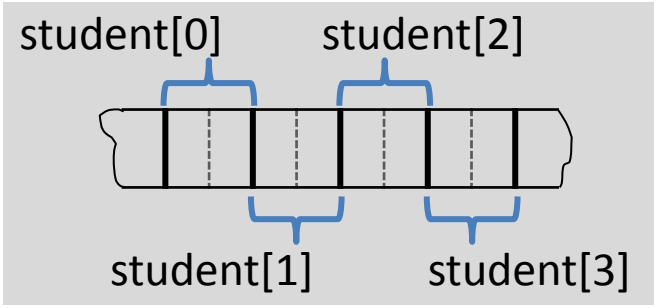
Lecture 2 – 3

Array

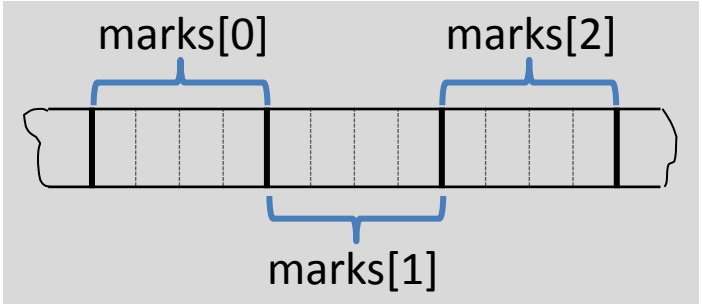
(Traversal, Search, Insertion, Deletion, Sorting)

Memory Storage – One Dimensional Array

```
int student[4];
```




```
float marks[3];
```



Memory Storage – Two Dimensional Array

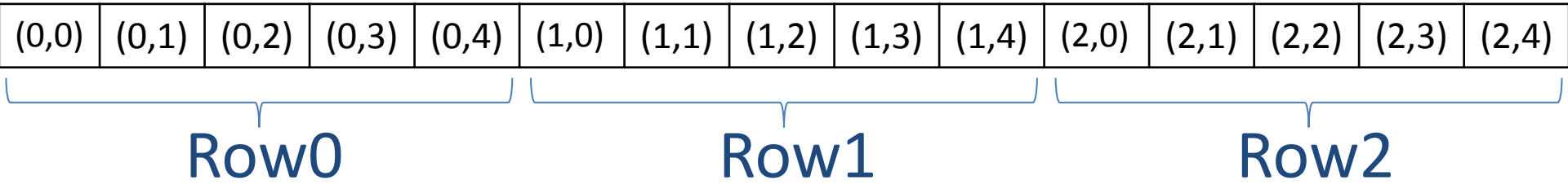
```
int marks[3][5];
```

- Can be visualized in the form of a matrix as

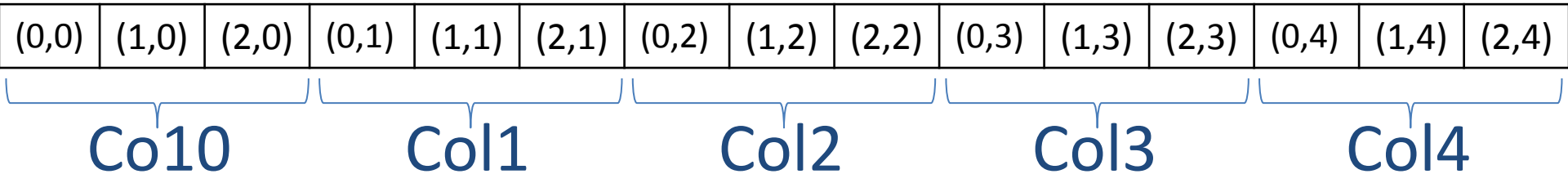
	Col 0	Col 1	Col 2	Col 3	Col 4
Row 0	marks[0][0]	marks[0][1]	marks[0][2]	marks[0][3]	marks[0][4]
Row 1	marks[1][0]	marks[1][1]	marks[1][2]	marks[1][3]	marks[1][4]
Row 2	marks[2][0]	marks[2][1]	marks[2][2]	marks[2][3]	marks[2][4]

