PERFORMANCE

Term	Remarks	Staff Member's Signature
I	DBMS Completed practical	M Bhagat 04/10/15
II	D.S.	03/03/16

4) print the new list.

5) Accept an element from the user that is to be searched in the list.

6) use FOR-LOOP in a range from '0' to the total no. of elements to search the elements from the list.

7) use if-loop that the elements in the list is equal to the element accepted from user.

8) If the element is found then print the statement that the element is found along with the elements position.

9) use another if-loop to print that the element is not found if the element which is accepted from user is not there in the list.

10) Draw the output of given algorithm.

Scanned with CamScanner

⑦ Sorted linear search

Sorting means to arrange the element in increasing or decreasing order.

Algorithm -

1) Create empty list & assign it to a variable.

2) Accept total no. of elements to be inserted into the list from user, say 'n'.

3) Use for-loop for using append() method to add the element in the list.

4) Use sort() method to sort the accepted element & assign in increasing order of the list, then print the list.

5) Use if statement to give the range in which element it found in given range then display "element not found".

6) Then use else statement, if element it not found in that then satisfy the given condition.

7) Use the loop in range from 0 to the total no. of elements to be searched before doing this accept and search number from the user using the input statement.

```
input = ("linear search").
a=[]
n=int(input("Enter the range"))
for s in range(0,n):
    s= int(input("Enter a number"))
    a.append(s).
    a.sort()
    print(a).
c= int(input("Enter a number to be searched"))
for i in range(0,n):
    if(a[i]==c):
        print("Number found at this position",i).
        break.
    else:
        print("Number not found")
```

output:

```
>>> Enter the range : 3
>>> Enter a number : 1
[1]
>>> Enter a number : 3
[1,3]
>>> Enter a number : 2
[1,2,3]
>>> Enter a number to be searched : 4
>>> Number not found.
```


Scanned with CamScanner

g) use a loop to see the element in the list is equal to the element accepted from the user.

h) if the element is found then print the statement that the element is found already with the current position.

i) use another a loop to print that the element is not found if any element which is accepted from the user is not there in the list.

j) Attach the input & output of above algorithm:

mn
21-1-19

* Practical No. 2 *

Aim: Implement Binary search to find an searched no. in the list.

Theory:

Binary search.

Binary search is also known as Half-interval search, logarithmic search or binary chop is a search algorithm that finds the position of a target value within a sorted array. If you are looking for the number which is at the end of the list then you need to search entire list and it needs to search entire list. In linear search, which is time consuming. This can be avoided by using Binary search fashion search.

Algorithm:

1) Create empty list and assign it to a variable
2) Using input method, accept the range of given list.
3) Use for-loop add elements in list using append() method.
4) Use sort() method to sort the accepted element & assign it in increasing ordered list print the list after sorting.

Scanned with CamScanner

Output:

Enter the range : 4

Enter the numbers: 2

[2]

Enter the numbers : 8

[2,8]

Enter the numbers: 6

[2,6,8].

Enter the numbers: 4

[2,4,6,8].

Enter the number to be searched: 4

Element found at : 2.

21/11/19

5) Use if-loop to give the range in which element is found in given range then display a message "Element not found".

6) Then use else element, if statements is not found in range then satisfy the below conditions.

7) Accept one argument & key of the element that element has to be searched.

8) Initialize first to 0 & last to last element of the list. as array is starting from 0 hence it is initialized 1 less than the total count.

9) Use for loop & assign the given range.

10) If statement in list & still the element to be searched is not found then find the middle element (m).

11) Else if the item to be searched is still less than the middle term then
Initialize last(h) = mid(m) - 1.
Else
Initialize first (l) = mid(m) - 1.

12) Repeat till you find the element. Check the input & output of above algorithm.

Scanned with CamScanner

No.

7.

8.

9.

10.

* Practical No. 3 *

Aim: Implementation of Bubble sort program on given list.

Theory:

Bubble - sort.

Bubble sort is based on the idea of repeatedly comparing pairs of adjacent elements & then swapping their position if they exist in the wrong order. This is the simplest form of sorting available. In this, we sort the given element in ascending or descending order by comparing two adjacent elements at a time.

Algorithm:

1) Bubble sort algorithm starts by comparing the first two elements of an array and swapping if necessary.

2) If we want to sort the elements of array in ascending order then first element is greater than second then we need to swap the element.

3) If the first element is smaller than second then we do not swap the element.

Scanned with CamScanner

ii) Again second and third elements are compared and swapped if it is necessary and same process upto second last & second last element is compared and swapped.

e) If there are n elements to be sorted then the process mentioned above should be repeated n-1 times to get the required output.

f) Write the output and input of above algorithm of bubble sort separately.

Pr
9/8/17

Scanned with CamScanner

SE...

