CHAPTER 3
DATA STRUCTURES: SORTING

3.1 Sorting

- "A process of arranging given data elements in some specific/sequential order."
- Order can be -
  - i) Numerical order: Ascending and Descending order
  - ii) Logical order: Alphabetical order
- Various sorting methods are: [BQ SIR MT]
  - i) **B**ubble Sort
  - ii) **Q**uick Sort
  - iii) **S**election Sort
  - iv) **I**nsertion Sort
  - v) **R**adix Sort
  - vi) **M**erge Sort
  - vii) **T**ree Sort

3.1.1 Bubble Sort

- **Basic Principle:**
  - "Smaller values move up in each pass as like the air bubbles move up in the water."
- **Working:**
  - (As covered in class.)
- **Example:**

Index	I/P	1st Pass				2nd Pass			3rd Pass		4th Pass
0	**20**	10	10	10	**10**	10	10	**10**	10	**10**	5
1	**10**	**20**	20	20	**20**	**20**	15	**15**	**15**	**5**	10
2	25	**25**	**25**	15	15	**15**	**20**	5	**5**	15	15
3	15	15	**15**	**25**	5	5	**5**	20	20	20	20
4	5	5	5	**5**	25	25	25	25	25	25	25

  - (Explain as covered in class.)

- **Function:**

```
void bubble_sort( int  a[ ] )
{
        int i,j,temp;

        for(i=0; i<MAX-1; i++)
        {
                for(j=0; j<MAX-1-i; j++)
                {
                        if( a[j] > a[j+1])
                        {
                                temp = a[j];
                                a[j] = a[j+1];
                                a[j+1] = temp;
                        }
                }
        }
}
```

- **Algorithm:**
  - **BUBBLE_SORT (A)**
    - [Sorts elements of given array A in ascending order.]
  - **Variables:**
    - **i) A:** Array having MAX elements.
    - **ii) MAX:** No. of maximum elements in the given array.
    - **iii) TEMP:** Stores an element temporarily for exchange purpose.
    - **iv) i, j:** Loop control variables.
  - **Steps:**
    - **Step-1:** [Outer Loop: Loop to control each pass.]
      Repeat Through Step-2
      FOR i = 0, 1, 2 … (MAX-1)
    - **Step-2:** [Inner Loop: Loop to control comparisons.]
      Repeat Through Step-3
      FOR j = 0, 1, 2 … (MAX-i-1)
    - **Step-3:** [Compare element with next element and exchange if required.]
      IF (A[j] > A[j+1]) THEN
      TEMP ← A[j]
      A[j] ← A[j+1]
      A[j+1] ← TEMP
      END IF
    - **Step-4:** [Finish]
      RETURN

3.1.2 Quick Sort / Partition Exchange Sort

- **Basic Principle:**
  - "Each time one key element (value) is placed to its position - dividing all elements into two parts, where
    - Left part contains elements smaller than the key element, and
    - Right part contains elements greater than the key element."
- **Working:** (As covered in class.)
- **Example:**

Data	Operation
25 15 10 35 30 5 40	Input
25 15 10 35 30 5 40 d u	Initialize (Key, down, up)
25 15 10 35 30 5 40 d → d → d → d u	Update 'down'
25 15 10 35 30 5 40 d → d → d → d u ← u	Update 'up'
25 15 10 5 30 35 40 d → d → d → d u ← u	Exchange
25 15 10 5 30 35 40 d → d u	Update 'down'
25 15 10 5 30 35 40 d → d u ← u ← u	Update 'up'
5 15 10 **25** 30 35 40 u d	Exchange 'key'
[5 15 10] **25** [30 35 40] u d	Divide all elements on left & right sides of the key element into two partitions
. . . Continue till end	Apply a quick sort on each part separately

  - (Explain as covered in class.)