Contd…

- Row-major order



- Column-major order



Operations on Linear Data Structures

- Traversal
- Search
- Insertion
- Deletion
- Merging
- Sorting

Traversal

- Processing each element in the array.
- Example: Find minimum element in the array

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

		cost	time		
1.	min = 0	c1	1	→	c1
2.	for i = 1 to n-1 do	c2	n	→	c2 n
3.	if A[min] > A[i]	c3	n – 1	→	c3 (n – 1)
4.	min = i	c4	$\sum_{i=1}^{n-1} t_i$	→	$c4\sum_{i=1}^{n-1} t_i$
5.	return A[min]	c5	1	→	c5

Contd…

- $T(n) = c1 + c2\ n + c3\ (n - 1) + c4 \sum_{i=1}^{n-1} t_i + c5$

$$= (c1 - c3 + c5) + (c2 + c3)\ n + c4 \sum_{i=1}^{n-1} t_i$$

- Best case: $\Omega(n) \rightarrow$ summation evaluates to 0.
- Worst case: $O(n) \rightarrow$ summation evaluates to n.
- Average case: $\theta(n) \rightarrow$ summation evaluates to n/2.

Example – Traversal

```
1.  //Determine smallest element in an array.
2.  #include<iostream>
3.  using namespace std;
4.  int arrayMinElement(int arr[], int n)
5.  {
6.     int min = 0;
7.     for (int i = 1; i < n; i++)
8.     {
9.        if (arr[i] < arr[min])
10.          min = i;
11.    }
12.    return arr[min];    }
```

Contd…

```
13. int main()
14. {
15.     int i, n, a[10];
16.     cout << "Enter the array size (<=10): ";
17.     cin >> n;
18.     for (i = 0; i < n; i++)
19.     { cout << "Enter " << i+1 << " element: ";
20.       cin >> a[i];
21.     }
22.     cout << "\nSmallest element is " << arrayMinElement(a,n);
23.     return 0;
24. }
```

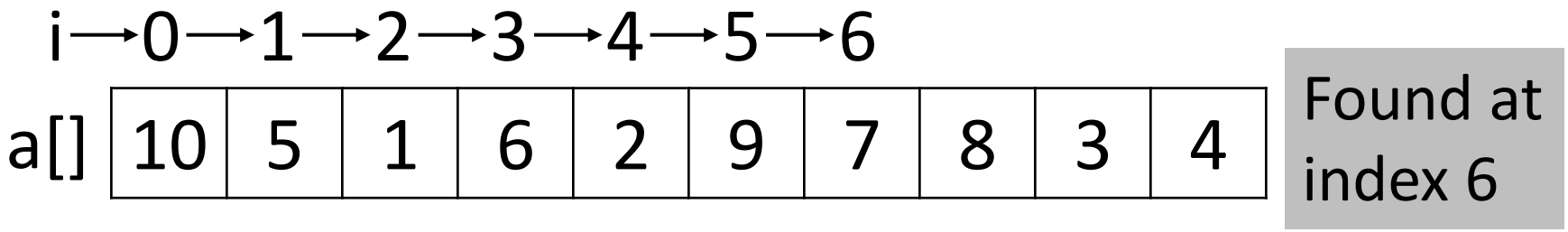
Search

Find the location of the element with a given value.

