KATHMANDU UNIVERSITY

DHULIKHEL, KAVREPALANCHOWK, NEPAL

SUBJECT CODE: COMP 314

In the partial fulfilment of "Introduction to Algorithms"

Lab Report #1

<u>Submitted To:</u>

Rajani Chulyadyo
Department of Computer Science and Engineering (DoCSE)

<u>Submitted By:</u>

Kamal Shrestha
Roll No.:49
6th Semester, 3rd Year
B.E. Computer Engineering

Submission Date: 29th April 2019

Lab 1: Searching (Linear and Binary Search)

Objectives:

- Implementation and analysis of Linear and Binary Search Algorithms.
- Implementing test cases in the algorithm
- Generate some random inputs for your program and apply both linear and binary search algorithms to and a particular element on the generated input. Record the execution times of both algorithms for best and worst cases on inputs of different size (e.g. from 10000 to 100000 with step size as 10000).
- Plot an input-size vs execution-time graph.
- Binary Search
    - Implementation of binary search in an unsorted list
    - Implementation of binary search in a sorted list
- Linear Search:
    - Implementation of linear search in an unsorted list
    - Implementation of linear search in a sorted list

Introduction:

1. Linear Search:

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

Algorithm:

Linear Search (Array A, Values x)

Step 1: Set counter to 1
Step 2: if counter > n then go to step 7
Step 3: if A[counter] = x then go to step 6
Step 4: Set counter to counter + 1
Step 5: Go to Step 2-Loop
Step 6: Print Element x Found at index counter and go to step 8
Step 7: Print element not found
Step 8: Exit

Pseudo Code:

```
procedure linear_search (list, value)
   for each item in the list
                  if match item == value
                                    return the item's location
                            end if
   end for
end procedure
```

Time Complexity:

Worst case time complexity: **O(N)**
Average case time complexity: **O(N)**
Best case time complexity: **O(1)**

Space complexity: **O(1)**

2. Binary Search:

Binary search is a fast search algorithm with run-time complexity of O (log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.

Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues on the sub-array as well until the size of the subarray reduces to zero.

Algorithm:

1. Start with the middle element:
   - If the **target** value is equal to the middle element of the array, then return the index of the middle element.
   - If not, then compare the middle element with the target value,
     - If the target value is greater than the number in the middle index, then pick the elements to the right of the middle index, and start with Step 1.
     - If the target value is less than the number in the middle index, then pick the elements to the left of the middle index, and start with Step 1.
2. When a match is found, return the index of the element matched.
3. If no match is found, then return -1

Pseudo Code:

```
Procedure binary_search
  A ← sorted array
  n ← size of array
  x ← value to be searched

  Set lowerBound = 1
  Set upperBound = n

  while x not found
    if upperBound < lowerBound
      EXIT: x does not exists.

    set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

    if A[midPoint] < x
      set lowerBound = midPoint + 1
```

```
    if A[midPoint] > x
      set upperBound = midPoint - 1

    if A[midPoint] = x
      EXIT: x found at location midPoint
  end while

end procedure
```

Time Complexity:

Worst case time complexity: **O(logN)**
Average case time complexity: **O(logN)**
Best case time complexity: **O(1)**

Space complexity: **O(1)**

Generating random numbers and sorting them

```
import random
randomnum = random.sample(range(1000000),100000)
sortedrandomnum = sorted(randomnum)
```

Randomly picking any number from the list:

```
randomnumberfromthelist = random.choice(sortedrandomnum)
```

How to record the times of execution of an algorithm

```
from time import time
start time = time( ) # record the starting time
run algorithm
end time = time( ) # record the ending time
elapsed = end time − start time
```

Python Code:

Lab_LinearSearch.py

```python

# Algorithm to find the target value using binarySerach for an unsorted array of values

def linearSearch(theValues,target):
    n=len(theValues)
    for i in range(n):
        if theValues[i]==target:
            return True
    return False

# Algorithm to find the target value using linear Serach for an sorted array of values

def sortedLinearSearch(theValues,target):
    n=len(theValues)
    for i in range(n):
        if theValues[i]==target:
            return True
        elif theValues[i]>target:
            return False
    return False
```