- **Function:**

```
void quick_sort(int a[ ], int start, int end)
{
        int key, down, up, temp;

        // intitialize…
        key = a[start];  down = start;   up = end;

        while( down < up)
        {
                // update down…
                while ( a[down]<=key && down < end )
                        down++;

                // update up…
                while ( a[up] > key)
                        up--;

                // check whether down & up crossed each other or combined.
                if ( down < up)
                {
                        // exchange a[down] and a[up]…
                        temp = a[down];     a[down] = a[up];  a[up] = temp;
                }
        }

        // exchange a[start] and a[up], i.e. key value…
        temp = a[start];          a[start] = a[up];          a[up] = temp;

        // apply quick sort on the left partition…
        if ( up-1 > start)
                quick_sort ( a, start, up-1);

        // apply quick sort on the right partition…
        if ( up+1 < end)
                quick_sort ( a, up+1, end);
}
```

- **Algorithm:**
  - **QUICK_SORT (A, START, END)**
    - [Sorts elements of given array A in ascending order.]
  - **Variables:**
    - **i) A:** Array having MAX elements.
    - **ii) MAX:** No. of maximum elements in the given array.
    - **iii) START:** Refers to first position (index) in current partition.
    - **iv) END:** Refers to the last position (index) in current partition.
    - **v) DOWN, UP:** Indices to access array elements.
    - **vi) TEMP:** Stores an element temporarily for exchange purpose.

❖ **Steps:**

- **Step-1:** [Initialize KEY, DOWN and UP]

  KEY ← A[START] DOWN ← START UP ← END

- **Step-2:** [Loop until the proper position of key element is found]

  Repeat through step-5 WHILE (DOWN < UP)

- **Step-3:** [Scan from left to right and update down]

  WHILE (A[DOWN] <= KEY AND DOWN<END)
  DO
  DOWN ← DOWN + 1
  END WHILE

- **Step-4:** [Scan from right to left and update up]

  WHILE (A[UP] > KEY)
  DO
  UP ← UP-1
  END WHILE

- **Step-5:** [Exchange A[UP] and A[DOWN]]

  IF (DOWN < UP) THEN
  TEMP←A[DOWN] A[DOWN]←A[UP] A[UP]←TEMP
  END IF

- **Step-6:** [Exchange A[UP] and KEY]

  TEMP←A[START] A[START]←A[UP] A[UP]←TEMP

- **Step-7:** [Apply quick sort on left partition]

  IF (UP-1 > START) THEN
  QUICK_SORT (A, START, UP-1)
  END IF

- **Step-8:** [Apply quick sort on right partition]

  IF (UP+1 < END) THEN
  QUICK_SORT (A, UP+1, END)
  END IF

- **Step-9:** [Finish]

  RETURN

3.1.3 Selection Sort

- **Basic Principle:**
  - "Smallest (minimum) elements are found from remaining elements and positioned at their proper locations."
- **Working:**
  - (As covered in class.)
- **Example:**

Index	Input	1st Pass	2nd Pass	3rd Pass	4th Pass
0	20	5	5	5	5
1	10	10	10	10	10
2	25	25	25	15	15
3	15	15	15	25	25
4	5	20	20	20	20

  - (Explain as covered in class.)

- **Function:**

```
void selection_sort( int a[ ] )
{
        int i, j, pos, min, temp;

        for(i=0; i<MAX-1; i++)
        {
                min=a[i];       pos=i;
                for(j=i+1; j<MAX; j++)
                {
                        if( min > a[j])
                        {
                                min = a[j];       pos = j;
                        }
                }
                if(pos != i)
                {
                        temp = a[i];     a[i] = a[pos];     a[pos] = temp;
                }
        }
}
```

- **Algorithm:**
  - **SELECTION_SORT (A)**
    - [Sorts elements of given array A in ascending order.]
  - **Variables:**
    - **i) A:** Array having MAX elements.
    - **ii) MAX:** No. of maximum elements in a given array.
    - **iii) MIN:** Stores minimum element during each pass.
    - **iv) POS:** Keeps position of minimum value.
    - **v) TEMP:** Stores an element temporarily for exchange purpose.
    - **vi) i, j:** Loop control variables.
  - **Steps:**

- **Step-1:** [Outer Loop: Loop to control each pass.]
  Repeat Through Step-4
  FOR i = 0, 1, 2 … (MAX-1)

