GROUP A
ASSIGNMENT NO-1

1 Problem Statement:-

Using Divide and Conquer Strategies design a cluster/Grid of BBB or Rasberi pi or Computers in network to run a function for Binary Search Tree using C /C++/ Java/Python/ Scala.

2 Theory :

Divide and Conquer Strategy:

Divide and conquer (D&C) is an algorithm design paradigm based on multi-branched recursion. A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.
This technique is the basis of efficient algorithms for all kinds of problems, such as sorting (e.g., quicksort, merge sort). multiplying large numbers (e.g. Karatsuba), syntactic analysis (e.g., top-down parsers), and computing the discrete Fourier transform (FFTs).
On the other hand, the ability to understand and design D&C algorithms is a skill that takes time to master. As when proving a theorem by induction, it is often necessary to replace the original problem by a more general or complicated problem in order to initialize the recursion, and there is no systematic method for finding the proper generalization. These D&C complications are seen when optimizing the calculation of a Fibonacci number with efficient double recursion.

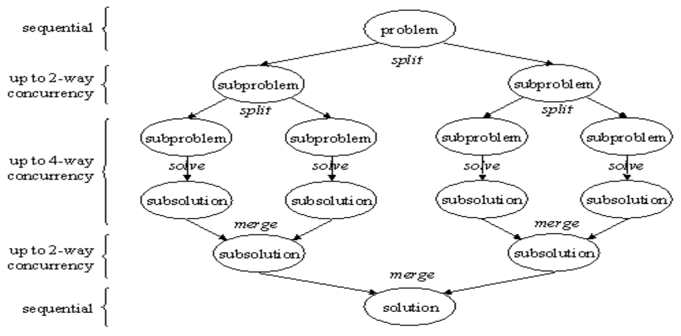
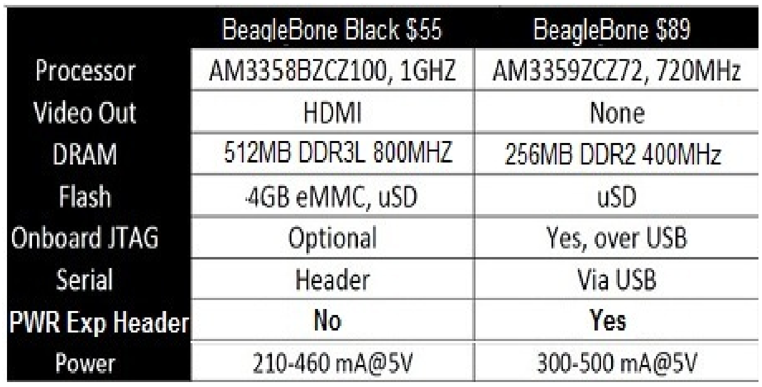