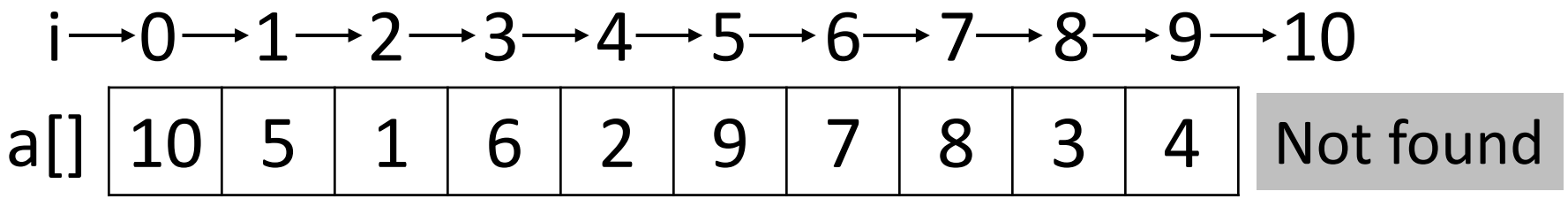
Linear Search

- 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.

	cost	time	
1. for i = 0 to n-1 do	c1	$\sum_{i=0}^{n} t_i$	$\rightarrow$ c1 $\sum_{i=0}^{n} t_i$
2. if A[i] == num	c2	$\sum_{i=0}^{n-1} t_i$	$\rightarrow$ c2 $\sum_{i=0}^{n-1} t_i$
3. return i	c3	1	$\rightarrow$ c3
4. return -1	c4	1	$\rightarrow$ c4

Contd…

- $T(n) = c1\sum_{i=0}^{n} t_i + c2\sum_{i=0}^{n-1} t_i + (c3 + c4)$

- Best case: $\Omega(1) \rightarrow$ summation evaluates to 1.
- Worst case: $O(n) \rightarrow$ summation evaluates to n.
- Average case: $\theta(n) \rightarrow$ summation evaluates to n/2.

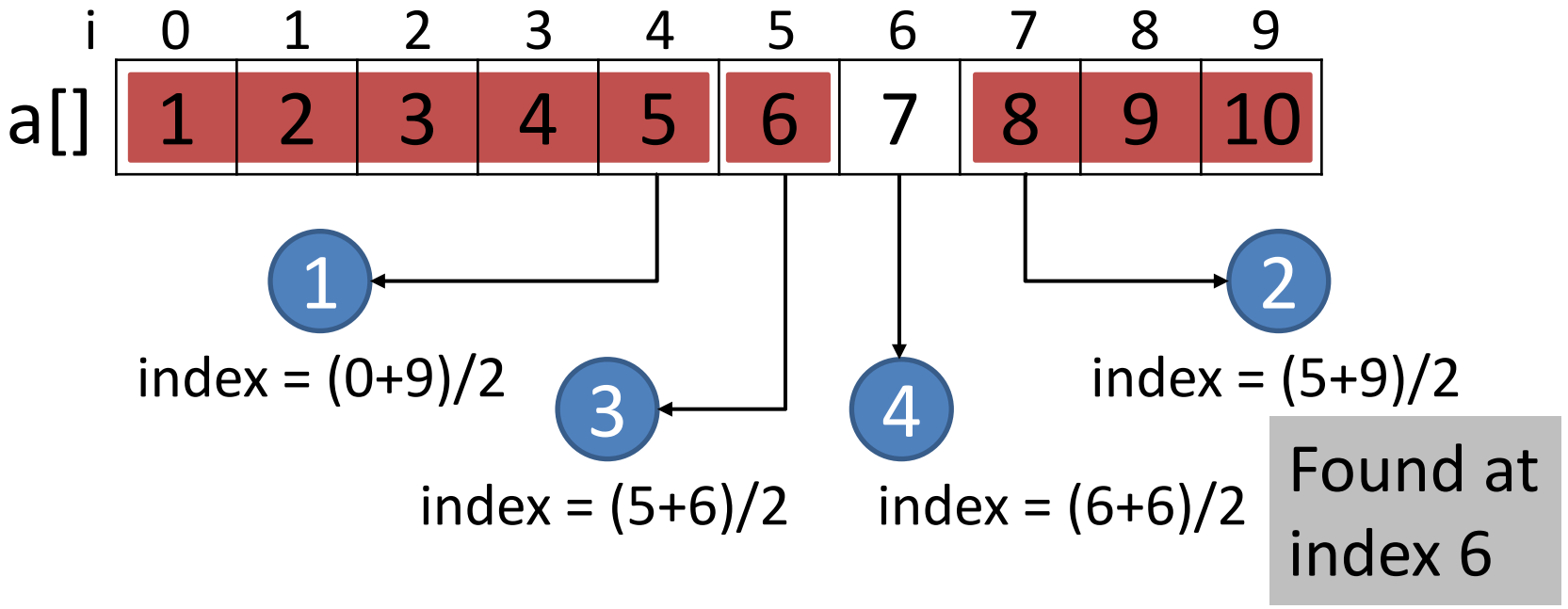
Example – Linear Search

```
1.  #include<stdio.h>
2.  int linearSearch(int a[], int n, int num)
3.  {  for (int i = 0; i < n; i++)
4.        if (a[i] == num) return i;
5.     return -1;  }
6.  int main()
7.  {  int i, n, a[10], num;
8.     printf("Enter the size of an array (<=10): ");     scanf("%d",&n);
9.     for (i = 0; i < n; i++)
10.    {  printf("Enter element at index %d: ",i);        scanf("%d", &a[i]);  }
11.       printf("\nEnter number to search: ");           scanf("%d",&num);
12.       int found = linearSearch(a,n,num);
13.       if(found != -1)   printf("Element found at index %d",found);
14.       else              printf("Element not found.");
15.       return 0;  }
```

