More Arrays

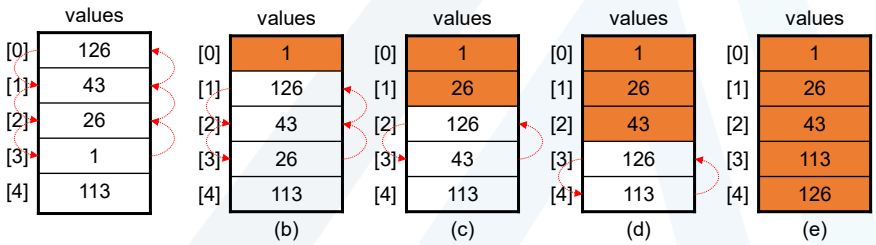
Lecture 8

4.6 Sorting Arrays

- Sorting data
  - Important computing application
  - Virtually every organization must sort some data
    - Massive amounts must be sorted
- Bubble sort (sinking sort)
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat these steps for every element

Bubble Sort

- Each iteration puts the smallest unsorted element into its correct place by comparing successive pairs of elements



4.6 Sorting Arrays

- Swapping variables

```
int x = 3, y = 4;
y = x;
x = y;
```

- What happened?
  -
  -
- Solution

fig04_16.cpp (1 of 3)

```
1  // Fig. 4.16: fig04_16.cpp
2  // This program sorts an array's values into ascending order.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14   const int arraySize = 10;  // size of array a
15   int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
16   int hold;  // temporary location used to swap array elements
17
18   cout << "Data items in original order\n";
19
20   // output original array
21   for ( int i = 0; i < arraySize; i++ )
22     cout << setw( 4 ) << a[ i ];
23
```


fig04_16.cpp (2 of 3)

```
24    // bubble sort
25    // loop to control number of passes
26    for ( int pass = 0; pass < arraySize - 1; pass++ )
27
28      // loop to control number of comparisons per pass
29      for ( int j = 0; j < arraySize - 1; j++ )
30
31        // compare side-by-side elements and swap them if
32        // first element is greater than second element
33        if ( a[ j ] > a[ j + 1 ] ) {
34          hold = a[ j ];
35          a[ j ] = a[ j + 1 ];
36          a[ j + 1 ] = hold;
37
38        } // end if
39
```

```
40    cout << "\nData items in ascending order\n";
41
42    // output sorted array
43    for ( int k = 0; k < arraySize; k++ )
44       cout << setw( 4 ) << a[ k ];
45
46    cout << endl;
47
48    return 0;  // indicates successful termination
49
50 } // end main
```

fig04_16.cpp
(3 of 3)

fig04_16.cpp
output (1 of 1)

```
Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
```


4.7 Case Study: Computing Mean, Median and Mode Using Arrays

- Mean
  - Average (sum/number of elements)
- Median
  - Number in middle of sorted list
  - 1, 2, 3, 4, 5 (3 is median)
  - If even number of elements, take average of middle two
- Mode
  - Number that occurs most often
  - 1, 1, 1, 2, 3, 3, 4, 5 (1 is mode)

fig04_17.cpp (1 of 8)

```
1  // Fig. 4.17: fig04_17.cpp
2  // This program introduces the topic of survey data analysis.
3  // It computes the mean, median, and mode of the data.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8  using std::fixed;
9  using std::showpoint;
10
11 #include <iomanip>
12
13 using std::setw;
14 using std::setprecision;
15
16 void mean( const int [], int );
17 void median( int [], int );
18 void mode( int [], int [], int );
19 void bubbleSort( int[], int );
20 void printArray( const int[], int );
21
22 int main()
23 {
24    const int responseSize = 99;  // size of array responses
25
```


