Lecture 3

Introduction to Data Structures

Data Structure

- A particular way of storing and organizing data in a computer so that it can be used efficiently by operations.

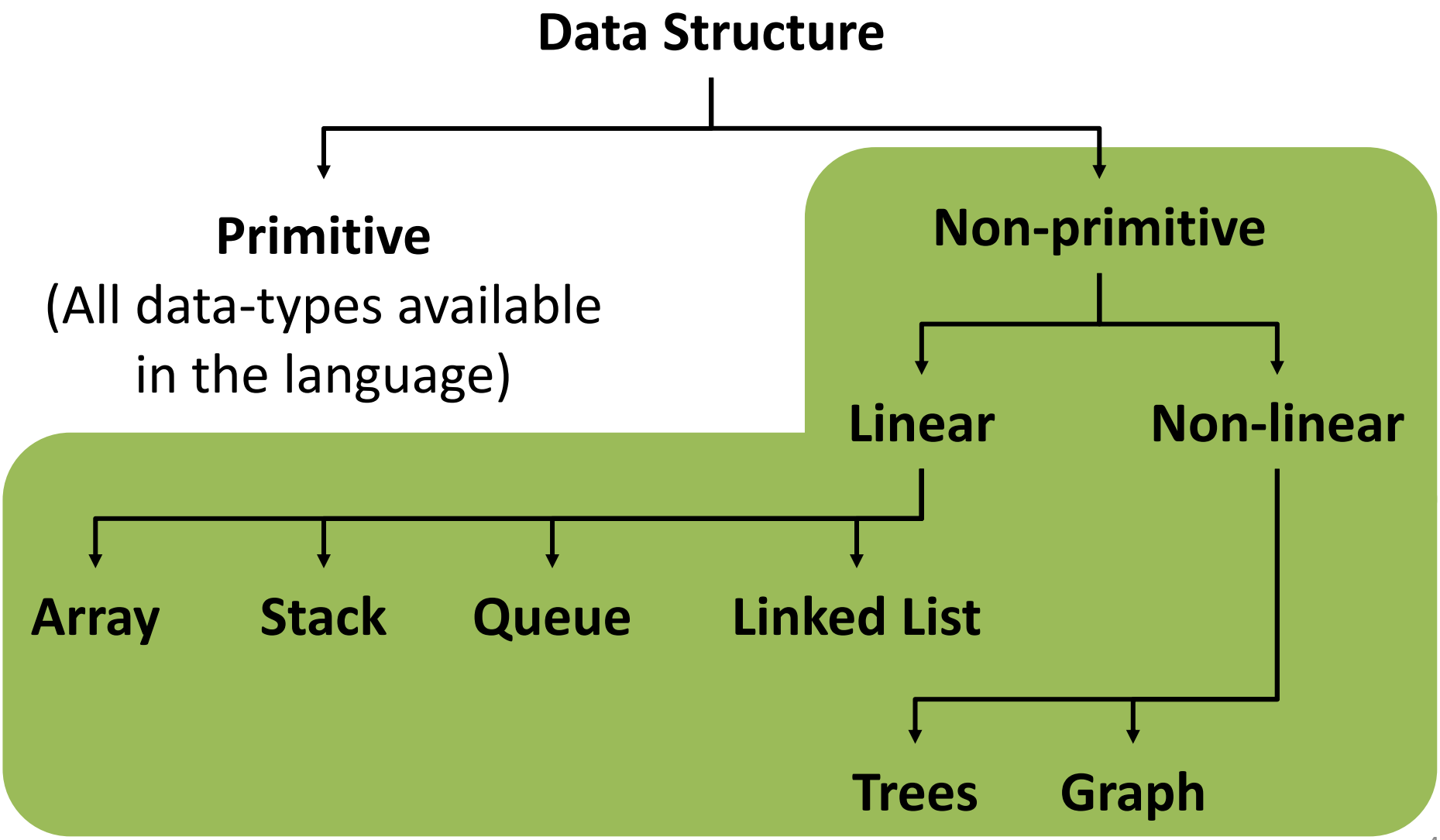
- They provide a means to manage large amounts of data efficiently, such as large databases.

