



# EXPERIMENT - 2

## Algorithms Design and Analysis Lab

### Aim

To implement Linear search and Binary search and analyse its time complexity.

Syeda Reeha Quasar

14114802719

4C7

## EXPERIMENT – 2

### Aim:

To implement Linear search and Binary search and analyse its time complexity.

### Theory:

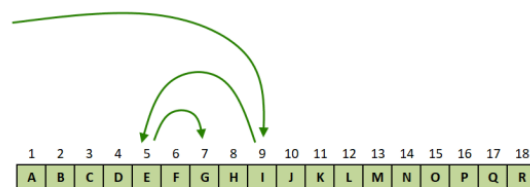
The search operation can be done in the following two ways:

#### Linear search:

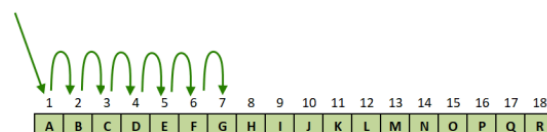
It's a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

#### Binary search:

This search algorithm works on the principle of divide and conquers. For this algorithm, 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 sub array reduces to zero.



Binary Search - Find 'G' in sorted list A-R



Linear Search - Find 'G' in sorted list A-R

**Pseudo code for linear search:**

```
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
```

**Pseudo code for Binary search:**






```
BinarySearch (A [0...N-1], value)
{
  low = 0
  high = N - 1
  while (low <= high) {
    // invariants: value > A[i] for all i < low
    //                value < A[i] for all i > high
    mid = (low + high) / 2
    if (A[mid] > value)
      high = mid - 1
    else if (A[mid] < value)
      low = mid + 1
    else
      return mid
  }
  return not_found // value would be inserted at index "low"
}
```

**Sample Example:****Example of Linear Search**

List : 10, 6, 222, 44, 55, 77, 24

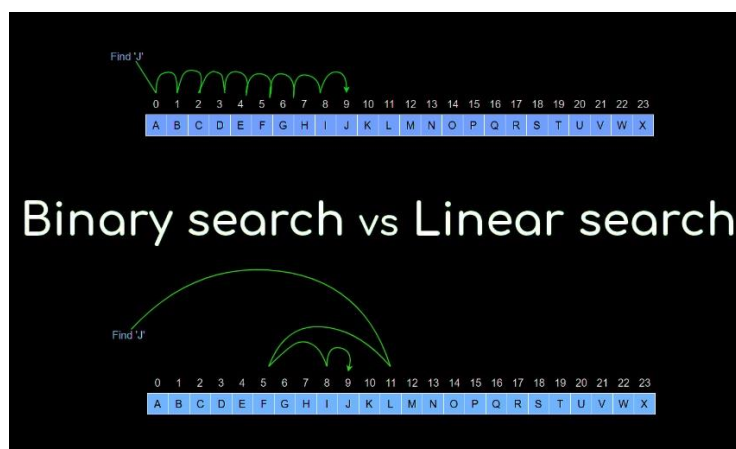
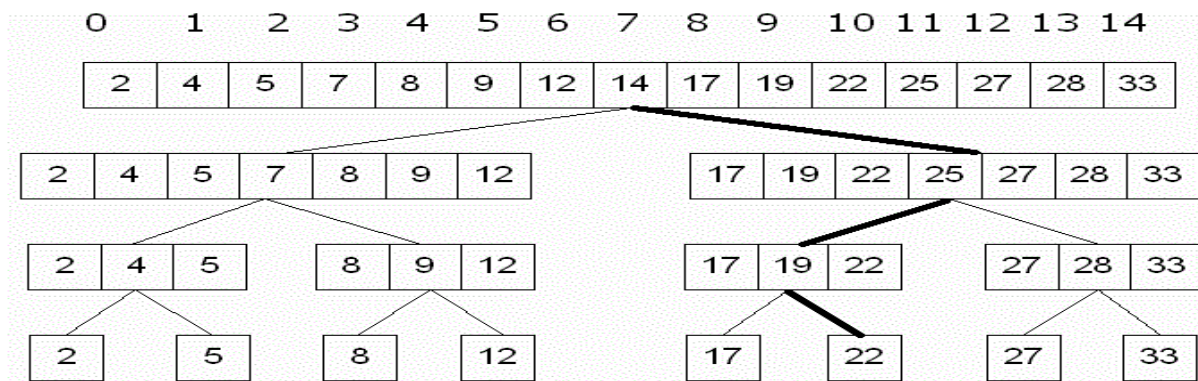
Key: 55

**Steps:**

1.   $\rightarrow K=1$
2.   $\rightarrow K=2$
3.   $\rightarrow K=3$
4.   $\rightarrow K=4$
5.   $\rightarrow K=5$

**So, Item 55 is in position 5.****Example of Binary Search:**

A sorted array is taken as input. The mid value is found out recursively.



