Data Structure

Prerequisite:-

a) Fundamentals of C Programming

b) Discrete Structures

Course Objectives:

1. This course introduces basic idea of data structure while making aware of
2. methods and structure used to organize large amount of data.
3. It’s also aimed at developing skill to implement methods to solve specific
4. problems using basic data structures.
5. The course also provides career opportunities in design of data, implementation
6. of data, technique to sort and searching the data

Course Outcomes

CO1: Illustrate various technique to for searching, Sorting and hashing
CO2: Explain the significance of dynamic memory management Techniques
CO3: Design and analyze different linear data structure techniques to solve real world problem.
CO4: Implement non-linear data structure to find solution for given engineering applications.
CO5: Summarize different categories of data Structure

Semester	III	Teaching Scheme				Evaluation Scheme				
						Theory			Practical	
Term	ODD	Th	Tu	Pr	Credits	TAE	CAE	ESE	INT	EXT
Course Category	C	3	--	2	4	10	15	50	25	--
Course Code	UCSL 201 UCSP 201									
Teaching Mode	Online	5			Total	75			25	
Duration of ESE						100				

Horowitz, Sahani, “Fundamentals of Data Structures in C” second edition, Universities Press.

R. Gilberg, B. Forouzan, "Data Structures: A pseudo Code Approach with C++", Cengage Learning, ISBN 9788131503140.

YashwantKanetkar, “Let us C” and “Pointers in C” , BPB Publication

Thomos H. Corman, Charls E. Leiserson, Ronald E. Rivest, Clifford Stein,“Introduction to Algorithms”, Third Edition, Prentice Hall India Learning Pvt. Ltd.

Data Structures using C, Aron M. Tanenbaum, Pearson Education, 1 Edition(2003).

M. Weiss, "Data Structures and Algorithm Analysis in C++", 2nd edition, Pearson Education, 2002, ISBN-81-7808-670-0

Unit -I	Introduction Sorting and Searching Arrays:
Unit-II	Introduction: Linked List
Unit-III	Stack and Queue
Unit-IV	Trees and Binary Tree
Unit-V	Graph and Their applications

Data Structure

- Data Structures
- Classifications (Primitive & Non Primitive)
- Data structure Operations,

- Data Structure
  - Data: are simply a value are set of values of different type which is called data types like string, integer, char etc.
  - Structure: Way of organizing information, so that it is easier to use

In simple words we can define data structures as

- A particular way of organizing data in a computer so that it can be used efficiently.
- A scheme for organizing related pieces of information.