Practical No. 4.

Aim: Implement quick sort to sort the given list

Theory:

QUICK SORT.

The quick sort is a reverse algorithm based on the divide and conquer technique.

Algorithm:

1) quick sort first selects a value which is called pivot value, first element serve as our first pivot value since we know that first will eventually end up as (our) in that list.

2) The partition process will happen next. It will find the split point and at the same time move other items to the appropriate side of the list, either less than or greater than pivot value.

3) partitioning begins by locating two position markers let us them leftmark & rightmark. At the beginning and end of the remaining items in the list. The goal of the partition process is to move items that are on wrong side with respect to pivot value.

coding:

```
def quick(alist):
    help(alist,0,len(alist)-1)
def help(alist, first, last):
    if first < last:
        split = part(alist, first, last).
        help(alist, first, split-1).
        help(alist, split+1, last).
def part(alist, first, last):
    pivot = alist[first]
    l = first+1.
    r = last
    done = false
    while not done:
        while l<=r and alist[l]<=pivot:
            l=l+1.
        while alist[r]>=pivot and r>=l:
            r=r-1
        if r<l:
            done = true.
        else:
            temp= alist[l]
            alist[l]= alist[r].
            alist[r] = temp.
```


Scanned with CamScanner

```
        temp = alist[finger].
        alist[finger] = alist[i].
        alist[i] = temp.
        return i.

x = int(input("enter range for the list:")).
alist = []
for b in range(0,x):
        b = int(input("enter element:"))
        alist.append(b).
        n = len(alist).
quick(alist).
print(alist).
```

45

4) we begin by incrementing leftmark until we locate a value that is greater than the pivot value. Then we decrement rightmark until we find value that is less than the pivot value. At this point, we have discovered two items that are out of place with respect to eventual split point.

5) At the point where rightmark becomes less than the leftmark, we stop. The position of rightmark is now the split point.

6) The pivot value can be exchanged with the content of split point and pivot value is now in place.

7) In addition, all the items to left of split point are less than pivot value & all the items to the right of split point are greater than pivot value. The list can now be divided at split point and quick sort can be invoked recursively on the two halves.

8) The quick sort function invoked a recursive function quick sorthelper.

9) quick sort helper begin with some base as the merge sort.

10) If length of the list is less than the equal one it is already sorted.

Scanned with CamScanner

11). If it is greater, then it can be partitioned & recursively sorted.

12) The partition function implements the process that are described earlier.

13) Display & stick the coding and output of the above algorithm.

output :

Enter range for the list: 4
Enter the element : 6
Enter the element : 8
Enter the element : 4
Enter the element : 9

[4, 4, 6, 9].

Scanned with CamScanner

ap

coding:

```
class stack:
    global tos
    def __init__(self):
        self.l = [0, 0, 0, 0, 0, 0, 0]
        self.tos = -1
    def push (self, data):
        n = len (self.l).
        if self.tos = n-1:
            print ("stack is full").
        else:
            self.tos = self.tos + 1.
            self.l [self.tos] = data
    def pop (self):
        if self.tos < 0:
            print ("stack is empty")
        else:
            k = self.l[self.tos].
            print ("data =", k)
            self.tos = self.tos - 1
    def peek (self):
        if self.tos < 0:
            print ("stack is empty").
        else:
            k = self.l [self.tos].
            print ("Data =", k).
            self.l [self.tos] = 0.
s = stack()
```


Scanned with CamScanner

4) Use if statement to give the condition that if length of given list is greater than the range of list then print stack is full.

5) Or else print statement to insert the element in the stack & initialize the value.

6) push method used to insert the element but pop method used to delete the element from the stack.

7) If in pop method, value is less than 1 then print the stack is empty or else delete the element from stack at topmost position.

8) Assign the element values in push method to add & print the given value is popped not.

10) Attach the input & output of above algorithm.

9) First condition checks whether the no. of elements are zero while the second one whether top is assigned any value. If top is not assigned any value, then it can be said that stack is empty.

output:

S.push(10)
S.push(20)
S.1
[10, 20, 0, 0, 0]
S.pop()
data = 20
[10, 0, 0, 0, 0]
S.peek()
Top element = 20.

Scanned with CamScanner

```
class queue:
    global R
    global F
    def __init__(self):
        self.R = 0
        self.F = 0
        self.l = [0,0,0]
    def enqueue(self, data):
        n = len(self.l)
        if self.R < n:
            self.l[self.R] = data
            self.R = self.R + 1
            print("Element inserted....", data)
        else:
            print("Queue is full")
    def dequeue(self):
        n = len(self.l)
        if self.F < n
            print(self.l[self.F])
            self.l[self.F] = 0
            print("Element deleted")
            self.F = self.F + 1.
        else:
            print("Queue is empty")

q = queue()
```


Scanned with CamScanner

CP

Algorithm

1) Define a class queue & assign global variables then define init() with self argument in init(), assign or initialize the initial value with the help of self argument.

