Experiment No: 02

Experiment Name: Comparison between Linear Search and Binary Search (Complexity Analysis)

Linear Search

Theory/Complexity: Suppose there are n elements: $a_1, a_2, ..., a_{n-1}, a_n$. It is required to search the element from the elements.

There may arise two cases:
(1) The element s is in n distinct indices from 0 to n-1.
(2) The element s may not be present in the list.

There are in total n+1 distinct case to consider in total. If element s is in index k, then Linear Search will do k+1 comparison. Number of comparisons for all case in case 1 can be defined as,

Comparisons for index 0 + Comparisons for index 1+ ... + Comparisons for index n-1

= 1 + 2 + 3 + 4 + ... + n

= $\mathbf{n*(n+1)/2}$ comparisons.

If element s in not present in the list, then there will be n comparisons. Number of comparisons for all cases in the 2nd case = n.

For Worst case the element will be present at the last index.
The **Average case** time complexity will be **O(n).**
For **Best case** the time complexity will be **O(1)**.
The **Worst case** complexity will be **O(n).**

Implementation:

```
#include <bits/stdc++.h>
#include <cstdlib>
#define ll long long int
using namespace std;

ll _linearSearch(ll arr[], ll n, ll key) {
    for (ll i = 1; i < n; i++)
        if(arr[i]==key) {return i ;}
    return -1;
}

ll _linearSearchRecursive(ll arr[], ll n, ll key) {
    n = n-1;
    if(n<0) return -1;
    if(arr[n]==key) {return n ;}
    return _linearSearchRecursive(arr, n ,key);}

int main(){
    #ifndef WORK_STATION
    freopen("in.txt","r",stdin);
    #endif
    ll n; cin >> n; ll arr[n+1];
    for(ll i=0; i<n; ++i) arr[i] = rand();
    ll key = rand();

    // Linear part
    ll res1 = _linearSearch(arr,n,key);
    (res1==-1)?cout<<"Key Not Found\n":cout<<"Key Found\n";
    // Recursion part
    ll res2 = _linearSearchRecursive(arr,n,key);
    (res2==-1)?cout<<"Key Not Found\n":cout<<"Key Found\n";
}
```

Input Output:

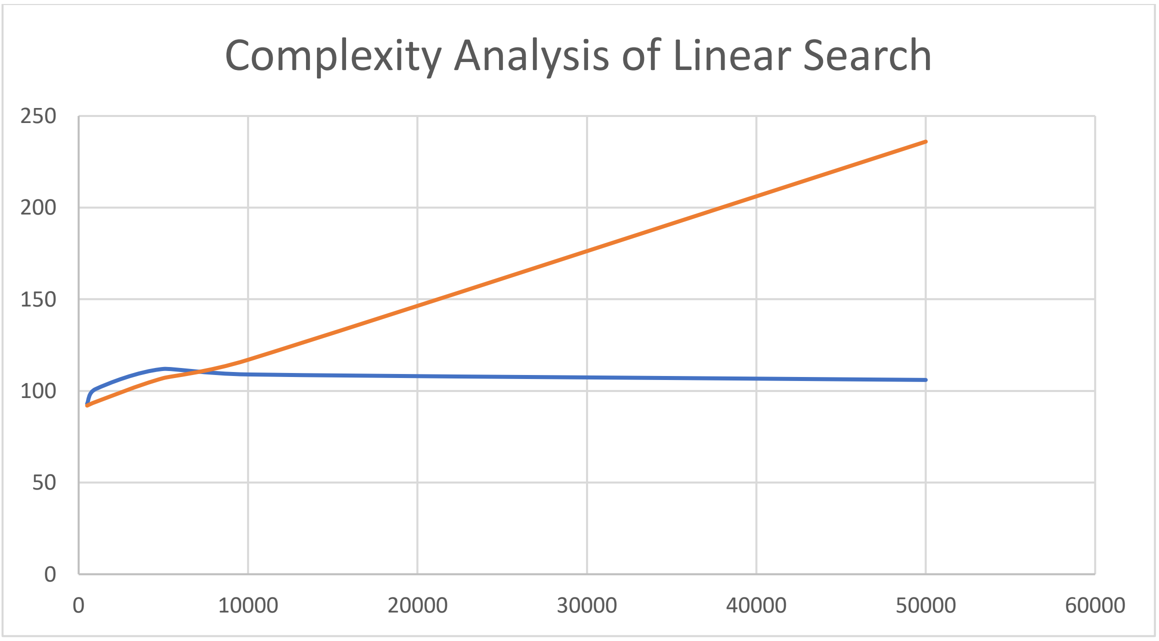
Time Taken by Iterative Process:

Input (Size)	**Output (milliseconds)**
500	93
1000	101
5000	112
10000	109
50000	106

Time Taken by Recursive Process:

Input (Size)	**Output (milliseconds)**
500	92
1000	94
5000	107
10000	117
50000	236

Output Analysis: Orange Recursive and Blue iterative



Binary Search

Theory: Suppose there are n elements: a_1, a_2, ..., a_{n-1}, a_n. It is required to search the element from the elements. There may arise two cases:
(1) The element s is in n distinct indices from 0 to n-1.
(2) The element s may not be present in the list.
There are in total n+1 distinct case to consider in total. If element s is in index k, then Linear Search will do k+1 comparison.

