## Sparse Matrices

- Matrices with relatively high proportion of zero entries are called sparse matrices.
- The natural method of representing matrices in memory as two dimensional arrays may not be suitable for sparse matrices.
- If the matrix is sparse, we must consider an alternate way of representing it that stores only the nonzero elements.
- The nonzero elements in a sparse matrix can be stored as arrays.

## Sparse Matrices

- For example,

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 2 | 0 | 0 | 1 | 0 |
| 0 | 6 | 0 | 0 | 7 | 0 | 0 | 3 |
| 0 | 0 | 0 | 9 | 0 | 8 | 0 | 0 |
| 0 | 4 | 5 | 0 | 0 | 0 | 0 | 0 |

Sparse Matrix 4 x 8

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Row[] | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 3 |
| Col[] | 3 | 6 | 1 | 4 | 7 | 3 | 5 | 1 | 2 |
| Value[] | 2 | 1 | 6 | 7 | 3 | 9 | 8 | 4 | 5 |

Array representation of non zero entries

## Sparse Matrices

❑To reconstruct the matrix structure, we need to record the row and column each nonzero entry comes.

❑So each element of the array into which the sparse matrix is mapped needs to have three fields:

i. Row(the row of the nonzero matrix entry)
ii. Col(the column number of the nonzero matrix entry)
iii. And value(the value of the nonzero matrix entry).

## One Dimensional array Programs

1. Reversing a 1D Array
   - ❖ The reverse operation reverses the entire array by swapping the elements form respective from the beginning and end of the array.
   - ❖ Note that the swapping should continue for n/2 times only, irrespective of whether the elements is even or odd.

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 9 | 6 | 8 | 4 |

| | | | | | |
|---|---|---|---|---|---|
| 4 | 8 | 6 | 5 | 3 | 1 |

## Algorithm

- Here arr is a linear array with n elements.
- This algorithm reverses by swapping the corresponding elements from the beginning and end of the arr.

```
Step 1: set i=0
Step 2: repeat step 3 to 4 while i<n/2
Step 3: interchange arr[i] and arr[n-1-i]
Step 4: set i=i+1
Step 5: set n=n-1
Step 6: Exit
```

Program : arrayreverse

## One Dimensional array Programs

2. Searching a 1D Array

- ❖ Let arr be a linear array of numbers in memory and suppose a specific element num of information is given.
- ❖ Searching refers to the operation of finding the position pos of num in arr or printing some message that num does not appear there.
- ❖ The search is said to be successful if num does appear in arr and unsuccessful otherwise.
- ❖ There are mean different searching algorithms.
- ❖ The search algorithm is generally chosen based on the way the information in arr is organized.
- ❖ Here we consider the simple linear search(or sequential search) algorithm.
- ❖ The linear search proceeds from the beginning towards the end of the linear array arr by comparing each element of the arr with the search item num one by one.

## 2. Searching a 1D Array

- If the element compared happens to be the same as the value we are looking for, then the search operations is paused to display the position at which the search content is found.
- The search operation is continued for looking for further match in the array, until the array get exhausted.
- If no match is found, a message to this effect need to be displayed.

## Algorithm

- ❖Here arr is a linear array with n elements.
- ❖This algorithm performs the linear search of the item num by comparing each element of arr with num one by one, proceeding from the beginning to the end of the array arr.
- ❖The algorithm use a flag to check whether a match is found or not in the search process.
- ❖If a match is found, the location information is displayed and a flag is set to this effect.

## Algorithm

```
Step 1: set i=0 anf flag=0
Step 2: repeat step 3 to 6 while i<n
Step 3: iff arr[i]=num then display I and set flag=1
Step 4: set i=i+1
Step 5: if flag=0 then display unsuccessful
Step 6: Exit
```

