

exact

80%

100% - 20% = 80%

100% - 20% = 80%

100% - 20% = 80%

80%

80%

100% - 20% = 80%

80%

100% - 20% = 80%

80%

100% - 20% = 80%

80%

100% - 20% = 80%

- 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.~~

No. 7
8
9
10

① Bound 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 elements in the list.
- 4) Use sort() method to sort the accepted elements & assign in increasing order of the list, then print the list.
- 5) Use if statement to give the range in which element is found in given range then display "Element not found".
- 6) Then use else statement, if element is not found in any 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 input statement.

```
input = ("linear search")
a = []
n = int(input("Enter the range"))
for i 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.
```

No.
1
2
3
4
5
6
7
8
9
10

- a) use if loop then the element in the list is equal to the element accepted from the user.
- b) if the element is found then print the statement that the element is found along with the current position.
- c) use another if loop to print that the element is not found if the element which is accepted from the user is not there in the list.
- d) Attach the input & output of above algorithm.

my
help

* 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~~ entire list many times 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.

40

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: 1.

✓
2/2/20

41

- 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 if NOT found in range then satisfy the below conditions.
- 7) Accept one requirement & 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).
 - i) Else if the item to be searched is still less than the middle term then
Initialize 'last(h) = mid(m)-1.'
 - ii) Else
Initialize 'first (h) = mid(m)-1.'
- 11) Repeat till you find the element. Atk the input & output of above algorithm.

* 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.

- 4) Again second and third elements are compared and swapped if it is necessary and this process goes on until last & second last element is compared and swapped.
- 5) If there are n elements to be sorted then the process mentioned above should be repeated $n-1$ times to get the required result.
- 6) ~~State the output and input of above algorithm of bubble sort step by step.~~

~~M
Ans~~

Practical NO. 4.

Aim: Implement quick sort to sort the given list.

Theory:

QUICK SORT.

The quick sort is a recursive 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, first will eventually end up at last 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 marks let call them leftmark & highmark. All 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
    u = last
    done = False
    while not done:
        while l <= u and alist[l] <= pivot:
            l = l + 1
        while alist[u] >= pivot and u >= l:
            u = u - 1
        if u < l:
            done = True
        else:
            temp = alist[l]
            alist[l] = alist[u]
            alist[u] = temp.
    
```

```

14
temp = alist[0]
alist[0] = alist[n]
alist[n] = temp
return n

n = int(input("Enter range for the list:"))
alist = []
for b in range(0,n):
    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 invokes a recursive function quick sort helper.
 - 9) quick sort helper begins with some base as the merge sort.
 - 10) If length of the list is less than the equal one, it is already sorted.

- Q
- i) If it is greater, than it can be partitioned & successively sorted.
 - ii) The partition function implements the process that was described earlier.
 - iii) Display & stick the coding and output of the above algorithm.

46

OUTPUT :

Enter range for the list: 4

Enter the element: 5

Enter the element: 8

Enter the element: 4

Enter the element: 9

[4, 4, 6, 9].

8

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()

- 4) use if statement to give the condition that if length of given list is greater than the length of list then print stack is full.
- 5) or else print statement & insert the element in the stack & initialize the value.
- 6) push method used to insert the element & pop method used to delete the element from the stack
- 7) if in pop method, value is less than 1 then return the stack is empty otherwise delete the element from stack at topmost position.
- 8) Assign the element values in push method to add & print the given value is popped or not.
- 9) Attach the input & output of above algorithm
- 10) First condition checks whether the no. of elements are zero or not. The second condition checks for if 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. I
[10, 20, 0, 0, 0]
S. POP ()
data = 20
[10, 0, 0, 0, 0]
S. PEAK ()

```

TOP ELEMENT = 20.

class queue:

 global H

 global F

 def __init__(self):

 self.H = 0

 self.F = 0

 self.L = [0, 0, 0]

 def enqueue(self, data):

 n = len(self.L)

 if self.H < n:

 self.L[self.H] = data

 self.H += 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 += 1

 else:

 print("Queue is empty")

q = Queue()

CB

- Algorithm
- 1) Define a class queue & assign global variables then define init() with self argument in init(), assign all 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 ~~less than~~ if queue is full or else insert the element in empty list & display that queue element added successfully & increment by 1.
 - 4) Define dequeue() with self argument under this we have to 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 1.
 - 5) Now call the queue() function & give the element which has to be added in the empty list by using enqueue() to print the list after the adding & some time after deleting & display the list after deleting the element from the list.

OUTPUT:

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

68

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 9 * + "

H = evaluate(S)

print(f"The evaluated value is, {H}.)

- 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 'm'.
- 8) When the input expression has been completely processing the result is on the stack. Pop them & return the value.
- 9) print the answer or string after the evolution of postfix.
- 10) Attach the input & output of above algorithm

25

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

- b) we may lose the reference to the 1st node in our linked list and hence most of one list go in order to avoid making some unwanted changes to the 1st node, we will use a temporary node to handle this.
- c) we will use this temporary node as a copy of the node, as we are currently transversing. Since we are making temporary node a copy of current node the datatype of the temporary node should also be node.
- d) BUT the 1st node is referenced by current so we can traverse to 2nd nodes as `h->next`.
- e) similarly, we can traverse rest of nodes in the linked list using same method by while loop
- f) our concern now is to find terminating condition for while loop
- g) The last node in the linked list is referred to as 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 is `None`.
- h) we can refer to the last of linked list as `h->next = None`.

2

OUTPUT :

20

30

40

50

60

70

80

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 call itself for the two halves & then merges the two sorted halves. The merge() function is used for merging two halves. The merge($a[l], m, r]$) is key process that assumes that $a[l..m]$ & $a[m+1..r]$ are sorted & merges the two sorted sub-arrays into one.

Algorithm:

- 1) Define the sort ($a[l], l, m, r)$)
- 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 manner including both parts.

25

def sort [arr, l, m, r]:

D1 = arr - R + 1

D2 = .

· merge sort (arr, m+1, R)

sort (arr, l, m, R)

arr = []

Print (arr)

n = len (arr)

merge sort (arr, 0, n-1)

Print (arr).

OUTPUT :

[12 11 13 5 6 7]

[5 6 7 11 12 13]

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 sets.
- 2) Now add() method used for adding the element according to given range, then print the sets after adding.
- 3) Find the union and intersection of above 2 sets using (comma), (or) method print the sets of union and info 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 in set using mathematical operation.
- 5) Use isdisjoint() to check that anything is common element is present or not. If not then display that it is mutually exclusive event.
- 6) Use clear() to remove or delete the sets and print the sets again clearing the element present in the set.

