Data Structures Using C Question Bank

Q1. List out the areas in which data structures are applied extensively?

Ans: Compiler Design, Operating System, Database Management System, Statistical analysis package, Numerical Analysis, Graphics, Artificial Intelligence, Simulation

Q2. What are the major data structures used in the following areas : RDBMS, Network data model & Hierarchical data model. ?

Ans: The major data structures used are as follows:

- RDBMS - Array (i.e. Array of structures)
- Network data model - Graph
- Hierarchical data model - Trees

Q3.If you are using C language to implement the heterogeneous linked list, what pointer type will you use?

Ans: The heterogeneous linked list contains different data types in its nodes and we need a link, pointer to connect them. It is not possible to use ordinary pointers for this. So we go for void pointer. Void pointer is capable of storing pointer to any type as it is a generic pointer type.

Q4. Minimum number of queues needed to implement the priority queue?

Ans: Two. One queue is used for actual storing of data and another for storing priorities.

Q5.What is the data structures used to perform recursion?

Ans: Stack. Because of its LIFO (Last In First Out) property it remembers its 'caller' so knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls.

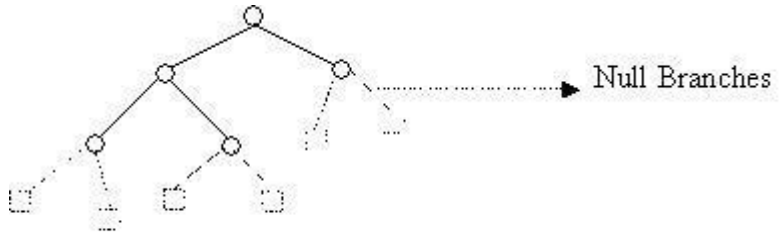
Every recursive function has its equivalent iterative (non-recursive) function. Even when such equivalent iterative procedures are written, explicit stack is to be used.

Q6. What are the notations used in Evaluation of Arithmetic Expressions using prefix and postfix forms?

Ans: Polish and Reverse Polish notations.

Q7. How many null nodes will a binary tree with 20 nodes have? Ans: 21

[Hint: Let us take a tree with 5 nodes (n=5)



It will have only 6 (ie,5+1) null branches.
A binary tree with n nodes has exactly n+1 null nodes.

Q8. What are the methods available in storing sequential files?

Ans: The methods available in storing sequential files are:

- Straight merging,
- Natural merging,
- Polyphase sort,
- Distribution of Initial runs.

Q9.List out few of the Application of tree data-structure?

Ans: The list is as follows:

- The manipulation of Arithmetic expression,
- Symbol Table construction,
- Syntax analysis.

Q10. List out few of the applications that make use of Multilinked Structures?

Ans: The applications are listed below:

- Sparse matrix,
- Index generation.

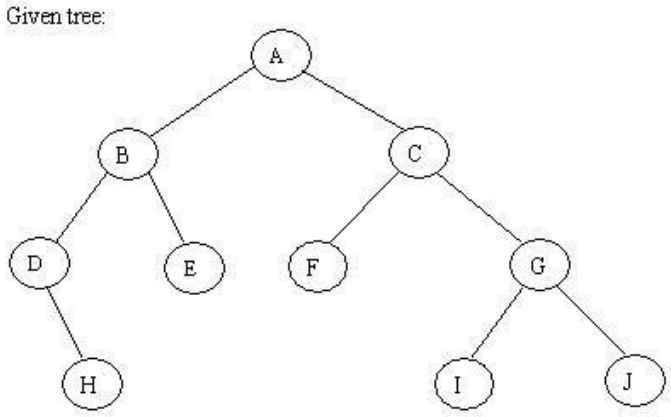
Q11.In tree construction which is the suitable efficient data structure?

Ans: Linked list is the efficient data structure.

Q12.What is the type of the algorithm used in solving the 8 Queens problem?

Ans: Backtracking

Q13.Traverse the given tree using Inorder, Preorder and Postorder traversals.



Ans:

•Inorder : D H B E A F C I G J

•Preorder: A B D H E C F G I J

•Postorder: H D E B F I J G C A

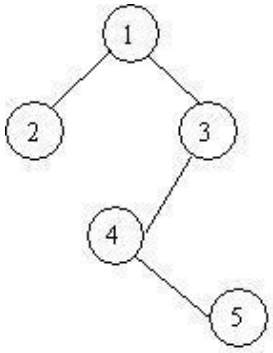
Q 14.There are 8, 15, 13, 14 nodes were there in 4 different trees. Which of them could have formed a full binary tree?

Ans: 15

[Hint : In general:There are 2n-1 nodes in a full binary tree.

By the method of elimination:Full binary trees contain odd number of nodes. So there cannot be full binary trees with 8 or 14 nodes, so rejected. With 13 nodes you can form a complete binary tree but not a full binary tree. So the correct answer is 15.]

Q15.In the given binary tree, using array you can store the node 4 at which location?



Ans: At location 6

1	2	3	-	-	4		--	5

Root	LC1	RC1	LC2	RC2	LC3	RC3	LC4	RC4

where LCn means Left Child of node n and RCn means Right Child of node n

Q16.Sort the given values using Quick Sort?

65	70	75	80	85	60	55	50	45

Ans: Sorting takes place from the pivot value, which is the first value of the given elements, this is marked bold. The values at the left pointer and right pointer are indicated using L and R respectively.

65	70^{L}	75	80	85	60	55	50	45^{R}

Since pivot is not yet changed the same process is continued after interchanging the values at L and R positions

65	45	75^{L}	80	85	60	55	50^{R}	70
65	45	50	80^{L}	85	60	55^{R}	75	70
65	45	50	55	85^{L}	60^{R}	80	75	70
65	45	50	55	60^{R}	85^{L}	80	75	70

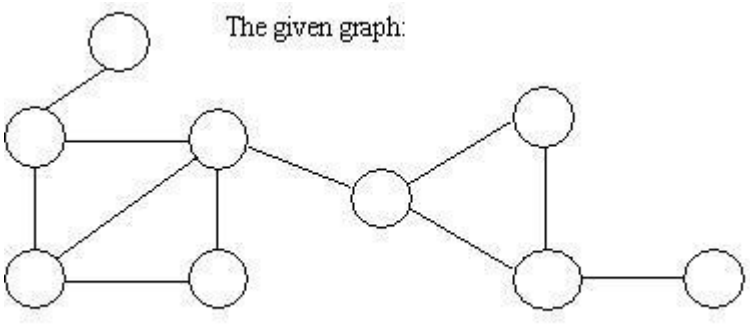
When the L and R pointers cross each other the pivot value is interchanged with the value at right pointer. If the pivot is changed it means that the pivot has occupied its original position in the sorted order (shown in bold italics) and hence two different arrays are formed, one from start of the original array to the pivot position-1 and the other from pivot position+1 to end.