## Result and Analysis

Input data needs to be sorted in Binary Search and not in Linear Search and it does the sequential access whereas Binary search access data randomly. So, time complexity of linear search is  $O(n)$  while Binary search has time complexity  $O(\log n)$  as search is done to either half of the given list.

Base of comparison	Linear search	Binary search
Time complexity	$O(N)$	$O(\log_2 N)$
Best case time	$O(1)$ first element	$O(1)$ center element
Prerequisite of an array	No prerequisite	Array must be sorted in order
Input data	No need to be sorted	Need to be sorted
Access	Sequential	random

LINEAR SEARCH	BINARY SEARCH
An algorithm to find an element in a list by sequentially checking the elements of the list until finding the matching element	An algorithm that finds the position of a target value within a sorted array
Also called sequential search	Also called half-interval search and logarithmic search
Time complexity is $O(N)$	Time complexity is $O(\log_2 N)$
Best case is to find the element in the first position	Best case is to find the element in the middle position
It is not required to sort the array before searching the element	It is necessary to sort the array before searching the element
Less efficient	More efficient
Less complex	More complex

# Linear search:

## Source Code:

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

int linearSearch(int arr[], int n, int x)
{
    for (int i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    // int arr[] = { 2, 3, 4, 10, 40 };
    // int n = sizeof(arr) / sizeof(arr[0]);

    int n = rand() % 100;
    int arr[n];

    for (int i = 0; i < n; i++)
    {
        arr[i] = rand() % 100;
    }

    cout << "Array: ";
    printArray(arr, n);

    int x;

    cout << "Enter element you want to search in the array: ";
```

```
cin >> x;

auto start = chrono::high_resolution_clock::now();
// unsync the I/O of C and C++.
ios_base::sync_with_stdio(false);

int result = linearSearch(arr, n, x);

auto end = chrono::high_resolution_clock::now();

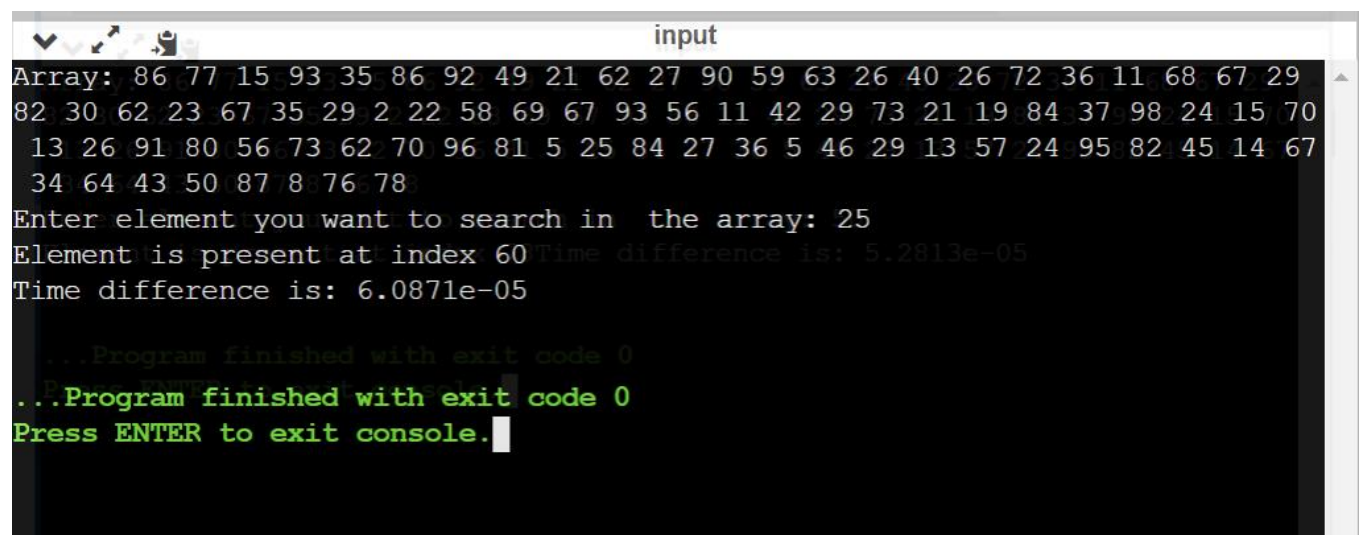
(result == -1)
    ? cout << "Element is not present in array"
    : cout << "Element is present at index " << result;