Classification of Data Structure

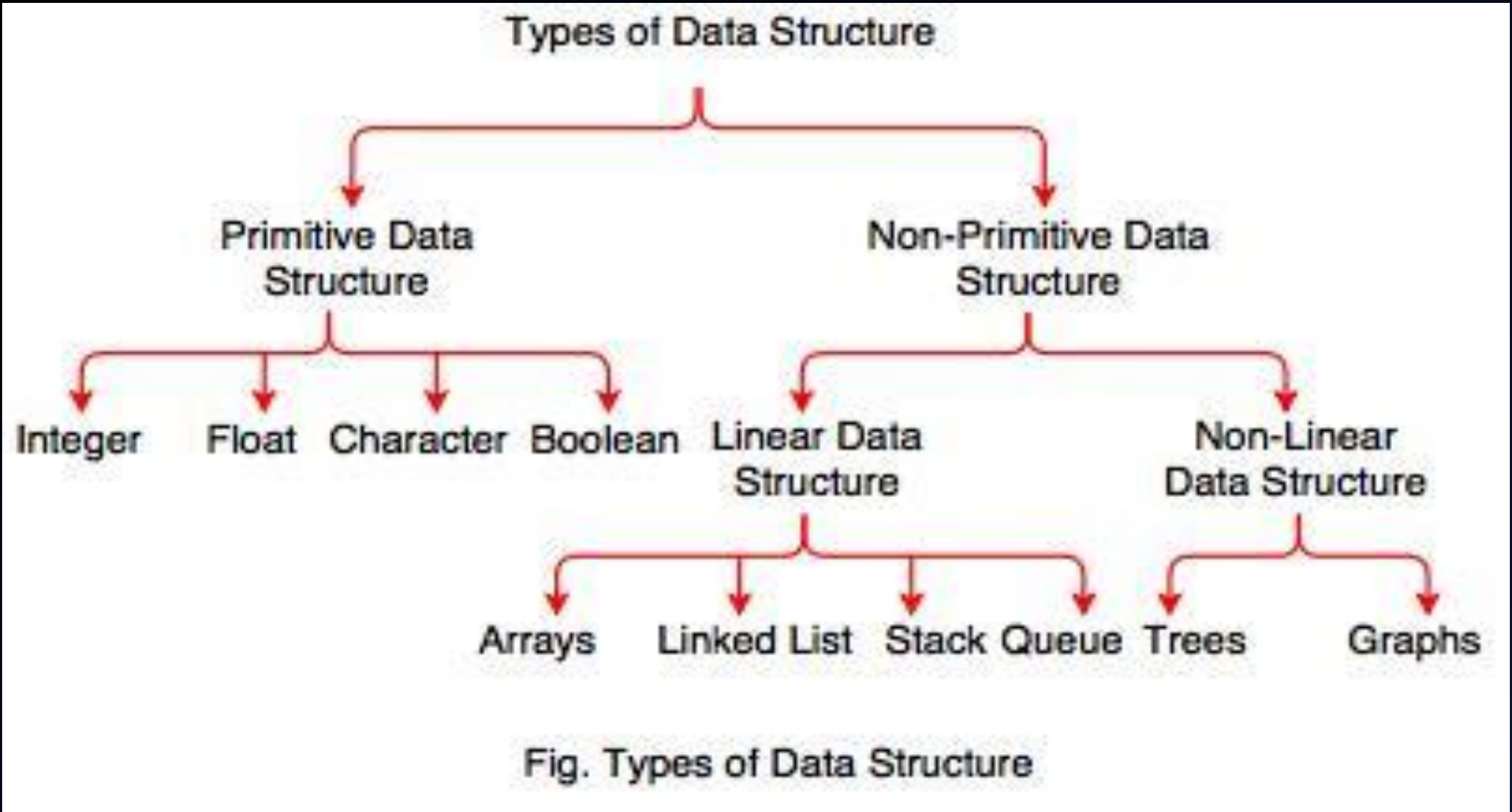


Fig. Types of Data Structure

- Primitive Data Structure:
  - A Primitive Data Structure used to represent the standard data types of any one of the computer languages (integer, Character, float etc.).
- Non - Primitive Data Structure :
  - Non Primitive Data Structure can be constructed with the help of any one of the primitive data structure. int a [10]
  - It can be structure and it is having a specific functionality.
  - It can be designed by user.
  - It can be classified as Linear and Non-Linear Data Structure.

Linear Data Structures:

A linear data structure traverses the data elements sequentially, in which only one data element can directly be reached.

Ex: Arrays, Linked Lists ,Stacks, Queues

Non-Linear Data Structures:

Every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure.

Ex: Trees, Graphs ,Heaps

Homogeneous and Non- Homogenous Data Structure

In Homogeneous Structure, all the elements are of same type. Example is arrays.

In Non-homogeneous structure, the elements may or may not be of same type. Example is records.

Static and Dynamic Data Structure

Static structures are ones whose sizes and structures, associated memory location are fixed at compile time. Int a[10]

Dynamic structures are ones which expand or shrink as required during the program execution and there associated memory location change.

Data Structure Operations

There are six basic operations that can be performed on data structure:-

- Traversing
- Searching
- Sorting
- Inserting
- Deleting
- Merging

(a) Traversing

Traversing means accessing and processing each element in the data structure exactly once.

This operation is used for counting the number of elements, printing the contents of the elements etc.

(b) Searching

Searching is finding out the location of a given element from a set of numbers.

(c) Sorting

Sorting is the process of arranging a list of elements in a sequential order. The sequential order may be descending order or an ascending order according to the requirements of the data structure.

(d) Inserting

Inserting an element is adding an element in the data structure at any position. After insert operation the number of elements are increased by one.

