18CS302/18IT302

Hall Ticket Number:

II/IV B.Tech (Regular) DEGREE EXAMINATION

November, 2019 **Common to CSE & IT**

Third Semester **Data Structures**

Time: Three Hours **Maximum:** 50 Marks

Answer Question No.1 compulsorily. (1X10 = 10 Marks)

Answer ONE question from each unit. (4X10=40 Marks)

1. Answer all questions (1X10=10 Marks)
   a) What do you mean by asymptotic notation? Name any three.
   b) Write the node structure for representing polynomials.
   c) What are the different notations to represent a mathematical expression? Which notation is most suitable for use in Computers?
   d) Give any two applications of linear queue.
   e) Differentiate binary tree and binary search tree.
   f) Construct expression tree for (-a)+b*c-d/e.
   g) Why do we need to balance the tree? Justify.
   h) What is meant by Binary heap?
   i) Write any two applications of hashing.
   j) Define disjoint set with an example.

UNIT I

2. a) Differentiate array and linked list with an example. 4M
   b) Write a program to create a singly linked list and a function to remove nth element whose location is specified? 6M

(OR)

3. a) Calculate the time complexity for the following recurrence relation
   $T(n) = 1 + T(n-1)$ if $n>1$
   $= 1$ if $n=1$ 4M
   b) Write an algorithm for the following operations in doubly linked list with neat diagrams.
   i) insert_at_specific() ii) delete_at_specific() 6M

UNIT II

4. Write an algorithm for converting infix expression to postfix expression. Consider the following expression
   (7+9)*2/4+3^5/6
   i) Convert the above infix expression to postfix. 10M
   ii) After conversion, Evaluate the postfix expression.

(OR)

5. a) Write a C-routine for enqueue and dequeue operations of Linear queue. 5M
   b) Compare Shell sort and Radix sort with the help of suitable example. 5M

Written by **K. Suresh Kumar, Asst.Professor, Dept.of IT,BEC**

UNIT III

6. a) Define the following with suitable examples
   i) Full binary tree ii) Complete binary tree iii) Perfect binary tree 6M
   b) Draw the binary tree whose pre-order and in-order traversals are given below
   preorder : A,B,D,G,C,E,H,I,F inorder: D,G,B,A,H,E,I,C,F 4M

(OR)

7. Define AVL tree. Start with an empty AVL search tree and insert the following keys in the given order: H, I, J, B, A, E, C, F, D, G, K, L
   a) Draw the figures depicting your tree immediately after each insertion and following rebalancing. 10M
   b) Remove the keys in order C, I, H. Draw your tree immediately after each delete.

UNIT IV

8. a) Write a program to implement Heap Sort. 4M
   b) Given input {4371, 1323, 6173, 4199, 4344, 9679, 1989} and hash function ***h(x)= x mod 10.*** Show the resulting hash table using linear probing and quadratic probing for bucket size 9? Compare their performance. [Take C1=2 and C2= 1 if necessary where C1 and C2 are arbitrary constants.] 6M

(OR)

9. a) Explain the concept of separate chaining. 4M
   b) What are the advantages of Disjoint Set data structure? Explain various operations performed on Disjoint Set data structure with an example. 6M

Written by **K. Suresh Kumar, Asst.Professor, Dept.of IT,BEC**

Scheme & Solutions

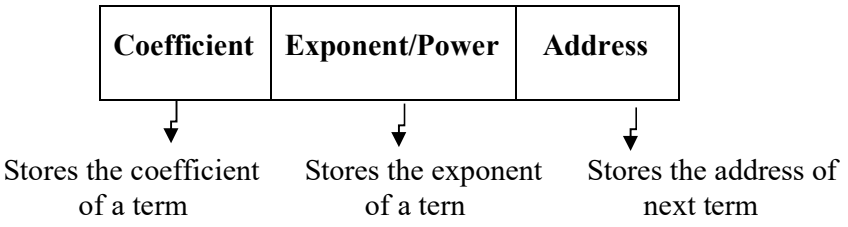
1. Answer all questions

a) What do you mean by asymptotic notation? Name any three.