- Data are simply values or set of values and Data Structure is organized collection of data.

Study of Data Structure Includes

- Logical description of data structure.
- Implementation of data structure.
- Quantitative analysis of data structure, this include amount of memory, processing time.

Classification of Data Structures



Algorithm

- A set of well defined instructions in sequence to solve a problem.
- Usually a high-level description of a procedure that manipulates well-defined input data to produce desired output data.
- Example: **Find the sum of two numbers**
  1. Take FIRST number as an input.
  2. Take SECOND number as an input.
  3. Add these two numbers.
  4. Output the result.

Contd…

Characteristics of a good algorithm:

- **Definiteness:** Has clear and unambiguous steps.
- **Finiteness:** Should terminate.
- **Input/Output:** Has a defined set of inputs and outputs.
- **Effectiveness:** Should be effective and correct.

Handling Arrays

Array ADT

```
int student [4];
float marks[10];
```

- The simplest but useful data structure.
- Assign single name to a homogeneous collection of instances of one abstract data type.
  - All array elements are of same type, so that a pre−defined equal amount of memory is allocated to each one of them.
- Individual elements in the collection have an associated index value that depends on array dimension.

Contd…

- One-dimensional and two-dimensional arrays are commonly used.
- Multi−dimensional arrays can also be defined.

- Usage:
  - Used frequently to store relatively permanent collections of data.
  - Not suitable if the size of the structure or the data in the structure are constantly changing.

Array Basic Operations

Operations on Linear Data Structures

- Traversal
- Insertion
- Deletion
- Search – Linear and Binary.

TRAVERSAL

Processing each element in the array.

Example – Print all the array elements.

Algorithm arrayTraverse(A,n)

Input: An array **A** containing **n** integers.

Output: All the elements in **A** get printed.

1. for i = 0 to n-1 do
2. Print A[i]

```
1. int arrayTraverse(int arr[], int n)
2. {
3.    for (int i = 0; i < n; i++)
4.          cout << "\n" << arr[i];
5. }
```

Example – Find minimum element in the array.

Algorithm arrayMinElement(A,n)
Input: An array **A** containing **n** integers.
Output: The minimum element in **A**.

```
1.  min = 0
2.  for i = 1 to n-1 do
3.        if A[min] > A[i]
4.              min = i
5.  return A[min]
```

```
1.  int arrayMinElement(int arr[], int n)
2.  {  int min = 0;
3.     for (int i = 1; i < n; i++)
4.     {   if (arr[i] < arr[min])
5.               min = i;
6.     }
7.     return arr[min];
8.  }
```

