```
theta = u'\u0398'
def makeAList(s):
    return map(int, s.split(' '))
def take_input():
    print "Please enter the elements separated by space"
    input_array = raw_input().split(' ')
    return map(int, input_array)
class Node:
    def __init__(self, data=-1, left = None, right=None):
        self.right = right
        self.left = left
        self.data = data
    def __str__(self):
        s = ""
        if self.left != None:
            s = s+ str(self.left.data)+"<--"
        s = s + "|"+str(self.data)+"|"
        if self.right!=None:
            s = s+ "-->"+str(self.right.data)
        #return "."+s+"."
        return s

class BinarySearchTree:
    def __init__(self):
        self.root = None
    def populateTree(self):
        input_array = take_input()
        #input_array = makeAList("4 3 6 1 2 5 7")
        for element in input_array:
            self.addElement(element)
    def addElement(self, element):
        if self.root == None:
            # this is the first element
            self.root = Node(element)
        else:
            # now find the parent here.
            parent = self.findParent(element)
            if parent != None:
                # now make the new node here
                newNode = Node(element)
                if element > parent.data:
                    parent.right = newNode
                else:
                    parent.left = newNode
            else:
                print "element already exists, cannot insert the new element"
    def inorderTraversal(self):
        if self.root == None:
            print "tree empty!"
```

```
            return
        self.inorder(self.root)
        print ""
    def inorder(self, node):
        if node == None:
            return
        self.inorder(node.left)
        print node,
        print " ",
        self.inorder(node.right)
    def findParent(self, element):
        '''
        this function returns the possible parent of the new element to be inserted
        it will return None if element already exists
        Note: root is not None
        '''
        current = self.root
        parent = current
        while current!=None:
            parent = current
            if element > current.data:
                current = current.right
            elif element < current.data:
                current = current.left
            else:
                return None
        return parent
    def showMenu(self):
        menu = ["add a new element", "search for an element", "sort", "exit"]
        while True:
            for i in range(len(menu)):
                print str(i+1)+". "+menu[i]
            print ">>>",
            choice = int(raw_input())
            if choice == 1:
                element = int(raw_input("Enter the element to be added: "))
                self.addElement(element)
                self.printComplexity()
            elif choice==2:
                element = int(raw_input("Enter the element to be searched: "))
                res = self.findParent(element)
                if res == None:
                    print "Found"
                    self.printComplexity()
                else:
                    print "Not found!"
            elif choice==3:
                self.inorderTraversal()
                print "Average case: "+theta+"(n)"
```

```
            elif choice==4:
                break
            else:
                print "wrong input!"
    def printComplexity(self):
        print "Worst case: "+theta+"(n)"
        print "Best case: "+theta+"(log n)"



def main():
    tree = BinarySearchTree()
    tree.populateTree()
    tree.showMenu()
    print "The program will now exit"
main()

'''
Output:
[exam1@localhost bst]$ python main.py
Please enter the elements separated by space
4 3 6 1 2 5 7
1. add a new element
2. search for an element
3. sort
4. exit
>>> 3
|1|-->2   |2|   1<--|3|   3<--|4|-->6   |5|   5<--|6|-->7   |7|
Average case: Θ(n)
1. add a new element
2. search for an element
3. sort
4. exit
>>> 1
Enter the element to be added: 8
Worst case: Θ(n)
Best case: Θ(log n)
1. add a new element
2. search for an element
3. sort
4. exit
>>> 3
|1|-->2   |2|   1<--|3|   3<--|4|-->6   |5|   5<--|6|-->7   |7|-->8   |8|
Average case: Θ(n)
1. add a new element
2. search for an element
3. sort
4. exit
>>> 2
```

```
Enter the element to be searched: 4
Found
Worst case: Θ(n)
Best case: Θ(log n)
1. add a new element
2. search for an element
3. sort
4. exit
>>> 4
The program will now exit

'''
```