The element at index n/2 can be found in one comparison as binary search starts from the middle.

At first Iteration length of array = n
At second Iteration length of array = n/2
At third Iteration length of array = (n/2)/2 = $n/2^2$
….   ….

….   ….

Therefore, after kth Iteration length of array = $n/2^k$
Suppose, after k iterations, the length of the array becomes 1.
Therefore, the Length of the array = 1

=> $n/2^k = 1$

=> $n = 2^k$

=> $\log_2 n = \log_2 2^k$ [Applying log on both sides]

=> $\log_2 n = k * \log_2 2$

As, ($\log_a a = 1$) Therefore, $k = \log_2(n)$

For Best case, when the element is in the middle index, Complexity = O(1)
For worst case, when the element is in the first or last index, Complexity= O(n)
The overall average complexity is $O(\log_2 n)$.
For recursion, the complexity is still the same.

Implementation:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;

int _mid (int x,int y){
    return ((x+y) / 2);
}

int _binarySearchLinear(int arr[], int start ,int end ,int key) {
    int mid;
    while (start <= end){
        mid = _mid(start,end);

        if(arr[mid] == key) return mid;

        if(arr[mid] > key) end = mid-1;
        else start = mid+1;
    }
    return -1;
}

int _binarySearchRecursive(int arr[], int start ,int end ,int key) {
    if(start > end) return -1;
    int mid = _mid(start,end);
    if(arr[mid] == key) return mid;

    if(arr[mid] > key) return _binarySearchRecursive(arr,start,mid-1,key);
    else return _binarySearchRecursive(arr,mid+1,end,key);
}

int main(){
    #ifndef WORK_STATION
    freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    #endif

    // Input Part
    int n; cin >> n; int arr[n+1];
    for(int i=0; i<n; ++i) cin >> arr[i];
    int key = rand();

    // Linear Part
    int res1 = _ binarySearchLinear (arr,0,n-1,key);
    (res1==-1)?cout<<"Key Not Found\n":cout<<"Key "<< key <<" Found at " << res1
<<"\n";

