1.**Python program to Use and demonstrate basic data structures.**

```
A=10
Y=110
Z=120
B=20.0
C=”welcome to python data structure”
D=[1,2,3,4,5,]
E={“name”:”maruthi”, “age”:18}
F=(“apple”, “banana”, “cherry”)
Print(“integer:”,a)
Print(“float:”,b)
Print(“string:”,c)
Print(“list”,d)
Print(“dictinary:”,e)
Print(“tuple:”,f)
d.append(7)
print(“list:”,d)
q=len(d)
```

2. **Implement an ADT with all its operations.**

```
class date:
  def __init__(self,a,b,c):
    self.d=a
    self.m=b
    self.y=c
  def day(self):
    print(“Day=”,self.d)
  def month(self):
    print(“Month=”,self.m)
  def year(self):
```

```
        print("year=",self.y)
    def monthName(self)
months=("unknown","january","febuary","march","april","may","june","july","august","septmber","october
","november","december")
        print("month name:",months[self.m])
d1=date(3,8,2000)
d1.day()
d1.month()
d1.year()
d1.monthName()
print(q)
w=d.index(2)
print(w)
for I in e:
    print(i)
print(y>z)
z=len©
print(z)
```

3.**Implement an ADT and compute space and time complexities.**

```
start=time.time()
class stack:
    def __init__(self):
        self.items=[]
    def is_empty(self):
        return self.items==[]
    def push(self, items):
        self.items.append(time)
        print(items)
    def pop(self):
```

```
        return self.items.pop()
    def peek(self):
        return self.items[len(self.items)-1]
    def size(self):
        return len (self.items)
s=stack()
print(s.is_empty)
print("push")
s.push(11)
s.push(12)
s.push(13)
print("peek",s.peek)
print("pop")
print(s.pop())
print(s.pop())
print("sixe",s.size())
end=time.time()
print("run time program is:", end-start)
```

4.Implement above solution using array and compute space

Andtime complexities and compare two solutions.

```
import time
start=time.time()
a=[1,2,3,4]
a.insert(4,5)
print(a)
a.pop(0)
print(a)
len=len(a)
print("length of the list:", len)
```

```
end=time.time()
print("runtime of the program is: ",end-start)
```

5.Implement Linear Search compute space and time complexities,

Plot graph using asymptomatic notations.

```
import time
import numpy as np
import matplotlib.pyplot as plt
def linear_search(A,X):
       for I in range(0,len(A)):
          if A[i]==X:
             print("search is success at position:",i)
             return
       print("search is not success")
elements = np.array([i*1000 for I in range (1,10)])
plt.xlabel ('list length')
plt.ylabel ('time complexity')
times=list()
for I in range(1,10):
   start=time.time()
   a= np.random.randint(1000, size=i*1000)
   linear_search(a,1)
   end=time.time()
   times.append(end-start)
   print("time taken for linear search in ",i*1000,"elements is",end-start,"s")
plt.plot(elements,times,label="linear search")
plt.grid()
plt.legend()
plt.show()
```

6.Implement Bubble, Selection algorithms compute space and

Time complexities, plot graph using asymptomatic notations.

```
import time
from numpy.random import randint
import matplotlib.pyplot as plt
def selection_sort(array):
  length=len(array)
  for I in range (length -1):
    minIndex =i
    for j in range (length -1):
      if array[j]<array[minIndex]:
        minIndex=j
    temp=array[i]
    array[i]=array[minIndex]
    array [minIndex]=temp
    return array
elements=list()
times=list()
for I in range(1,10):
  a=randint(0,1000*I,1000*i)
  start=time.time()
  selection_sort(a)
  end=time.time()
  print(len(a), “elements sorted selection_sort in”, end-start)
  elements.append(len(a))
  times.append(end-start)
plt.xlabel(‘list length’)
plt.ylabel(‘time complexity’)
plt.plot(elements,times,label=”selection_sort”)
plt.grid()
```

```
plt.legend()
plt.show()
```

7.**Selection sort:**

```
import time
from numpy.random import seed
from numpy.random import randint
import matplotlib.pyplot as plt
def bubble_sort(list1):
    for I in range(0,len(list1)-1):
        for j in range(len(list1)-1):
            if(list1[j]>list1[j+1]):
                temp = list1[j]
                list1[j] = list1[j+1]
                list1[j+1] = temp
    return list1
elements=list()
times=list()
for I in range(1,10):
    a = randint(0,1000*I,1000*i)
    start=time.time()
    bubble_sort(a)
    end=time.time()
    print(len(a), “elements sorted by bubble sort in”,end-start)
    elements.append(len(a))
    times.append(end-start)
plt.xlabel(‘list length’)
plt.ylabel(‘time complexity’)
plt.plot(elements,times,label=’bubble sort’)
plt.grid()
```

```
plt.legend()

plt.show()
```

8.**Implement Binary Search using recursion Compute space and**

Time complexities, plot graph using asymptomatic notations

And compare two.

```
import time

import numpy as np

import matplotlib.pyplot as plt

def binary_search(arr,target):

   low,high=0,len(arr)

   while low<high:

      mid=(low+high)//2

      if target>arr[mid]:

         high=mid

      elif target>arr[mid]:

         low=mid+1

      else:

         return mid

   return -1

elements = np.array([i*1000 for I in range (1,10)])

plt.xlabel('list length')

plt.ylabel('time complexity')

times=list()

for I in range(1,10):

   start=time.time()

   a= np.random.randint(1000, size=i*1000)

   binary_search(a,1)

   end=time.time()

   times.append(end-start)
```

```
    print(“time taken for binarysearch in “,i*1000,
          “elements is”,end-start,”s”)
Plt.plot(elements,times,label=”binary search”)
Plt.grid()
Plt.legend()
Plt.show()
```