Figure.1

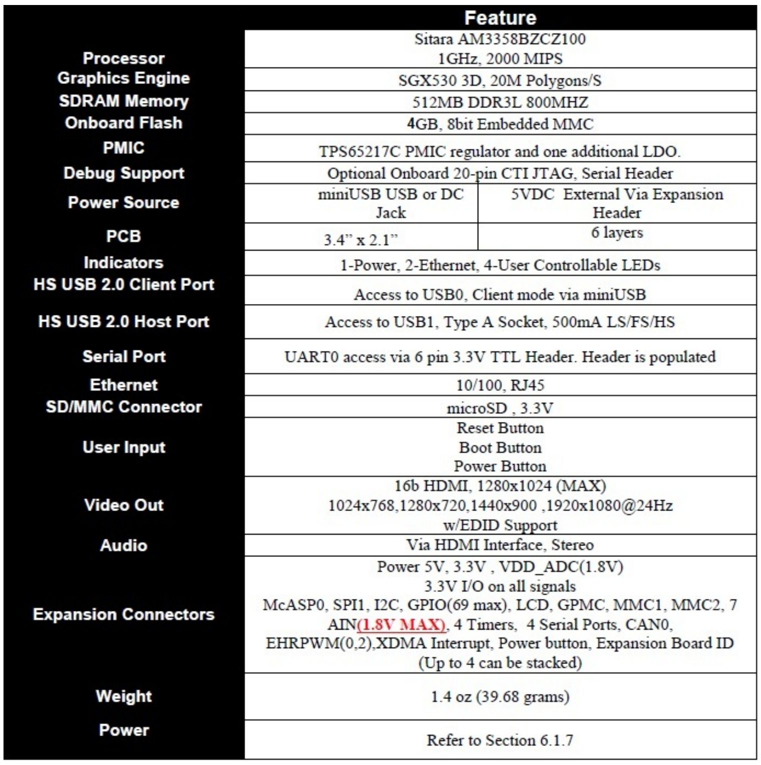
Beagle Bone Black

The BeagleBone Black is the newest member of the BeagleBoard family. It is a lower-cost, high-expansion focused BeagleBoard using a low cost Sitara XAM3359AZCZ100 Cortex A8 ARM processor from Texas Instruments. It is similar to the Beaglebone,but with some features removed and some features added. The table below gives the high points on the differences between the BeagleBone and BeagleBone Black.



	BeagleBone Black $55	BeagleBone $89
Processor	AM3358BZCZ100, 1GHZ	AM3359ZCZ72, 720MHz
Video Out	HDMI	None
DRAM	512MB DDR3L 800MHZ	256MB DDR2 400MHz
Flash	4GB eMMC, uSD	uSD
Onboard JTAG	Optional	Yes, over USB
Serial	Header	Via USB
PWR Exp Header	No	Yes
Power	210-460 mA@5V	300-500 mA@5V

BeagleBone Black Features



	Feature	
Processor	Sitara AM3358BZCZ100 1GHz, 2000 MIPS	
Graphics Engine	SGX530 3D, 20M Polygons/S	
SDRAM Memory	512MB DDR3L 800MHZ	
Onboard Flash	4GB, 8bit Embedded MMC	
PMIC	TPS65217C PMIC regulator and one additional LDO.	
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header	
Power Source	miniUSB USB or DC Jack	5VDC External Via Expansion Header
PCB	3.4" x 2.1"	6 layers
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB	
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS	
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
Ethernet	10/100, RJ45	
SD/MMC Connector	microSD , 3.3V	
User Input	Reset Button Boot Button Power Button	
Video Out	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support	
Audio	Via HDMI Interface, Stereo	
Expansion Connectors	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Weight	1.4 oz (39.68 grams)	
Power	Refer to Section 6.1.7	

BeagleBone Black Picture

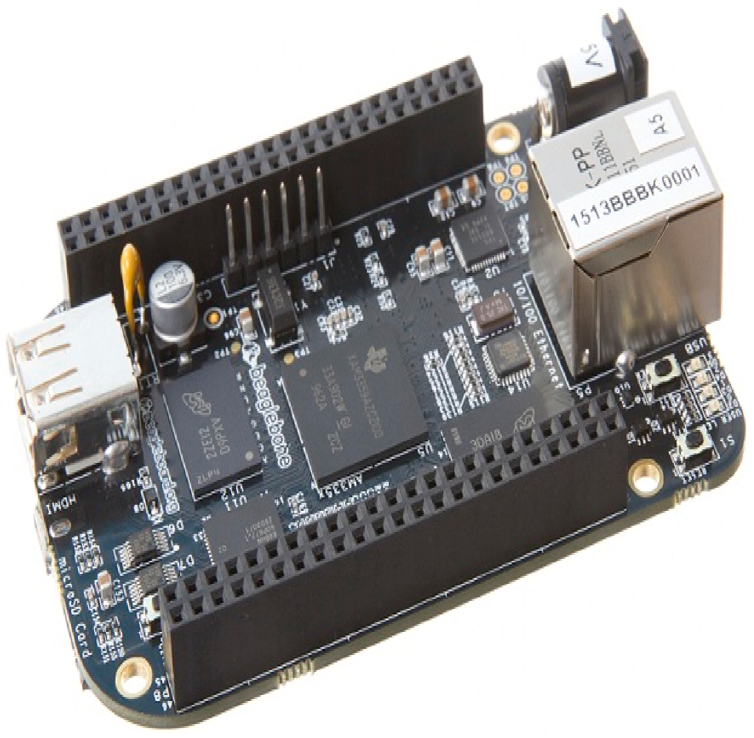


Figure.2

Binary Search Tree

A binary search tree is a rooted binary tree, whose internal nodes each store a key (and optionally, an associated value) and each have two distinguished sub-trees, commonly denoted left and right. The tree additionally satisfies the binary search tree property, which states that the key in each node must be greater than all keys stored in the left sub-tree, and smaller than all keys in right sub-tree.