Program : search

## One Dimensional array Programs

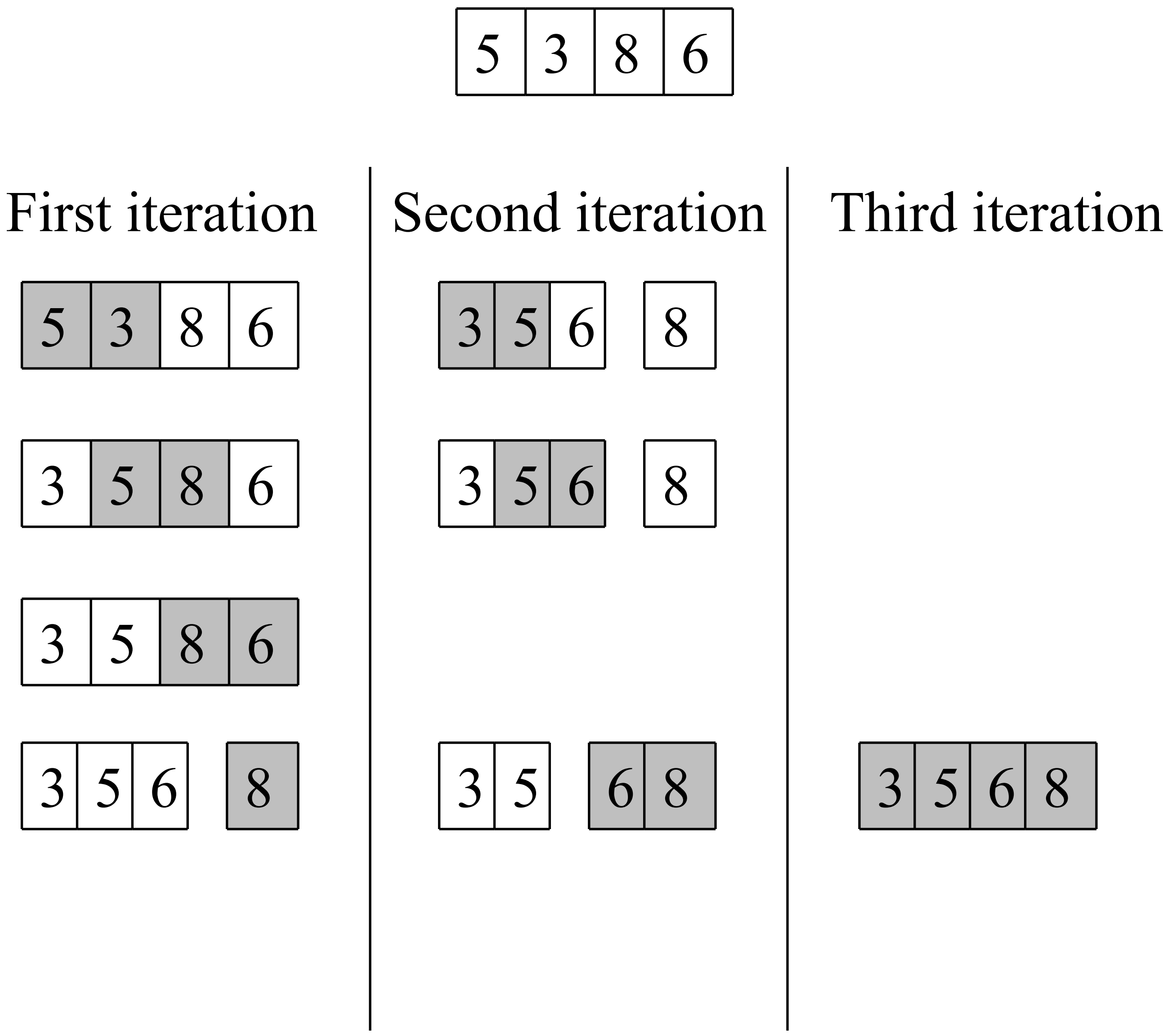
3. Sorting a 1D Array

- The sort operation arranges the elements of the array either in ascending or descending order.
- There are many sorting algorithms available to perform the sort operation, of which we are considering here a simple sorting algorithm called bubble sort.
- Bubble sort, also known as exchange sort, is the oldest and simplest sorting algorithm in use.
- Bubble sort works by comparing each item in the array with the next to it, and swapping these two items if they are in the wrong order.
- The pass through the list is repeated until no swaps are needed, which means the list is sorted.

## 3. Sorting a 1D Array – Bubble sort

- The first iteration of this algorithm begins with comparing 0th element with the 1st element.
- If it is found to be greater than the 1st element, then they are interchanged.
- Next, the 1st element is compared with the 2nd element, if it is found to be greater, then they are interchanged.
- In the same way all the elements(excluding the last) are compared with their next element and are interchanged, if required.
- On completing the first iteration, the largest element gets placed at the last position.
- Similarly, in the second iteration, comparisons are made till the last but one element and this time the second largest element gets placed at the second last position in the list.
- As a result, after all iterations, the list becomes a sorted list.