Ans: Asymptotic notation of an algorithm is a mathematical representation of its complexity. Majorly, three types of asymptotic notations are used for representing complexity of an algorithm. They are 1. Big-oh (O) 2. Big-Omega (Ω) 3. Big-Theta(Θ) ----- **1M**

b) Write the node structure for representing polynomials.

Ans: The node structure for representing polynomial is as follows:



----- **1M**

c) What are the different notations to represent a mathematical expression? Which notation is most suitable for use in Computers?

Ans: There are three notations are used to represent mathematical expression in computer science. They are 1. infix notation 2. Prefix notation 3. Postfix notation

Most suitable notation used in computers is "Postfix Notation" ----- **1M**

d) Give any two applications of linear queue.

Ans: Queue is used in the following applications:

1. CPU Scheduling 2. Radix sort 3. Call centre phone system, etc. **Any two valid ----- 1M**

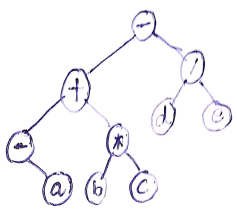
e) Differentiate binary tree and binary search tree.

Ans: Binary tree is the tree in which every node has zero or one or utmost two children. There is no condition or relationship between the values of the parent and children nodes.

Binary search tree (which also inherits the properties of a binary tree), the node with value smaller than or equal to the parent node must become the left child and the node with value greater than the parent node must become the right child. ----- **1M**

f) Construct expression tree for (-a)+b*c-d/e.

Ans: Expression tree for the given expression is as follows:



---- 1M

g) Why do we need to balance the tree? Justify.

Ans: In Binary search tree (BST), the time complexity for search operation in worst case (i.e., in skewed trees) is O(n). To overcome this, BST needs to be height balanced. In Height Balanced trees (AVL trees), time complexity for search operation is O(logn). ----- **1M**

h) What is meant by Binary heap?

Ans: A **binary heap** is a complete **binary** tree which satisfies the **heap** ordering property. The ordering can be one of two types:

The min-**heap** property: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.

The *max-heap property*: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root. ----- **1M**

i) Write any two applications of hashing.

Ans: The following are the various applications of hashing

1. Data base system 2. Data dictionaries 3. Tagged buffers. Any two valid ---- **1M**

j) Define disjoint set with an example.

Ans: A **disjoint-set data structure** is a **data structure** that keeps track of a **set** of elements partitioned into a number of **disjoint** (non-overlapping) subsets. This can be used for determining if two elements are in the same subset. E.g: if A={1,2,3} and B={4,5,6} are disjoint sets. ----- **1M**

UNIT-I

2. a) Differentiate array and linked list with an example. [4M]

S.no.	Array	Linked List
1.	Insertions and deletions are difficult.	Insertions and deletions can be done easily.
2.	It needs movements of elements for insertion and deletion.	It does not need movement of nodes for insertion and deletion.
3.	In it space is wasted.	In it space is not wasted.
4.	It is more expensive.	It is less expensive.
5.	It requires less space as only information is stored.	It requires more space as pointers are also stored along with information.
6.	Its size is fixed.	Its size is not fixed.
7.	It cannot be extended or reduced according to requirements.	It can be extended or reduced according to requirements.
8.	Same amount of time is required to access each element.	Different amount of time is required to access each element.