60^{L}	45	50	55^{R}	65	85^{L}	80	75	70^{R}
55^{L}	5	45	50^{R}	60	65	70^{R}	80^{L}	75	85
50^{L}	45^{R}	55	60	65	70	80^{L}	75^{R}	85

In the next pass we get the sorted form of the array.

45	50	55	60	65	70	75	80	85

Q17. For the given graph, draw the DFS and BFS?



Ans:

- BFS: A X G H P E M Y J
- DFS: A X H P E Y M J G

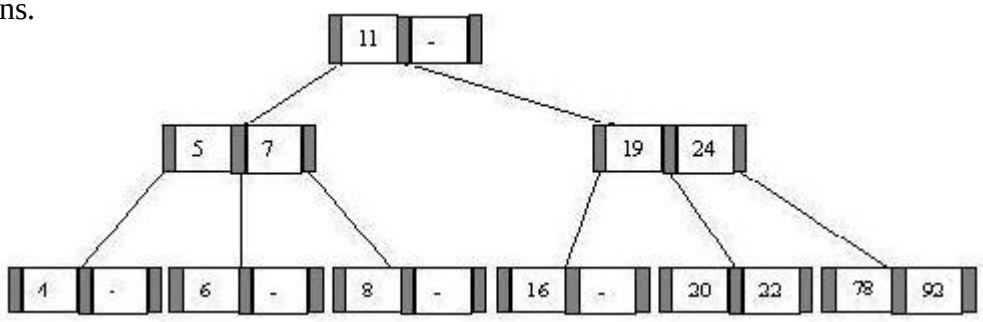
Q18. Classify the Hashing Functions based on the various methods by which the key value is found.
Ans: The list of Hashing functions is as follows:
•Direct method
•Subtraction method
•Modulo-Division
method •Digit-Extraction
method •Mid-Square
method •Folding method
•Pseudo-random method

Q19.What are the types of Collision Resolution Techniques and the methods used in each of the type?
Ans: The types of Collision Resolution Techniques are:
•Open addressing (closed hashing)
The methods used include:
•Overflow block
•Closed addressing (open
hashing) The methods used
include: •Linked list
•Binary tree

Q20.In RDBMS, what is the efficient data structure used in the internal
storage representation?
Ans: B+ tree. Because in B+ tree, all the data is stored only in leaf nodes, that makes searching easier. This corresponds to the records that shall be stored in leaf nodes.

Q21.Draw the B-tree of order 3 created by inserting the following data arriving in sequence - 92 24 6 7 11 8 22 4 5 16 19 20 78

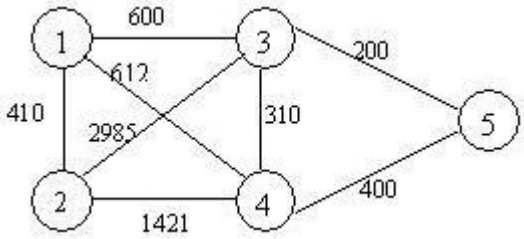
Ans.



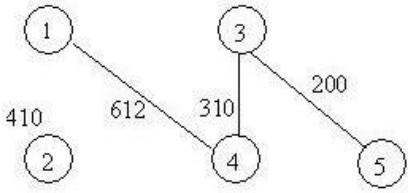
Q22. What is a spanning Tree?
Ans: A spanning tree is a tree associated with a network. All the nodes of the graph appear on the tree once. A minimum spanning tree is a spanning tree organized so that the total edge weight between nodes is minimized.

Q23. Does the minimum spanning tree of a graph give the shortest distance between any 2 specified nodes?
Ans: No. Minimal spanning tree assures that the total weight of the tree is kept at its minimum. But it doesn't mean that the distance between any two nodes involved in the minimum-spanning tree is minimum.

Q24.Convert the given graph with weighted edges to minimal spanning tree.

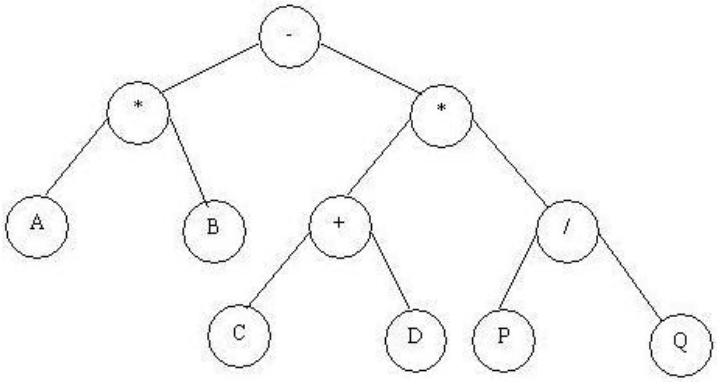


Ans: the equivalent minimal spanning tree is:

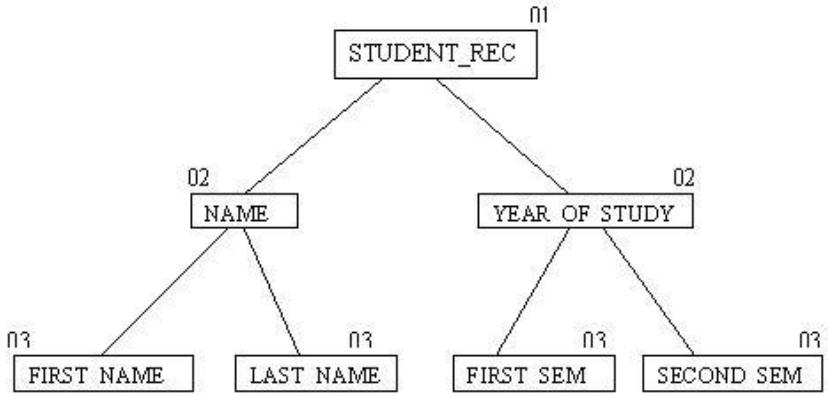


Q25. Whether Linked List is linear or Non-linear data structure?
Ans: According to Access strategies Linked list is a linear one.
According to Storage Linked List is a Non-linear one.