- **Step-2:** [Initialize MIN and POS.]
  MIN ← A[i] POS ← i

- **Step-3:** [Inner Loop: Find minimum value.]
  Repeat FOR j := (i+1) TO (MAX)
  DO
  IF (A[j] < MIN) THEN
  MIN ← A[j] POS ← j
  END IF
  END FOR

- **Step-4:** [Exchange minimum value.]
  IF (POS ≠ i) THEN
  TEMP ← A[i]
  A[i] ← A[POS]
  A[POS] ← TEMP
  END IF

- **Step-5:** [Finish]
  RETURN

3.1.4 Insertion Sort

- **Basic Principle:**
  - "Choose each element one by one, and, insert it at its proper position."
- **Working:**
  - (As covered in class.)
- **Example:**

Index	Input	1st Pass	2nd Pass	3rd Pass	4th Pass
0	20	10	10	10	5
1	10	20	20	15	10
2	25	25	25	20	15
3	15	15	15	25	25
4	5	5	5	5	20

  - (Explain as covered in class.)

- **Function:**

```
void insertion_sort( int a[ ] )
{
        int i, j, temp;

        for (i=1; i<MAX; i++)
        {
                temp = a[i];
                for (j=i-1; j>=0 && a[j]>temp; j - -)
                {
                        a[j+1] = a[j];
                }
                a[j+1] = temp;
        }
}
```

- **Algorithm:** (Prepare on your own.)

3.1.5 Radix Sort / Bucket Sort / Digital Sort

- **Basic Principle:**
  - Non-comparative sorting algorithm.
  - Uses buckets to sort data based on digits – starting from least significant to most significant.
- **Working:**
  - (As covered in class.)
- **Example:**
  - Input: 27, 75, 19, 71, 43

Bucket	1st Pass	2nd Pass
0		
1	71	19
2		27
3	43	
4		43
5	75	
6		
7	27	71, 75
8		
9	19	

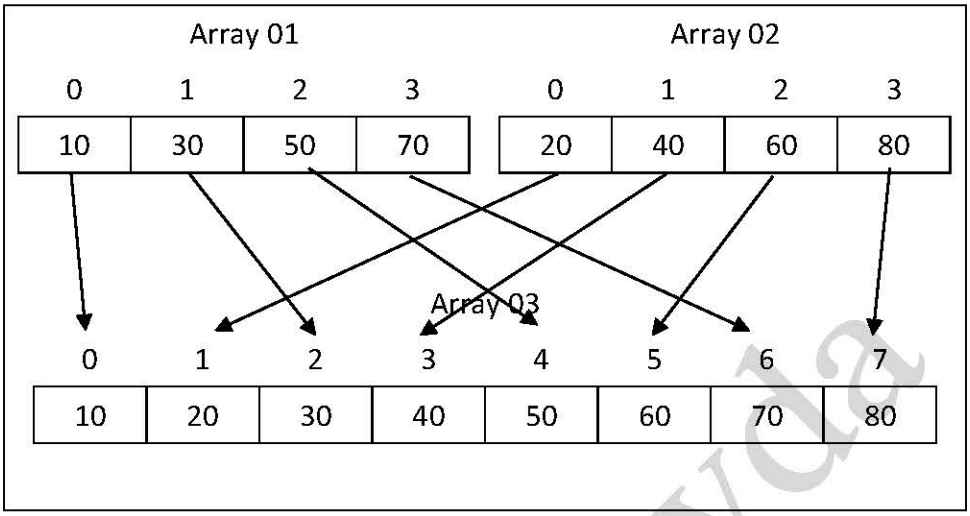
Sequence					
I/P:	27	75	19	71	43
After P1:	71	43	75	27	19
After P2:	19	27	43	71	75
O/P:	19	27	43	71	75

  - (Explain as covered in class.)

3.1.6 Merge Sort

- **Basic Principle:**
  - Divide and Conquer.
  - Divide list into several sub-lists, until each sub-list has only single element.
  - Then, compare pair of elements, place into order, and combine. Repeat this step until entire list is formed again.
- **Working:**
  - (As covered in class.)