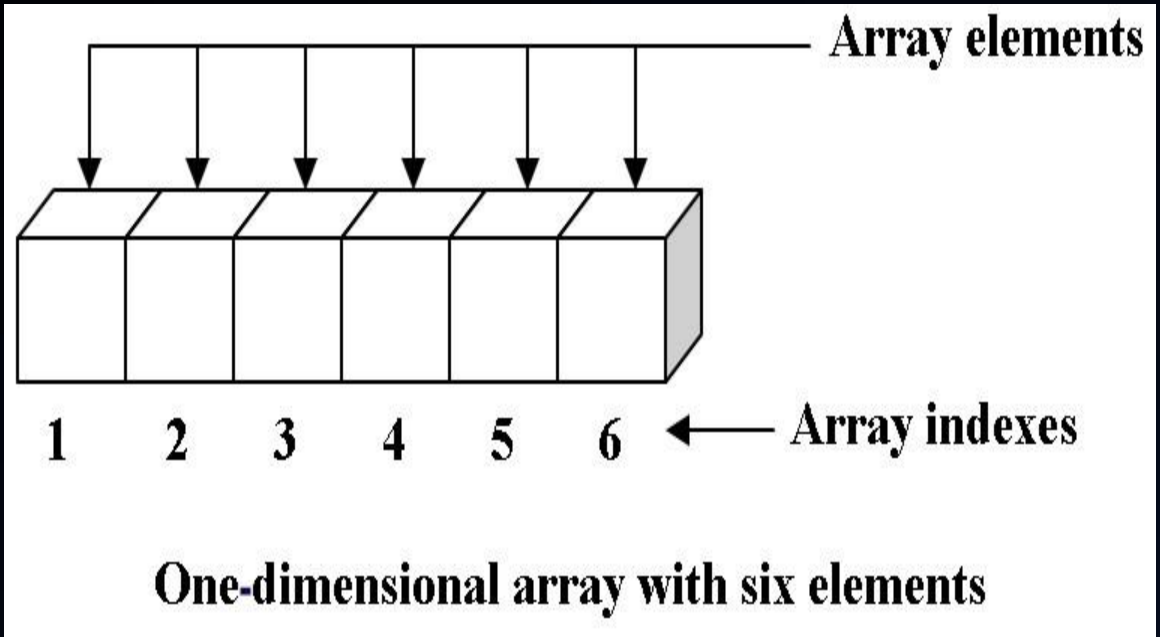
(e) Deleting

Deleting an element is removing an element in the data structure at any position. After deletion operation the number of elements are decreased by one.

(f) Merging

The process of combining the elements of two data structures into a single data structure is called merging.