Lab_BinarySearch.py

```python

# Algorithm to find the target value using linear Serach for an unsorted array of values
def binarySearch(theValues,target):
    low=0
    high=len(theValues)-1
    while low<=high:
        mid=(high+low)/2
        mid=int(mid)
        if theValues[mid]==target:
            return True
        elif target<theValues[mid]:
            high=mid-1
        else:
            low=mid+1
    return False

# Algorithm to find the target value using linear Search for an sorted array of values

def sortedbinarySearch(theValues,target):
    low=0
    high=len(theValues)-1
    while low<=high:
        mid=(high+low)/2
        mid=int(mid)
        if theValues[mid]==target:
            return True
        elif target<theValues[mid]:
            high=mid-1
        elif target >theValues[mid]:
            low=mid+1
        else:
            return False
    return False
```

Test_Cases.py

```python
import unittest
from Lab_LinearSearch import linearSearch, sortedLinearSearch
from Lab_BinarySearch import binarySearch, sortedbinarySearch

class SearchTestCase(unittest.TestCase):
    # test for linear Search

    def test_linearsearch(self):
        values=[5,3,6,1,2,9,0]
        self.assertEqual(linearSearch(values,5),True)
        self.assertEqual(linearSearch(values,1),True)
        self.assertEqual(linearSearch(values,7),False)

    def test_sortedLinearSearch(self):
        values=[5,3,6,1,2,9,0]
        values=sorted(values)
        self.assertEqual(linearSearch(values,5),True)
        self.assertEqual(linearSearch(values,1),True)
        self.assertEqual(linearSearch(values,7),False)

    def test_binarySearch(self):
        values=[5,3,6,1,2,9,0]

        self.assertEqual(linearSearch(values,5),True)
        self.assertEqual(linearSearch(values,1),True)
        self.assertEqual(linearSearch(values,7),False)

    def test_sortedbinearSearch(self):
        values=[5,3,6,1,2,9,0]
        values=sorted(values)
        self.assertEqual(linearSearch(values,5),True)
        self.assertEqual(linearSearch(values,1),True)
        self.assertEqual(linearSearch(values,7),False)

if __name__=='__main__':
    unittest.main()
```

Output:

```
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
[Finished in 1.4s]
```

Lab_Report.py

```
import random
from time import time
from Lab_LinearSearch import linearSearch, sortedLinearSearch
from Lab_BinarySearch import binarySearch, sortedbinarySearch


randomnum = random.sample(range(1000000),100000)
sortedrandomnum = sorted(randomnum)

target = random.choice(sortedrandomnum)

elapsed_unsorted=[]
elapsed_sorted=[]
elapsed_binary_unsorted=[]
elapsed_binary_sorted=[]

j=0
total_time=0



# Applying Algorithm for Unsorted Linear Search
for i in range(10000,100001,10000):
    start_time = time( )


    linearSearch(randomnum[0:i],target)

    end_time = time( )

    elapsed_unsorted.insert(j,end_time-start_time)
    j+=1


print ("\nUnsorted Linear Search times")
for i in range(10):
    print (elapsed_unsorted[i])
    total_time=total_time+elapsed_unsorted[i]
print("Total Time Taken:",total_time)



# Applying Algorithm for Sorted Linear Search
for i in range(10000,100001,10000):
    start_time = time( )

    sortedLinearSearch(sortedrandomnum[0:i],target)

    end_time = time( )

    elapsed_sorted.insert(j,end_time-start_time)
    j+=1

print ("\nSorted Linear Search times")
for i in range(10):
    print (elapsed_sorted[i])
    total_time=total_time+elapsed_sorted[i]
print("Total Time Taken:",total_time)

# Applying Algorithm for Unsorted Binary Search

for i in range(10000,100001,10000):
    start_time = time( )

    binarySearch(randomnum[0:i],target)

```

```python
    end_time = time( )

    elapsed_binary_unsorted.insert(j,end_time-start_time)
    j+=1

print ("\nBinary Unsorted Search times")
for i in range(10):
    print (elapsed_binary_unsorted[i])
    total_time=total_time+elapsed_binary_unsorted[i]
print("Total Time Taken:",total_time)

# Applying Algorithm for Sorted Binary Search
for i in range(10000,100001,10000):
    start_time = time( )

    sortedbinarySearch(sortedrandomnum[0:i],target)

    end_time = time( )

    elapsed_binary_sorted.insert(j,end_time-start_time)
    j+=1

print ("\nBinary Sorted Search times")
for i in range(10):
    print (elapsed_binary_sorted[i])
    total_time=total_time+elapsed_binary_sorted[i]
print("Total Time Taken:",total_time)
```

Link to Github: [KamalShrest]()

Output:

Unsorted Linear Search times
0.0029876232147216797
0.0069963932037353516
0.009009838104248047
0.01198434829711914
0.015002250671386719
0.017972946166992188
0.014010906219482422
0.015973806381225586
0.018002033233642578
0.015993118286132812
Total Time Taken: 0.12793326377868652

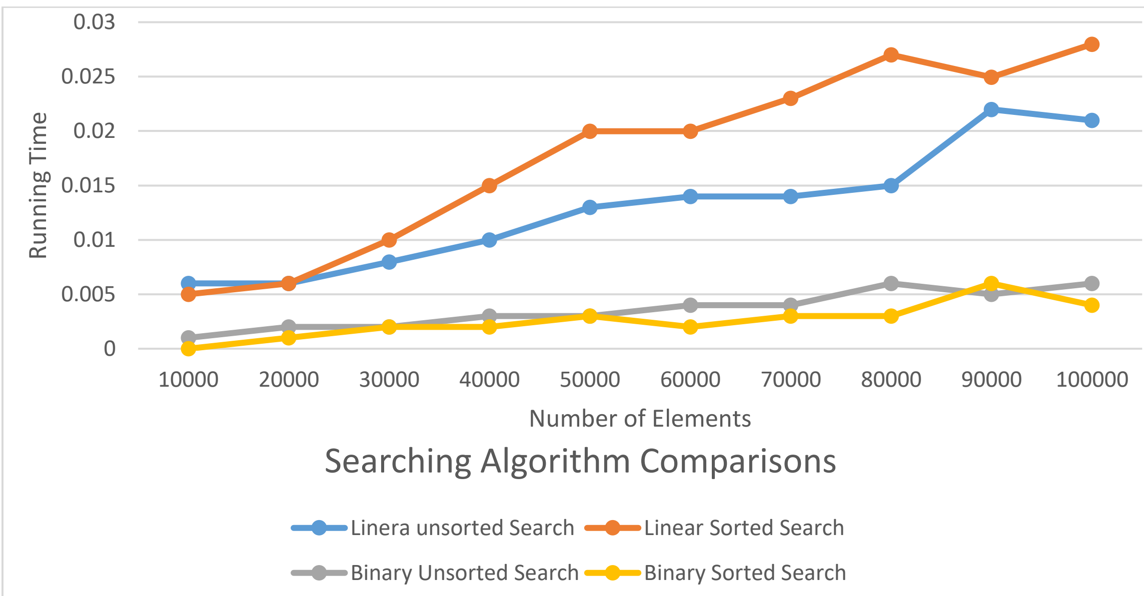
Sorted Linear Search times
0.0050106048583984375
0.005995988845825195
0.009993791580200195
0.014975547790527344
0.02100539207458496
0.01797175407409668
0.01500844955444336
0.01597428321838379
0.016989946365356445
0.016991376876831055
Total Time Taken: 0.267850399017334

Binary Unsorted Search times
0.0010001659393310547
0.0010013580322265625
0.0010159015655517578
0.0019860267639160156
0.0030095577239990234
0.0030074119567871094
0.0049991607666015625
0.003988742828369141
0.004979848861694336
0.006995677947998047
Total Time Taken: 0.2998342514038086

Binary Sorted Search times
0.0009980201721191406
0.0010197162628173828
0.0009784698486328125
0.002999544143676758
0.0030193328857421875
0.002991914749145508
0.0039806365966796875
0.0030210018157958984
0.0029845237731933594
0.003992319107055664
Total Time Taken: 0.325819730758667

Graph:

Input size	Linera unsorted Search	Linear Sorted Search	Binary Unsorted Search	Binary Sorted Search
10000	0.006003141	0.005001307	0.001000166	0
20000	0.006005526	0.005995035	0.001998663	0.000999928
30000	0.007978916	0.00999856	0.002002716	0.002001047
40000	0.009994507	0.014986277	0.002994299	0.001996756
50000	0.01299715	0.019989729	0.003004789	0.002999544
60000	0.01398921	0.019988775	0.00399828	0.002003431
70000	0.013992071	0.023001909	0.004002094	0.002994537
80000	0.014991999	0.027004242	0.005992889	0.002999544
90000	0.021986961	0.024951935	0.004998922	0.005992174
100000	0.020986557	0.027981758	0.005994081	0.00400281



Conclusion

Hence, from the graph we can see that search times correspond to O(1) and O(n) for linear search and O(1) and O(log n) for binary search.