- **Example – Basic Merge Sort:**

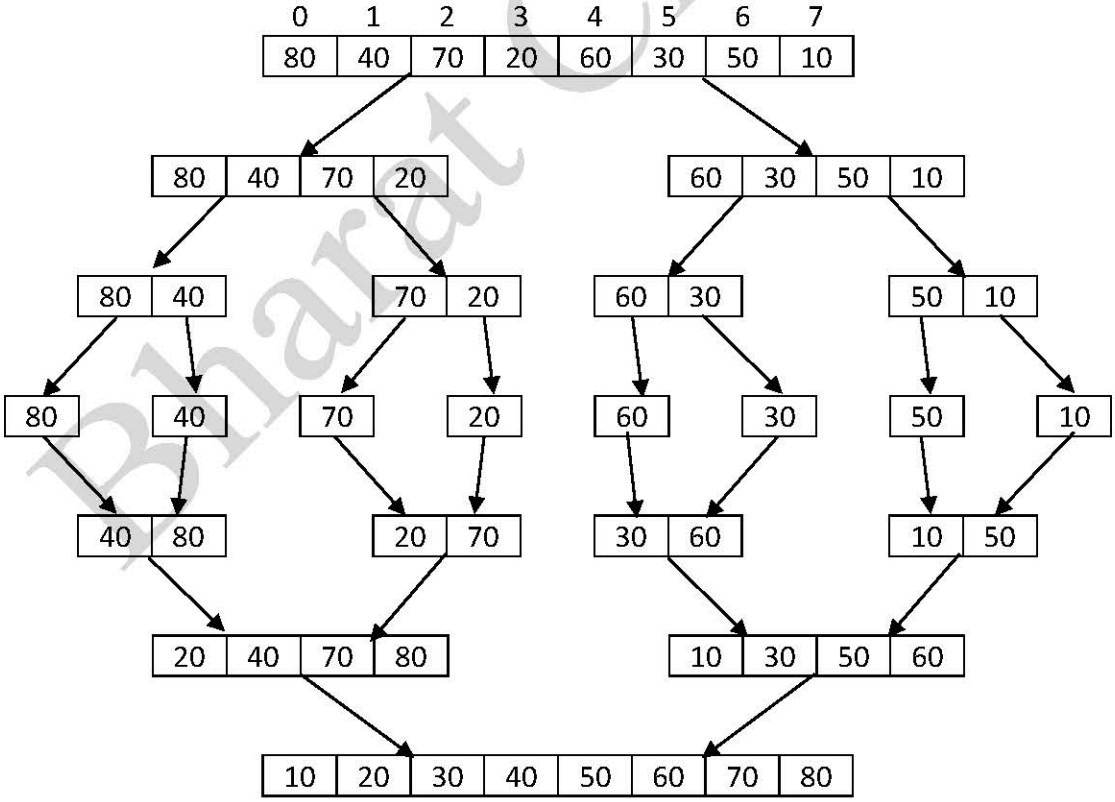


- **Implementation:**

```
#include<stdio.h>

#define N1 4
#define N2 4
#define N3 8

void main ( )
{
        int a1[N1], a2[N2], a3[N3], i;
        void merge_sort ( int *,int *,int *);

        printf("Enter elements for First Array:\n");
        for( i=0; i<N1; i++)
                scanf("%d", &a1[i]);

        printf("Enter elements for Second Array:\n");
        for( i=0; i<N2; i++)
                scanf("%d", &a2[i]);

        merge_sort (a1, a2, a3 );

        printf("Sorted Elements in Third Array: ");
        for(i=0; i<N3; i++)
                printf("   %d", a3[i]);
}
```

```
void merge_sort ( int  a1[ ], int  a2[ ], int  a3[ ] )
{
        int c1=0, c2=0, c3=0;
        while ( c1<N1 && c2<N2)
        {
                if( a1[c1] < a2[c2] )
                {       a3[c3] = a1[c1];        c1++;   c3++;   }
                else
                {       a3[c3] = a2[c2];        c2++;   c3++;   }
        }
        while( c1 < N1 )
        {       a3[c3] = a1[c1];        c1++;   c3++;   }
        while( c2 < N2 )
        {       a3[c3] = a2[c2];        c2++;   c3++;   }

}
```

- **Example – Merge Sort:**
  - Input: 80, 40, 70, 20, 60, 30, 50, 10



● ● ●