Algorithm Lab : Assignment - 1

> **Name** : Sandeep Shaw
> **Roll** : GCECTB-R19-3022
> **Dept** : Computer Science & Engineering
> **4th Semester, 2nd Year**

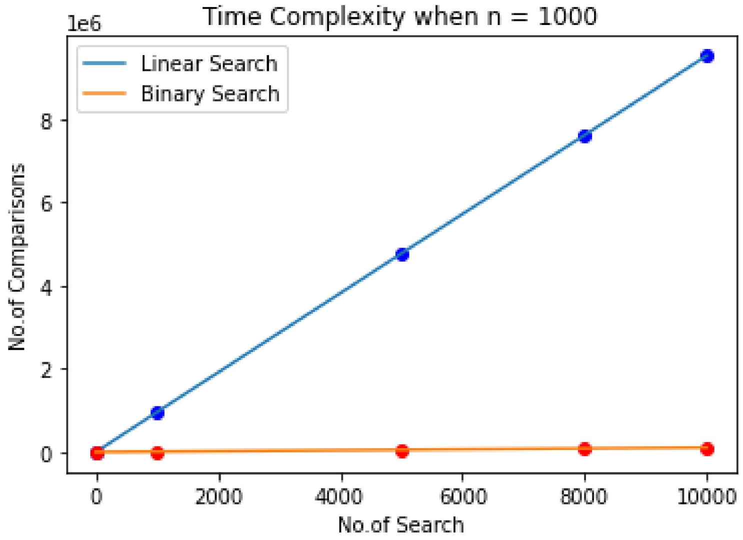
QUESTION : Write two search functions linear search and binary search.
Create a random array of size n, perform m searches on the array, where the element to be searched is also determined randomly. Calculate the no. of comparisons your function makes. Vary the value of m from m=1,1000, 5000, 8000, 10000. Plot the graph m vs no. of comparisons for both linear and binary search on the same graph. For binary search also consider the number of comparisons of your sorting algorithm on the array which you use only once. Make 3 search graphs for n=1000, 5000, 10000.

Solution

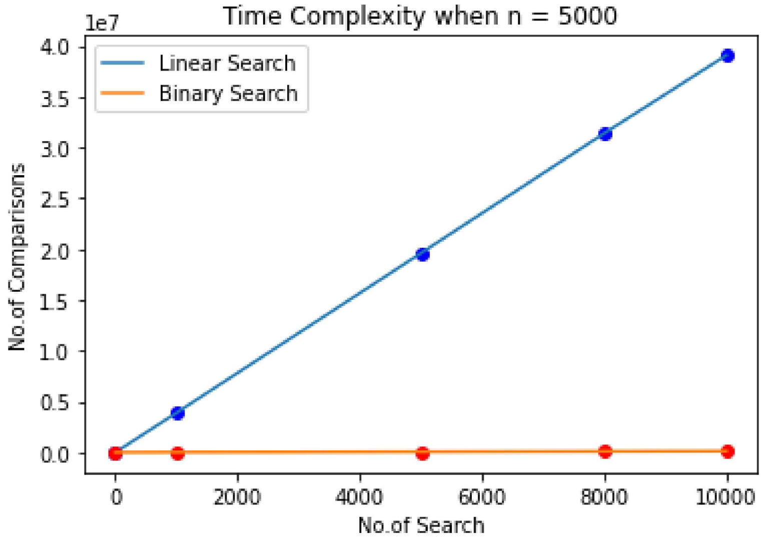
In [1]:

```
import numpy as np
import random
import matplotlib.pyplot as plt

class Search:

    #Constructor to initialise the values
    def __init__(self, n, limit):
        self.size = n
        self.searchArray = [1,1000, 5000, 8000, 10000]
        self.linearCounter = 0
        self.binaryCounter = 0
        self.linearData = [0, 0, 0, 0, 0]
        self.binaryData = [0, 0, 0, 0, 0]
        self.upperLimit = limit

        #generate Random Array using numpy
        self.generateRandomList()

        #execution function
        self.executeFunction()

    #Collect Data
    def executeFunction(self):
        for i in range(5):
            for j in range(0,self.searchArray[i]):
                key = self.generateKey()
                self.linearSearch(key)
                self.binarySearch(0, self.size-1, key)
            self.linearData[i] = self.linearCounter
            self.binaryData[i] = self.binaryCounter
            self.linearCounter = 0
            self.binaryCounter = 0

        self.plotData()
```