fig04_17.cpp (2 of 8)

```
26    int frequency[ 10 ] = { 0 };  // initialize array frequency
27
28    // initialize array responses
29    int response[ responseSize ] =
30       { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
31         7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
32         6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
33         7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
34         6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
35         7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
36         5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
37         7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
38         7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
39         4, 5, 6, 1, 6, 5, 7, 8, 7 };
40
41    // process responses
42    mean( response, responseSize );
43    median( response, responseSize );
44    mode( frequency, response, responseSize );
45
46    return 0;  // indicates successful termination
47
48 } // end main
49
```

fig04_17.cpp
(3 of 8)

```
50 // calculate average of all response values
51 void mean( const int answer[], int arraySize )
52 {
53   int total = 0;
54
55   cout << "********\n  Mean\n********\n";
56
57   // total response values
58   for ( int i = 0; i < arraySize; i++ )
59     total += answer[ i ];
60
61   // format and output results
62   cout << fixed << setprecision( 4 );
63
64   cout << "The mean is the average value of the data\n"
65      << "items. The mean is equal to the total of\n"
66      << "all the data items divided by the number\n"
67      << "of data items (" << arraySize
68      << "). The mean value for\nthis run is: "
69      << total << " / " << arraySize << " = "
70      << static_cast< double >( total ) / arraySize
71      << "\n\n";
72
73 } // end function mean
74
```


fig04_17.cpp
(4 of 8)

```
75 // sort array and determine median element's value
76 void median( int answer[], int size )
77 {
78   cout << "\n********\n Median\n********\n"
79      << "The unsorted array of responses is";
80
81   printArray( answer, size );  // output unsorted array
82
83   bubbleSort( answer, size );  // sort array
84
85   cout << "\n\nThe sorted array is";
86   printArray( answer, size );  // output sorted array
87
88   // display median element
89   cout << "\n\nThe median is element " << size / 2
90      << " of\nthe sorted " << size
91      << " element array.\nFor this run the median is ";
92   if (size%2 == 0)
93      cout << ((answer[ size / 2-1 ]  + answer[ size / 2 ])/2.0) << "\n\n";
94   else
95      cout << answer[ size / 2 ] << "\n\n";
96
97 } // end function median
98
```

```
96  // determine most frequent response
97  void mode( int freq[], int answer[], int size )
98  {
99    int largest = 0;   // represents largest frequency
100   int modeValue = 0;  // represents most frequent response
101
102   cout << "\n********\n  Mode\n********\n";
103
104   // initialize frequencies to 0
105   for ( int i = 1; i <= 9; i++ )
106     freq[ i ] = 0;
107
108   // summarize frequencies
109   for ( int j = 0; j < size; j++ )
110     ++freq[ answer[ j ] ];
111
112   // output headers for result columns
113   cout << "Response" << setw( 11 ) << "Frequency"
114        << setw( 19 ) << "Histogram\n\n" << setw( 55 )
115        << "1    1    2    2\n" << setw( 56 )
116        << "5    0    5    0    5\n\n";
117
```



```
118   // output results
119   for ( int rating = 1; rating <= 9; rating++ ) {
120     cout << setw( 8 ) << rating << setw( 11 )
121          << freq[ rating ] << "         ";
122
123     // keep track of mode value and largest frequency value
124     if ( freq[ rating ] > largest ) {
125       largest = freq[ rating ];
126       modeValue = rating;
127
128     } // end if
129
130     // output histogram bar representing frequency value
131     for ( int k = 1; k <= freq[ rating ]; k++ )
132       cout << '*';
133
134     cout << '\n';  // begin new line of output
135
136   } // end outer for
137
138   // display the mode value
139   cout << "The mode is the most frequent value.\n"
140        << "For this run the mode is " << modeValue
141        << " which occurred " << largest << " times." << endl;
142
143 } // end function mode
```

fig04_17.cpp (7 of 8)

```
144
145 // function that sorts an array with bubble sort algorithm
146 void bubbleSort( int a[], int size )
147 {
148   int hold;  // temporary location used to swap elements
149
150   // loop to control number of passes
151   for ( int pass = 1; pass < size; pass++ )
152
153     // loop to control number of comparisons per pass
154     for ( int j = 0; j < size - 1; j++ )
155
156       // swap elements if out of order
157       if ( a[ j ] > a[ j + 1 ] ) {
158         hold = a[ j ];
159         a[ j ] = a[ j + 1 ];
160         a[ j + 1 ] = hold;
161
162       } // end if
163
164 } // end function bubbleSort
165
```


