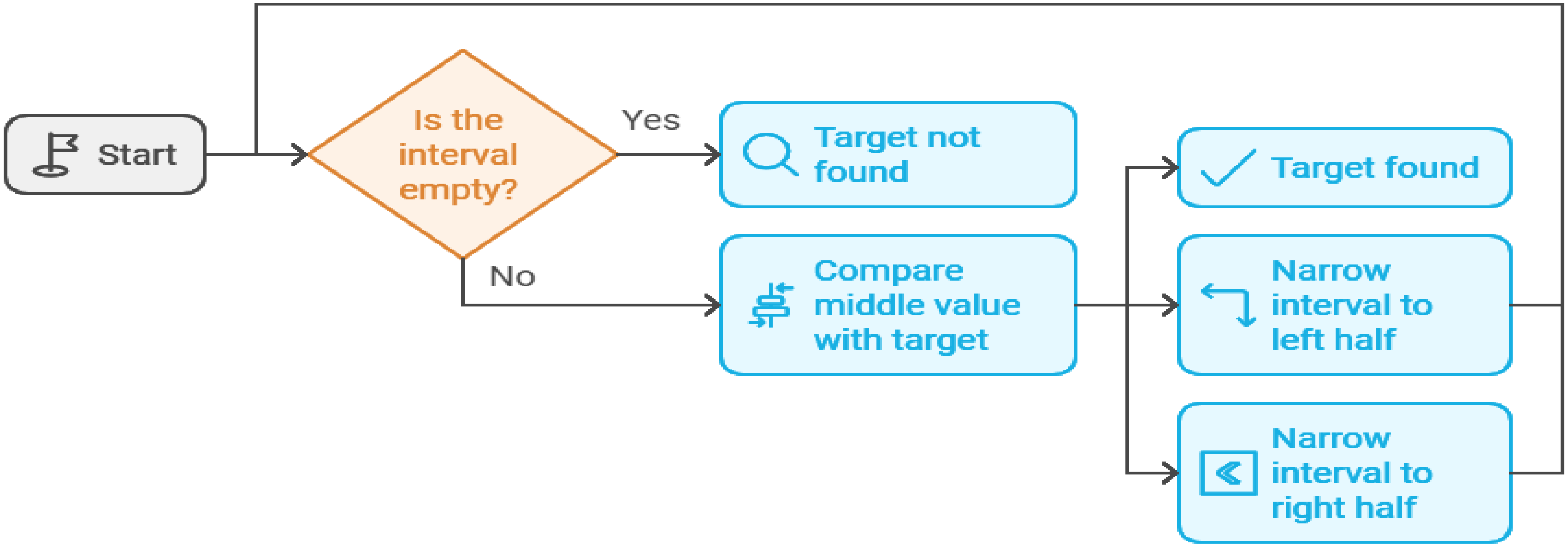
```
using System;// C# code to linearly search x in arr[].
class GFG {
    public static int search(int[] arr, int N, int x)
    {   for (int i = 0; i < N; i++) {  if (arr[i] == x)     return i;    }   return -1;
    }
    public static void Main() // Driver's code
    {   int[] arr = { 2, 3, 4, 10, 40 };
        int x = 10;
        int result = search(arr, arr.Length, x); // Function call
        if (result == -1)
          Console.WriteLine("Element is not present in array");
        else
          Console.WriteLine("Element is present at index "+ result);
    }
}
```

