UCS1403 Design and Analysis of Algorithms
AY: 2021-22

Name: *Krithika Swaminathan*
Reg. no.: *205001057*

DAA Lab – Assignment 5

Binary Search and Primality Check using Divide and Conquer strategy

1. To Implement Binary search and analyze its time complexity.

Code:

```
#Python program to implement binary search

def binSearch(arr,low,high,key,comp):
      if (high>=low):
            mid = (low+high)//2
            comp += 1

            if (arr[mid]==key):
                  comp += 1
                  return mid, comp
            elif (arr[mid]<key):
                  comp += 2
                  return binSearch(arr,mid+1,high,key,comp)
            else:
                  comp += 2
                  return binSearch(arr,low,mid-1,key,comp)

      else:
            return -1, comp

arr = list(map(int,input("Enter array: ").split()))
key = int(input("Enter element to search for: "))
arr.sort()

print("\nArray: ",arr)
print("Key: ",key)
comp = 0
res, comp = binSearch(arr,0,len(arr)-1,key,comp)
print("Comparisons:",comp)

if (res==-1):
      print("Element not found in array")
else:
      print("Found at:",res)
```

Output:

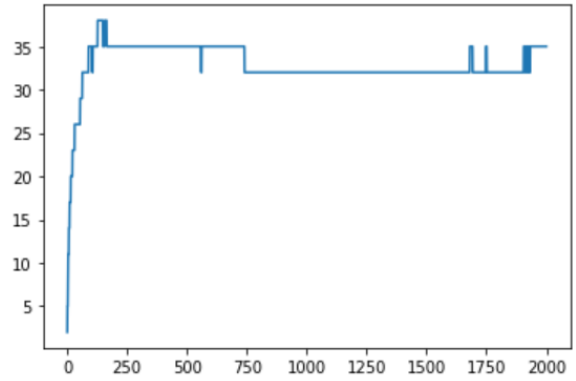
```
Enter array: 23 45 67 21 98
Enter element to search for: 98
Array: [21, 23, 45, 67, 98]
Key: 98
Comparisons: 8
Found at: 4
```

Name: *Krithika Swaminathan*
Reg. no.: *205001057*

Code:

```
#Python program to analyse the time complexity of binary search

import random
from matplotlib import pyplot as plt
import numpy as np

def binSearch(arr,low,high,key,comp):
      if (high>=low):
            mid = (low+high)//2
            comp += 1

            if (arr[mid]==key):
                  comp += 1
                  return mid, comp
            elif (arr[mid]<key):
                  comp += 2
                  return binSearch(arr,mid+1,high,key,comp)
            else:
                  comp += 2
                  return binSearch(arr,low,mid-1,key,comp)

      else:
            return -1, comp

arr = []
n = 2000
nums,comps = [],[]
for i in range(1,n+1):
  nums.append(i)
  for j in range(i):
    arr.append(random.randint(1,n))
  arr.sort()
  key = arr[0]
  comp = 0
  res, comp = binSearch(arr,0,len(arr)-1,key,comp)
  comps.append(comp)

xpoints = np.array(nums)
ypoints = np.array(comps)

plt.plot(xpoints,ypoints)
```

Output:



Name: *Krithika Swaminathan*
Reg. no.: *205001057*

2. To Implement Miller Rabin Algorithm.

Code:

```
#Python program to implement the Miller-Rabin algorithm for primality
testing

import random

def power(x, y, p):
      res = 1;
      x = x % p;
      while (y > 0):
            if (y & 1):
                  res = (res * x) % p;
            y = y>>1;
            x = (x * x) % p;

      return res;

def millerTest(d, n):
      a = 2 + random.randint(1, n - 4);
      x = power(a, d, n);

      if (x == 1 or x == n - 1):
            return True;

      while (d != n - 1):
            x = (x * x) % n;
            d *= 2;

            if (x == 1):
                  return False;
            if (x == n - 1):
                  return True;

      return False;

def isPrime( n, k):
      #base cases
      if (n <= 1 or n == 4):
            return False;
      if (n <= 3):
            return True;

      d = n - 1;
      while (d % 2 == 0):
            d //= 2;

      for i in range(k):
            if (millerTest(d, n) == False):
                  return False;

      return True;
```

Name: *Krithika Swaminathan*
Reg. no.: *205001057*

```
k = 2;
print("List of prime numbers lesser than 100: ");
for n in range(1,100):
      if (isPrime(n, k)):
            print(n , end=" ");
print()
```

Output:

```
~/DAA-Exercise5$ python3 miller.py
List of prime numbers lesser than 100:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
~/DAA-Exercise5$
```