Q26.Draw a binary Tree for the expression : A * B - (C + D) * (P /
Ans:



Q27. For the following COBOL code, draw the Binary tree?
01 STUDENT_REC.
02 NAME.
03 FIRST_NAME PIC X(10).
03 LAST_NAME PIC X(10).
02 YEAR_OF_STUDY.
03 FIRST_SEM PIC XX.
03 SECOND_SEM PIC XX.
Ans:



Q28. What is the difference between a queue and a stack?
Ans: A queue is typically FIFO (priority queues don't quite follow that) while a stack is LIFO. Elements get inserted at one end of a queue and retrieved from the other, while the insertion and removal operations for a stack are done at the same end.

Q29.What is the difference between storing data on the heap vs. on the stack?
Ans: The stack is smaller, but quicker for creating variables, while the heap is limited in size only by how much memory can be allocated. Stack would include most compile time variables, while heap would include anything created with malloc or new. (This is for C/C++, and not strictly the case.)

Q30. Let W be the string ABCD (a) Find the length of W. (b) List all substings of W.

Answer: (a) Length = 4

(b) ABCD, ABC,BCD, AB,BC, CD, A,B,C,D and null string

Q31.Consider the linear arrays AAA(5:50), BBB (-5:10) and CCC(18).

(a) Find the number of elements in each array

(b) Suppose Base(AAA) = 300 and w=4 words per memory cell for AAA. Find the address of AAA[15], AAA[35] and AAA[55]

Ans : (a) Length = Upper Bound (UB) – Lower Boun d (LB) +1

Length(AAA) = 50-5+1 = 46

Length(BBB) = 10-(-5)+1 = 16

Length(CCC) = 18-1+1 = 18

(b) Using the formula: LOC(AAA[k]) = Base(AAA)+w(K-LB) LOC(AAA[15])= 300+4(15-5) = 340

LOC(AAA[35])= 300+4(35-5) = 420

AAA[55] is not an element of AAA since 55 exceeds UB = 50.

Q32.Consider the following stack of characters, where STACK is allocated N = 8 mmory cells

STACK : A,C,D,F,K,_,_,_. (_ means empty allocated cell)

Describe the stack as the following operations takes place:

(a) POP(STACK, ITEM)
(b) POP(STACK, ITEM)
(c) POP(STACK, ITEM)
(d) PUSH(STACK, R)
(e) PUSH(STACK,L)
(f) PUSH(STACK, S)
(g) PUSH(STACK,P)
(h) POP(STACK, ITEM)

Ans: (a) STACK: A,C,D,F,_,_,_,_

(b) STACK: A,C,D,_,_,_,_,_

(c) STACK: A,C,_,_,_,_,_,_

(d) STACK: A,C,R,_,_,_,_,_

(e) STACK:A,C,R,L,_,_,_,_

(f) STACK:A,C,R,L,S, _ ,_,_

(g) STACK: A,C,R,L,S,P,_,_

(h) STACK: A,C,R,L,S,_,_,_

Q33. Consider the problem in Q-32. (a) When will overflow occur? (b) When will C be deleted before R?

Ans : (a) Since the stack has been allocated 8 memory cells, overflow will occur when STACK contains 8 elements and there is a PUSH operation to add an element

(b) Since STACK is implemented as a STACK, C will never be deleted before R.

Q34.Translate infix expression into its equivalent post fix expression: (A-B)*(D/E)
Ans: (A-B)*(D/E) = [AB-]*[DE/] = AB-DE/*

Q35. Translate infix expression into its equivalent post fix expression: (A+B^D)/(E-F)+G
Ans : (A+B^D)/(E-F)+G = (A+BD^])/[EF-]+G = [ABD^+]/[EF-]+G
= [ABD^+EF-/]+G = ABD^+EF-/G

Q36.Translate infix expression into its equivalent post fix expression:
A*(B+D)/E-F*(G+H/K)
Ans: A*(B+D)/E-F*(G+H/K) = A*[BD+]/E-F*(G+[HK/]) = [ABD+*]E-F*[GHK/+] = [ABD+*E/]-[FGHK/+*] = ABD+*E/FGHK/+*-

Q37.Consider the following arithmetic expression P, written in postfix notation :
P: 12 , 7 , 3 , - , / , 2 , 1 , 5 , + , * , +
Translate P into infix expression.
Ans : P: 12,[7-3],/,2,1,5,+,*,+ =
[12/(7-3)],2,1,5,+,*,+ = [12/(7-
3)],2,[1+5],*,+ = [12/(7-
3)],[2*(1+5)],+ =12/(7-
3)+2*(1+5)

Q38.Write an algorithm to traverse a linked list.
Answer : 1. Set PTR : = START
2. Repeat steps 3 and 4 while PTR is not equal to NULL.
3. Apply PROCESS to INFO [PTR]
4. Set PTR: = LINK[PTR]
5. Exit

Q39. Evaluate
P: 12 , 7 , 3 , - , / , 2 , 1 , 5 , + , * , + ,)

Ans :

Symbol	STACK
12	12
7	12,7
3	12,7,3
-	12,4
/	3
2	3,2
1	3,2,1
5	3,2,1,5
+	3,2,6

*	3,12
+	15
)	15

Q40. List out few of the Application of tree data structure?

Ans: The list is as follows:

- The manipulation of Arithmetic expression,
- Symbol Table construction,
- Syntax analysis.

Q41.What are priority queues?

Ans: A priority queue is a collection of elements such that each element has been assigned a priority

Q42. What is the condition to be checked for the multiplication of two matrices?

Ans :If matrices are to be multiplied, the number of columns of first matrix should be equal to the number of rows of second matrix.

Q43. Write the syntax for multiplication of matrices?

Ans :

```
for (=0; < value;++)
{
for (=0; < value;++)
{
for (=0; < value;++)
{
arr[var1][var2] += arr[var1][var3] * arr[var3][arr2];
}
}
}
```

Q44.What is a string?

Ans:A sequential array of characters is called a string.

Q45. What is use terminating null character?

Ans :Null character is used to check the end of string.

Q46. What is an empty string?

Ans :A string with zero character is called an empty string.

Q47. What are the operations that can be performed on a string?

Ans:The following are the operations that can be performed on a string: finding the length of string, copying string, string comparison, string concatenation, finding substring etc.

Q48. What is Brute Force algorithm?
Ans :Algorithm used to search the contents by comparing each element of array is called Brute Force algorithm.

Q49.What are the limitations of arrays?
Ans :The following are the limitations of arrays:
Arrays are of fixed size.
Data elements are stored in continuous memory locations which may not be available always.
Adding and removing of elements is problematic because of shifting the locations.