Output: Element is present at index 3

III. Binary Search



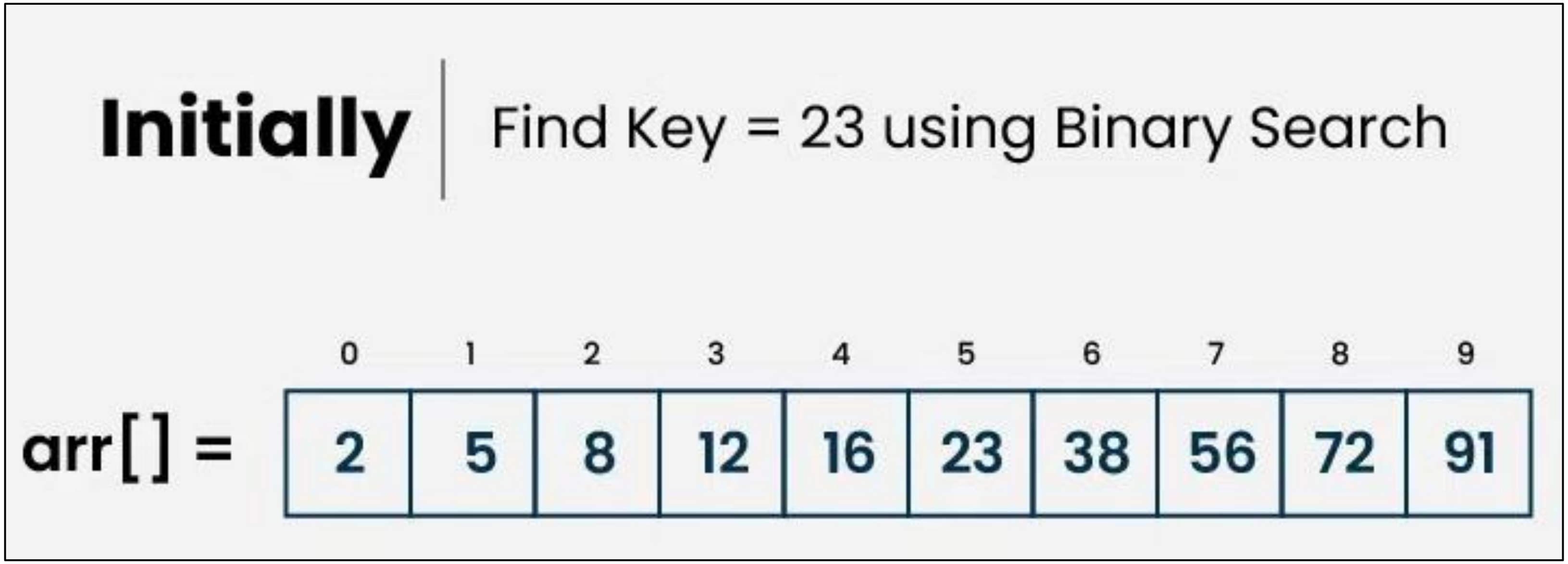
0	1	2	3	4	5	6	7	8	9
−5	−2	0	1	2	4	5	6	7	10
Low				Middle					High
−5	−2	0	1	2	4	5	6	7	10
					Low		Middle		High
−5	−2	0	1	2	4	5	6	7	10
								Low Middle	High


Start
Is the interval empty?
Yes
No
Target not found
Compare middle value with target
Target found
Narrow interval to left half
Narrow interval to right half

III. Binary Search

How does Binary Search Algorithm work?

Consider an array arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91}, and the target = 23.