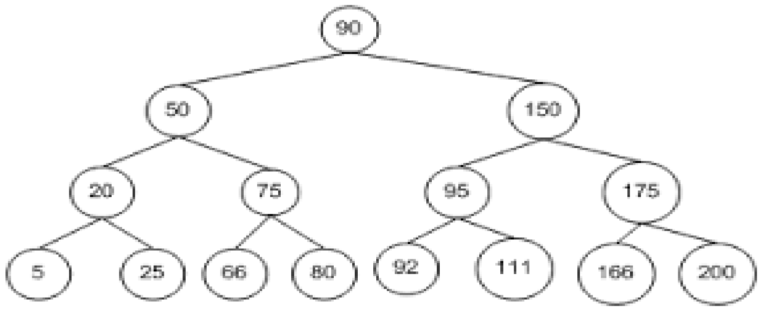


Figure.3

Traversal of the Nodes in a BST

By traversal we mean visiting all the nodes in a graph. Traversal strategies can be specified by the ordering of the three objects to visit: the current node, the left subtree, and the right subtree. We assume the the left subtree always comes before the right subtree. Then there are three strategies.

1. Inorder.

The ordering is: the left subtree, the current node, the right subtree.

2. Preorder.

The ordering is: the current node, the left subtree, the right subtree.

3. Postorder.

The ordering is: the left subtree, the right subtree, the current node.

Operations on BST

1. Searching for a key

We assume that a key and the subtree in which the key is searched for are given as an input. We'll take the full advantage of the BST-property.
Suppose we are at a node. If the node has the key that is being searched for, then the search is over. Otherwise, the key at the current node is either strictly smaller than the key that is searched for or strictly greater than the key that is searched for. If the former is the case, then by the BST property, all the keys in th left subtree are strictly less than the key that is searched for. That means that we do not need to search in the left subtree. Thus, we will examine only the right subtree. If the latter is the case, by symmetry we will examine only the right subtree.

2. Insertion

Suppose that we need to insert a node z such that k = key[z].Using binary search we nd a nil such that replacing it by z does not break the BST-property.

3. Deletion

Suppose we want to delete a node z.
1. If z has no children, then we will just replace z by nil.
2. If z has only one child, then we will promote the unique child to z's place.
3. If z has two children, then we will identify z's successor. Call it y. The successor y either is a leaf or has only the right child. Promote y to z's place. Treat the loss of y using one of the above two solutions.

Computers network

A computer network or data network is a telecommunications network which allows computers to exchange data. In computer networks, networked computing devices exchange data with each other using a data link. The connections between nodes are established using either cable media or wireless media

Figure.4

3 Algorithm :

1.start.
2.create a master(pc) and slaves (BBB) connection.
3.Check the if the all are connected properly.
4.Create a network using router or switch.
5.Create a cluster and test the there connection(using ping).
6.For creating nodes we use BBB input address.
7.Accept the array in randomly.
6.Enter input number in random manner to the system.
7.Now system parallel apply binary search tree algorithm in two Beagle Bone Black.
8.If system find the number and show the ip address of Beagle Bone Black in which number is found .
9.If system not find the number then shows no result found.
10.Stop.

4 Mathematical model:

Let S be a binary search tree system:
S ={Q,$\sum$,O,δ,I,F(me)}
Where ,
1.Q is set of states that performs operations on system(S)
2.$\sum$ is input given by user to select state(q)
3.O is output generated by states (Q)
4.δ is transition function which maps Q*$\sum - >$Q
5.I is input values x given by user

1).δ:Q[$\sum$ x=0] $- > q_0$

Create a cluster
Master system is desktop computer system(DCS).
Beagle Bone Black->BBB.
Router(R)
        |———————-BBB_1
        |connection
DCS——R———————-BBB_2
        |connection
        |———————-BBB_3
Test the connection(using ping)

2).δ:Q[$\sum$ x=1] $- > q_1$

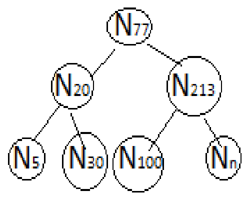
Accept the array A[] and array element(e) randomly
A[$e_5, e_{20}, e_{77}, e_{213}$............$e_n$]

3).δ:Q[$\sum$ x=2] $- > q_2$

Enter input(I) from user(U)
Uε[$I_1 I_2 I_3 I_4$..........I_n]