## Steps in bubble sort



## Algorithm

Here arr is an array with n elements.

```
Step 1:repeat step 2 and 3 for i=0 to n-1
Step 2:repeat step 2 and 3 for j=0 to n-i-1
Step 3:if arr[j] > arr[j+1], then;
                    interchange arr[j] and arr[j+1]
Step 4:set j= j+1
Step 5:set i=i+1
Step 6:Exit
```

Program : bubblesort

## One Dimensional array Programs

4. Merging two 1D Arrays

- ❑ Merging operation combines two arrays.
- ❑ We have to merge the two arrays to another one, otherwise append first array elements to second array or second array elements to first array.
- ❑ If both the arrays are ordered, then the ordered merged array is created from the component arrays by copying the elements from the component arrays in the sorted order and adding them to the third merged array.

- ❑ Program: merge(unordered)
- ❑ Program: mergeorder(ordered)

## One Dimensional array Programs

5. Get total – return the total of all the values in the array
6. Get average – return the average of all the values in the array
7. Get biggest – return the biggest values in the array
8. Get smallest – return the smallest values in the array.

Program: total

## Two Dimensional array Programs

1. Traverse – processing each element in the array
2. Transpose – interchanging the rows and columns
3. Search – finding the location of an element with a given value
4. Add – find sum of two arrays
5. Product - find product of two arrays
6. Get total and average – return the total and average of all the values in the array.
7. Biggest and smallest – find the biggest and smallest element in array.

## 1. Traverse - Algorithm

- Here arr is a 2D array with r rows and c columns.
- The algorithm traverses arr applying an operation to each element of arr.

```
Step 1: set I and j as 0
Step 2: repeat step 3 to 6 if i<r
Step 3: repeat step 4 to 5 if j<c
Step 4: display arr[i][j]
Step 5: set j=j+1
Step 6: set i=i+1
Step 7: exit.
```

Program : 2dtraverse

## 2. Transpose - Algorithm

- Here arr is a 2D array with r rows and c columns.

```
Step 1: set I and j as 0
Step 2: repeat step 3 to 6 if i<c
Step 3: repeat step 4 to 5 if j<r
Step 4: display arr[j][i]
Step 5: set j=j+1
Step 6: set i=i+1
Step 7: exit.
```

Program : transpose

## 3. Search - Algorithm

- ❑ Here arr is a 2D array with r rows and c columns.
- ❑ This algorithm performs the linear search of the item num by comparing each element of arr with num one by one.
- ❑ The algorithm uses a flag to check whether a match is found or not in the search process.
- ❑ If a match is found, the location information is displayed and a flag is set to this effect.

```
Step 1: set I ,j and flag as 0
Step 2: repeat step 3 to 6 if i<r
Step 3: repeat step 4 to 5 if j<c
Step 4: if arr[i][j] = num then display location[i][j] and set flag=1
Step 5: set j=j+1
Step 6: set i=i+1
Step 7: if flag=0 then display not found.
Step 8: exit
```

Program : search2d

## 4. Add - Algorithm

- Here arr1, arr2, and arr3 are 2D arrays with r rows and c columns.

```
Step 1: set I ,j and flag as 0
Step 2: repeat step 3 to 6 if i<r
Step 3: repeat step 4 to 5 if j<c
Step 4:arr3[i][j]=arr1[i][j]+arr2[i][j]
Step 5: set j=j+1
Step 6: set i=i+1
Step 7: exit
```

Program : add2d

## 5.Product - Algorithm

- Here arr1 is a 2D array with r1 rows and c1 columns, arr2 is a 2D with r2 rows and c2 columns and arr3 is a 2D array with r1 rows and c2 columns.

```
Step 1: set I, j and k as 0
Step 2: repeat step 3 to 9 if i<r1
Step 3: repeat step 4 to 8 if j<c1
Step 4:arr3[i][j]=0
Step 5: repeat step 6 to 7 if k<c1
Step 6: arr3[i][j]=arr3[i][j]+arr1[i][k] * arr2[k][j]
Step 7: set k= k+1
Step 8: set j=j+1
Step 9: set i= i+1
Step 10: exit
```

Program : product

## 6.Get Total and average

Program : 2ndtotal

## 6.Biggest and smallest

Program : bigsmall