Initially | Find Key = 23 using Binary Search

	0	1	2	3	4	5	6	7	8	9
arr[] =	2	5	8	12	16	23	38	56	72	91

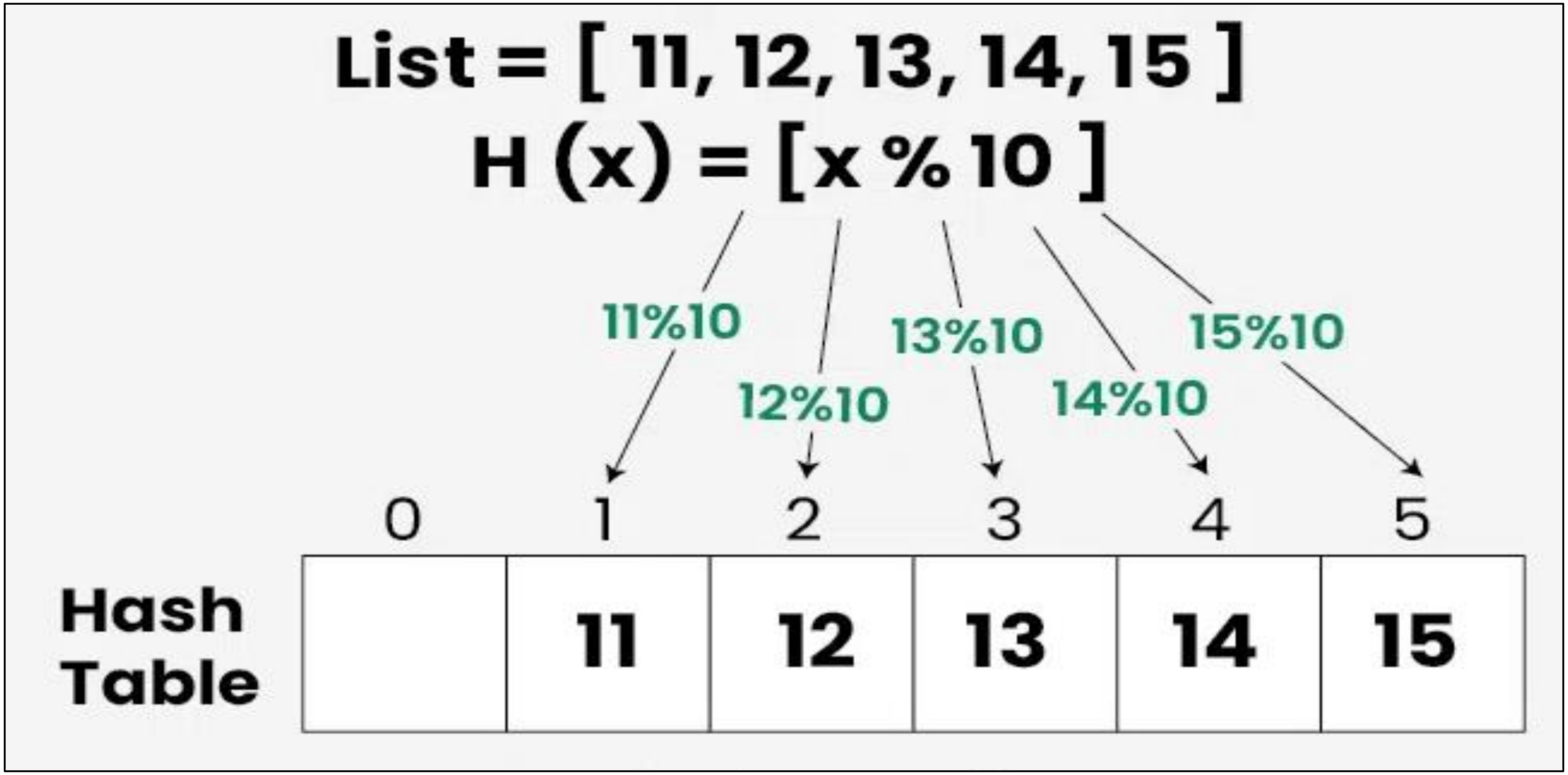
III. Binary Search

```
using System; // imple. of iterative Binary Search
class GFG {// Returns index of x if it is present in arr[]
   static int binarySearch(int[] arr, int x)
   {   int low = 0, high = arr.Length - 1;
      while (low <= high) { // Check if x is at mid
         int mid = low + (high - low) / 2;
        if (arr[mid] == x)  return mid;
         // If x greater, ignore left half
         if (arr[mid] < x)
            low = mid + 1;
         // If x is smaller, ignore right half
         else
            high = mid - 1;
      } // If we reach here, element was not present
```

```
        return -1;    }
public static void Main()// Driver code
   {   int[] arr = { 2, 3, 4, 10, 40 };
        int n = arr.Length;
        int x = 10;
        int result = binarySearch(arr, x);
        if (result == -1)
         Console.WriteLine("Element is not
present in array");
        else        Console.WriteLine("Element is
present at "+ "index " + result);    }
}
```