fig04_17.cpp (8 of 8)

```
166 // output array contents (20 values per row)
167 void printArray( const int a[], int size )
168 {
169   for ( int i = 0; i < size; i++ ) {
170
171     if ( i % 20 == 0 )  // begin new line every 20 values
172       cout << endl;
173
174     cout << setw( 2 ) << a[ i ];
175
176   } // end for
177
178 } // end function printArray
```

```
********
  Mean
********
The mean is the average value of the data
items. The mean is equal to the total of
all the data items divided by the number
of data items (99). The mean value for
this run is: 681 / 99 = 6.8788
********
 Median
********
The unsorted array of responses is
 6 7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8
 6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9
 6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
 5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8
 7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

The sorted array is
 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5
 5 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

The median is element 49 of
the sorted 99 element array.
For this run the median is 7
```


fig04_17.cpp output (2 of 2)

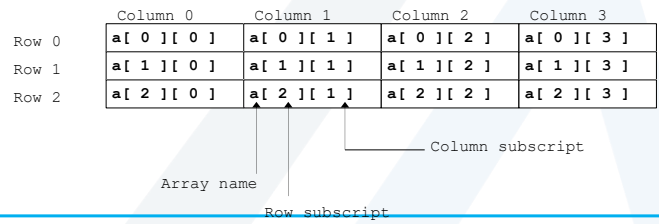
```
********
  Mode
********
Response  Frequency        Histogram

                                    1    1    2    2
                               5    0    5    0    5


       1          1          *
       2          3          ***
       3          4          ****
       4          5          *****
       5          8          ********
       6          9          *********
       7         23          ***********************
       8         27          ***************************
       9         19          *******************
The mode is the most frequent value.
For this run the mode is 8 which occurred 27 times.
```

4.9 Multiple-Subscripted Arrays

- Multiple subscripts
  - `a[ i ][ j ]`
  - Tables with rows and columns
  - Specify row, then column
  - "Array of arrays"
    - `a[0]` is an array of 4 elements
    - `a[0][0]` is the first element of that array



4.9 Multiple-Subscripted Arrays

- To initialize
  - Default of 0
  - Initializers grouped by row in braces

```
int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };
```

1	2
3	4

```
int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };
```

1	0
3	4

4.9 Multiple-Subscripted Arrays

- Referenced like normal
  ```
  cout << b[ 0 ][ 1 ];
  ```
  - Outputs **0**
  - Cannot reference using commas
    ```
    cout << b[ 0, 1 ];
    ```
    - Syntax error
- Function prototypes
  - Must specify sizes of subscripts
    - First subscript not necessary, as with single-scripted arrays
  - **`void printArray( int [][ 3 ] );`**

1	0
3	4

fig04_22.cpp (1 of 2)

```
1  // Fig. 4.22: fig04_22.cpp
2  // Initializing multidimensional arrays.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  void printArray( int [][ 3 ] );
9
10 int main()
11 {
12    int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
13    int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
14    int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
15
16    cout << "Values in array1 by row are:" << endl;
17    printArray( array1 );
18
19    cout << "Values in array2 by row are:" << endl;
20    printArray( array2 );
21
22    cout << "Values in array3 by row are:" << endl;
23    printArray( array3 );
24
25    return 0;  // indicates successful termination
26
27 } // end main
```

(2 of 2)

fig04_22.cpp output (1 of 1)

```
28
29  // function to output array with two rows and three columns
30  void printArray( int a[][ 3 ] )
31  {
32     for ( int i = 0; i < 2; i++ ) {    // for each row
33
34        for ( int j = 0; j < 3; j++ )   // output column values
```

```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```


4.9 Multiple-Subscripted Arrays

- Next: program showing initialization
  - After, program to keep track of students grades
  - Multiple-subscripted array (table)
  - Rows are students
  - Columns are grades

	Quiz1	Quiz2
Student0	95	85
Student1	89	80

fig04_23.cpp
(1 of 6)