Any valid 4 differences ----- 4M

2. b) Write a program to create a singly linked list and a function to remove nth element whose location is specified? [Structure definition—1M, Creation---- 3M Nth node---- 2M] [6M]

Ans:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int num;
   struct node *next;
};
void create(struct node **);
void nthnode(struct node *, int);
void release(struct node **);
int main()
{
   struct node *p = NULL;
   int n;
   printf("Enter data into the list\n");
   create(&p);
   printf("Enter the value n to find nth position from the last node: ");
   scanf("%d", &n);
   nthnode(p, n);
   release (&p);
   return 0;
}
```

------ **1M**

```
void nthnode(struct node *head, int n)
{
   struct node *p, *q;
   int i;

   q = p = head;

   for (i = 0; i < n && q != NULL; i++)
   {
      q = q->next;
   }
   if (i < n)
   {
printf("Entered n = %d is larger than the number of elements = %d in list. Please try again.\n", n, i);
   }
   else
   {
      while (q->next != NULL)
      {
         q = q->next;
```

```
            p = p->next;
        }
        printf("%d is %d nodes from the last node.\n", p->num, n);
    }
}
```

------- **2M**

```
void create(struct node **head)
{
    int c, ch;
    struct node *temp, *rear;

    do
    {
        printf("Enter number: ");
        scanf("%d", &c);
        temp = (struct node *)malloc(sizeof(struct node));
        temp->num = c;
        temp->next = NULL;
        if (*head == NULL)
        {
            *head = temp;
        }
        else
        {
            rear->next = temp;
        }
        rear = temp;
        printf("Do you wish to continue [1/0]: ");
        scanf("%d", &ch);
    } while (ch != 0);
    printf("\n");
}
```

------- **3M**