Output: Element is present at index 3

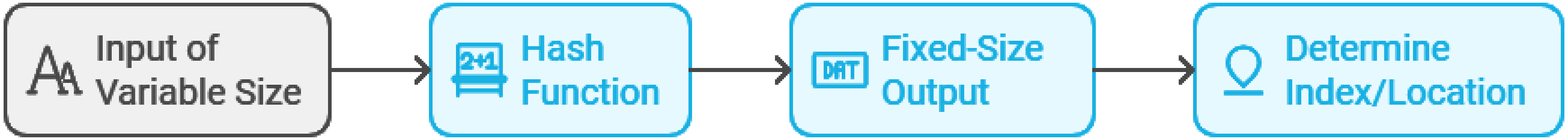
IV. Hashing Search



List = [11, 12, 13, 14, 15]

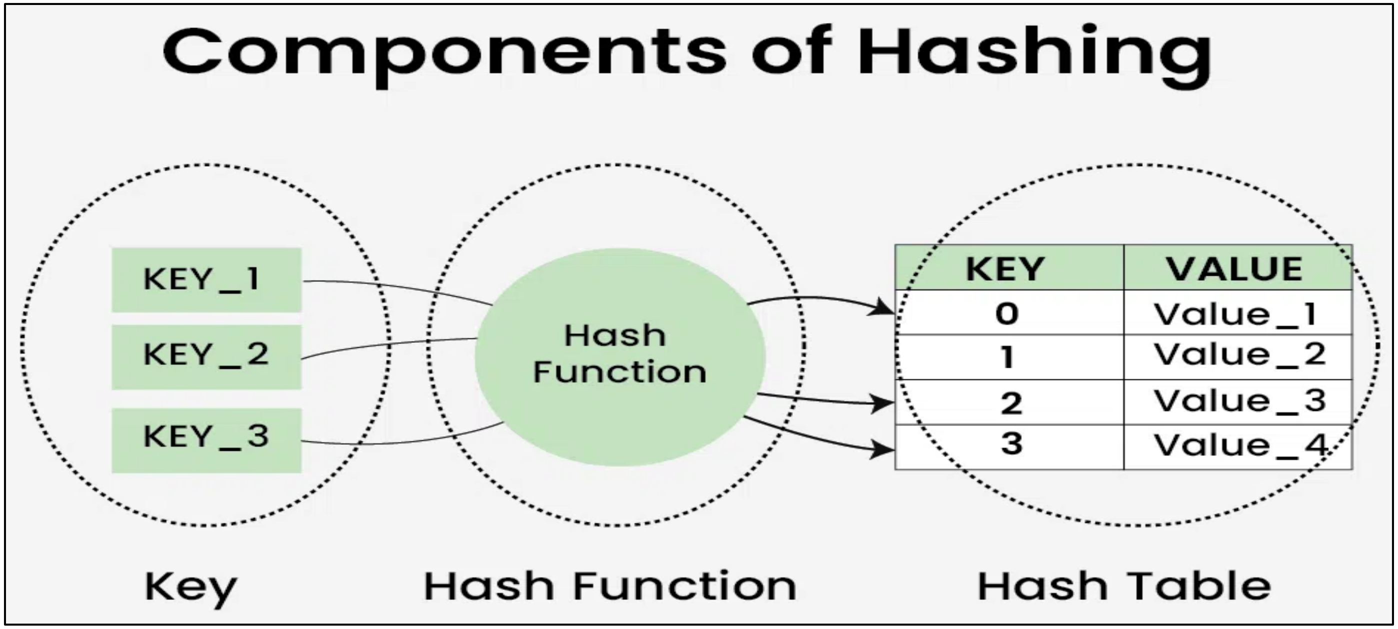
H (x) = [x % 10]