4).δ:Q[$\sum$ x=3] $- > q_3$

Create binary search tree is accept array A[]
First step selected randomly root node(N)
Nε[$N_5, N_{20}, N_{77}, N_{213}, N_{100}$............$N_n$]
Input Element-> I



if I=N then find element
if I>N then input shift right node
if I>N then input shift left node
if I≠N then not find element
Parallel apply binary search tree(BST) algorithm in two Beagle Bone Black(BBB).

```
        |——BBB1
BST–|
        |——BBB2
```

5).δ:Q[$\sum$ x=4] $->q_4$

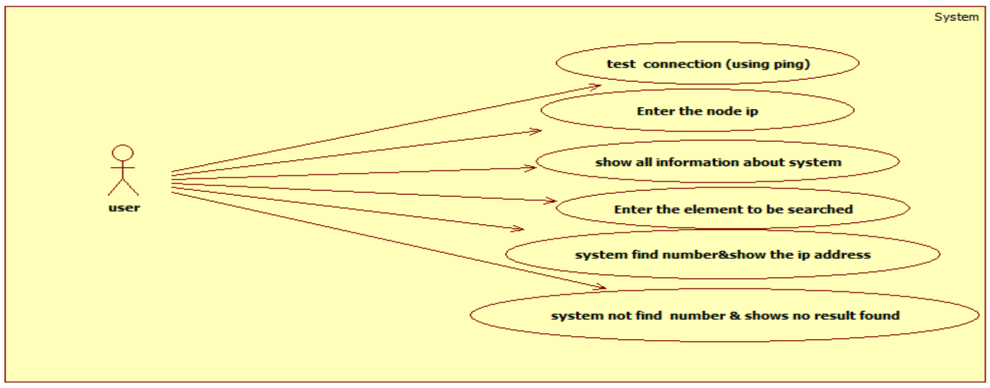
If system
(i) Find the number(N) and show the ip address(IPA) of Beagle Bone Black(BBB)
N->BBB(IPA)
(ii)Not find the number then shows no result found

6).δ:Q[$\sum$ x=5] $->q_5$

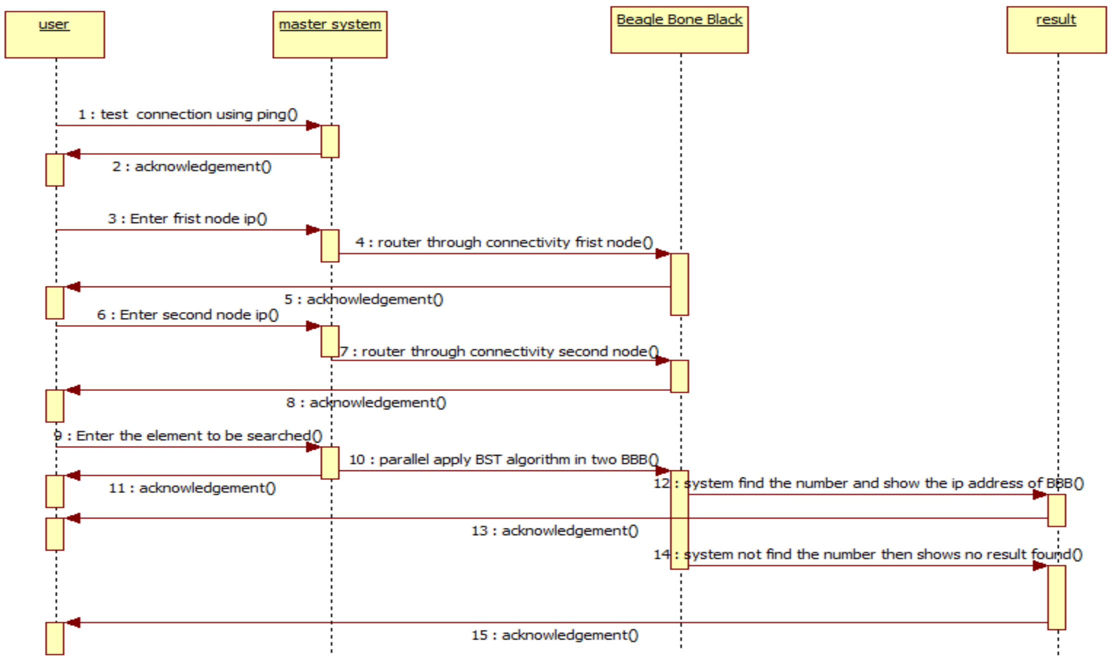
Stop

5 UML Diagram

Use Case Diagram


System
user
test connection (using ping)
Enter the node ip
show all information about system
Enter the element to be searched
system find number&show the ip address
system not find number & shows no result found