double time_taken = chrono::duration_cast<chrono::nanoseconds>(end -
start).count();
time_taken *= 1e-9;

cout << "Time difference is: " << time_taken << setprecision(6) << endl;

return 0;
}
```

## Output:



```
input
Array: 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 68 67 29
82 30 62 23 67 35 29 2 22 58 69 67 93 56 11 42 29 73 21 19 84 37 98 24 15 70
13 26 91 80 56 73 62 70 96 81 5 25 84 27 36 5 46 29 13 57 24 95 82 45 14 67
34 64 43 50 87 8 76 78
Enter element you want to search in the array: 25
Element is present at index 60 Time difference is: 5.2813e-05
Time difference is: 6.0871e-05
...Program finished with exit code 0
...Program finished with exit code 0
Press ENTER to exit console.
```

```
input
Array: 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 68 67 29
82 30 62 23 67 35 29 2 22 58 69 67 93 56 11 42 29 73 21 19 84 37 98 24 15 70
13 26 91 80 56 73 62 70 96 81 5 25 84 27 36 5 46 29 13 57 24 95 82 45 14 67
34 64 43 50 87 8 76 78
Enter element you want to search in the array: 0
Element is not present in array
Time difference is: 4.7726e-05

...Program finished with exit code 0
Press ENTER to exit console.
```

```
input
Array: 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 68 67 29
82 30 62 23 67 35 29 2 22 58 69 67 93 56 11 42 29 73 21 19 84 37 98 24 15 70
13 26 91 80 56 73 62 70 96 81 5 25 84 27 36 5 46 29 13 57 24 95 82 45 14 67
34 64 43 50 87 8 76 78
Enter element you want to search in the array: 12
Element is not present in array
Time difference is: 6.6163e-05

...Program finished with exit code 0
Press ENTER to exit console.
```



## Batch Analysis:

### Source Code:

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

int linearSearch(int arr[], int n, int x)
{
    for (int i = 0; i < n; i++)
        if (arr[i] == x)
            return i;

    return -1;
}

void printArray(double arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << ", ";

    cout << endl;
}

void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << ", ";

    cout << endl;
}

void printArray(float arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << ", ";

    cout << endl;
}

double search(int n)
{
    int arr[n];
```

```
    for (int i = 0; i < n; i++)
    {
        arr[i] = rand() % 100;
    }
    sort(arr, arr + n);

    int x = rand() % 100;

    auto start = chrono::high_resolution_clock::now();
    ios_base::sync_with_stdio(false);

    linearSearch(arr, n, x);

    auto end = chrono::high_resolution_clock::now();

    double time_taken = chrono::duration_cast<chrono::nanoseconds>(end -
start).count();
    time_taken *= 1e-9;

    return time_taken;
}

int main()
{
    double times[10];
    int ns[10];

    for (int x = 0; x < 10; x++)
    {
        int n = rand() % 100;
        ns[x] = n;
        times[x] = search(n);
    }

    cout << "value of n's: " << endl;
    printArray(ns, 10);

    cout << "time for each n: " << endl;
    printArray(times, 10);
}
```