	0	1	2	3	4	5
Hash Table		11	12	13	14	15



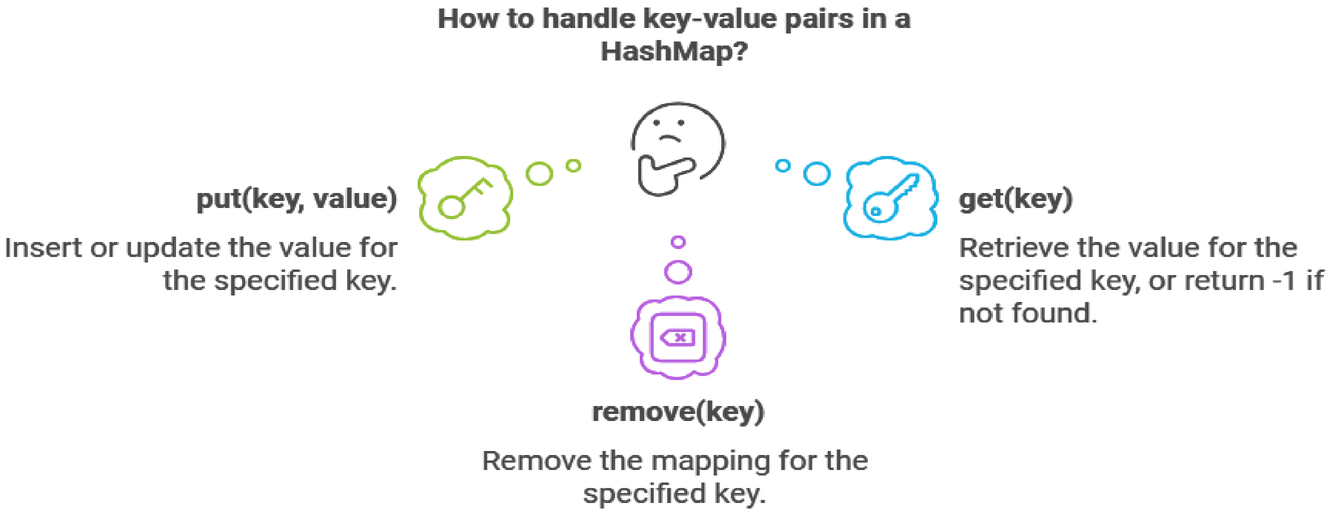
Input of Variable Size → Hash Function → Fixed-Size Output → Determine Index/Location

IV. Hashing Search



Components of Hashing

KEY	VALUE
0	Value_1
1	Value_2
2	Value_3
3	Value_4



How to handle key-value pairs in a HashMap?

put(key, value)

Insert or update the value for the specified key.

get(key)

Retrieve the value for the specified key, or return -1 if not found.

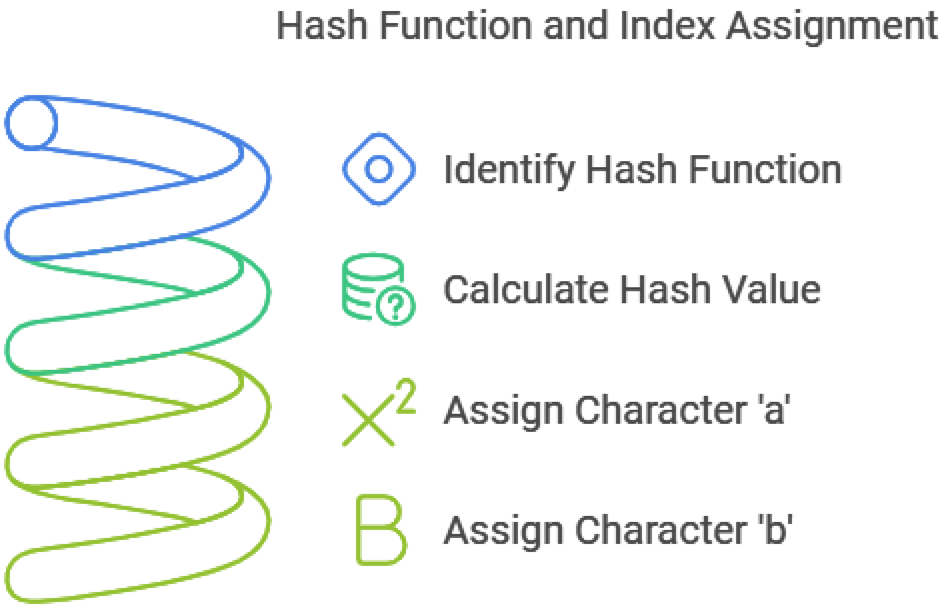
remove(key)