Review of Arrays

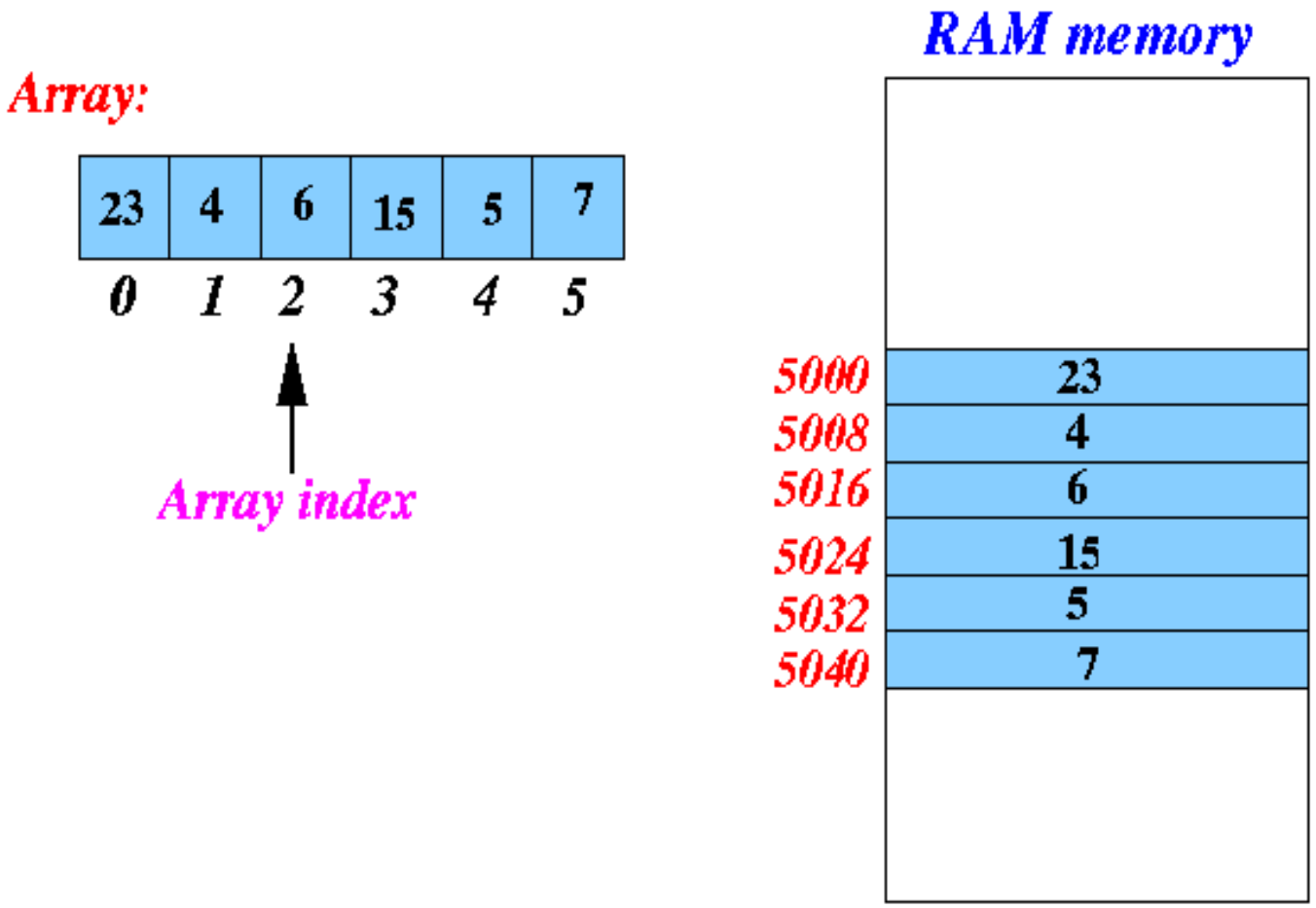


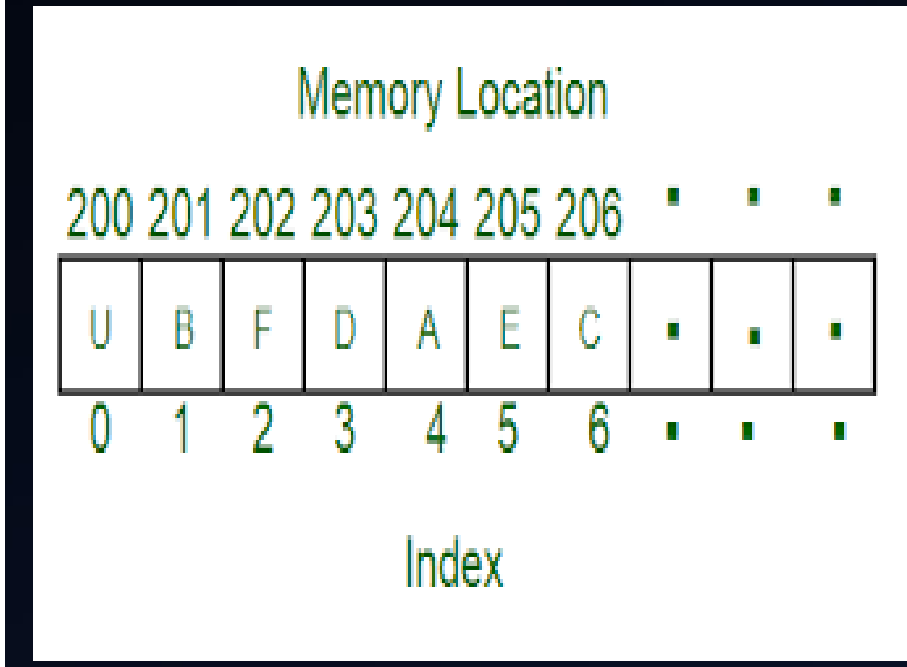
One-dimensional array with six elements

The array is an **abstract data type (ADT)** that holds a collection of elements accessible by an index.