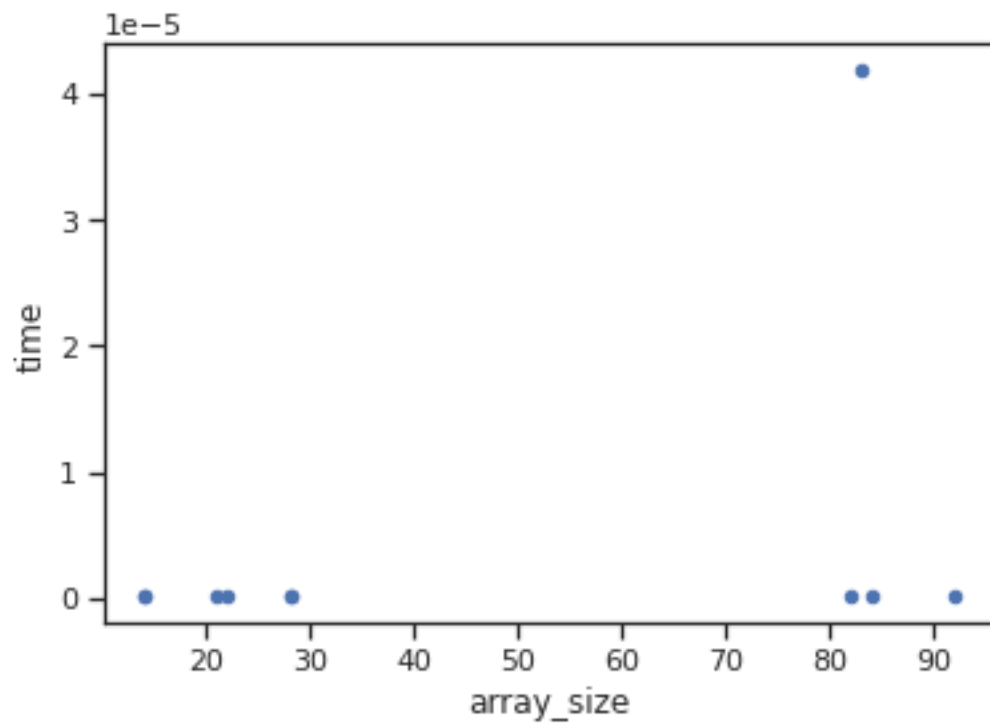
**Output:**

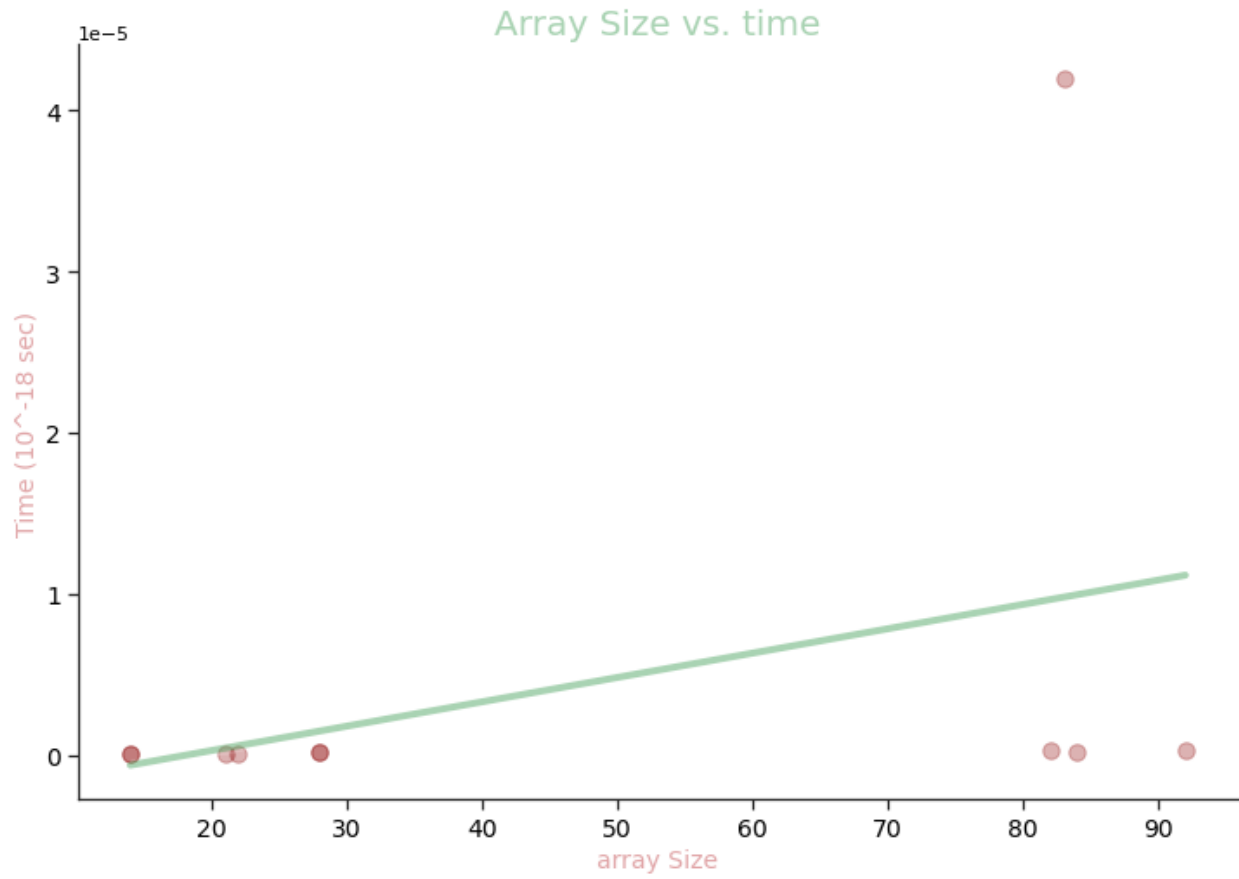
```
input
value of n's:
83, 84, 14, 21, 28, 22, 82, 92, 14, 28,
time for each n:
3.1241e-05, 1.35e-07, 1.01e-07, 9.3e-08, 1.43e-07, 1.14e-07, 2.44e-07, 2.25e-07, 5.8e-08, 1.33e-07,

...Program finished with exit code 0
Press ENTER to exit console.
```

```
arraySize = [83, 84, 14, 21, 28, 22, 82, 92, 14, 28]
```

```
time = [4.1967e-05, 1.51e-07, 1.03e-07, 1.02e-07, 1.65e-07, 1.46e-07, 2.93e-07, 2.79e-07, 1.04e-07, 1.68e-07]
```





