|  |  |
| --- | --- |
| **Ex. 8** | **EXPLORING LISTS AND TUPLES** |
| **Date: 28/03/24** |  |

**Aim:**

To explore lists and tuples in Python by writing programs for the following:

1. Sort a list of numbers obtained from the user using Bubble sort, without using built- in functions. Find the kth largest number in the list. Obtain k from the user.
2. Sort a list of numbers obtained from the user using Insertion sort, without using built-in functions.
3. Perform a linear search on a given list to find a given element and return its position if present in the list, without using built-in functions.
4. Use the binary search algorithm to check if a given element is present in a given tuple, without using built-in functions, except the “sorted” function.
5. Count the frequency of occurrence of each element of a tuple and display the elements and their count. Find the elements that appear only once in the tuple. (Write 2 programs, one that makes use of tuple methods and functions, and another that does not.)

**Algorithm:**

**(a)**

STEP 1: Prompt the user to input a list of elements.

STEP 2: Request the user to enter the value of k to determine the k-th largest element.

STEP 3: Implement the bubble sort algorithm:

- Iterate through the list multiple times.

- In each iteration, compare adjacent elements and swap them if they are in the wrong order.

STEP 4: Print the sorted list once all elements are in the correct order.

STEP 5: Retrieve and print the k-th largest element using the sorted list's indexing.

**(b)**

STEP 1: Prompt the user to input a list of elements.

STEP 2: Iterate through the list starting from the second element.

STEP 3: For each element (key), compare it with the elements before it.

STEP 4: Shift all elements greater than the key one position ahead.

STEP 5: Insert the key into its correct position.

STEP 6: Print the sorted list after all elements have been processed.

**(c)**

STEP 1: Prompt the user to input a list of elements.

STEP 2: Request the user to enter the element they want to search for.

STEP 3: Initialize a flag variable to 0, indicating the element has not been found.

STEP 4: Iterate over the list:

- If the current element matches the search element, store its index, set the flag to 1, and break the loop.

STEP 5: If the flag is set to 1 after the loop, print that the element is found and its position.

STEP 6: If the flag remains 0, print that the element is not found

**(d)**

STEP 1: Prompt the user to input a list of elements and convert this list to a tuple.

STEP 2: Request the user to enter the number they wish to search for in the tuple.

STEP 3: Sort the tuple to ensure it is in ascending order, necessary for binary search.

STEP 4: Define the `search` function to implement the binary search:

- Use two pointers, `low` and `high`, to maintain the current search range.

- Calculate the middle index and compare the middle element with the target number.

- Adjust the `low` or `high` pointers based on the comparison to narrow the search range.

- Return `1` if the element is found.

STEP 5: Call the `search` function with initial values for `low` (0) and `high` (length of the tuple minus one).

STEP 6: Print the result based on the return value of the `search` function; print "Element is found" if the function returns `1`, otherwise print "Element is not found".

**(e)**

STEP 1: Prompt the user to input elements, which are then stored in a list and converted to a tuple `tup1`.

STEP 2: Initialize two empty lists, `lst2` for tracking unique elements and `lst3` for elements that appear only once.

STEP 3: Iterate over each element in `tup1`:

- If the element is not already in `lst2`, append it to `lst2` and use the `count` method of the tuple to print the number of occurrences of the element.

STEP 4: Iterate over elements in `lst2`:

- Use the `count` method on the original list `lst1` to check if any element appears exactly once.

- Append elements with exactly one occurrence to `lst3`.

STEP 5: Print `lst3`, which contains the list of elements that occur only once in the tuple.

**Program: (a)**

#bubble sort

lst=eval(input("Enter the elements :")) k=int(input("Enter the value of k:")) for j in range(len(lst)):

for i in range(len(lst)-1) :

if lst[i]>lst[i+1]: lst[i],lst[i+1]=lst[i+1],lst[i]

print("the sorted list is",lst) print("The",k,"largest element is ",lst[len(lst)-k])

**(b)**

#insertion sort

lst=eval(input("Enter the elements in the list")) for i in range(1,len(lst)):

key=lst[i]

j=i-1

while(j>=0 and lst[j]>key): lst[j+1]=lst[j]

j-=1

lst[j+1]=key

print("The sorted list is:",lst)

**©**

lst=eval(input("Enter the elements in the list:")) num=int(input("Enter the element to search:")) flag=0

for i in range(len(lst)): if lst[i]==num:

index=i flag=1 break

if(flag==1):

print("The element is found at",index,"position") else:

print("The element is not found")

**(d)**

def search(tup,low,high,num):

while (low<=high): middle=(low+high)//2 if tup[middle]==num:

return 1

elif( tup[middle]<num): low=middle+1

elif (num<tup[middle]): high=middle-1

lst=eval(input("Enter the elements:")) num=int(input("Enter the number to search:")) low=0

high=len(lst) lst.sort() tup=tuple(lst)

if(search(tup,low,high,num)): print("Element is found")

else:

print("Element is not found")

**(e)**

#using methods

lst1=eval(input("Enter the elements in the tuple:")) tup1=tuple(lst1)

lst2=[] lst3=[]

for element in tup1:

if element not in lst2: lst2.append(element)

print("count of",element,"is",tup1.count(element))

for i in lst2:

if lst1.count(i)==1: lst3.append(i)

print("list of elements with only one occurence:",lst3)