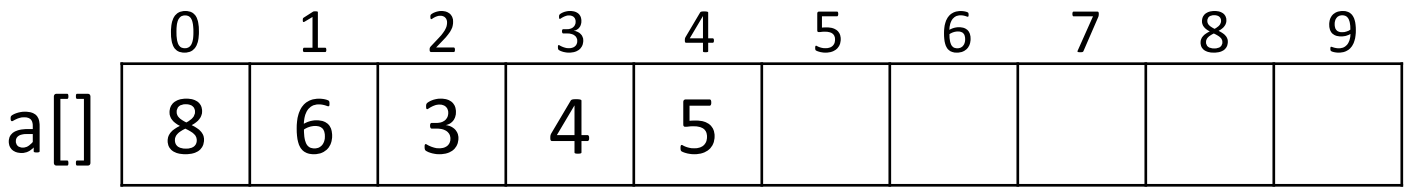
Insertion

Insert an element in the array

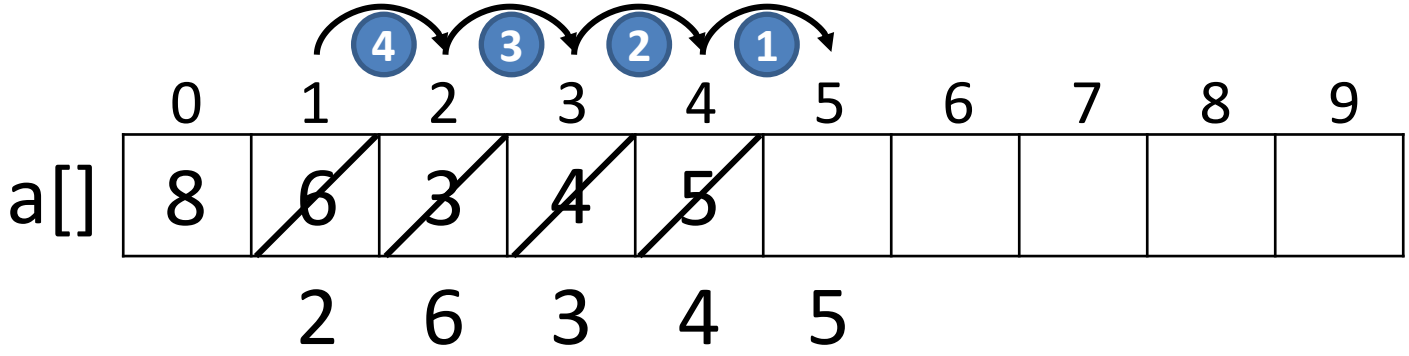
Deletion

Delete an element from the array

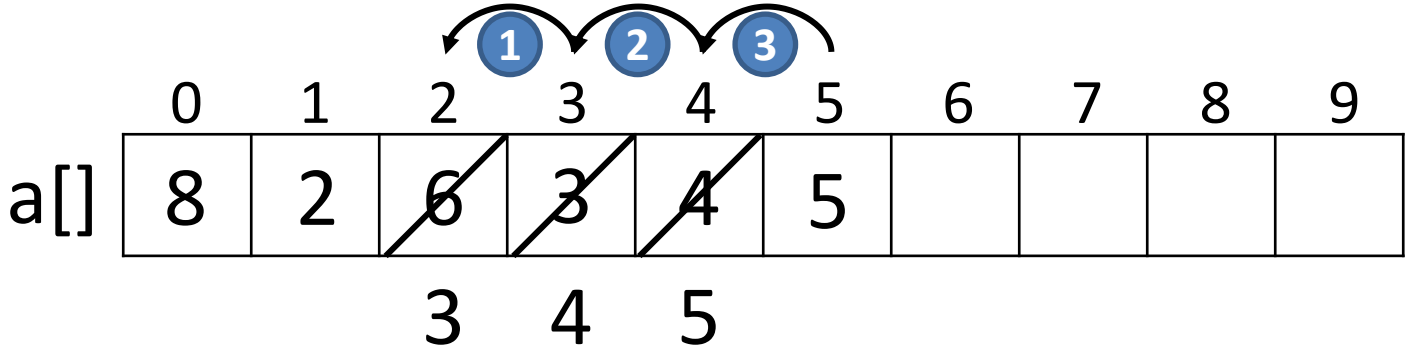
Insertion and Deletion



- Insert 2 at index 1



- Delete the value at index 2



Algorithm – Insertion

Algorithm insertElement(A,n,num,indx)

Input: An array **A** containing **n** integers and the number **num** to be inserted at index **indx**.

Output: Successful insertion of **num** at **indx**.

1. for i = n – 1 to indx do
2. A[i + 1] = A[i]
3. A[indx] = num
4. n = n + 1

```
1.  void insert(int a[], int num, int pos)
2.  {   for(int i = n-1; i >= pos; i--)
3.          a[i+1] = a[i];
4.      a[pos] = num;
5.      n++;
6.  }
```

Algorithm – Deletion

Algorithm deleteElement(A,n,indx)
Input: An array **A** containing **n** integers and the index **indx** whose value is to be deleted.
Output: Deleted value stored initially at **indx**.