- The elements stored in an array can be anything from primitives types
- An element is stored in a given index and they can be retrieved at a later time by specifying the same index.
- The Array (ADT) have one property, they store and retrieve elements using an index.

The Array data structure





Declaration of Arrays: Single and Multi

```
int group [10] ;
```

```
int table[2][3];
```

Initialization of Arrays

```
type array-name[size] = { list of values };
```

```
int number[3] = {4,5,9};
```

```
char name[ ] = {'j','o','h','n','\0'};
```

```
char name[ ] = "john";
```

```
int table[2][3] = {0,0,0,1,1,1};
```

```
int table[2][3] = {{0,0,0}, {1,1,1}};
```

```
int table[ ][3] = {{0,0,0}, {1,1,1}};
```

```
int table[2][3] = {1,1,2};
```

```
#include<stdio.h>
void main()
{
    int array[5];
    printf("Enter 5 numbers to store them in array \n");
    for(i=0;i<5;i++)
  {
    Scanf("%d", &array[i]);
  }
    printf("Element in the array are: \n");
    For(i=0;i<5;i++)
  {
    printf("Element stored at a[%d]=%d \n", i, array[i]);
  }
    getch();
}
```

```
#include<stdio.h>
int main()
{
int a[5][2]={{0,0},{1,2},{2,4},{3,6},{4,8}};
int i,j;
for(i=0;i<5;i++)
   {
for(j=0;j<2;j++)
 {
printf("a[%d][%d] = %d\n",i,j,a[i][j]);
   }
}
return 0;
}
```

```
for(i=0;i<r;++i)
for(j=0;j<c;++j)
{
printf("\n b[%d][%d]:",i+1,j+1);
scanf("%d",&b[i][j]);
}
```

Complexities

Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

Best Case − Minimum time required for program execution.

Average Case − Average time required for program execution.

Worst Case − Maximum time required for program execution.

Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

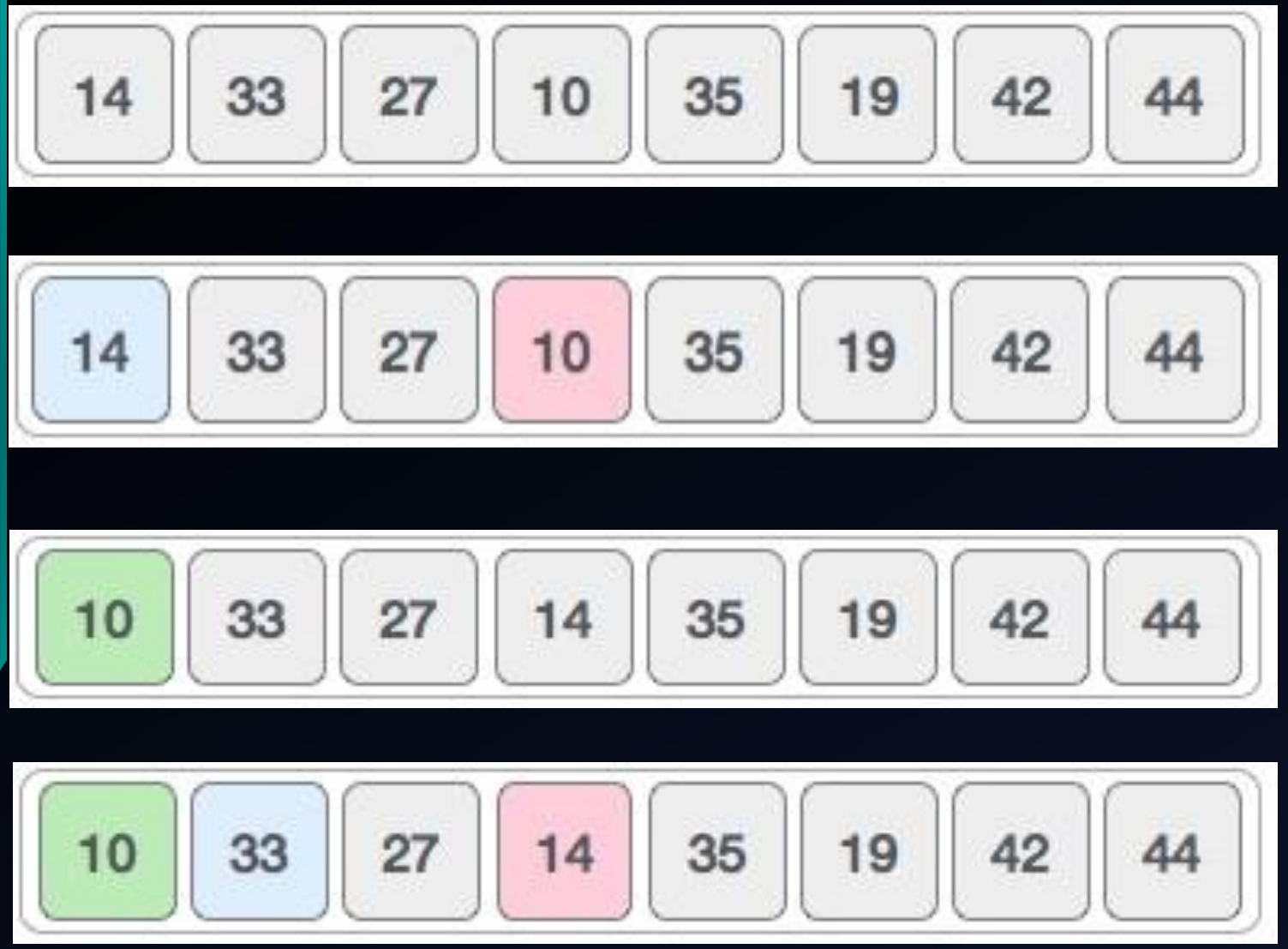
O Notation
Ω Notation
θ Notation

Common Asymptotic Notations

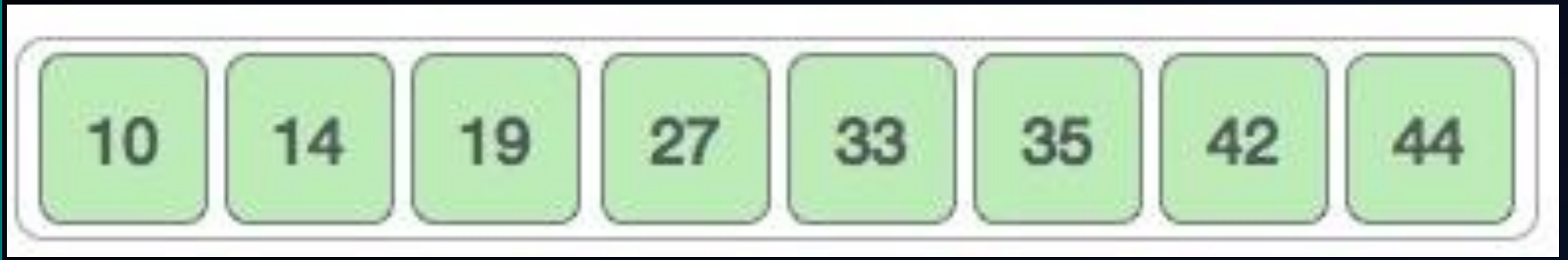
constant	–	$O(1)$
logarithmic	–	$O(\log n)$
linear	–	$O(n)$
n log n	–	$O(n \log n)$
quadratic	–	$O(n^2)$
cubic	–	$O(n^3)$
polynomial	–	$n^{O(1)}$
exponential	–	$2^{O(n)}$