Binary Searching

- Used if the array is sorted.
- Example:

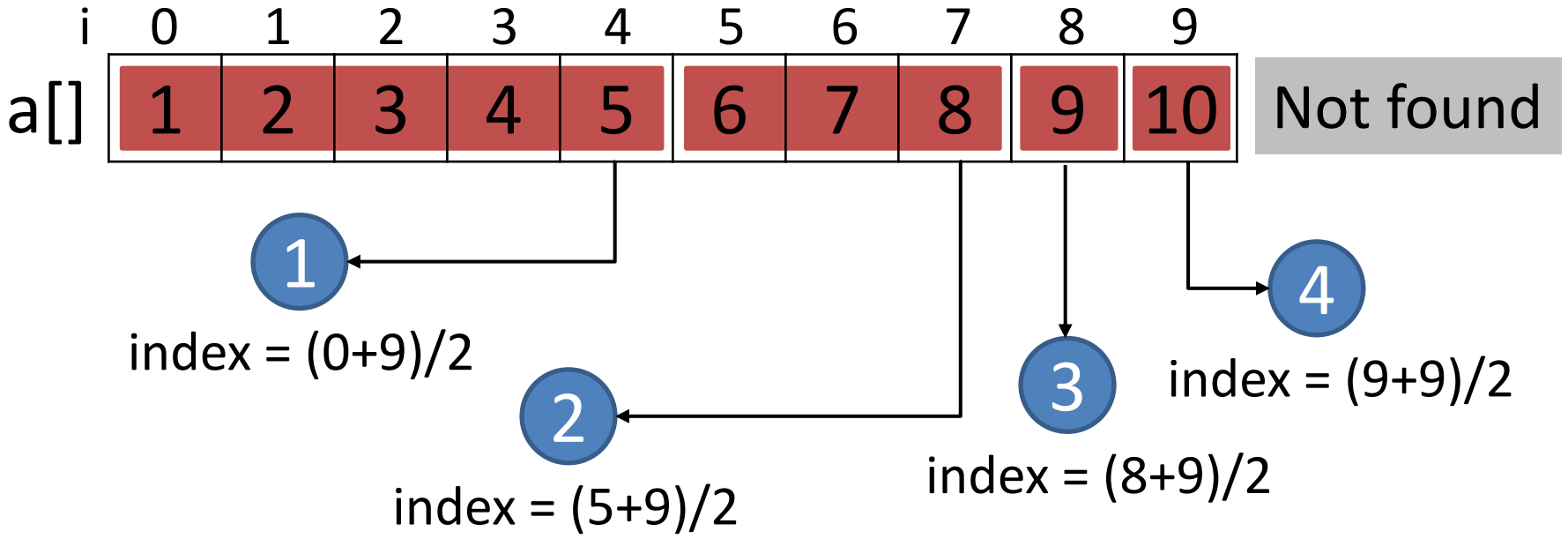
  Search 7 in the following array



Contd…

- Used if the array is sorted.
- Example:

  Search 11 in the following array



Contd…

Algorithm binarySearch(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.  beg = 0, end = n-1
2.  while  beg <= end  do
3.        mid = (beg + end)/2
4.        if  A[mid] == num
5.          return mid
6.        else if   A[mid] > num
7.                  end = mid - 1
8.        else if   A[mid] < num
9.                  beg = mid + 1
10. return -1
```

Contd…

- In each iteration, the number of elements to be searched from gets reduced by half. This process continues till the number of elements (to be searched) reaches to 1.
  - 1st iteration – $n/2^0$
  - 2nd iteration – $n/2^1$
  - 3rd iteration – $n/2^2$
  - …
  - m^{th} iteration – $n/2^m$

Let the termination condition (i.e. single element to search from) reaches at the m^{th} iteration, thus

$$1 = \frac{n}{2^m}$$

$$2^m = n$$

$$m \log_2 2 = \log_2 n$$

$$m = \lg n$$

$$\Rightarrow \text{complexity} \quad \text{is} \quad O(\lg n)$$

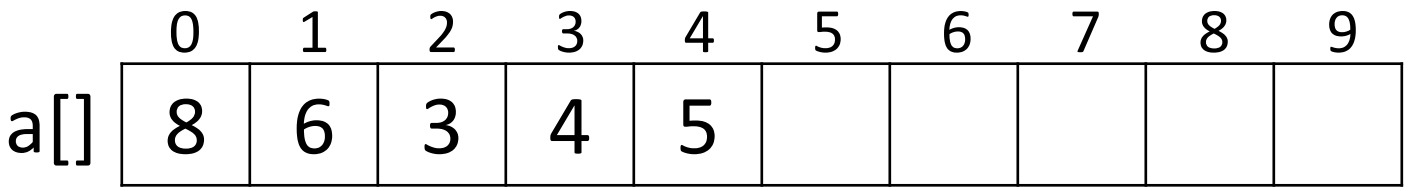
Example – Binary Search

```
1.  #include<stdio.h>
2.  int binarySearch(int arr[], int n, int num)
3.  {   int beg = 0, end = 9;
4.      while (beg <= end)
5.      {   int mid = (beg + end)/2;
6.          if (arr[mid] == num)
7.              return mid;
8.          else if (arr[mid] > num)
9.              end = mid - 1;
10.         else if (arr[mid] < num)
11.             beg = mid + 1;
12.     }
13.     return -1;
14. }
```

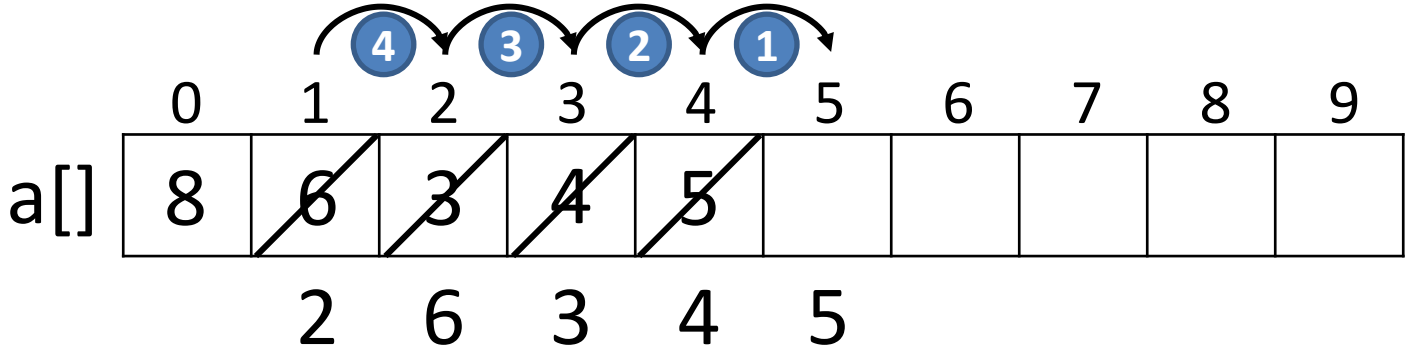
Contd…