Q50. How can you overcome the limitations of arrays?
Ans :Limitations of arrays can be solved by using the linked list.

Q51. What is a linked list?
Ans :Linked list is a data structure which store same kind of data elements but not in continuous memory locations and size is not fixed. The linked lists are related logically.

Q52. What is the difference between an array and a linked list?
Ans :The size of an array is fixed whereas size of linked list is variable.
In array the data elements are stored in continuous memory locations but in linked list it is non continuous memory locations.
Addition, removal of data is easy in linked list whereas in arrays it is complicated.

Q53. What is a node?
Ans :The data element of a linked list is called a node.

Q54.What does node consist of?
Ans :Node consists of two fields:
data field to store the element and link field to store the address of the next node.

Q55.Write the syntax of node creation?
Answer : Syntax:

```
struct node
{
data type ;
struct node *ptr; //pointer for link node
}
temp;
```

Q56.Write the syntax for pointing to next node?

Ans : Syntax:

node->link=node1;

Q57.What is sorting?

Ans : Sorting is the process of arranging elements in some logical order.Sorting methods are classified into the following categories: External sorting: This deals with sorting of data stored in external files. This method is used when the volume of data is very large and cannot be held in a computer's RAM . Internal sorting: This deals with sorting of data held in the RAM of a computer.

Q58.List some popular sorting methods.

Ans: Popular sorting methods:

- Bubble sort
- Bucket sort
- Insertion sort
- Merge sort
- Quick sort
- Selection sort
- Shell sort

Q59. Explain bubble sort.

Ans : It requires n-1 passes to sort an array.

- In each pass every element a[i] is compared with a[i+1], for i=0 to (n-k-1), where k is the pass number and if they are out of order i.e. if a[i]>a[i+1], they are swapped.
- This will cause the largest element to move up or bubble up.
- Thus after the end of the first pass the largest element in the array will be placed in the last or nth position and on the next pass, the next largest element will be placed at position (n-1). This continues for each successive pass until the last or (n-1)th pass when the second smallest element will be placed at the second position.

Pass1.

Step 1. if a[0]>a[1] then swap a[0] and a[1].

Step 2. if a[1]>a[2] then swap a[1] and a[2].

…

Step n-1. if a[n-2]>a[n-1] then swap a[n-2] and a[n-1]. Pass2.

Step 1. if a[0]>a[1] then swap a[0] and a[1].

Step 2. if a[1]>a[2] then swap a[1] and a[2].

…

Step n-2. if a[n-3]>a[n-2] then swap a[n-3] and a[n-2].

…

Pass k.

Step 1. if a[0]>a[1] then swap a[0] and a[1].
Step 2. if a[1]>a[2] then swap a[1] and a[2].
…
Step n-k. if a[n-(k+1)]>a[n-k] then swap a[n-(k+1)] and a[n-k].
…
Pass n-1
Step 1. if a[0]>a[1] then swap a[0] and a[1].

Q60. What is the complexity of bubble sort?
Ans : First pass requires n-1 comparison

- Second pass requires n-2 comparison
- kth pass requires n-k comparisons
- Last pass requires only one comparison

Therefore the total comparisons are:

$$(n-1)+(n-2)+\ldots(n-k)+\ldots 3+2+1$$
$$=\frac{n(n-1)}{2}$$
$$=O(n^2)$$

Q61. What is insertion sort?
Ans : This algorithm is very popular with bridge players when they sort their cards. In this procedure, we pick up a particular value and then insert it at the appropriate place in the sorted sub list.

Q62. Explain the procedure for insertion sort.
Ans: This algorithm requires n-1 passes.
Pass1: a[1] is inserted either before or after a[0] so that a[0] and a[1] are sorted.
Pass2: a[2] is inserted either before a[0] or between a[0] and a[1] or after a[1] so that the elements a[0], a[1], a[2] are sorted.
Pass3: a[3] is inserted either before a[0] or between a[0] and a[1] or between a[1] and a[2] or after a[2] so that the elements a[0], a[1], a[2], a[3] are sorted.
Pass k: a[k] is inserted in its proper place in the sorted sub array a[0], a[1], a[2],…a[k-1] so that the elements a[0], a[1], a[2],…a[k-1],a[k] are sorted.
Pass n-1:a[n-1] is inserted in its proper place in the sorted sub array a[0], a[1], a[2],…a[n-2] so that the elements a[0], a[1], a[2],…a[n-1] ar e sorted.

Q63. Sort 20,35,40,100,3,10,15 using insertion sort.

Ans:

Given array: 20 35 40 100 3 10 15

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
20	35	40	100	3	10	15

Pass 1:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
35	20	40	100	3	10	15

Since a[1] <a[0], insert element a[1] before a[0]

Pass 2

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
20	35	40	100	3	10	15

Since a[2]> aa[1], no action is taken

Pass3

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
20	35	40	100	3	10	15

Since a[3]>a[2], no action is performed

Pass 4

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
20	35	40	100	3	10	15

Since [4] is less than a[3],a[2],aa[1] as well as a[0] therefore insert a[4] before a[0]

Pass 5

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	20	35	40	100	10	15

Since [5] is less than a[4],a[3],a[2],a[1] therefore insert a[5] before a[1]

Pass 6

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	10	20	35	40	100	15

Since [6] is less than a[5], a[4],a[3],a[2] therefore insert a[5] before a[2]

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	10	15	20	35	40	100

Q64.What is the complexity of insertion sort?

Ans : The worst-case performance occurs when the elements of the input array are in descending order. In the worst-case, the first pass will require one comparison, the second

pass will require 2 comparisons, and so on until the last pass which will require (n-1) comparisons. In general, the kth pass will require k-1 comparisons.

Therefore the total number of comparisons is:

$F(n)=1+2+3+\ldots..+(n-k)+\ldots.+(n-3)+(n-2)+(n-1)$

$=n(n-1)/2$

$=O(n^2)$

Q65. What is bucket sort?

Ans : Most people use the bucket sort method when sorting a list of names in alphabetical order. The procedure is:

- First, the names are grouped according to the first letter, thus the names are arranged in 26 classes, one for each letter of the alphabet. The first class consists of those names that begin with letter A, the second class consists of those names that begin with letter B, and so on.
- Next, the names are grouped according to the second letter. After this step, the list of names will be sorted on the first two letters.
- This process is continued for a number of times equal to the length of the name with maximum letters.

Since there are 26 letters in the alphabet, we make use of 26 buckets, one for each letter of the alphabet. After grouping these names according to their specific letter, we collect them according to the order of the buckets. This new list becomes the input for the next pass when we consider the next letter.