```
1  // Fig. 4.23: fig04_23.cpp
2  // Double-subscripted array example.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::fixed;
8  using std::left;
9
10 #include <iomanip>
11
12 using std::setw;
13 using std::setprecision;
14
15 const int students = 3;  // number of students
16 const int exams = 4;     // number of exams
17
18 // function prototypes
19 int minimum( int [][ exams ], int, int );
20 int maximum( int [][ exams ], int, int );
21 double average( int [], int );
22 void printArray( int [][ exams ], int, int );
23
```


fig04_23.cpp
(2 of 6)

```
24 int main()
25 {
26   // initialize student grades for three students (rows)
27   int studentGrades[ students ][ exams ] =
28     { { 77, 68, 86, 73 },
29       { 96, 87, 89, 78 },
30       { 70, 90, 86, 81 } };
31
32   // output array studentGrades
33   cout << "The array is:\n";
34   printArray( studentGrades, students, exams );
35
36   // determine smallest and largest grade values
37   cout << "\n\nLowest grade: "
38        << minimum( studentGrades, students, exams )
39        << "\nHighest grade: "
40        << maximum( studentGrades, students, exams ) << '\n';
41
42   cout << fixed << setprecision( 2 );
43
```

```
44   // calculate average grade for each student
45   for ( int person = 0; person < students; person++ )
46     cout << "The average grade for student " << person
47          << " is "
48          << average( studentGrades[ person ], exams )
49          << endl;
50
51   return 0;  // indicates successful termination
52
53 } // end main
54
55 // find minimum grade
56 int minimum( int grades[][ exams ], int pupils, int tests )
57 {
58   int lowGrade = 100; // initialize to highest possible grade
59
60   for ( int i = 0; i < pupils; i++ )
61
62     for ( int j = 0; j < tests; j++ )
63
64       if ( grades[ i ][ j ] < lowGrade )
65         lowGrade = grades[ i ][ j ];
66
67   return lowGrade;
68
69 } // end function minimum
```

fig04_23.cpp
(3 of 6)


```
70
71 // find maximum grade
72 int maximum( int grades[][ exams ], int pupils, int tests )
73 {
74   int highGrade = 0;  // initialize to lowest possible grade
75
76   for ( int i = 0; i < pupils; i++ )
77
78     for ( int j = 0; j < tests; j++ )
79
80       if ( grades[ i ][ j ] > highGrade )
81         highGrade = grades[ i ][ j ];
82
83   return highGrade;
84
85 } // end function maximum
86
```

fig04_23.cpp
(4 of 6)

fig04_23.cpp (5 of 6)

```
87  // determine average grade for particular student
88  double average( int setOfGrades[], int tests )
89  {
90     int total = 0;
91
92     // total all grades for one student
93     for ( int i = 0; i < tests; i++ )
94        total += setOfGrades[ i ];
95
96     return static_cast< double >( total ) / tests;  // average
97
98  } // end function maximum
```


fig04_23.cpp (6 of 6)

```
99
100 // Print the array
101 void printArray( int grades[][ exams ], int pupils, int tests )
102 {
103    // set left justification and output column heads
104    cout << left << "            [0]  [1]  [2]  [3]";
105
106    // output grades in tabular format
107    for ( int i = 0; i < pupils; i++ ) {
108
109       // output label for row
110       cout << "\nstudentGrades[" << i << "] ";
111
112       // output one grades for one student
113       for ( int j = 0; j < tests; j++ )
114          cout << setw( 5 ) << grades[ i ][ j ];
115
116    } // end outer for
117
118 } // end function printArray
```

fig04_23.cpp output (1 of 1)

```
The array is:
            [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```