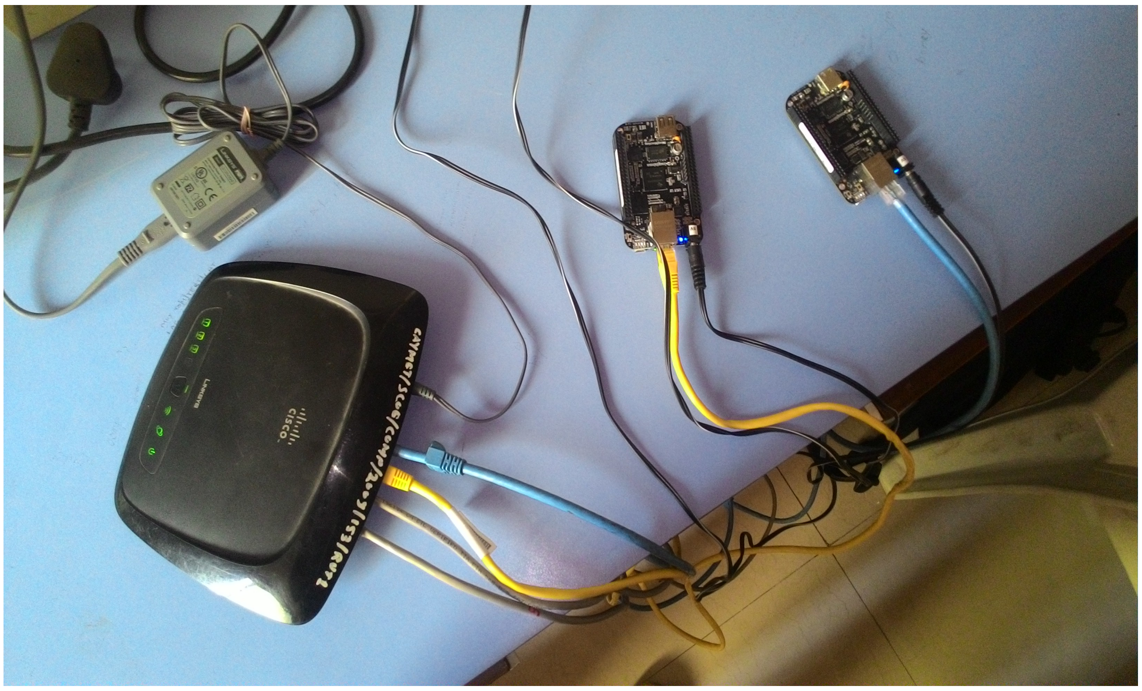
Sequence Diagram


user
master system
Beagle Bone Black
result
1 : test connection using ping()
2 : acknowledgement()
3 : Enter frist node ip()
4 : router through connectivity frist node()
5 : acknowledgement()
6 : Enter second node ip()
7 : router through connectivity second node()
8 : acknowledgement()
9 : Enter the element to be searched()
10 : parallel apply BST algorithm in two BBB()
11 : acknowledgement()
12 : system find the number and show the ip address of BBB()
13 : acknowledgement()
14 : system not find the number then shows no result found()
15 : acknowledgement()