Q66. Write the algorithm for bucket sort.

Ans: Bucket sort (a, n)

Here a is a linear array of integers with n elements, the variable digit count is used to store the number of digits in the largest number in order to control the number of passes to be performed.

```
Begin
   find the largest number of the array
   set digit count=no. of digits of the largest
   no. for pass=1 to digit count by 1 do
      initialize buckets
      for i=1 to n-1 by 1 do
         set digit=obtain digit no. pass of
         a[i] put a[i] in bucket no. digit
         increment bucket count for bucket no.
      digit endfor
      collect all the numbers from buckets in
   order endfor
end
```

Q67. Give the complexity of bucket sort.

Ans : Suppose that the number of digits in the largest number of the given array is s and the number of passes to be performed is n. Then, the number of comparisons, f(n), needed to sort the given array are:
f(n)=n*s*10, where 10 is the decimal number base.
Though s is independent of n, but if s=n, then in worst case,
$f(n)=O(n^2)$. But on the other hand, if $s=\log_{10} n$, then $f(n)=O(n \log_{10} n)$.
From the above discussion, we conclude that bucket sort performs well only when the number of digits in the elements are very small.

Q68. What is merge sort?
Ans: Merging means combining elements of two arrays to form a new array. The simplest way of merging two arrays is to first copy all the elements of one array into a new array and then append all the elements of the second array to the new array. If you want the resultant array to be sorted, you can sort it by any of the sorting techniques. If the arrays are originally in sorted order, they can be merged in such a way as to ensure that the combined array is also sorted. This technique is known as merge sort.

Q69. Explain the procedure of merge sort.
Ans : The technique of merge sort (i.e., sorting during merging) is much more efficient than sorting after merging for arrays that are already sorted.
STEP 1: Let us consider two arrays say, A[7] and B[5] to be merged to form a new array. The new array say C will be having 7+5=12 elements.
STEP 2: Compare A[0] and B[0]; if A[0]<B[0] (say); move A[0] to C[0]. Increment the pointers of array A and array C (the pointer of that array is incremented whose element is moved in the third array).
STEP 3: Now compare the elements of A and B where the pointers are pointing. That is, compare A[1] and B[0].
Suppose that we find B[0]<A[1], so we move B[0] to C[1] and increment B's pointer to point to the next element in array B.

Q70. What is quicksort algorithm?
Ans : Quick sort is a divide-and-conquer sorting algorithm. To understand quick-sort, let us look at a high-level description of the algorithm.

1. Divide: If the sequence S has 2 or more elements, select an element x from S to be your pivot. Any arbitrary element, like the last, will do. Remove all the elements of S and divide them into 3 sequences:
    a. L, which holds S's elements less than x
    b. E, which holds S's elements equal to x
    c. G, which holds S's elements greater than x
2. Recurse: Recursively sort L and G
3. Conquer: Finally, to put elements back into S in order, first insert the elements of L, then those of E, and then those of G.

Q71. What is the complexity of quicksort algorithm?

Ans : Finding the location of the element that splits the array into two sections is an O(n) operation, because every element in the array is compared to the pivot element. After the division, each section is examined separately.

If the array is split approximately in half (an optimistic assumption), then there will be log2n splits.

Therefore, the total comparisons will be: $f(n) = n*\log_2 n = O(n\log_2 n)$

Q72.What is selection sort?

Ans: Selection sort requires (n-1) passes to sort an array.

In the first pass, we find the smallest element from a[0], a[1], a[2], … a[n-1] and swap it with the first element, i.e. a[0]. In the second pass, we find the smallest element from a[1], a[2], a[3]….a[n-1] and swap it with a[1] and so on.

Q73. Write the process of selectionsort.

Ans : Pass1.

1. Find the location loc of the smallest element in the entire array, i.e. a[0],[1],a[2]…a[n-1].
2. Interchange a[0] & a[loc]. Then, a[0] is trivially sorted.

Pass2.

1. Find the location loc of the smallest element in the entire array, i.e. a[1],a[2]…a[n-1].
2. Interchange a[1] & a[loc]. Then a[0], a[1] are sorted.

Pass k.

1. Find the location loc of the smallest element in the entire array, i.e. a[k],a[k+1],a[k+2]…a[n-1].
2. Interchange a[k] & a[loc]. Then a[0],a[1],a[2],…a[k] are sorted.

Pass n-1.

1. Find the location loc of the smaller of the elements a[n-2],a[n-1].
2. Interchange a[n-2] & a[loc]. Then elements a[0],a[1],a[2]….a[n-1] are sorted.
3. sorted.

Q74.Sort: 20,35,40,100,3,10,15

Ans :

Given array: 20 35 40 100 3 10 15

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
20	35	40	100	3	10	15

Pass 1:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
20	35	40	100	3	10	15

Loc=4

Interchange elements a[0] & a[4] i.e. 20 and 3

Pass 2

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	35	40	100	20	10	15

Loc=5

Interchange elements a[1] & a[5] i.e. 35 and 10

Pass3

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	10	40	100	20	35	15

Loc=6

Interchange elements a[2] & a[6] i.e. 40 and 15

Pass 4

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	10	15	100	20	35	15

Loc=4

Interchange elements a[3] & a[4] i.e. 100 and 20

Pass 5

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	10	15	20	100	35	40

Loc=5

Interchange elements a[4] & a[5] i.e. 100 and 35

Pass 6

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	10	15	20	35	100	40

Loc=6

Interchange elements a[5] & a[6] i.e. 100 and 40

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	10	15	20	35	40	100

Q75. What is the complexity of insertion sort?

- Ans: The first pass requires n-1 comparisons to find the location of the smallest element.
- The second pass requires n-2 comparisons.
- The kth pass requires n-k comparisons.
- The last pass requires only one comparison.

Therefore, the total number of comparisons are:
F(n)=(n-1)+(n-2)+……+(n-k)+…3+2+1
=n(n-1)/2
Thus, the complexity is $O(n_2)$

Q76. Write a Sub Algorithm to Find the Smallest Element in the Array. Ans: Smallest element(a, n, k, loc)

Here a is linear array of size n. This sub algorithm finds the location loc of smallest element among a[k-1],a[k+1],a[k+2]…a[n-1]. A temporary variable "small" is used to hold the current smallest element and j is used as the loop control variable.

```
Begin
   set small=a[k-
   1] set loc=k-1
     for j=k to (n-1) by 1
    do if(a[j]<small) then
        set small = a[j]
         set
      loc=j endif
   endfor
end
```

Q77.Write Selection Sort Algorithm
Ans: Selectionsort(a,n)

Here a is the linear array with n elements. This algorithm sorts elements into ascending order. It uses a temporary variable temp to facilitate the exchange of two values and variable i is used as a loop control variable

```
Begin
   for i=1 to (n-1) by 1 do
      call Smallestelement(a,n,I,loc)
      set temp=a[i-1]
      set a[i-1]=a[loc]
      set a[loc]=temp
   endfor
end
```

Q78.How do you create an empty linked list?