```
void release(struct node **head)
{
    struct node *temp = *head;
    *head = (*head)->next;
    while ((*head) != NULL)
    {
        free(temp);
        temp = *head;
        (*head) = (*head)->next;
    }
}
```

(OR)

3. a) Calculate the time complexity for the following recurrence relation
T(n) = 1 + T(n-1) if n>1

= 1 if n=1 **[4M]**

Ans: Given T(n) = 1+ T(n-1) if n>1
= 1 if n=1

Consider T(n) = 1+ T(n-1) ---------- (eq.1)

Back substitution:

Substitute n = n-1 in eq.1 then

T(n-1) = 1 + T(n-2) ---------- (eq.2)

Similarly substitute n = n-2 in eq.1 then

T(n-2) = 1 + T(n-3) ---------- (eq.3) -------- **2M**

Now, substitute eq.2 in eq.1 then eq.1 becomes

T(n) = 1 + T(n-1)
= 1 + (1 + T(n-2)) (:. From eq.2)
= 2 + T(n-2)
= 2 + (1 + T(n-3)) (:. From eq.3)
= 3 + T(n-3)

Similarly = 3 + (1 + T(n-4))
= 4 + T(n-4)
.
.

T(n) = k + T(n-k) -------- (eq.4) this function will end only when n-k=1 then k=n-1.

Now, Substitute k = n-1 in (eq.4) then eq.4 becomes

T(n) = n-1 + T(n-(n-1))
T(n) = n-1 + T(1)
T(n) = n - 1 + 1 (:. From given function T(1) = 1)
T(n) = n

The time complexity for given function in worst case is ***O(n)*** ------- **2M**

3. b) Write an algorithm for the following operations in doubly linked list with neat diagrams.

i) insert_at_specific() ii) delete_at_specific() **[6M]**

Ans: i) inserting at Specific location in the list :

The following steps to insert a new node after a specific node in the double linked list...

Step 1: Create a newNode with given value.

Step 2: Check whether list is Empty (head == NULL)

Step 3: If it is Empty then, assign NULL to newNode → previous & newNode → next and newNode to head.

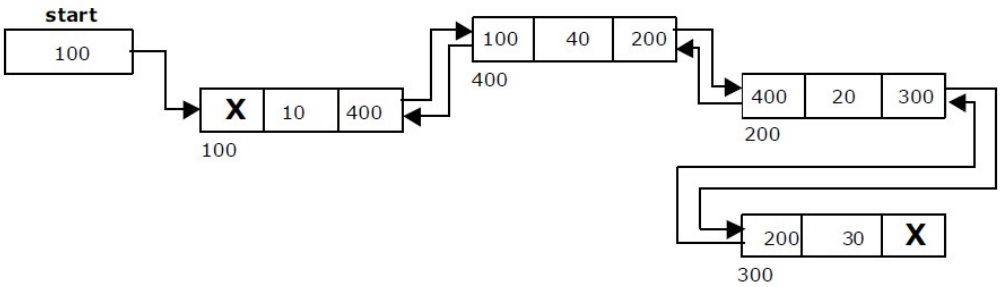
Step 4: If it is not Empty then, define two node pointers temp1 & temp2 and initialize temp1 with head.

Step 5: Keep moving the temp1 to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).

Step 6: Every time check whether temp1 is reached to the last node. If it is reached to the last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp1 to next node.

Step 7: Assign temp1 → next to temp2, newNode to temp1 → next, temp1 to newNode → previous, temp2 to newNode → next and newNode to temp2 → previous.

Example: Insert a node at intermediate position in double linked list is shown in below



insertion method -----→ 3Marks

ii) Deleting a Specific Node from the list:

The following steps to delete a specific node from the double linked list...

Step 1: Check whether list is Empty (head == NULL)

Step 2: If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3: If it is not Empty, then define a Node pointer 'temp' and initialize with head.

