Paper code: U218-144(T2)

OCTOBER 2018/ IN-SEM (T2)

S. Y. B. TECH. (~~PROGRAM~~ IT) (SEMESTER - I)

COURSE NAME: FUNDAMENTALS OF DATA STRUCTURES

COURSE CODE: ITUA21174

Marking scheme:

Q.1) a) Merge sort: 2M
Merge two sorted arrays: 3M
Complexity:1M

b) Linear search: 3M (Algo:2M, example:1M)
Binary search: 3M (Algo:2M, example:1M)

c) Importance of searching-sorting:2M
Sort stability: 2M

Q.2) a) Correct passes with pivot: 6M

b) Sorted Input: 2M
Searching 28: 2M
Searching 80: 2M

c) Methodology and efficiency: 3M, Example:1M

Q.3) a) Representation: 2M Simple transpose algo: 3M Complexity:1M

b) Data:1M and Methods with correct syntax:3M

c) Row major matrix:2M, Memory address calculation:2M

Q.4)a) Ordered list: 2M
Each polynomial representation: 2M

b) Pseudo C code: 4M Example:2M
c) C function to convert a conventional matrix to sparse matrix: 4M

Paper code: U218-144(T2)

OCTOBER 2018/ IN-SEM (T2)

S. Y. B. TECH. (~~PROGRAM~~ IT) (SEMESTER - I)

COURSE NAME: FUNDAMENTALS OF DATA STRUCTURES

COURSE CODE: ITUA21174

Solution:

Q.1) a)

```
void merge_sort_recursive(int Ar[30], int low, int high)
{
 if(low<high)
 {
  mid=(low+high)/2;
  merge_sort(Ar,low,mid);
  merge_sort(Ar,mid+1,high);
  merge(Ar,low,mid,mid+1,high);
    }
 }
```

or

```
 void merge_sort_iterative (int Ar[30], int n)
{
int size,i,j,k,u1,u2,l1,l2,temp[30];
  for(size=1;size<n;size=2*size)
{
        l1=0;k=0;
        while (l1+size<n)
        {
        u1=l1+size-1;
        l2=u1+1;
        u2=l2+size-1;
        if(u2>=n)
                u2=n-1;
        merge(Ar,l1,u1,l2,u2);
        l1=u2+1;
        }
}
void merge(int A[30], int l1,int u1,int l2,int u2)
{
 int temp[40],i,j,k=0;
 i=l1;   j=l2;
 while(i<=u1 && j<=u2)
  {
   if(A[i]<A[j])
    {
     temp[k]=A[i];
      k++;          i++;              //Smaller elements is copied
    }
   else
    {
```

```
        temp[k]=A[j];
        k++;        j++;
       }
    }
 while(i<=u1)                    //Copy remaining elements
  {
   temp[k]=A[i];
   k++;    i++;
  }
  while(j<=u2)
  {
   temp[k]=A[j];
   k++;    j++;
  }
  for(i=0,j=l1;j<=u2;i++,j++)
  {
   A[j]=temp[i];
  }
}
}
```

At each level in the binary tree created for Merge Sort, there are n elements, with O(1) time spent at each element

→ O(n) running time for processing one level

The height of the tree is O(log n)

Therefore, the time complexity is ***O***(nlog n)

b) Linear search:

- Array **numlist** contains 17,23,5,11,2,29,3
- Searching for the the value **11**, linear search examines **17**, **23**, **5**, and **11**
- Searching for the the value **7**, linear search examines **17**, **23**, **5**, **11**, **2**, **29**, and **3**

```
Set found to false
  Set position to –1
  Set index to 0
       While index < number of elts and found is false
              If list [index] is equal to search value
                 found = true
                 position = index
              End If
              Add 1 to index
       End While
       Return position
```

Binary search:

- Array **numlist** contains 17,23,5,11,2,29,3
- Searching for the the value **11**, binary search examines **11** and stops
- Searching for the the value **7**, binary search examines **11**, **3**, **5**, and stops
- The binary search is much more efficient than the linear search.
- It requires the list to be in order.

- The algorithm starts searching with the middle element. (If data is sorted in ascending order)
    - If the item is less than the middle element, it starts over searching the first half of the list.
    - If the item is greater than the middle element, the search starts over starting with the middle element in the second half of the list.
    - It then continues halving the list until the item is found.

c) Fundamental problems in computer science and programming. There are some very common problems that we use computers to solve.