Ans: This variable is used to point to the first element of the list

Since the list will be empty in the beginning, the variable head is assigned a sentinel value to indicate that the list is empty.

```
void createemptylist(node **head)
{
   *head=NULL;
}
```

Q79. Name some operations on Linked Lists.

The main operations that are performed on a linked list are the following:

- Traversing and searching
- Inserting
- Deleting

Q80.What is a heap?

Ans: A binary tree has the **heap property** *iff*

a. it is empty *or*
b. the key in the root is larger than that in either child and both subtrees have the heap property.

A heap can be used as a priority queue: the highest priority item is at the root and is trivially extracted. But if the root is deleted, we are left with two sub-trees and we must *efficiently* re-create a single tree with the heap property.

The value of the heap structure is that we can both extract the highest priority item and insert a new one in **O(log*n*)** time.

Q81. Define AVL trees.

Ans : An AVL tree is a binary search tree which has the following properties:

1. The sub-trees of every node differ in height by at most one.
2. Every sub-tree is an AVL tree.

Q82. Explain Djiksatras algorithm.

Answer: Djikstra's algorithm (named after its discover, E.W. Dijkstra) solves the problem of finding the shortest path from a point in a graph (the *source*) to a destination. It turns out that one can find the shortest paths from a given source to *all* points in a graph in the same time, hence this problem is sometimes called the **single-source shortest paths** problem.

Q83. Give mode of operation in Dijkstra's algorithm .

Ans: Dijkstra's algorithm keeps two sets of vertices:

S the set of vertices whose shortest paths from the source have already been determined *and*

V-S the remaining vertices.

The other data structures needed are:

d array of best estimates of shortest path to each vertex

pi an array of predecessors for each vertex

The basic mode of operation is:

1. Initialise **d** and **pi**,
2. Set **S** to empty,
3. While there are still vertices in **V-S**,

i. Sort the vertices in **V-S** according to the current best estimate of their distance from the source,
ii. Add **u**, the closest vertex in **V-S**, to **S**,
iii. **Relax** all the vertices still in **V-S** connected to **u**

Q84. Explain greedy algorithm.

Ans: Many algorithms can be formulated as a finite series of guesses, *eg* in the Travelling Salesman Problem, we try (guess) each possible tour in turn and determine its cost. When we have tried them **all**, we know which one is the optimum (least cost) one. However, we must try them all before we can be certain that we know which is the optimum one, leading to an ***O(n!)*** algorithm.

Intuitive strategies, such as building up the salesman's tour by adding the city which is closest to the current city, can readily be shown to produce sub-optimal tours. As another example, an experienced chess player will not take an opponent's pawn with his queen - because that move produced the maximal gain, the capture of a piece - if his opponent is guarding that pawn with another pawn. In such games, you must look at *all* the moves ahead to ensure that the one you choose is in fact the optimal one. All chess players know that short-sighted strategies are good recipes for disaster!

There is a class of algorithms, the *greedy algorithms*, in which we can find a solution by using only knowledge available at the time the next choice (or guess) must be made. The problem of finding the Minimum Spanning Tree is a good example of this class.

Q85. What are partitions?

Ans: A partitions is a set of sets of elements of a set.

- Every element of the set belong to one of the sets in the partition.
- No element of the set belong to more than one of the sub-sets.

Q86..What are spanning trees?

Ans: A *spanning tree* of a graph, G, is a set of **|V|**-1 edges that connect all vertices of the graph.

Q87. What is a minimum spanning tree?

Ans: If a cost, $\mathbf{c_{ij}}$, is associated with each edge, $\mathbf{e_{ij} = (v_i,v_j)}$, then the minimum spanning tree is the set of edges, $\mathbf{E_{span}}$, forming a spanning tree, such that:

$$\mathbf{C = sum(\ c_{ij}} \mid \text{all } \mathbf{e_{ij}} \text{ in } \mathbf{E_{span}}\ )$$

is a minimum.

Q88.What does Kruskals's algorithm do?

Ans: This algorithm creates a *forest* of trees. Initially the forest consists of **n** single node trees (and no edges). At each step, we add one (the cheapest one) edge so that it joins two trees together. If it were to form a cycle, it would simply link two nodes that were already part of a single connected tree, so that this edge would not be needed.

Q89. List the steps to construct a forest. Ans: The steps are:

1. Construct a forest - with each node in a separate tree.
2. Place the edges in a priority queue.
3. Until we've added **n**-1 edges,
    1. Continue extracting the cheapest edge from the queue, until we find one that does not form a cycle,
    2. Add it to the forest. Adding it to the forest will join two trees together.

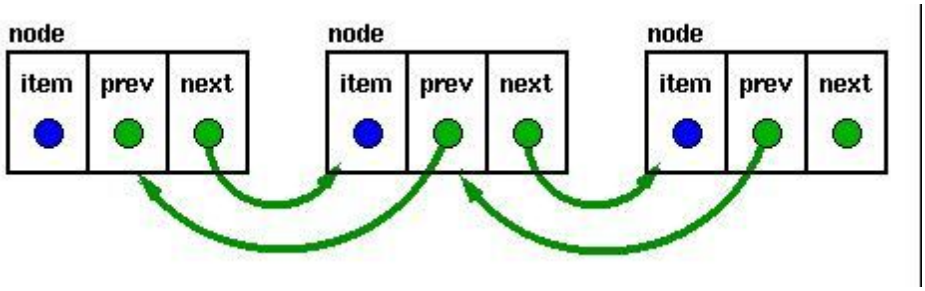
Every step joins two trees in the forest together, so that, at the end, only one tree will remain in T.

Q90. Explain circularly linked lists.

Ans: By ensuring that the tail of the list is always pointing to the head, we can build a circularly linked list. If the external pointer (the one in `struct t_node` in our implementation), points to the current "tail" of the list, then the "head" is found trivially via `tail->next`, permitting us to have either LIFO or FIFO lists with only one external pointer. In modern processors, the few bytes of memory saved in this way would probably not be regarded as significant. A circularly linked list would more likely be used in an application which required "round-robin" scheduling or processing.