Step 4: Keep moving the temp until it reaches to the exact node to be deleted or to the last node.

Step 5: If it is reached to the last node, then display 'Given node not found in the list! Deletion not possible!!!' and terminate the fuction.

Step 6: If it is reached to the exact node which we want to delete, then check whether list is having only one node or not

Step 7: If list has only one node and that is the node which is to be deleted then set head to NULL and delete temp (free(temp)).

Step 8: If list contains multiple nodes, then check whether temp is the first node in the list (temp == head).

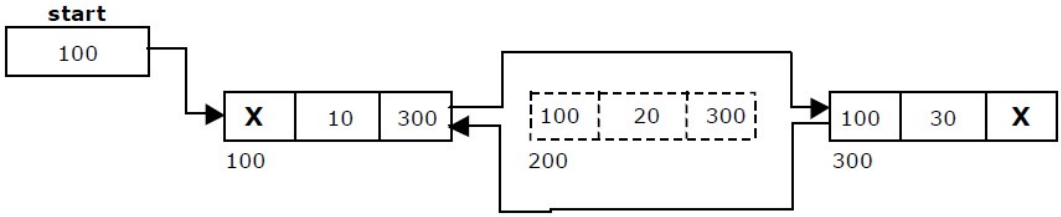
Step 9: If temp is the first node, then move the head to the next node (head = head → next), set head of previous to NULL (head → previous = NULL) and delete temp.

Step 10: If temp is not the first node, then check whether it is the last node in the list (temp → next == NULL).

Step 11: If temp is the last node then set temp of previous of next to NULL (temp → previous → next = NULL) and delete temp (free(temp)).

Step 12: If temp is not the first node and not the last node, then set temp of previous of next to temp of next (temp → previous → next = temp → next), temp of next of previous to temp of previous (temp → next → previous = temp → previous) and delete temp (free(temp)).

Example: Deleting a node at specified position in double linked list is shown in below



Any one deletion method -----→ 3Marks

UNIT II

4. a) Write an algorithm for converting infix expression to postfix expression. Consider the following expression

(7+9)*2/4+3^5/6

i) Convert the above infix expression to postfix.

ii) After conversion, Evaluate the postfix expression. [10M]

Ans: Procedure to convert from infix expression to postfix expression is as follows:

1. Scan the infix expression from left to right.
2. a) If the scanned symbol is left parenthesis, push it onto the stack.

   b) If the scanned symbol is an operand, then place directly in the postfix expression (output).

   c) If the symbol scanned is a right parenthesis, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parenthesis.

   d) If the scanned symbol is an operator, then go on removing all the operators from the stack and place them in the postfix expression, if and only if the precedence of the operator which is on the top of the stack is greater than (or greater than or equal) to the precedence of the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack. ------ **3M**

i) Postfix form for given infix expression is 79+2*4/35^6/+ ------ 4M

ii) Evaluation of post fix expression is 48.5 ------ 3M

5. a) Write a C-routine for enqueue and dequeue operations of Linear queue. **[5M]**