4.8 Searching Arrays: Linear Search and Binary Search

- Search array for a key value
- Linear search
  - Compare each element of array with key value
    - Start at one end, go to other
  - Useful for small and unsorted arrays
    - Inefficient
    - If search key not present, examines every element

4.8 Searching Arrays: Linear Search and Binary Search

- Binary search
  - Only used with sorted arrays
  - Compare middle element with key
    - If equal, match found
    - If key < middle
      - Repeat search on first half of array
    - If key > middle
      - Repeat search on last half
  - Very fast
    - At most N steps, where 2 > # of elements
    - 30 element array takes at most 5 steps
      2 > 30


```
1  // Fig. 4.19: fig04_19.cpp
2  // Linear search of an array.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int linearSearch( const int [], int, int );  // prototype
10
11 int main()
12 {
13   const int arraySize = 100;  // size of array a
14   int a[ arraySize ];         // create array a
15   int searchKey;              // value to locate in a
16
17   for ( int i = 0; i < arraySize; i++ )  // create some data
18     a[ i ] = 2 * i;
19
20   cout << "Enter integer search key: ";
21   cin >> searchKey;
22
23   // attempt to locate searchKey in array a
24   int element = linearSearch( a, searchKey, arraySize );
25
```

```
26    // display results
27    if ( element != -1 )
28       cout << "Found value in element " << element << endl;
29    else
30       cout << "Value not found" << endl;
31
32    return 0;  // indicates successful termination
33
34 } // end main
35
36 // compare key to every element of array until location is
37 // found or until end of array is reached; return subscript of
38 // element if key or -1 if key not found
39 int linearSearch( const int array[], int key, int sizeOfArray )
40 {
41    for ( int j = 0; j < sizeOfArray; j++ )
42
43       if ( array[ j ] == key )  // if found,
44          return j;              // return location of key
45
46    return -1;  // key not found
47
48 } // end function linearSearch
```


fig04_19.cpp output (1 of 1)

```
Enter integer search key: 36
Found value in element 18

Enter integer search key: 37
Value not found
```

```
1  // Fig. 4.20: fig04_20.cpp
2  // Binary search of an array.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::setw;
12
13 // function prototypes
14 int binarySearch( const int [], int, int, int, int );
15 void printHeader( int );
16 void printRow( const int [], int, int, int, int );
17
18 int main()
19 {
20   const int arraySize = 15;  // size of array a
21   int a[ arraySize ];        // create array a
22   int key;                   // value to locate in a
23
24   for ( int i = 0; i < arraySize; i++ )  // create some data
25     a[ i ] = 2 * i;
26
```



```
27   cout << "Enter a number between 0 and 28: ";
28   cin >> key;
29
30   printHeader( arraySize );
31
32   // search for key in array a
33   int result =
34     binarySearch( a, key, 0, arraySize - 1, arraySize );
35
36   // display results
37   if ( result != -1 )
38     cout << '\n' << key << " found in array element "
39          << result << endl;
40   else
41     cout << '\n' << key << " not found" << endl;
42
43   return 0;  // indicates successful termination
44
45 } // end main
46
```

fig04_20.cpp
(2 of 6)

fig04_20.cpp (3 of 6)

```
47 // function to perform binary search of an array
48 int binarySearch( const int b[], int searchKey, int low,
49   int high, int size )
50 {
51   int middle;
52
53   // loop until low subscript is greater than high subscript
54   while ( low <= high ) {
55
56     // determine middle element of subarray being searched
57     middle = ( low + high ) / 2;
58
59     // display subarray used in this loop iteration
60     printRow( b, low, middle, high, size );
61
```


fig04_20.cpp (4 of 6)

```
62     // if searchKey matches middle element, return middle
63     if ( searchKey == b[ middle ] )  // match
64       return middle;
65
66     else
67
68       // if searchKey less than middle element,
69       // set new high element
70       if ( searchKey < b[ middle ] )
71         high = middle - 1;  // search low end of array
72
73       // if searchKey greater than middle element,
74       // set new low element
75       else
76         low = middle + 1;   // search high end of array
77   }
78
79   return -1;  // searchKey not found
80
81 } // end function binarySearch
```