Remove the mapping for the specified key.

IV. Hashing Search

How does Hashing work?

Suppose we have a set of strings {“ab”, “cd”, “efg”} and we would like to store it in a table.



IV. Hashing Search

Hashing Strings into a Table

Calculate Numerical Value of "ab

Calculate Numerical Value of "cd

Calculate Numerical Value of "efg

Compute Hash for "ab

Compute Hash for "cd

Compute Hash for "efg

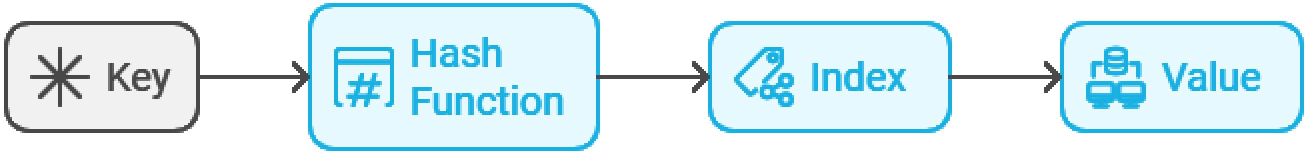
Store "ab" in Table

Store "cd" in Table

IV. Hashing Search

```
using System;  using System.Text;
using System.Security.Cryptography;
class HashingSearchExample
{
    static void Main()
    {   string message = "This is the original message!";
        // Convert the string into an array of bytes
        byte[ ] messageBytes = Encoding.UTF8.GetBytes(message);
        // Create the hash value from the array of bytes
        byte[ ] hashValue = SHA256.HashData(messageBytes);
        // Display the hash value to the console
        Console.WriteLine(Convert.ToHexString(hashValue));
    }
}
```