- ☐ Placing records in order, which we call **sorting**
- ☐ Sorting done to make searching easier
    - Sorting Books in Library
    - Sorting Individuals by Height (Feet and Inches)
    - Sorting songs(Alphabetical)
    - Arranging folders in order
    - Sorting Numbers (Sequential)
- ☐ **Searching** through a lot of records for a specific record or set of records.
    - Finding telephone number in directory
    - Search a book in library
    - Searching a particular customer record

Q.2) a)

10	8	-9	12	1	-5	7	20	11	15
pivot	i								j
-5	8	-9	12	1	**10**	12	20	11	15
-9	**-5**	8	12	1	10	11	**12**	20	15
-9	-5	1	**8**	12	10	11	12	**15**	20

b) Sorted Input: 2,9,11,12,15,21,30,32,40,54,55,58,66,80,95
Searching 28: Mid=32, 12, 21, 30 Not found
Searching 80: Mid= 32, 58, 80 Found at position 13

c) Selection sort:

	Selection sort:	Insertion sort:

Metho dology	Selects the next smallest item in the array each time from unsorted elements and places it at the beginning of unsorted array
	for i ← 0 to N-2 do {
	min ← i;
	for j ← i+1 to N-1 do
	if (A[j] < A[min]) min ← j
	}
	swap A[i] and A[min] ;
	}
	Given an array of length n,
		-For each array element from the second to the last (nextPos = 1)
		-Insert the element at nextPos where it belongs in the array, increasing the length of the sorted subarray by 1
		for(k=1;k<n;k++)
		//passes
		{
		y=a[k]; //current element

	• Search elements 0 through n-1 and select the smallest
	• Swap it with the element in location 0
	• Search elements 1 through n-1 and select the smallest
	• Swap it with the element in location 1
	Continue in this fashion until there's nothing left to search
		for(i=k-1;i>=0
		&& y<a[i];i--) //check previous elements
		{
		a[i+1]=a[i]; //shift greater elements to right
		a[i]=y;
		}

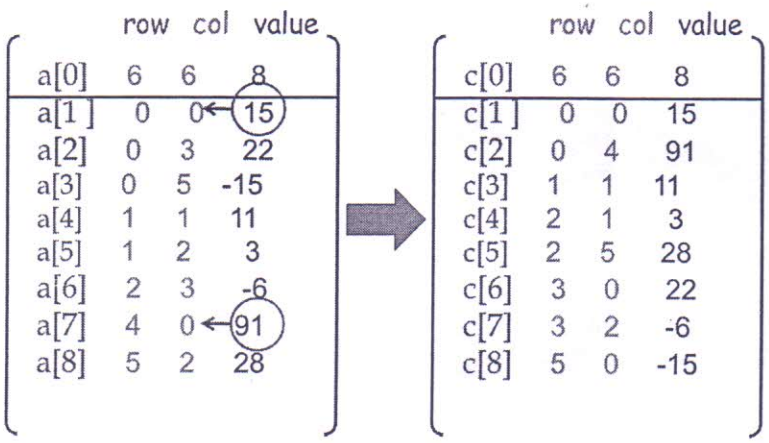
Example	No. of swapping is reduced than bubble sort. One swap per pass. • Ex. 45 -40 190 99 11 • Pass1:-40 45 190 99 11 • Pass2:-40 11 190 99 45 • Pass3:-40 11 45 190 99 • Pass4:-40 11 45 99 190 • Pass4:-40 11 45 99 190	0 1 2 3 4 a 45 -40 190 99 11 b 0 0 0 0 0 Step1:a 45 -40 190 99 11 b 45 0 0 0 0 Step2:a 45 -40 190 99 11 b -40 45 0 0 0 Step3:a 45 -40 190 99 11 b -40 45 190 0 0 Step4:a 45 -40 190 99 11 b -40 45 99 190 0 Step5:a 45 -40 190 99 11 b -40 11 45 99 190
Efficiency	O(n^2)	O(n^2)

Q.3) a) A lot of "zero" entries. Thus large memory space is wasted.

- Use triple <row, col, value> to characterize an element in the matrix.
- Use array of triples a[] to represent a matrix.

$$A = \begin{pmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{pmatrix}$$

	row	col	value
a[0]	6	6	8
a[1]	0	0	15
a[2]	0	3	22
a[3]	0	5	-15
a[4]	1	1	11
a[5]	1	2	3
a[6]	2	3	-6
a[7]	4	0	91
a[8]	5	2	28



Simple transpose algorithm: Find all elements in col. 0, and store them in row 0;
Find all elements in col. 1, and store them in row 1; ... ... etc

```
for (j=0; j<#col; j++) {     //O(#col)
   for all elements in col j {  //O(#terms)
      place element (i, j, value) in the
            next position of array c[];
   }
}
```

b) Abstract Data Types for Polynomial.