2) Define a empty list & define enqueue() method with 2 argument define the length of empty_list.

3) use IF statement that length is equal to queue if full or use else insert the element in empty list or display that queue element added successfully & increment by 1.

4) Define dequeue() with self argument under that use if statement that front is equal to length of list then display queue is empty or else, given that front is at zero & using the delete the element from the front side & increment it by

5) Now call the queue() function & give the element that has to be added in the empty list by using enqueue() & print the list option for adding & some for deleting & display the list after deleting the element from the list.

output :

```
>>> Q.add (10)
    element inserted = 10
>>> Q.add (20)
    element inserted = 20
>>> Q.add (30)
    element inserted = 30
>>> Q.add (40)
    queue is full
>>> Q.remove()
    10 element deleted.
```


Scanned with CamScanner

```
def evaluate:
    k = s.split()
    n = len(k)
    stack = []
    for i in range(n):
        if k[i].isdigit():
            stack.append(int(k[i]))
        elif k[i] == "+":
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) + int(a))
        elif k[i] == "-":
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) - int(a))
        elif k[i] == "*":
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) * int(a))
        else:
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) / int(a))
    return stack.pop()

s = "8 6 4 * +"
q = evaluate(s)
print("The evaluated value is:", q)
```


Scanned with CamScanner

6) If the token is an operator +, -, /, * it will need two operands. Pop the P twice the first pop is second operand & the second pop is the first operand.

7) Perform the arithmetic operation. Push the result back on the 'n'.

8) When the input expression has been completely processing the result is on the stack. Pop the & return the value.

9) Print the result of string after the evolution of postfix.

10) Attach the input & output of above algorithm.

Scanned with CamScanner

```
class node:
    global data
    global next
    def __init__(self, item):
        self.data = item
        self.next = None
class linkedlist:
    global s
    def __init__(self):
        self.s = NONE
    def addL(self, item):
        newnode = node(item):
        if self.s == None:
            self.s = newnode
        else:
            head = self.s
            while head.next != None:
                head = head.next
                head.next = newnode.
    def addB(self, item):
        newnode = node(item)
        if self.s = None:
            self.s = newnode
        else:
            newnode.next = self.s
            self.s = newnode
```


Scanned with CamScanner

6) we may lose the reference to the 1st node in our linked list and hence most of one list so in order to avoid making some unwanted changes to the 1st node, we will use a temporary node to traverse.

7) we will use this temporary node as a copy of the node, we are currently traversing. Since we are making temporary node a copy of current node the datatype of the temporary node should also be node.

8) BUT the 1st node is referenced by current so we can traverse to 2nd nodes as h = h.next.

9) Similarly, we can traverse rest of nodes in the linked list using same method by while loop

10) our concern now is to find terminating condition for while loop.

11) The last node in the linked list is referred to the tail of linked list since the last node of linked list does not have any next node, the value in the next field of the last node

12) we can refer to the last of linked list as h.next.

Scanned with CamScanner

12

OUTPUT :

20

30

40

50

60

70

80

Scanned with CamScanner

Practical No. 9.

Aim : Implementation of merge sort.

Theory: like quicksort, merge sort is a divide & conquer algorithm it divides input array in two halves calls itself for the two halves & then merges the two sorted halves. The merge() function is used for merging two halves. The mergeCarr, l, m, r) is key process that assumes that arr[l..m] & arr[m+1..r] are sorted & merges the two sorted sub-arrays into one.

Algorithm:

1) Define the sort (arr, l, m, r)
2) Stores the starting position of both parts in temporary variables.
3) Checks if first part comes to an end or not.
4) checks if second part comes to an end or not.
5) checks which part has smaller element.
6) Now the real array has element in sorted m including both parts.

Scanned with CamScanner

```
def mergeSort(arr, l, r):
    if l < r:
        m = (l+(r-1))//2
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)

arr = [12, 11, 13, 5, 6, 7]
print("Given array is", arr)
n = len(arr)
mergeSort(arr, 0, n-1)
print("\n\nSorted array is", arr)
```

OUTPUT :

```
Given array is [12, 11, 13, 5, 6, 7]

Sorted array is [5, 6, 7, 11, 12, 13]
```


Scanned with CamScanner

PRACTICAL NO. 10.

Algorithm.

1) Define two empty set as set 1 and set 2, now use for statement providing the range of above 2 set :

2) Now add() method used for adding the element according to given range, then print the set after adding.

3) Find the union and intersection of above 2 sets using (comma), ; (semi) method print the sets of union and into of set 3 and set 4.