#without methods frequency={}

lst1=eval(input("Enter the elements in the list")) unique=[]

for i in lst1:

if i in frequency: frequency[i]+=1

else:

frequency[i]=1

for element,count in frequency.items(): if count==1:

unique.append(element)

for i ,j in frequency.items(): print("count of",i,"is",j)

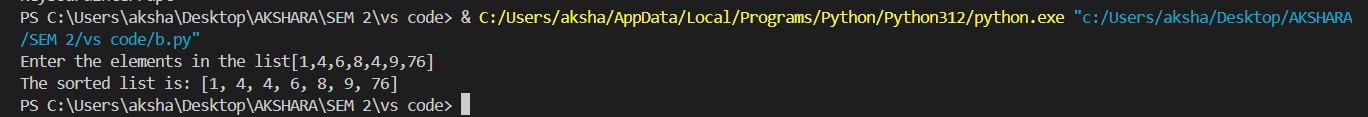
print("Numbers with unique occurence is",unique)

**Screenshot of Output:**

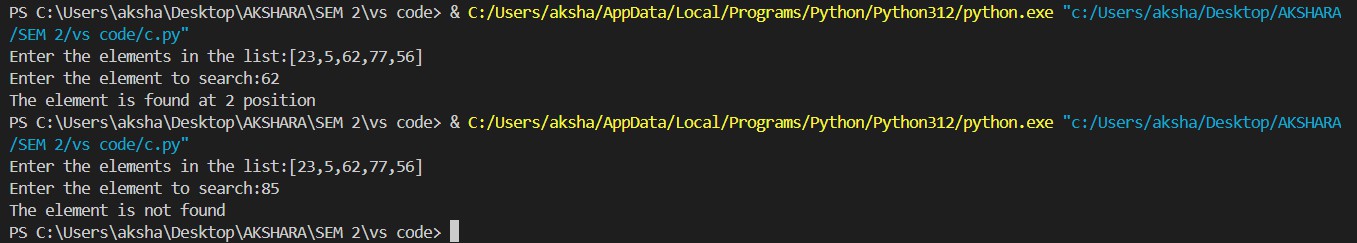
(a)



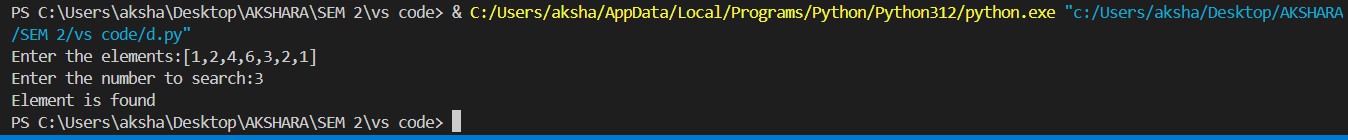
(b)



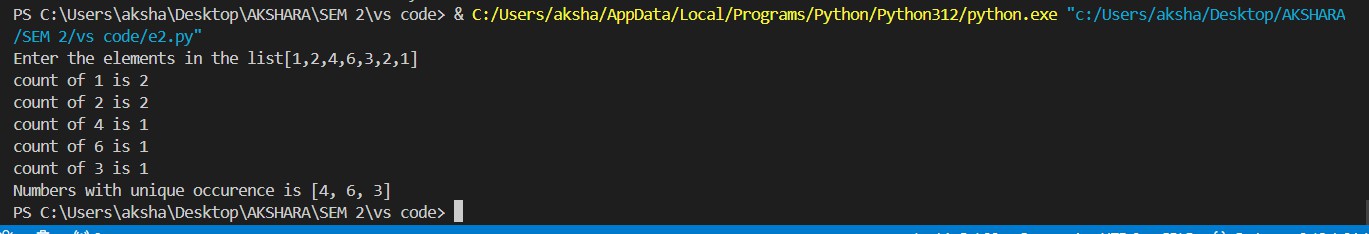
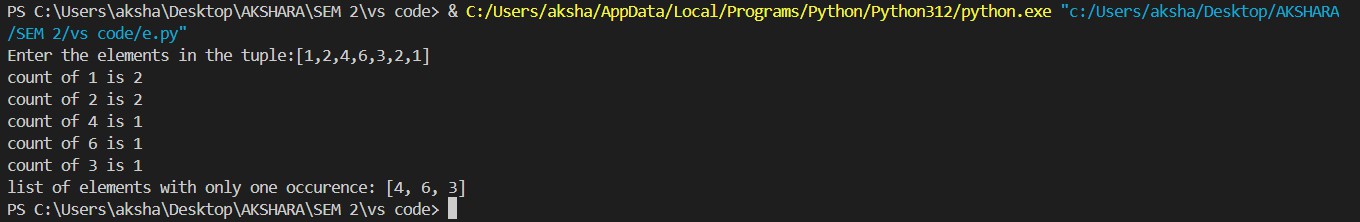
**(c)**



**(d)**



**(e)**



**Result:**

Thus, programs have been written and executed to explore lists and tuples in Python.