```

58
class Node:
    global n1
    global l
    global data
    def __init__(self, l):
        self.l = None
        self.data = l
        self.n = 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.data)
                        break
                else:
                    if h.r != None:
                        h = h.r
                    else:
                        h.r = newnode
    def:
        if h.n != None:
            h = h.n
        else:

```

59

Practical No. 11.

Aim: program based on binary search tree by implementing Inorder, Preorder & Postorder Transversal.

Theory:

- Binary tree is a tree which supports maximum of 2 children for any node within the tree. Then 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 1st child is identified as left child & other as right child.

1. Inorder:

- Transverse the left subtree, the left subtree in turn might have left & right subtrees.
- Visit the root node.
- Transverse the right subtree and repeat it.

2. Preorder:

- Visit the root node.
- Transverse the left subtree. The left subtree in turn might have left and right subtree.
- Transverse the right subtree. Repeat it.

3. Postorder:

- Transverse the left subtree. The left subtree in turn might have left and right subtrees.

6.2

- (i) Traverse the right subtree.
- (ii) visit the root node.

* Algorithm :

- 1) Define class node and define init() method with argument. Initialize the value in this method.
- 2) Again, define a class BT that is binary search tree with init() method with self argument and assign the root to 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 statement for if node is less than the main node then put on left otherwise that in right.
- 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 use statement for checking that node is greater than main root then put it into right side.

63

```
T.add(3)  
T.add(7)  
print ("Preorder")  
T.preorder(T.root)  
print ("inorder")  
T.inorder(T.root)  
print ("postorder")  
T.postorder(T.root)
```

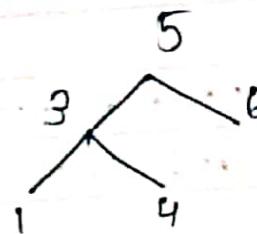
OUTPUT:

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
12 added on right of 15

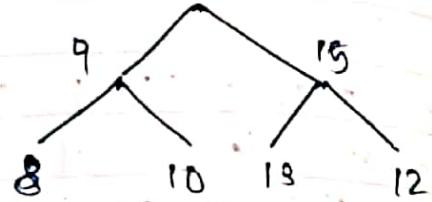
13

* preOrder:(LRV)

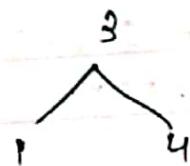
1. 7



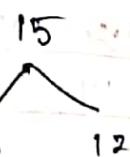
12



2. 7 5



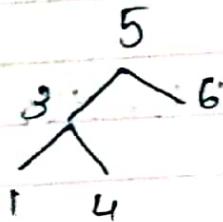
12



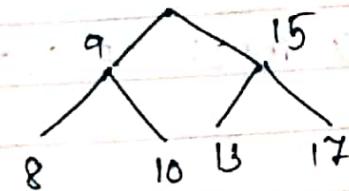
3. 7 5 3 1 4 6 12 9 8 10 15 13 17

* postOrder:(LVR)

1.

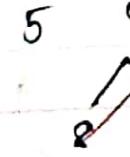


12



7

2.



12

7

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