```
1.  int main()
2.  {  int i, n, a[10], num;
3.     printf("Enter the size of an array (<=10): ");
4.     scanf("%d",&n);
5.     for (i = 0; i < n; i++)
6.       {  printf("Enter element at index %d: ",i);
7.          scanf("%d", &a[i]);  }
8.       printf("\nEnter number to search: ");
9.       scanf("%d",&num);
10.      int found = linearSearch(a,n,num);
11.      if(found != -1)
12.        printf("Element found at index %d",found);
13.      else
14.        printf("Element not found.");
15.      return 0;  }
```

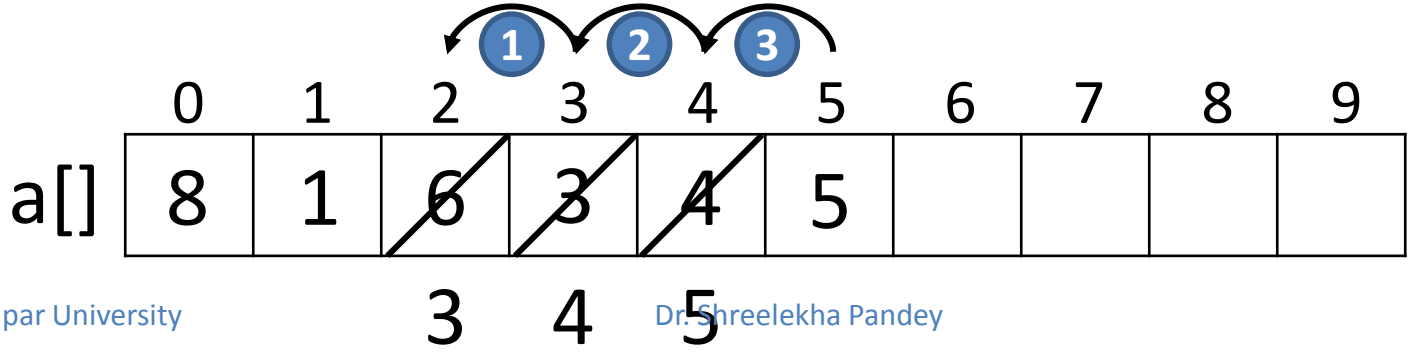
Insertion and Deletion



- 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**.

	cost	time
1. for i = n – 1 to indx do	c1	n – indx + 1
2. A[i + 1] = A[i]	c2	n – indx
3. A[indx] = num	c3	1
4. n = n + 1	c4	1

Contd…

- $T(n) = c1\ (n - indx + 1) + c2\ (n - indx) + c3 + c4$

- Best case: $\Omega(1) \rightarrow indx = n - 1$
- Worst case: $O(n) \rightarrow indx = 0$
- Average case: $\theta(n) \rightarrow indx = n/2$

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**.

	cost	time
1. temp = A[indx]	c1	1
2. for i = indx to n – 2 do	c2	n – indx
3. A[i] = A[i + 1]	c3	n – indx – 1
4. n = n – 1	c4	1
5. return temp	c5	1

Contd…

- $T(n) = c1 + c2\ (n - indx) + c3\ (n - indx - 1) + c4 + c5$

- Best case: $\Omega(1) \rightarrow indx = n - 1$
- Worst case: $O(n) \rightarrow indx = 0$
- Average case: $\theta(n) \rightarrow indx = n/2$

Example – Insertion and Deletion