1. temp = A[indx]
2. for i = indx to n – 2 do
3. A[i] = A[i + 1]
4. n = n – 1
5. return temp

```
1.  int deleteElement(int a[], int pos)
2.  {   int temp = a[pos];
3.      for(int i = pos; i <= n-2; i++)
4.          a[i] = a[i+1];
5.      n--;
6.      return temp;
7.  }
```

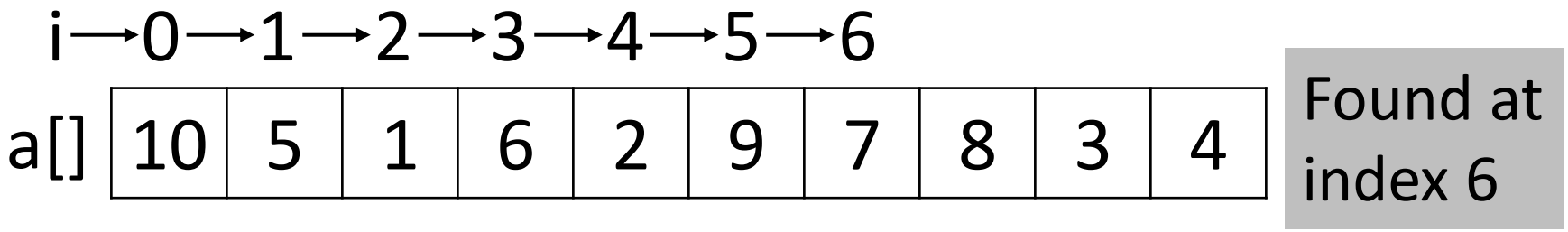
Search

Find the location of the element with a given value.

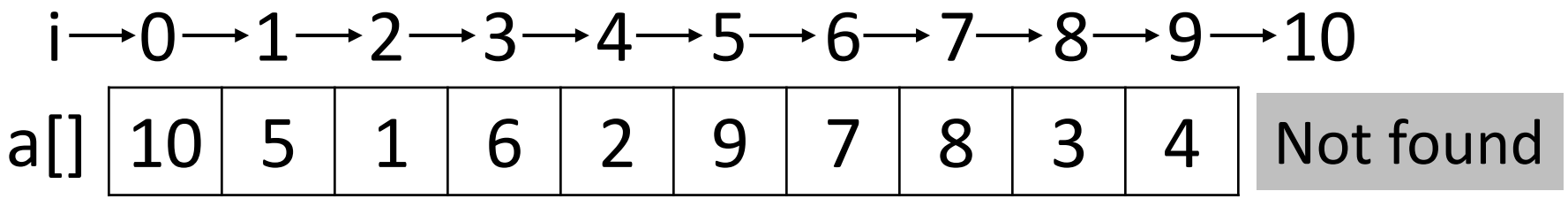
Linear Search

- Used if the array is unsorted.
- Example:

Search 7 in the following array



Search 11 in the following array



Contd…

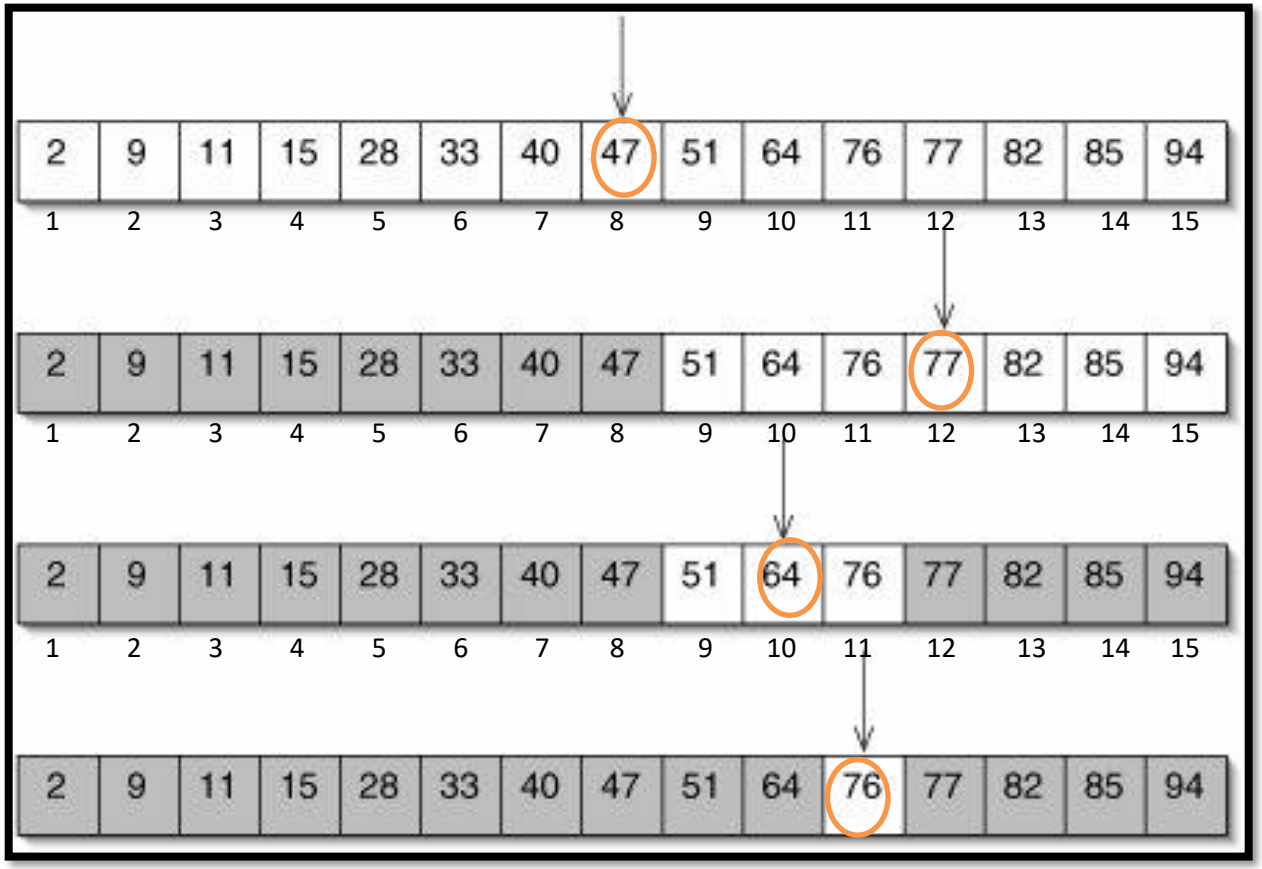
Algorithm linearSearch(A,n,num)
Input: An array **A** containing **n** integers and number **num** to be searched.
Output: Index of **num** if found, otherwise -1.

```
1.  for i = 0 to n-1 do
2.      if A[i] == num
3.          return i
4.  return -1
```

```
1.  int linearSearch(int a[], int n, int num)
2.  {  for (int i = 0; i < n; i++)
3.       if (a[i] == num)
4.            return i;
5.     return -1;
6.  }
```

Binary Search

- Works on sorted or alphabetically arranged data.
- Used to find the location of a given item of information in data efficiently. *Example:* ***Search for 76 in the given array***



Binary Search Algorithm

Algorithm binarySearch(A,n,num)

Input: Array **A** containing **n** integers and number **num** to be searched.

Output: Index of **num** if found, otherwise -1.

```
1.  low= 0
2.  high = n-1
3.  while (low < high)
4.      mid = (low + high) / 2
5.      if A[mid] == num
6.              return mid
7.      else if A[mid] > num
8.              high = mid
9.      else
10.             low = mid + 1
11. return -1
```

```
1.  int binarySearch(int a[], int n, int num)
2.  {
3.      int m, l=0, h=n-1;
4.      while(l < h)
5.      {
6.          m=(l + h) / 2;
7.          if (a[m]==num)
8.              return m;
9.          else if (a[m]>num)
10.             h=m;
11.         else
12.             l=m+1;
13.     }
14.     return -1;
15. }
```

Thank You