```
#include <stdio.h>
 #define MAX 50
 int queue_array[MAX];
int rear = - 1;
int front = - 1;                                          Declaration ------ 1M
void enqueue()
{
   int add_item;
   if (rear == MAX - 1)
   printf("Queue Overflow \n");
   else
   {
      if (front == - 1)
      /*If queue is initially empty */
      front = 0;
      printf("Inset the element in queue : ");
      scanf("%d", &add_item);
      rear = rear + 1;
      queue_array[rear] = add_item;
   }
} /* End of enqueue() */                                  -------- 2M
 void dequeue()
{
   if (front == - 1 || front > rear)
   {
      printf("Queue Underflow \n");
      return ;
   }
   else
   {
      printf("Element deleted from queue is : %d\n", queue_array[front]);
      front = front + 1;
   }
} /* End of dequeue() */                                  ---------- 2M
```

5. b) Compare Shell sort and Radix sort with the help of suitable example. [5M]

Shell Sort	Radix Sort
1. Founded by Donald Shell in 1959	1. Founded by Harold H. Seward in 1954
2. Shell sort works by comparing elements that are distant rather than adjacent elements in an array or list where adjacent elements are compared. Shell sort also called as diminishing increment sort.	2. Radix sort is non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value.
3. It uses increment sequence for comparing elements	3. It uses Queues for grouping same significant position and value elements.
4. The worst case time complexity is $O(n^2)$	4. The worst case time complexity is $O(n^2)$
5. It is a complex algorithm. Still a less efficient than merge, heap or quick sorting techniques.	5. Radix sort is very fast compared to other sorting techniques but it uses additional space for sorting the elements.

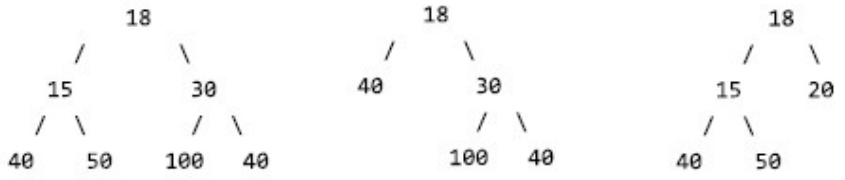
Any two valid comparisons ---- **2M**

UNIT III

6. a) Define the following with suitable examples

i) Full binary tree ii) Complete binary tree iii) Perfect binary tree [6M]

Ans: i) Full Binary tree: A binary tree is full if every node has 0 or 2 children. Following are examples of full binary tree.

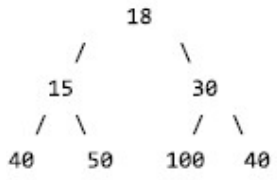


--------- **2M**

ii) Complete Binary Tree: A binary tree is complete binary tree if all levels are filled except possibly the last level and the last level has all keys as left as possible. Following are examples of complete binary tree.

---------- **2M**

iii) Perfect Binary Tree: A binary tree is perfect binary tree in which all internal nodes have two children and all leaves are at same level. Following are examples of perfect binary tree.

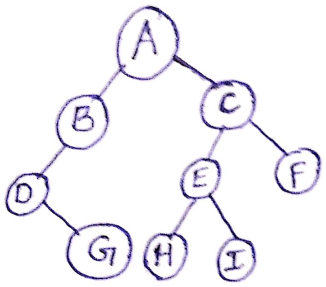


A perfect binary tree of height H (where height is number of nodes on path from root to leaf) has 2^H-1. --------- **2M**

6. b) Draw the binary tree whose pre-order and in-order traversals are given below

preorder : A,B,D,G,C,E,H,I,F inorder: D,G,B,A,H,E,I,C,F [4M]

Ans:



[Procedure --- 2M + Construction – 2M]

(OR)

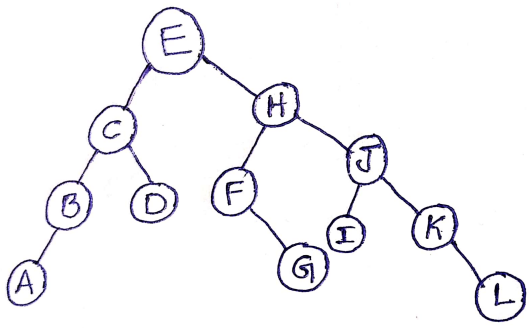
7. Define AVL tree. Start with an empty AVL search tree and insert the following keys in the given order: H, I, J, B, A, E, C, F, D, G, K, L

a) Draw the figures depicting your tree immediately after each insertion and following rebalancing.

b) Remove the keys in order C, I, H. Draw your tree immediately after each delete. [10M]

Ans: AVL Tree: AVL Tree is balanced binary search tree. Invented by G.M. Adelson-Velsky and E.M. Landison in 1962. A binary tree is said to be balanced if the difference between the heights of left and right sub trees of every node in the tree is either -1, 0, or +1. -------- **1M**

a) After inserting all the given keys, the final AVL Tree is shown in below:

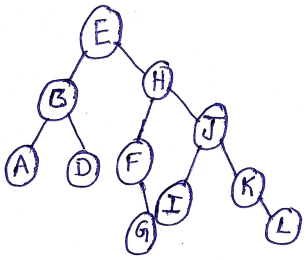


Procedure ----- **3M**

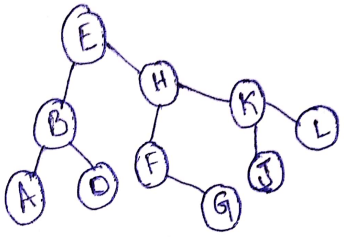
Final tree ------ **2M**

b) After deleting C from final AVL tree is as follows

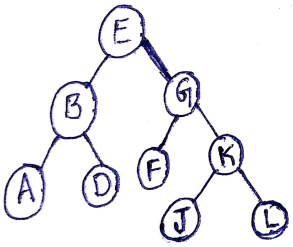
After deleting 'C'



After deleting 'I'



After deleting 'H'



1M+1M+1M=3M

UNIT-IV

8. a) Write a program to implement Heap Sort. **[4M]**

Ans:

```
void heapsort(int[], int);
void buildheap(int [], int);
void satisfyheap(int [], int, int);
void main()
{
 int a[10], i, size;
 printf("Enter size of list");    // less than 10, because max size of array is 10
 scanf("%d",&size);
 printf("Enter elements");
 for( i=0; i < size; i++)
 {
  scanf("%d",&a[i]);
 }
 heapsort(a, size);
 getch();
```

```
}

void heapsort(int a[], int length)
{

  int heapsize, i, temp;

  buildheap(a, length);

  heapsize = length - 1;
  for( i=heapsize; i >= 0; i--)
  {
    temp = a[0];
    a[0] = a[heapsize];
    a[heapsize] = temp;
    heapsize--;
    satisfyheap(a, 0, heapsize);
  }
  for( i=0; i < length; i++)
  {
    printf("\t%d",a[i]);
  }
}
void buildheap(int a[], int length)
{
  int i, heapsize;
  heapsize = length - 1;
  for( i=(length/2); i >= 0; i--)
  {
    satisfyheap(a, i, heapsize);
  }
}
```

-------- **2M**

```
void satisfyheap(int a[], int i, int heapsize)
{
 int l, r, largest, temp;
 l = 2*i;
 r = 2*i + 1;
 if(l <= heapsize && a[l] > a[i])
 {
  largest = l;
 }
 else
 {
  largest = i;
 }
 if( r <= heapsize && a[r] > a[largest])
 {
  largest = r;
 }
 if(largest != i)
 {
  temp = a[i];
  a[i] = a[largest];
  a[largest] = temp;
  satisfyheap(a, largest, heapsize);
 }
}
```

------- **2M**

8. b) Given input {4371, 1323, 6173, 4199, 4344, 9679, 1989} and hash function *h(x)= x mod 10.* Show the resulting hash table using linear probing and quadratic probing for bucket size 9? Compare their performance. [Take C1=2 and C2= 1 if necessary where C1 and C2 are arbitrary constants.] [6M]

Ans: Given input {4371, 1323, 6173, 4199, 4344, 9679, 1989}

Hash table size=9 i.e., m=9

H(x)= x mod 10

In Linear probing, Hash function is as follows

H(x)= (H^1(x) + i) mod m where m denotes hash table size or bucket size and i=0,1,2,…

Insert 4371 into hash table using linear probing

H(4371) = ((4371 mod 10) + i) mod 9

if i=0, H(4371) = (1 + 0) mod 9 = 1

then the key 4371 will be placed at index 1 in hash table.

Similarly calculate indexes for remaining keys using **linear probing**.

After inserting all keys into hash table, hash table becomes

index	Keys
0	4199
1	4371
2	9679
3	1323
4	6173
5	4344
6	1989
7	
8	

Hash Table ------- **3M**

In Quadratic probing, Hash function is as follows

$H(x) = (H^1(x) + C1.i + C2.i^2) \ mod \ m$. where m denotes hash table size or bucket size.

i=0,1,2,….. and C1, C2 are arbitrary constants.

To insert 4371, hash function will be

H(4371) = ((4371 mod 10) + 2 .0 + 1.0) mod 9 [:.Here C1=2,C2=1 and i=0)

= (1+0+0) mod 9 = 1

Similarly calculate indexes for remaining keys using **Quadratic probing,** hash table becomes

index	Keys
0	4199
1	4371
2	
3	1323
4	4344
5	9679
6	6173
7	1989
8	

Hash Table

Comparison : In linear probing, difference of probe sequences is fixed i.e., 1 when collision occurs. Where as in Quadratic probing, difference of probe sequences is not fixed. In the above problem to insert 1989 into hash table using linear probing, its takes 7 probe sequences. Where as in quadratic probing, it takes only 4 probe sequences. By seeing this, to insert 1989 , linear probing technique take more time than quadratic probing. --------- **3M**

(OR)

9. a) Explain the concept of separate chaining. [5M]

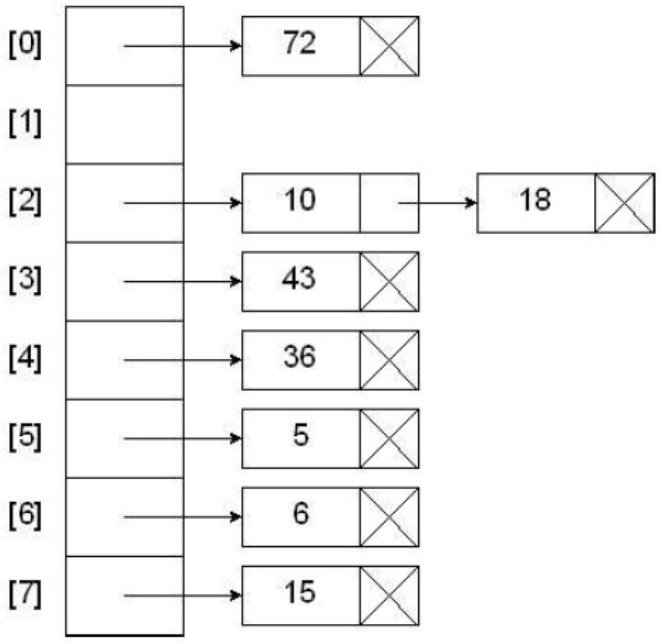
Ans: Separate chaining is defined as a method by which linked lists of values are built in association with each location within the hash table when a collision occurs. ----- **1M**

To understand the concept of Separate chaining, consider the following example

Inserting the keys {36, 18, 72, 43, 6, 10, 5, 15} into hash table with table size 8 is as follows:

Hash key = key % table size