fig04_20.cpp (5 of 6)

```
82
83  // print header for output
84  void printHeader( int size )
85  {
86     cout << "\nSubscripts:\n";
87
88     // output column heads
89     for ( int j = 0; j < size; j++ )
90        cout << setw( 3 ) << j << ' ';
91
92     cout << '\n';  // start new line of output
93
94     // output line of - characters
95     for ( int k = 1; k <= 4 * size; k++ )
96        cout << '-';
97
98     cout << endl;  // start new line of output
99
100 } // end function printHeader
101
```


fig04_20.cpp (6 of 6)

```
102 // print one row of output showing the current
103 // part of the array being processed
104 void printRow( const int b[], int low, int mid,
105    int high, int size )
106 {
107    // loop through entire array
108    for ( int m = 0; m < size; m++ )
109
110       // display spaces if outside current subarray range
111       if ( m < low || m > high )
112          cout << "   ";
113
114       // display middle element marked with a *
115       else
116
117          if ( m == mid )         // mark middle value
118             cout << setw( 3 ) << b[ m ] << '*';
119
120          // display other elements in subarray
121          else
122             cout << setw( 3 ) << b[ m ] << ' ';
123
124    cout << endl;  // start new line of output
125
126 } // end function printRow
```

fig04_20.cpp output (1 of 2)

```
Enter a number between 0 and 28: 6

Subscripts:
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
---------------------------------------------------------
 0  2  4  6  8 10 12 14* 16 18 20 22 24 26 28
 0  2  4  6* 8 10 12

6 found in array element 3



Enter a number between 0 and 28: 25

Subscripts:
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
---------------------------------------------------------
 0  2  4  6  8 10 12 14* 16 18 20 22 24 26 28
                   16 18 20 22* 24 26 28
                               24 26* 28
                               24*

25 not found
```


fig04_20.cpp output (2 of 2)

```
Enter a number between 0 and 28: 8

Subscripts:
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
---------------------------------------------------------
 0  2  4  6  8 10 12 14* 16 18 20 22 24 26 28
 0  2  4  6* 8 10 12
            8 10* 12
            8*

8 found in array element 4
```

Radix Sort Example

Original Array	Array After 1st Pass	Array After 2nd Pass	Array After 3rd Pass
762	800		
124	100		
432	761		
761	001		
800	762		
402	432		
976	402		
100	124		
001	976		
999	999		

- Insert original array into the table as shown
- Collect the values so they are ordered as shown in Array After 1st Pass

Table

	[0]	[1]	[2]	[3]	…	[n-1]
[0]	800	100				
[1]	761	001				
[2]	762	432	402			
[3]						
[4]	124					
[5]						
[6]	976					
[7]						
[8]						
[9]	999					

Example

Original Array	Array After 1st Pass	Array After 2nd Pass	Array After 3rd Pass
762	800	800	
124	100	100	
432	761	001	
761	001	402	
800	762	124	
402	432	432	
976	402	761	
100	124	762	
001	976	976	
999	999	999	

- Doing the same process for the tens place

Table

	[0]	[1]	[2]	[3]	…	[n-1]
[0]	800	100	001	402		
[1]						
[2]	124					
[3]	432					
[4]						
[5]						
[6]	761	762				
[7]	976					
[8]						
[9]	999					

Example

Original Array	Array After 1st Pass	Array After 2nd Pass	Array After 3rd Pass
762	800	800	001
124	100	100	100
432	761	001	124
761	001	402	402
800	762	124	432
402	432	432	761
976	402	761	762
100	124	762	800
001	976	976	976
999	999	999	999

- Doing the same process for the hundreds place

Table

	[0]	[1]	[2]	[3]	…	[n-1]
[0]	001					
[1]	100	124				
[2]						
[3]						
[4]	402	432				
[5]						
[6]						
[7]	761	762				
[8]	800					
[9]	976	999				