Q91. What are doubly linked lists?

Ans: Doubly linked lists have a pointer to the preceding item as well as one to the next**.**



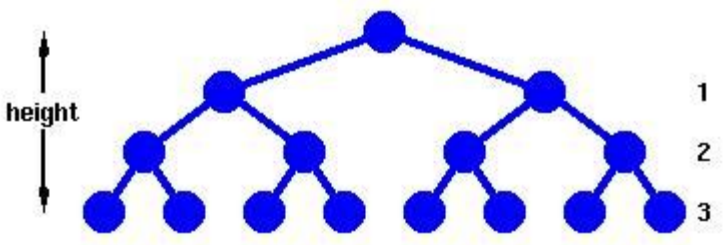
Q92. What is a binary tree?

Ans: The simplest form of tree is a **binary tree**. A binary tree consists of

a. a *node* (called the **root** node) and
b. left and right sub-trees.
   Both the sub-trees are themselves binary trees.

Q93. What are complete trees?

Ans:



This forms a *complete tree*, whose *height* is defined as the number of links from the root to the deepest leaf.

A complete tree

Q94. What is an m-way search tree?

Ans: An **m-way search tree**

a. is empty or
b. consists of a root containing **j** (1<=**j**<**m**) keys, $\mathbf{k_j}$, and a set of sub-trees, $\mathbf{T_i}$, (**i** = 0..**j**), such that
   i. if **k** is a key in $\mathbf{T_0}$, then **k** <= $\mathbf{k_1}$
   ii. if **k** is a key in $\mathbf{T_i}$ (0<**i**<**j**), then $\mathbf{k_i}$ <= **k** <= $\mathbf{k_{i+1}}$
   iii. if **k** is a key in $\mathbf{T_j}$, then **k** > $\mathbf{k_j}$ and
   iv. all $\mathbf{T_i}$ are nonempty m-way search trees or all $\mathbf{T_i}$ are empty

Q95.What is quicksort algorithm?

Ans: Quicksort is a very efficient sorting algorithm invented by C.A.R. Hoare. It has two phases:

- the partition phase and
- the sort phase.

As we will see, most of the work is done in the partition phase - it works out where to divide the work. The sort phase simply sorts the two smaller problems that are generated in the partition phase.

This makes Quicksort a good example of the **divide and conquer** strategy for solving problems. (You've already seen an example of this approach in the binary search procedure.) In quicksort, we divide the array of items to be sorted into two partitions and then call the quicksort procedure recursively to sort the two partitions, *ie* we *divide* the problem into two smaller ones and *conquer* by solving the smaller ones.

Q96. Program to allocate memory dynamically for strings, and store their addresses in array of pointers to strings.

Ans.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <string.h>

void main( )
{
        char *name[5] ;
        char str[20] ;
        int i ;

        clrscr( ) ;

        for ( i = 0 ; i < 5 ; i++ )
        {
                printf ( "Enter a String: " ) ;
```

```
                gets ( str ) ;
                name[i] = ( char * ) malloc ( strlen ( str ) + 1 ) ;
                strcpy ( name[i], str ) ;
        }

        printf ( "\nThe strings are:" ) ;

        for ( i = 0 ; i < 5 ; i++ )
                printf ( "\n%s", name[i] ) ;

        for ( i = 0 ; i < 5 ; i++ )
                free ( name[i] ) ;

        getch( ) ;
}
```

Q97. Program to add a new node to the ascending order linked list. */

Ans.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

/* structure containing a data part and link part */
struct node
{
        int data ;
        struct node *link ;
} ;

void add ( struct node **, int ) ;
void display ( struct node * ) ;
int count ( struct node * ) ;
void delete ( struct node **, int ) ;

void main( )
{
        struct node *p ;
        p = NULL ;  /* empty linked list */

        add ( &p, 5 ) ;
        add ( &p, 1 ) ;
        add ( &p, 6 ) ;
        add ( &p, 4 ) ;
        add ( &p, 7 ) ;

        clrscr( ) ;
        display ( p ) ;
        printf ( "\nNo. of elements in Linked List = %d", count ( p ) ) ;
}

/* adds node to an ascending order linked list */
void add ( struct node **q, int num )
{
        struct node *r, *temp = *q ;

        r = malloc ( sizeof ( struct node ) ) ;
        r -> data = num ;

        /* if list is empty or if new node is to be inserted before the first node */
        if ( *q == NULL || ( *q ) -> data > num )
        {
                *q = r ;
                ( *q ) -> link = temp ;
        }
        else
        {
                /* traverse the entire linked list to search the position to insert the
                   new node */
                while ( temp != NULL )
```

```
                {
                        if ( temp -> data <= num && ( temp -> link -> data > num ||
                                                                        temp -> link ==
NULL ))
                        {
                                r -> link = temp -> link ;
                                temp -> link = r ;
                                return ;
                        }
                        temp = temp -> link ;  /* go to the next node */
                }
        }
}

/* displays the contents of the linked list */
void display ( struct node *q )
{
        printf ( "\n" ) ;

        /* traverse the entire linked list */
        while ( q != NULL )
        {
                printf ( "%d ", q -> data ) ;
                q = q -> link ;
        }
}

/* counts the number of nodes present in the linked list */
int count ( struct node *q )
{
        int c = 0 ;

        /* traverse the entire linked list */
        while ( q != NULL )
        {
                q = q -> link ;
                c++ ;
        }
        return c ;
}
```

Q98. Program that implements depth first search algorithm.