    // Recursive Part
    int res2 = _binarySearchRecursive(arr,0,n-1,key);
    (res2==-1)?cout<<"Key Not Found\n":cout<<"Key "<< key <<" Found at " << res2
<<"\n";
}
/*
 @_author
 Md. Shefat Zeon
 1903147,CSE,RUET
*/
```

Input Output:

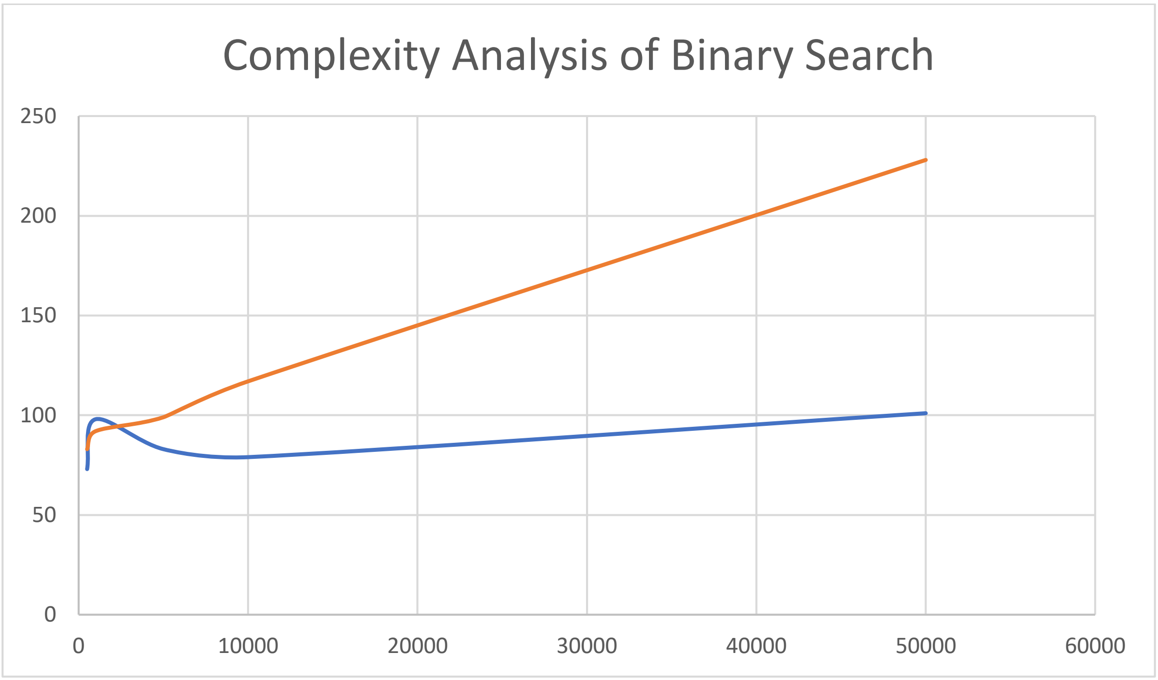
Time Taken by Iterative Process:

Input (Size)	**Output (milliseconds)**
500	73
1000	98
5000	83
10000	79
50000	101

Time Taken by Recursive Process:

Input (Size)	**Output (milliseconds)**
500	83
1000	92
5000	99
10000	117
50000	228

Output Analysis: Orange Recursive and Blue iterative

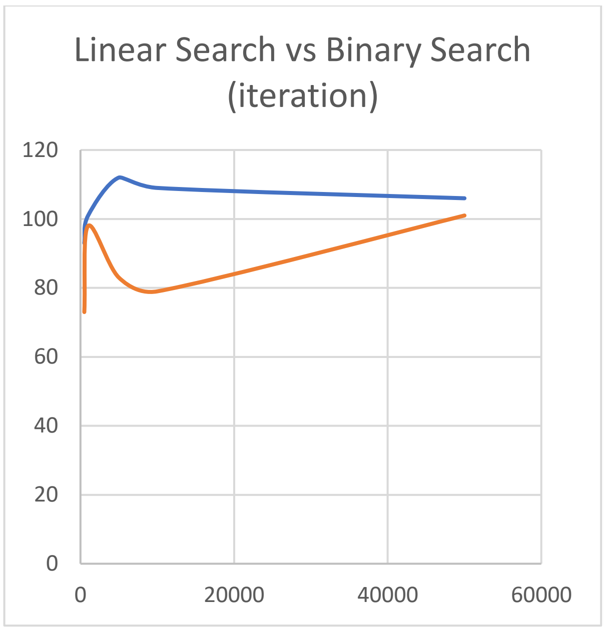


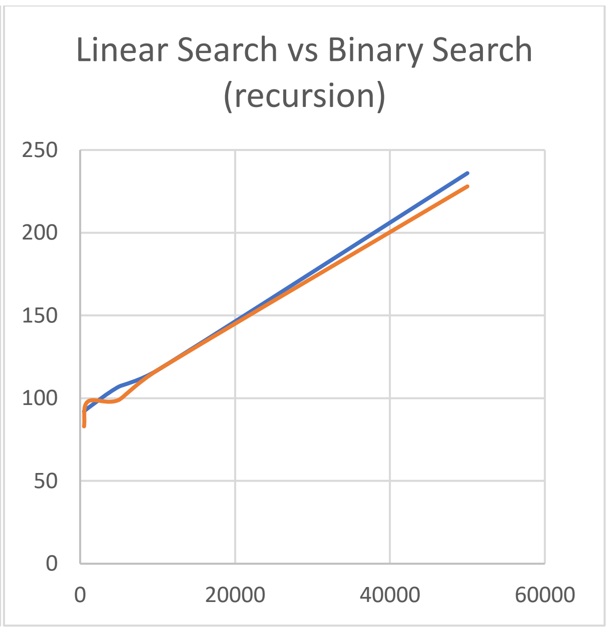
Comparison Between Linear Search and Binary Search:

Runtime for different values:

No of Inputs	Time taken by Algorithms (milliseconds)			
	Linear Search		Binary Search	
	Iterative	Recursion	Iterative	Recursion
500	93	92	73	83
1000	101	94	98	92
5000	112	107	83	99
10000	109	117	79	117
50000	106	236	101	228

Graph: Orange Represents Binary Search & Blue Represents Linear Search





Analysis:

Here we can see, both linear and binary search algorithms can be useful depending on their application.
For Binary Search, if the array's elements are arranged in sorted order, then binary search is preferred for searching, that means the search can be done only when array is sorted.
On the other hand, the linear search can search using trivial process to search the element. It does not need to be a sorted array. But for sorted list array when the size is big (n = 20000, 50000, ... etc.) the time gets more for linear search which takes a lot of time for the machine to search the desired element. Whereas, for binary search, the time is relatively lower (as shown in the graph). Therefore, we can conclude that binary search is more efficient for sorted arrays, otherwise linear search has the faster execution time.