```
4    = 36 % 8
2    = 18 % 8
0    = 72 % 8
3    = 43 % 8
6    =  6 % 8
2    = 10 % 8
5    =  5 % 8
7    = 15 % 8
```



Example ------- 4M

9. b) What are the advantages of Disjoint Set data structure? Explain various operations performed on Disjoint Set data structure with an example. [5M]

Ans: Advantages of Disjoint Data Structure:

1. It is used to identify cycles in undirected graph.
2. Used to identify whether an element is belongs to which subset. ----- **1M**

Implementation 4

To implement the operations of union and find, do the following:

1. Find(A, B): Check whether arr[A] = arr[B]
2. Union(A, B): Connect A to B and merge the components that comprise A and B by replacing elements that have a value of arr[A] with the value of arr[B].

Initially there are 10 subsets and each subset has one element in it.

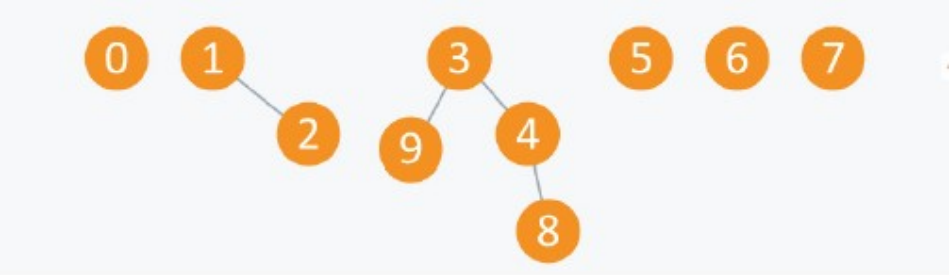
When each subset contains only one element, the array *arr* is as follows:

Let's perform some operations.

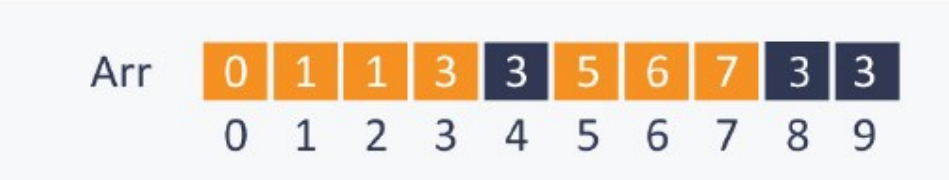
1. Union(2, 1)

The *arr* is as follows:

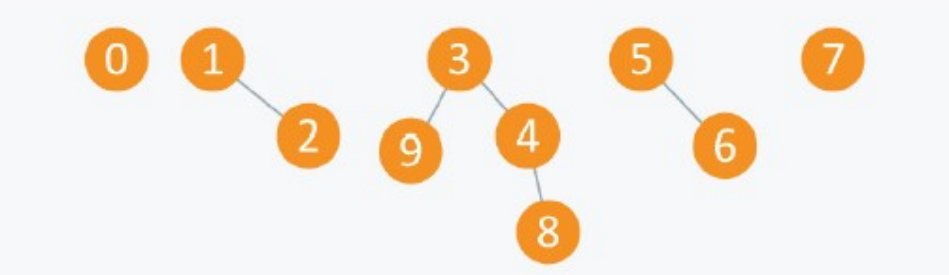
1. Union(4,3)
2. Union(8,4)
3. Union(9,3)



The *arr* is as follows:



1. Union(6, 5)



The *arr* is as follows:

Let's perform some find(A, B) operations.

1. Find(0, 7): 0 and 7 are disconnected, and therefore, you will get a false result
2. Find(8, 9): Although 8 and 9 are not connected directly, there is a path that connects both the elements, and therefore, you will get a true result

------ **4M**

Prepared By

K. Suresh Kumar,

Assistant Professor, **Signature of HOD**

Department of IT.

S.NO.	Examiners	College	Contact Number	Sign.