4) Use if statement to find out the subset and superset of set 3 and set 4 display the above ans

5) Display that element in set 3 and set 4 display set using mathematical operation.

6) Use isdisjoint() to check that anything is comm element is present or not. If not then display that it is mutually exclusive event.

7) Use clear() to remove or delete the sets and print the set after clearing the element present in the set.

Scanned with CamScanner

```
class Node :
    global rl
    global l
    global data
    def __init__(self,i) :
        self.l = None
        self.data = i
        self.r = None
class tree:
    global root
    def __init__(self):
        self.root = None
    def add (self,val):
        if self.root == None:
            self.root = Node(val)
        else:
            newnode = Node(val)
            h = self.root
            while true:
                if newnode.data < h.data:
                    if h.l != None:
                        h = h.l
                    else :
                        h.l = newnode
                        print(newnode.data, "added on left of",h.dat
                        break
                else :
                    if h.r != None:
                        h = h.r
                    else:
```


Practical No. 11.

Aim : Program based on binary search tree by implementing Inorder, Preorder & Postorder Traversal.

Theory :

Binary tree is a tree which supports maximum of 2 children for any node within the tree. Thus any particular node can have either 0 or 1 or 2 children. There is another identity of binary tree that it is ordered such that 1 child is identified as left child & other as right child.

1. Inorder :

(i) Traverse the left subtree, the left subtree in turn might have left & right subtrees.
(ii) Visit the root node.
(iii) Traverse the right subtree and repeat it.

2. Preorder :

(i) Visit the root node.
(ii) Traverse the left subtree. The left subtree in turn might have left and right subtree.
(iii) Traverse the right subtree. Repeat it.

3. Postorder :

(i) Traverse the left subtree. The left subtree in turn might have left and right subtrees.

Scanned with CamScanner

(ii) Traverse the right subtree
(iii) visit the root node.

* Algorithm :

1) Define class node and define init() method with 2 argument. Initialize the value in this method.

2) Again, define a class BST that is binary search tree with init() method with self argument and assign the root is None.

3) Define add() method for adding the node. Define a variable p that p = node(value)

4) use if statement for checking the condition that root is none then use else statement for if node is less than the main node then put on average that in left

5) use while loop for checking the node is less than or greater than the main node and break the loop if it is not satisfying.

6) use if statement within that else statement for checking that node is greater than main root then put it into rightside.

Scanned with CamScanner

(a)

```
T.add(13).
T.add(17).
print("preorder")
T.preorder(T.root)
print("inorder")
T.inorder(T.root)
print("postorder")
T.postorder(T.root)
```

output:

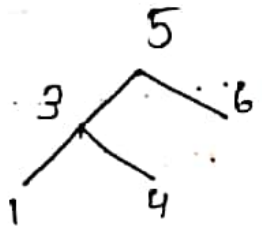
5 added on left of 7.
12 added on right of 7
8 added on left of 5
6 added on right of 5
9 added on left of 12.
15 added on right of 12
1 added on left of 3
4 added on right of 3
8 added on left of 9.
10 added on right of 9
13 added on left of 15
17 added on right of 15

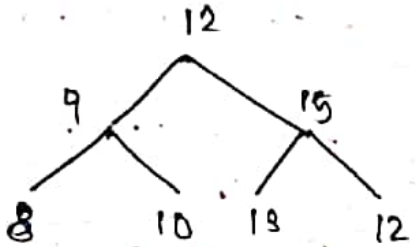
Scanned with CamScanner

ia

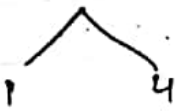
* preorder: (VLR)

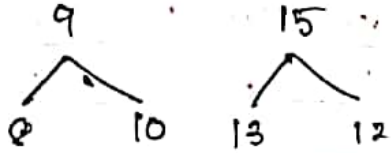
1. 7





2. 7 5 3 6 12 9 15

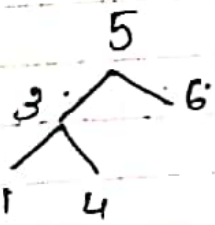


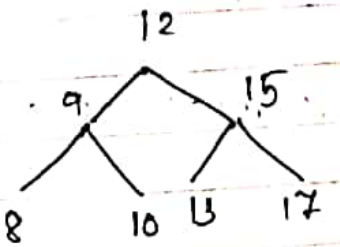


3. 7 5 3 1 4 6 12 9 8 10 15 13 17

* postorder: (LRV)

1.

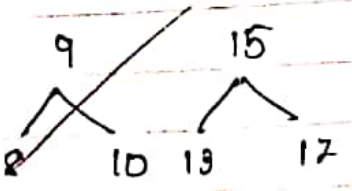




7

2.

6 5



12 7

3. 1 4 3 6 5 8 10 9 13 17 15 12 7

Scanned with CamScanner