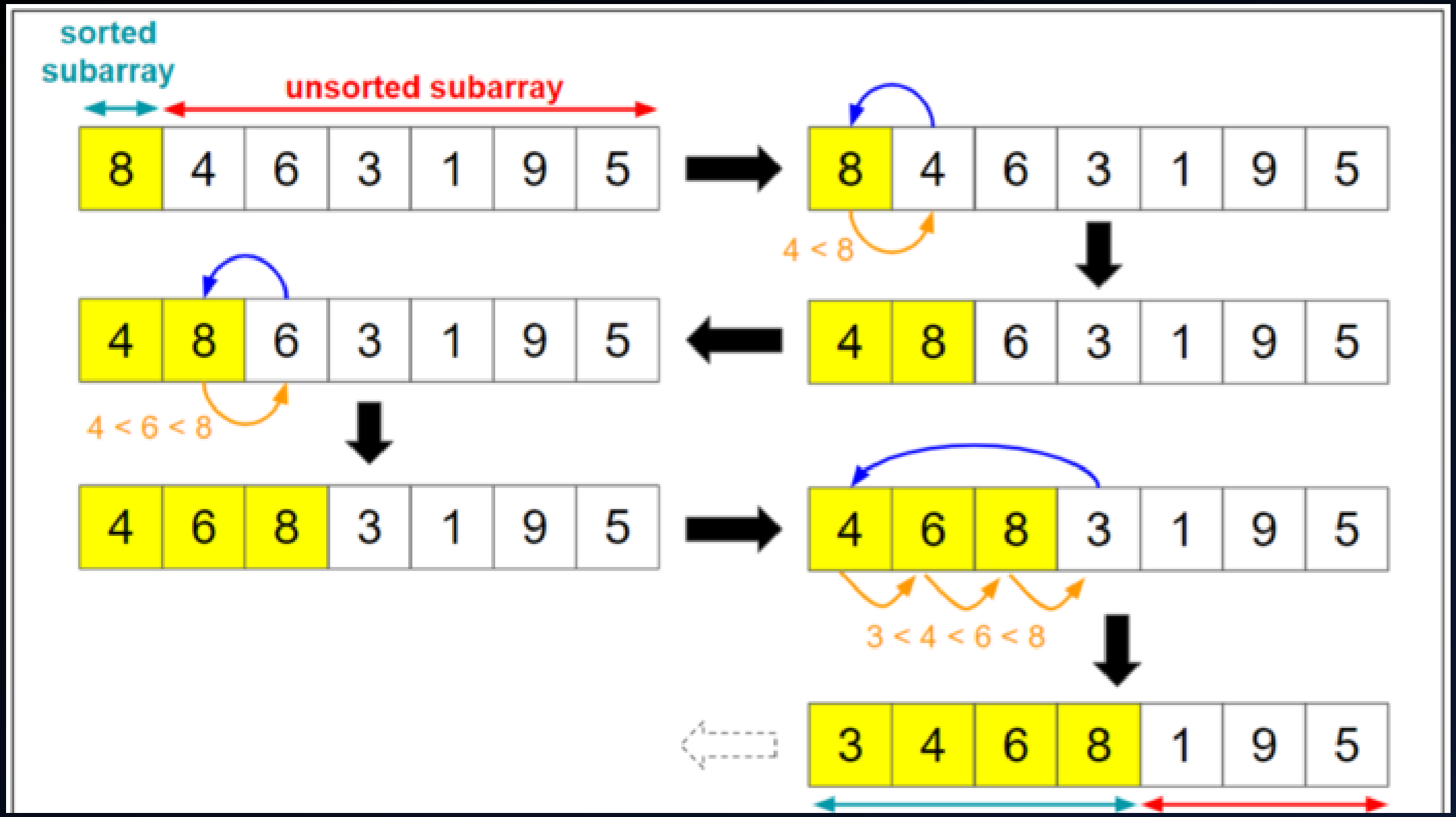
The selection sort algorithm **sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning**.

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.


14 33 27 10 35 19 42 44
14 33 27 10 35 19 42 44
10 33 27 14 35 19 42 44
10 33 27 14 35 19 42 44


10
14
27
33
35
19
42
44


10
14
19
27
33
35
42
44


sorted subarray
unsorted subarray
8 4 6 3 1 9 5
4 < 8
4 8 6 3 1 9 5
4 < 6 < 8
4 6 8 3 1 9 5
3 < 4 < 6 < 8
3 4 6 8 1 9 5

Best case scenario