**Linear Search**

# Binary search:

## Source Code:

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

int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l)
    {
        int mid = l + (r - l) / 2;

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

        else if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        return binarySearch(arr, mid + 1, r, x);
    }
    return -1;
}

int binarySearchIter(int arr[], int l, int r, int x)
{
    while (l <= r)
    {
        int m = l + (r - l) / 2;

        if (arr[m] == x)
            return m;

        if (arr[m] < x)
            l = m + 1;

        else
            r = m - 1;
    }
    return -1;
}
```

```
void printArray(int arr[], int size)
{
    int i;

    for (i = 0; i < size; i++)
        cout << arr[i] << " ";

    cout << endl;
}

int main()
{
    // int arr[] = { 2, 3, 4, 10, 40 };
    // int n = sizeof(arr) / sizeof(arr[0]);

    int n = rand() % 100;
    int arr[n];

    for (int i = 0; i < n; i++)
    {
        arr[i] = rand() % 100;
    }

    sort(arr, arr + n);

    cout << "Array: ";
    printArray(arr, n);

    int x;

    cout << "Enter element you want to search in the array: ";
    cin >> x;

    cout << "\n Recursive Binary Search" << endl;

    auto start = chrono::high_resolution_clock::now();
    // unsync the I/O of C and C++.
    ios_base::sync_with_stdio(false);

    int result = binarySearch(arr, 0, n, x);

    auto end = chrono::high_resolution_clock::now();

    (result == -1)
        ? cout << "Element is not present in array" << endl
```

```
        : cout << "Element is present at index " << result << endl;

    double time_taken = chrono::duration_cast<chrono::nanoseconds>(end -
start).count();
    time_taken *= 1e-9;

    cout << "Time difference is: " << time_taken << setprecision(6) << endl;

    cout << "\n Iterative Binary Search" << endl;

    start = chrono::high_resolution_clock::now();

    int resultIter = binarySearchIter(arr, 0, n, x);

    end = chrono::high_resolution_clock::now();