- Object: Array of PolynomialTerms (coefficient and power of each term)
- Operations:
  - Boolean IsZero(poly)

::= return FALSE or TRUE.

  - Coefficient Coeff(poly, expon)

::= return coefficient of x^{expon}

  - int Eval(poly, ValueX)

::= return value of polynomial

  - Polynomial Add(poly1, poly2)

::= return poly1 + poly2

  - Polynomial multiply (poly1, poly2)

::= return poly1 * poly2

c) If the elements are stored in row wise manner then it is called "Row Major Representation".

Address calculation for any element will be as follows

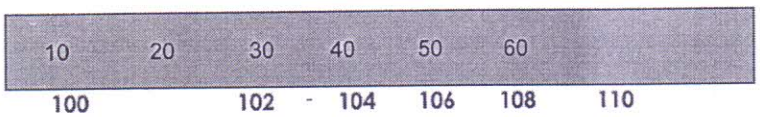
In row major matrix, the element a[i][j] will be:

base address + (i * total number of columns + j)*size of each element

A[2][4] in A[5][6]= 1000+ (2*6+4)*2 =1032

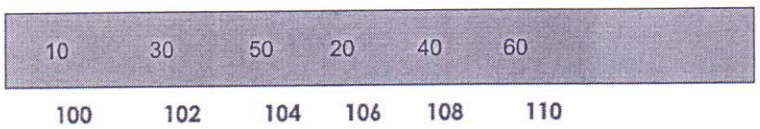
- Then in a row major matrix

a[0][0] a[0][1] a[1][0] a[1][1] a[2][0] a [2][1]



And in the column major matrix

a[0][0] a[0][1] a[1][0] a[1][1] a[2][0] a [2][1]

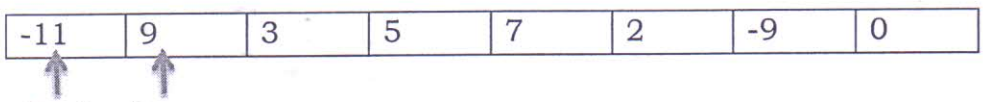


Q.4) a)ordered list is: ordered by "position" i.e. this is a "position oriented list"

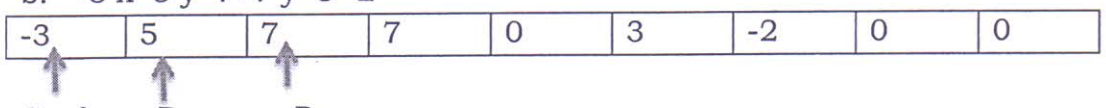
- Linear list
- lists ordered by time,
- grocery lists as in the order we think about it
- to-do lists: In priority order...
- a *sorted list* is a "value oriented list",
- Eg- days of week, digits etc.

- integer = {0, +1, -1, +2, -2, +3, -3, ...}
- daysOfWeek = {S,M,T,W,Th,F,Sa}
  - Data structures used: Array(static & dynamic) & linked list
  - Operations: Display, Search, Insert, Delete

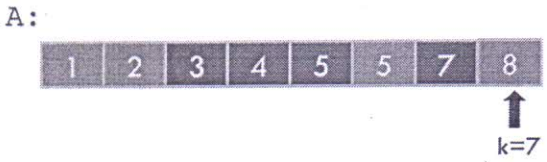
a. -11 x^9 + 3 x^5 + 7 x^2 - 9



b. -3 x^5 y^7+7 y^3- 2



b)



```
void merge(int A[30], int l1,int u1,int l2,int u2)
{
 int temp[40],i,j,k=0;
 i=l1; j=l2;
 while(i<=u1 && j<=u2)
  {
   if(A[i]<A[j])
    {
     temp[k]=A[i];
      k++;          i++;          //Smaller elements is copied
    }
   else
    {
     temp[k]=A[j];
     k++;       j++;
     }
  }
 while(i<=u1)                //Copy remaining elements
 {
  temp[k]=A[i];
  k++;
  i++;
 }
  while(j<=u2)
  {
   temp[k]=A[j];
   k++;
   j++;
  }
}
```

c) C function to convert a conventional matrix to sparse matrix:

```
void Convert(int M[5][5], int r, int c,  int S[ ][3])
{       k=1;
        S[0][0]=r;
        S[0][1]=c;
        for (i=0; i<r; i++)
                for (j=0; j<c; j++)
                        {
                                if ( M[i][j] !=0)
                                        {
                                        S[k][0]=i;
                                        S[k][1]=j;
                                        S[k][2]=M[i][j];
                                        k++;
                                        }
                        }
S[0][2]=k-1;
}
```