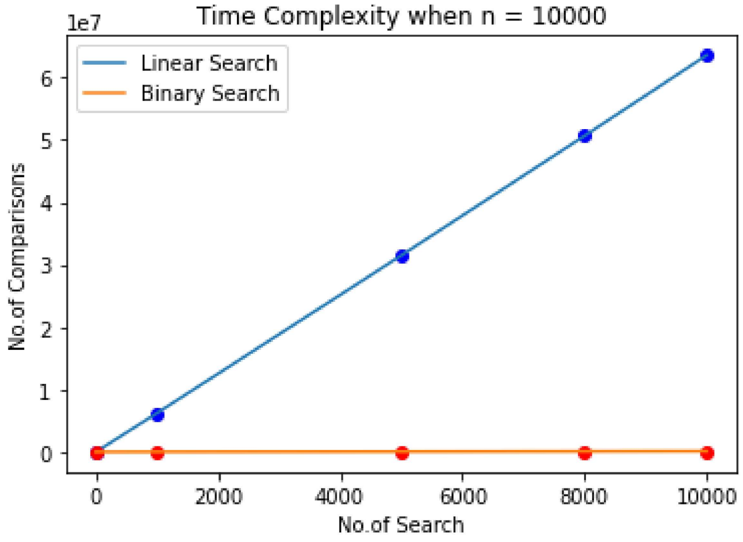
```python
    #Generate Random Key
    def generateKey(self):
        key = random.randint(0,self.upperLimit)
        return(key)

    #Generate Random Array
    def generateRandomList(self):
        self.array = np.random.randint(self.upperLimit, size=self.size)
        self.sortedArray = np.sort(self.array)
        self.generateKey()

    #Linear Search
    def linearSearch(self, key):
        for i in range(self.size):
            self.linearCounter += 1
            if(self.array[i] == key):
                break

    #Binary Search
    def binarySearch(self, l, r, key):
        self.binaryCounter += 1
        if (r >= l):
            mid = l + (r - l) // 2
            if (self.sortedArray[mid] == key):
               pass
            elif (self.sortedArray[mid] > key):
                return self.binarySearch(l, mid-1, key)
            else:
                return self.binarySearch(mid + 1, r, key)
        else:
            pass

    #Plotting the Graph
    def plotData(self):
        plt.plot(self.searchArray, self.linearData, "bo" )
        plt.plot(self.searchArray, self.linearData, label = "Linear Search" )

        plt.plot(self.searchArray, self.binaryData, "ro" )
        plt.plot(self.searchArray, self.binaryData, label = "Binary Search" )

        plt.xlabel('No.of Search')
        plt.ylabel('No.of Comparisons')
        plt.title(f'Time Complexity when n = {self.size}')
        plt.legend()
        plt.show()

        print("Linear Search : ",self.linearData)
        print("Binary Search : ",self.binaryData)


if(__name__ == "__main__"):
    graph1 = Search(1000, 10000)  # n = 1000   range (0,10000)
    graph2 = Search(5000, 10000)  # n = 5000   range (0,10000)
    graph3 = Search(10000, 10000) # n = 10000  range (0,10000)
```



```
Linear Search :  [1000, 958701, 4762729, 7605104, 9512427]
Binary Search :  [11, 10823, 53983, 86315, 107837]
```



```
Linear Search :  [291, 3885967, 19600280, 31399339, 39083952]
Binary Search :  [12, 12444, 62374, 99749, 124575]
```



```
Linear Search :  [6921, 6221614, 31475961, 50641265, 63556852]
Binary Search :  [13, 12752, 63830, 102142, 127955]
```

Searching Algorithm

1. Linear Search

```
def linearSearch(self, key):
 for i in range(self.size):
     self.linearCounter += 1
     if(self.array[i] == key):
         break
```

2. Binary Search

```
def binarySearch(self, l, r, key):
 self.binaryCounter += 1
 if (r >= l):
     mid = l + (r - l) // 2
     if (self.sortedArray[mid] == key):
        pass
     elif (self.sortedArray[mid] > key):
         return self.binarySearch(l, mid-1, key)
     else:
         return self.binarySearch(mid + 1, r, key)
 else:
     pass
```

Creating Random Array and Generating Random Key

1. Generate Random Key

```
def generateKey(self):
 key = random.randint(0,self.upperLimit)
 return(key)
```

2. Generate Random Array

```
def generateRandomList(self):
 self.array = np.random.randint(self.upperLimit, size=self.size)
 self.sortedArray = np.sort(self.array)
 self.generateKey()
```

Passing the Test Case n = 1000, 5000, 10000 to class Search

```
if(__name__ == "__main__"):
    graph1 = Search(1000, 10000)  # n = 1000   range (0,10000)
    graph2 = Search(5000, 10000)  # n = 5000   range (0,10000)
    graph3 = Search(10000, 10000) # n = 10000  range (0,10000)
```

Constructor and Execution Function

```
#Constructor to initialise the values
def __init__(self, n, limit):
    self.size = n
    self.searchArray = [1,1000, 5000, 8000, 10000]  # Value of Searching Case m
    self.linearCounter = 0
    self.binaryCounter = 0
    self.linearData = [0, 0, 0, 0, 0]
    self.binaryData = [0, 0, 0, 0, 0]
    self.upperLimit = limit
```

```
        #generate Random Array using numpy
        self.generateRandomList()

        #execution function
        self.executeFunction()
```