    (resultIter == -1)
        ? cout << "Element is not present in array" << endl
        : cout << "Element is present at index " << resultIter << endl;

    time_taken = chrono::duration_cast<chrono::nanoseconds>(end - start).count();
    time_taken *= 1e-9;

    cout << "Time difference is: " << time_taken << setprecision(6) << endl;

    return 0;
}
```

**Output:**

```
input
Array: 2 5 5 8 11 11 13 13 14 15 15 19 21 21 22 23 24 24 25 26 26 26 27 27 2
9 29 29 29 30 34 35 35 36 36 37 40 42 43 45 46 49 50 56 56 57 58 59 62 62 62
63 64 67 67 67 67 68 69 70 70 72 73 73 76 77 78 80 81 82 82 84 84 86 86 87
90 91 92 93 93 95 96 98
Enter element you want to search in the array: 43

Recursive Binary Search
Element is present at index 37
Time difference is: 6.3252e-05

Iterative Binary Search
Element is present at index 37
Time difference is: 2.71e-07

...Program finished with exit code 0
Press ENTER to exit console.
```

```
input
Array: 2 5 5 8 11 11 13 13 14 15 15 19 21 21 22 23 24 24 25 26 26 26 27 27 2
9 29 29 29 30 34 35 35 36 36 37 40 42 43 45 46 49 50 56 56 57 58 59 62 62 62
63 64 67 67 67 67 68 69 70 70 72 73 73 76 77 78 80 81 82 82 84 84 86 86 87
90 91 92 93 93 95 96 98
Enter element you want to search in the array: 0

Recursive Binary Search
Element is not present in array
Time difference is: 5.0812e-05

Iterative Binary Search
Element is not present in array
Time difference is: 1.85e-07

...Program finished with exit code 0
Press ENTER to exit console.
```



## Batch Analysis:

### Source Code:

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

int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l)
    {
        int mid = l + (r - l) / 2;

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

        else if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        return binarySearch(arr, mid + 1, r, x);
    }
    return -1;
}

int binarySearchIter(int arr[], int l, int r, int x)
{
    while (l <= r)
    {
        int m = l + (r - l) / 2;
        if (arr[m] == x)
```

```
        return m;

    if (arr[m] < x)
        l = m + 1;

    else
        r = m - 1;
}
return -1;
}

void printArray(double arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << ", ";

    cout << endl;
}

void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << ", ";

    cout << endl;
}

void printArray(float arr[], int size)
{
    for (int i = 0; i < size; i++)
```

```
        cout << arr[i] << ", ";

    cout << endl;
}

double search(int n)
{
    int arr[n];
    for (int i = 0; i < n; i++)
    {
        arr[i] = rand() % 100;
    }
    sort(arr, arr + n);

    int x = rand() % 100;

    auto start = chrono::high_resolution_clock::now();
    ios_base::sync_with_stdio(false);

    binarySearch(arr, 0, n, x);

    auto end = chrono::high_resolution_clock::now();

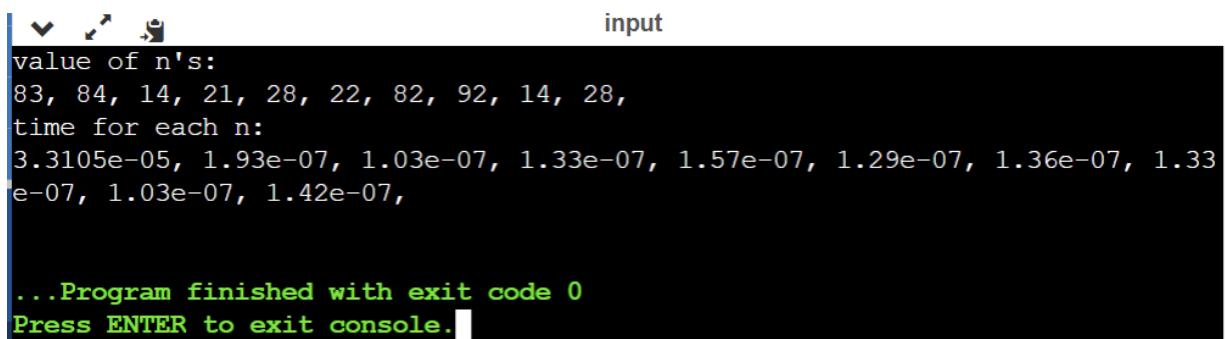
    double time_taken = chrono::duration_cast<chrono::nanoseconds>(end -
start).count();
    time_taken *= 1e-9;

    return time_taken;
}

int main()
```

```
{  
    double times[10];  
    int ns[10];  
  
    for (int x = 0; x < 10; x++)  
    {  
        int n = rand() % 100;  
        ns[x] = n;  
        times[x] = search(n);  
    }  
  
    cout << "value of n's: " << endl;  
    printArray(ns, 10);  
  
    cout << "time for each n: " << endl;  
    printArray(times, 10);  
}
```