V. Hash Tables

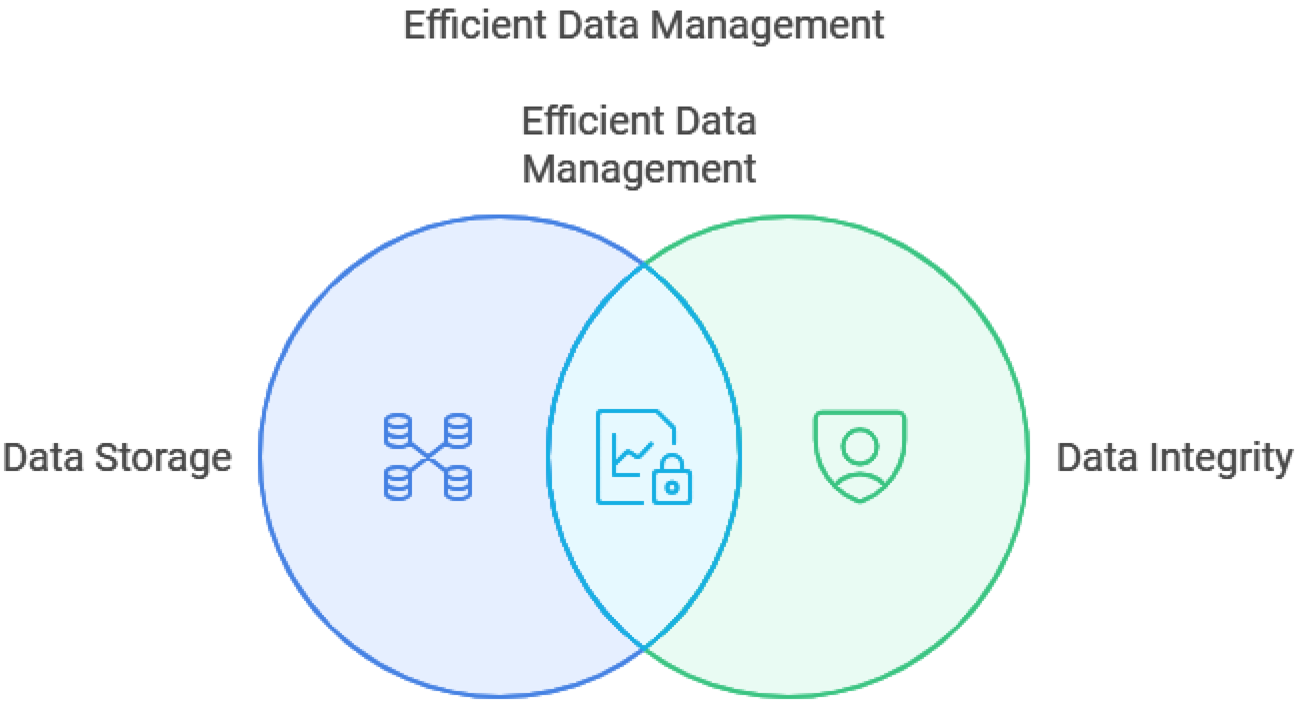


```
using System;
class Program {
   static void Main(string[] args) {
//initialize a string
      string s = "geeksforgeeks";
 // Using an array to store the count of each alphabet
 // by mapping the character to an index value
      int[] arr = new int[26];
 //Storing the count
     for (int i = 0; i < s.Length; i++) {
         arr[s[i] - 'a']++;
       }
       //Search the count of the character
       char ch = 'e';
       Console.WriteLine("The count of " + ch + " is " +
arr[ch - 'a']);  // get count
   }
}
```

Output: The count of e is 4

V. Hash Tables

Applications of Hash Table:



V. Quizzes

1. What is a sorting algorithm?
2. What are the different types of sorting algorithms?
3. What is the Quick Sort algorithm? How does QuickSort work?
4. What are the advantages and dis of using Quick Sort over other sorting algorithms?
5. What is the difference between Quick Sort and Merge, Quick and Heap Sort?
6. What type of the following sorting code:

```
public int[] SortArray(int[] array, int leftIndex, int rightIndex)
{       var i = leftIndex;
        var j = rightIndex;
        var pivot = array[leftIndex];
   while (i <= j)
   {  while (array[i] < pivot)    {   i++;       }
       while (array[j] > pivot)    {   j--;       }
```

V. Quizzes

```
     if (i <= j)  {   int temp = array[i];
                 array[i] = array[j];  array[j] = temp;  i++; j--; }
    }//While
      if (leftIndex < j)  SortArray(array, leftIndex, j);
      if (i < rightIndex)  SortArray(array, i, rightIndex);
      return array;
    }
```

7. What is Searching? What is the Binary Search?
8. What is Linear Search (Sequential Search), Binary VS Linear Search ?

V. Quizzes

7. What type of the following searching code:

```
int[] Y = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int x = 5,  i = 0, j = Y.Length - 1, k;
do {     k = (i + j) / 2;
   if (Y[k] < x)    i = k;
      else     j = k;
    } while (Y[k] != x && i < j);
    if (Y[k] == x)  Console.WriteLine("x is in the array");
    else Console.WriteLine("x is not in the array");
}
```

THANK
YOU!