```
1.  #include<stdio.h>
2.  int n;
3.  void insert(int a[], int num, int pos)
4.  {   for(int i = n-1; i >= pos; i--)          a[i+1] = a[i];
5.      a[pos] = num;
6.      n++;  }
7.  int deleteElement(int a[], int pos)
8.  {   int temp = a[pos];
9.      for(int i = pos; i <= n-2; i++)          a[i] = a[i+1];
10.     n--;
11.     return temp;  }
12. void printArray(int a[])
13. {    for (i = 0; i < n; i++)
14.          printf("%d ",a[i]);  }
```

Contd…

```
15. int main()
16. {  int i, a[10], num, pos;
17.    printf("Enter the size of an array (<10): ");
18.    scanf("%d",&n);
19.    for (i = 0; i < n; i++)
20.    {    printf("Enter element at index %d: ",i);
21.         scanf("%d",&a[i]);    }
22.   printf("\nEnter number to be inserted: ");
23.   scanf("%d",&num);
24.   printf("Enter the desired index: ");
25.   scanf("%d",&pos);
26.   insert(a,num,pos);
```

Contd…

```
27.  printf("\nArray after insertion is...\n");
28.  printArray(a);
29.  printf("\nEnter the index whose value is to be deleted: ");
30.  scanf("%d",&pos);
31.  printf("\nThe deleted element is %d.", deleteElement(a,pos));
32.  printf("\nArray after deletion is...\n");
33.  printArray(a);
34.  return 0;
35. }
```

Sorting

- Rearranging elements of an array in some order.
- Various types of sorting are
  - Bubble Sort.
  - Insertion Sort.
  - Selection Sort.
  - Quick Sort.
  - Merge Sort.
  - Heap Sort.

Algorithm – Bubble Sort

Algorithm bubbleSort(A,n)
Input: An array **A** containing **n** integers.
Output: The elements of **A** get sorted in increasing order.

	cost	time
1. for i = 1 to n – 1	c1	n
2. for j = 0 to n – i – 1 do	c2	$\sum_{i=1}^{n-1} t_i$
3. if A[j] > A[j + 1]	c3	$\sum_{i=1}^{n-1} (t_i - 1)$
4. Exchange A[j] with A[j+1]	c4	$\sum_{i=1}^{n-1} (t_i - 1)$

In all the cases, complexity is of the order of n^2.

Algorithm – Optimized Bubble Sort

Algorithm bubbleSortOpt(A,n)
Input: An array **A** containing **n** integers.
Output: The elements of **A** get sorted in increasing order.

```
1.  for  i = 1  to  n – 1
2.      flag = true
3.      for j = 0  to  n – i – 1  do
4.          if A[j] > A[j + 1]
5.              flag = false
6.              Exchange A[j] with A[j+1]
7.      if  flag == true
8.          break;
```

The best case complexity reduces to the order of n, but the worst and average is still n^2. So, overall the complexity is of the order of n^2 again.

Example – Bubble Sort

```
1.  #include<stdio.h>
2.  void bubbleSortOpt(int a[],int n);
3.  int main()
4.  {
5.     int a[10];
6.     printf("Enter 10 numbers: \n");
7.     for(int i = 0; i < 10; i++)
8.           scanf("%d",&a[i]);
9.     bubbleSortOpt(a,10);
10.    printf("\n");
11.    for(int i = 0; i < 10; i++)
12.       printf("%d ",a[i]);
13.    return 0;
14. }
```

Example – Bubble Sort

```
15. void bubbleSortOpt(int a[], int n)
16. {  int i, j, flag;
17.    for (i = 1; i <= n-1; i++)
18.        {     flag = 1;
19.             for (j = 0; j <= n-1-i; j++)
20.             {       if (a[j] > a[j+1])
21.                     {       flag = 0;
22.                             a[j] = a[j] + a[j+1];
23.                             a[j+1] = a[j] - a[j+1];
24.                             a[j] = a[j] - a[j+1];
25.                     }
26.             }
27.             if(flag) break;
28.       }          }
```