6 Program:-

```
import  random,sys,dispy,time

#node class
class Node(object):
    def __init__(self, val):
        self.l_child = None
        self.r_child = None
        self.data = val
#insert
def binary_insert(root, node):
    if root is None:
        root = node
    else:
        if root.data > node.data:
            if root.l_child is None:
                root.l_child = node
            else:
                binary_insert(root.l_child, node)
        else:
            if root.r_child is None:
                root.r_child = node
            else:
                binary_insert(root.r_child, node)



def make_tree_and_return_root(arr):
    r=Node(arr[0])
    for i in range(1,arr.__len__()):
        binary_insert(r, Node(arr[i]))

    return r

#search
def search_tree(root,ele):
    print "root is: "+str(root.data)
    if root.data==ele :
        print "Element found:"+str(root.data)
    elif root.data>ele:
        print "left "
        if root.l_child!=None:
            search_tree(root.l_child, ele)
        else:print "not found"
    elif root.data<ele:
        print "right "
        if root.r_child!=None:
            search_tree(root.r_child, ele)
        else:print "not found"


#for dividing list
def divide_list(lst, n):
    return [lst[i::n] for i in range(n)]


#tree


if __name__ == '__main__':

    arr=[]
    nodes=[]
    n=int(raw_input("Enter the total size of elements : "))
    for i in range(0,n):
        #sys.maxint
        temp=random.randrange(1,100)
        arr.append(temp)

    #initializing the nodes dynamically
    '''while True:
        temp=raw_input("Enter the node ip or ok to initialize :")
        if temp=="ok":
            break
        else:
            nodes.append(temp) '''

    #make cluster now
```

```
nodes=["192.168.1.102","192.168.1.104"]
cluster = dispy.JobCluster(search_tree,nodes,depends=[Node])
cluster.stats()
#making trees according to the nodes
n=nodes.__len__()
z=divide_list(arr, n)
print z
#dynamic b trees list
bst_lst=[]
for array in z:
    bst_lst.append(make_tree_and_return_root(array))
#bst_lst stores the roots of binary trees

elementToSearch=int(input("Enter the element to be searched: "))

#search_tree(bst_lst[0], elementToSearch)







# now time to submit the jobs and get the element
jobs = []
for i in range(n):
    job = cluster.submit_node(nodes[i],bst_lst[i],elementToSearch)
    job.id = nodes[i]
    jobs.append(job)

for job in jobs:
    job()
    if job.status != dispy.DispyJob.Finished:
        print('job %s failed: %s' % (job.id, job.exception))
    else:
        print('%s: %s :output: %s' % (job.id, job.result,job.stdout)

cluster.print_status()
cluster.close()
```



7 Output-

```
Enter the total size of elements : 200
Enter the node ip or ok to initialize :192.168.1.102
Enter the node ip or ok to initialize :192.168.1.104
Enter the node ip or ok to initialize :ok
2016-03-08 15:33:36,519 - dispy - Storing fault
recovery information in "_dispy_20160308153336"
 Node |                          CPUs |Jobs|Sec/Job   |Node Time Sec
--------------------------------------------------------------------------
 192.168.1.102 (beaglebone)| 1   |  0 | 0.000    | 0.000
 192.168.1.104 (beaglebone)| 1   |  0 | 0.000    | 0.000
Total job time: 0.000 sec
[[348, 690, 45, 932, 231, 859, 828, 836, 367, 22, 630, 517, 733, 688,
303, 353, 934, 134, 338, 757, 668, 554, 585, 399, 530, 229, 419, 267,
533, 153, 948, 367, 598, 606, 208, 852, 707, 446, 97, 160, 285, 43,
891, 424, 67, 275, 681, 608, 942, 179, 711, 769, 46, 798, 922, 70, 772,
 799, 67, 310, 368, 551, 563, 85, 845, 280, 837, 855, 340, 869, 104,
405, 304, 338, 792, 962, 589, 438, 685, 760, 368, 945, 714, 103, 132,
986, 514, 903, 867, 959, 400, 883, 102, 905, 976, 910, 112, 395,
728, 66], [786, 187, 449, 136, 382, 125, 76, 342, 225, 398, 373, 337,
728, 245, 869, 965, 85, 790, 679, 999, 925, 102, 721, 37, 987, 481, 60,
 500, 826, 171, 338, 10, 95, 664, 490, 453, 702, 614, 464, 273, 569,
184,731, 65, 253, 663, 621, 65, 361, 440, 802, 178, 398, 377, 628, 683
, 842, 62, 915, 896, 73, 532, 951, 368, 523, 75, 574, 743, 247, 146,
98, 387, 137,710, 417, 458, 930, 947, 875, 184, 76, 440, 863, 241,
464, 530, 217, 215, 293, 704, 2, 623, 718, 533, 935, 447, 151, 146,
67, 403]]
Enter the element to be searched: 733
192.168.1.102: None :output: root is: 348
right
root is: 690
right
root is: 932
left
root is: 859
left
root is: 828
left
root is: 733
Element found:733

192.168.1.104: None :output: root is: 786
left
root is: 187
right
root is: 449
right
root is: 728
right
root is: 731
right
root is: 743
left
not found
Node                          |CPUs|Jobs|Sec/Job| Node Time Sec
----------------------------------------------------------------
 192.168.1.102 (beaglebone)| 1   | 1   |0.111   | 0.111
 192.168.1.104 (beaglebone)| 1   | 1   |0.061   | 0.061

Total job time: 0.172 sec
```

8 Conclusion:

Thus we studied and implemented using divide and conquer strategies design a cluster Grid of BBB Computers in network to run a function for Binary Search Tree using Python.

Signature of Subject Teacher:-______________