Ans.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

#define TRUE 1
#define FALSE 0
#define MAX 8

struct node
{
        int data ;
        struct node *next ;
} ;

int visited[MAX] ;

void dfs ( int, struct node ** ) ;
struct node * getnode_write ( int ) ;
void del ( struct node * ) ;

void main( )
{
        struct node *arr[MAX] ;
        struct node *v1, *v2, *v3, *v4 ;
   int i ;
```

```
        clrscr( ) ;

        v1 = getnode_write ( 2 ) ;
        arr[0] = v1 ;
        v1 -> next = v2 = getnode_write ( 3 ) ;
        v2 -> next = NULL ;

        v1 = getnode_write ( 1 ) ;
        arr[1] = v1 ;
        v1 -> next = v2 = getnode_write ( 4 ) ;
        v2 -> next = v3 = getnode_write ( 5 ) ;
        v3 -> next = NULL ;

        v1 = getnode_write ( 1 ) ;
        arr[2] = v1 ;
        v1 -> next = v2 = getnode_write ( 6 ) ;
        v2 -> next = v3 = getnode_write ( 7 ) ;
        v3 -> next = NULL ;

        v1 = getnode_write ( 2 ) ;
        arr[3] = v1 ;
        v1 -> next = v2 = getnode_write ( 8 ) ;
        v2 -> next = NULL ;

        v1 = getnode_write ( 2 ) ;
        arr[4] = v1 ;
        v1 -> next = v2 = getnode_write ( 8 ) ;
        v2 -> next = NULL ;

        v1 = getnode_write ( 3 ) ;
        arr[5] = v1 ;
        v1 -> next = v2 = getnode_write ( 8 ) ;
        v2 -> next = NULL ;

        v1 = getnode_write ( 3 ) ;
        arr[6] = v1 ;
        v1 -> next = v2 = getnode_write ( 8 ) ;
        v2 -> next = NULL ;

        v1 = getnode_write ( 4 ) ;
        arr[7] = v1 ;
        v1 -> next = v2 = getnode_write ( 5 ) ;
        v2 -> next = v3 = getnode_write ( 6 ) ;
        v3 -> next = v4 = getnode_write ( 7 ) ;
        v4 -> next = NULL ;

        dfs ( 1, arr ) ;

    for ( i = 0 ; i < MAX ; i++ )
            del ( arr[i] ) ;

    getch( ) ;
}

void dfs ( int v, struct node **p )
{
        struct node *q ;
        visited[v - 1] = TRUE ;

        printf ( "%d\t", v ) ;

        q = * ( p + v - 1 ) ;

        while ( q != NULL )
        {
                if ( visited[q -> data - 1] == FALSE )
                        dfs ( q -> data, p ) ;
                else
                        q = q -> next ;
        }
}
```

```
struct node * getnode_write ( int val )
{
   struct node *newnode ;
         newnode = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
         newnode -> data = val ;
         return newnode ;
}

void del ( struct node *n )
{
         struct node *temp ;

   while ( n != NULL )
   {
      temp = n -> next ;
      free ( n ) ;
      n = temp ;
   }
}
```

Q99. Program to find the minimum cost of a spanning tree.

Ans.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

struct lledge
{
         int v1, v2 ;
         float cost ;
         struct lledge *next ;
} ;

int stree[5] ;
int count[5] ;
int mincost ;

struct lledge * kminstree ( struct lledge *, int ) ;
int getrval ( int ) ;
void combine ( int, int ) ;
void del ( struct lledge * ) ;

void main( )
{
         struct lledge *temp, *root ;
         int i ;

         clrscr( ) ;

         root = ( struct lledge * ) malloc ( sizeof ( struct lledge ) ) ;

         root -> v1 = 4 ;
         root -> v2 = 3 ;
         root -> cost = 1 ;
         temp = root -> next = ( struct lledge * ) malloc ( sizeof ( struct lledge ) ) ;

         temp -> v1 = 4 ;
         temp -> v2 = 2 ;
         temp -> cost = 2 ;
         temp -> next = ( struct lledge * ) malloc ( sizeof ( struct lledge ) ) ;

         temp = temp -> next ;
         temp -> v1 = 3 ;
         temp -> v2 = 2 ;
         temp -> cost = 3 ;
         temp -> next = ( struct lledge * ) malloc ( sizeof ( struct lledge ) ) ;

         temp = temp -> next ;
         temp -> v1 = 4 ;
```

```
        temp -> v2 = 1 ;
        temp -> cost = 4 ;
        temp -> next = NULL ;

        root = kminstree ( root, 5 ) ;

        for ( i = 1 ; i <= 4 ; i++ )
                printf ( "\nstree[%d] -> %d", i, stree[i] ) ;
        printf ( "\nThe minimum cost of spanning tree is %d", mincost ) ;
        del ( root ) ;

        getch( ) ;
}
struct lledge * kminstree ( struct lledge *root, int n )
{
        struct lledge *temp = NULL ;
        struct lledge *p, *q ;
        int noofedges = 0 ;
        int i, p1, p2 ;

        for ( i = 0 ; i < n ; i++ )
                stree[i] = i ;
        for ( i = 0 ; i < n ; i++ )
                count[i] = 0 ;

        while ( ( noofedges < ( n - 1 ) ) && ( root != NULL ) )
        {
                p = root ;

                root = root -> next ;

                p1 = getrval ( p -> v1 ) ;
                p2 = getrval ( p -> v2 ) ;

                if ( p1 != p2 )
                {
                        combine ( p -> v1, p -> v2 ) ;
                        noofedges++ ;
                        mincost += p -> cost ;
                        if ( temp == NULL )
                        {
                                temp = p ;
                                q = temp ;
                        }
                        else
                        {
                                q -> next = p ;
                                q = q -> next ;
                        }
                        q -> next = NULL ;
                }
        }
        return temp ;
}

int getrval ( int i )
{
        int j, k, temp ;
        k = i ;
        while ( stree[k] != k )
                k = stree[k] ;
        j = i ;
        while ( j != k )
        {
                temp = stree[j] ;
                stree[j] = k ;
                j = temp ;
        }
        return k ;
}

void combine ( int i, int j )
```

```
{
        if ( count[i] < count[j] )
                stree[i] = j ;
        else
        {
                stree[j] = i ;
                if ( count[i] == count[j] )
                        count[j]++ ;
        }
}

void del ( struct lledge *root )
{
        struct lledge *temp ;

        while ( root != NULL )
        {
                temp = root -> next ;
                free ( root ) ;
                root = temp ;
        }
}
```

Q100. Program that creates random numbers in a given file.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

void main( )
{
        FILE *fp ;
        char str [ 67 ] ;
        int i, noofr, j ;

        clrscr( ) ;

        printf ( "Enter file name: " ) ;
        scanf ( "%s", str ) ;

        printf ( "Enter number of records: " ) ;
        scanf ( "%d", &noofr ) ;

        fp = fopen ( str, "wb" ) ;
        if ( fp == NULL )
        {
                printf ( "Unable to create file." ) ;
                getch( ) ;
                exit ( 0 ) ;
        }
```

```
        randomize( ) ;

        for ( i = 0 ; i < noofr ; i++ )
        {
                j = random ( 1000 ) ;
                fwrite ( &j, sizeof ( int ), 1, fp ) ;
                printf ( "%d\t", j ) ;
        }

        fclose ( fp ) ;

        printf ( "\nFile is created. \nPress any key to continue." ) ;

        getch( ) ;
}
```