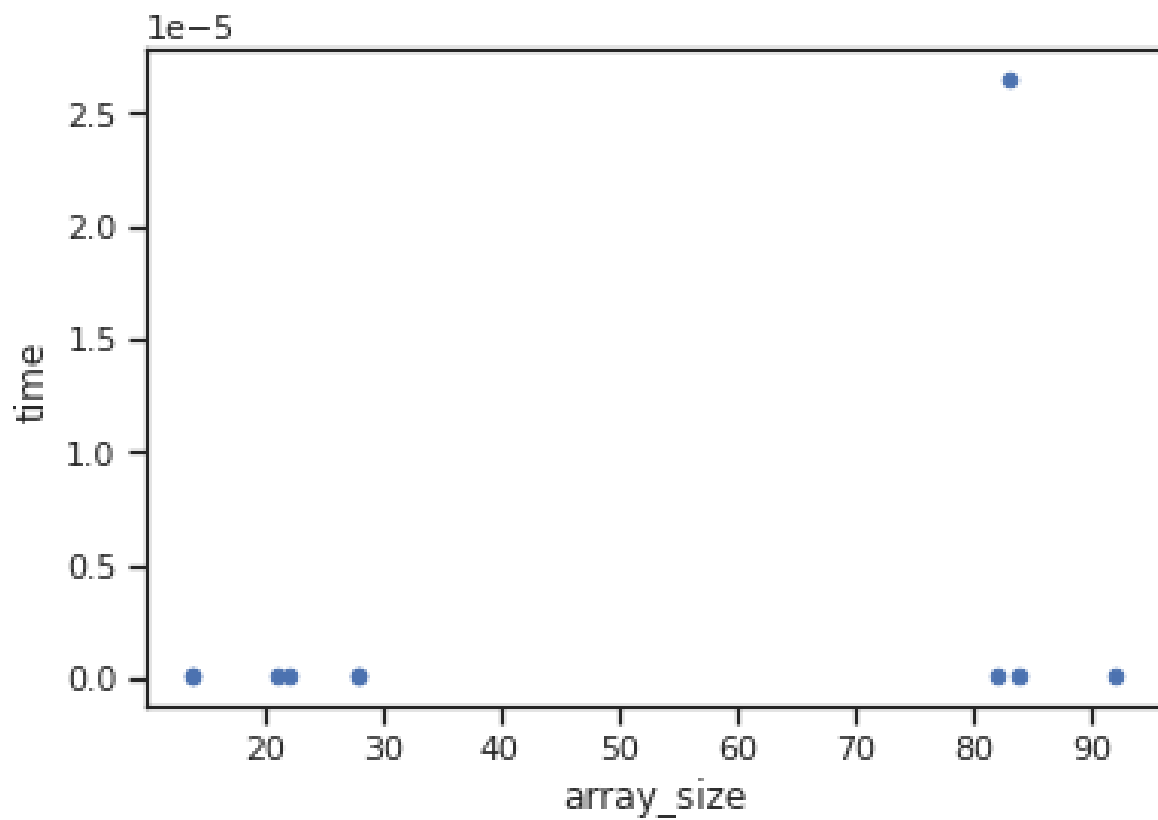
## Output:

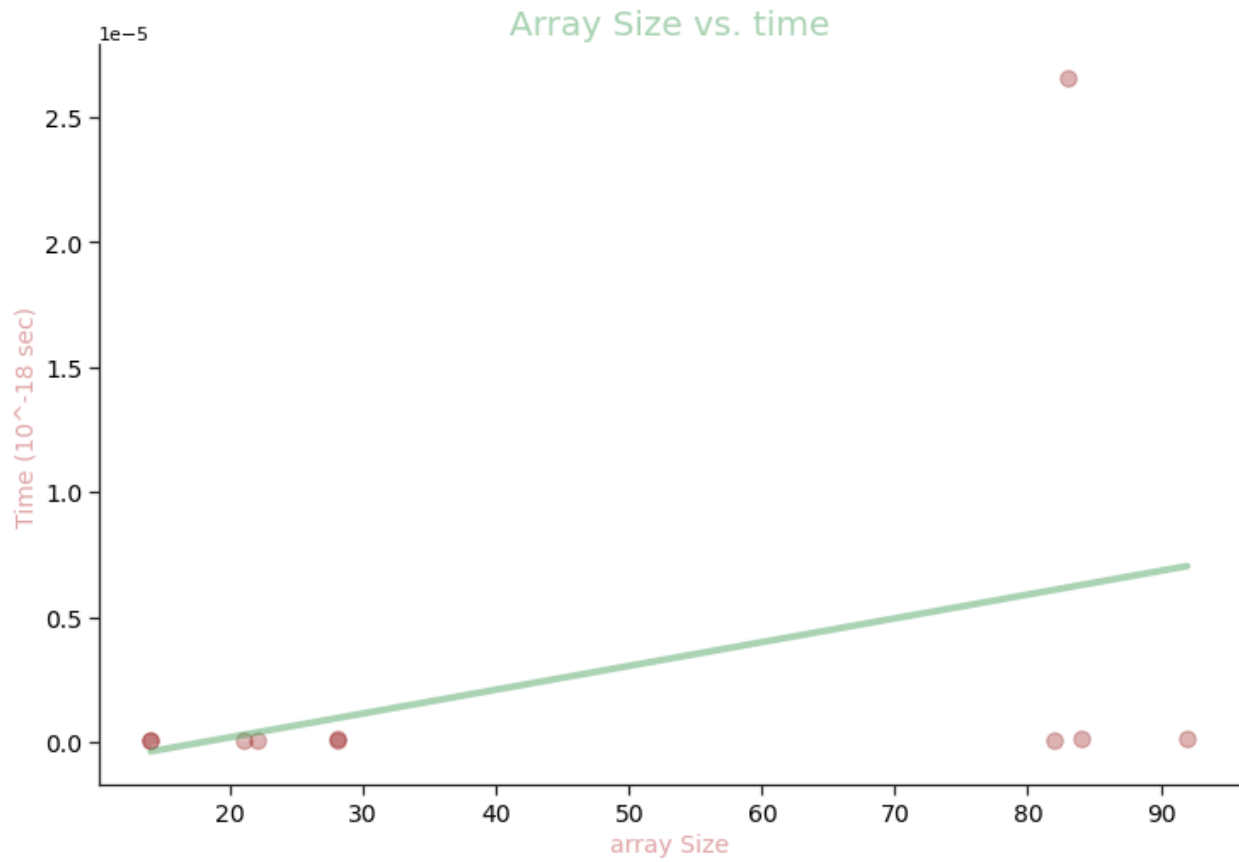


```
input  
value of n's:  
83, 84, 14, 21, 28, 22, 82, 92, 14, 28,  
time for each n:  
3.3105e-05, 1.93e-07, 1.03e-07, 1.33e-07, 1.57e-07, 1.29e-07, 1.36e-07, 1.33  
e-07, 1.03e-07, 1.42e-07,  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

```
arraySize = [83, 84, 14, 21, 28, 22, 82, 92, 14, 28]
```

```
time = [2.6536e-05, 1.6e-07, 7.9e-08, 9.3e-08, 1.31e-07, 8.7e-08, 1e-07,  
1.15e-07, 8.9e-08, 1.06e-07]
```





**Binary Search**

## Viva Questions

1. The sequential search, also known as \_\_\_\_\_?

Ans.

### Linear Search

One of the most straightforward and elementary searches is the sequential search, also known as **a linear search**.

2. What is the primary requirement for implementation of binary search in an array?

Ans.

The one pre-requisite of binary search is that an array should be in sorted order, whereas the linear search works on both sorted and unsorted array. The binary search algorithm is based on the divide and conquer technique, which means that it will divide the array recursively.

3. Is binary search applicable on array and linked list both?

Ans.

Yes, Binary search is possible on the linked list if the list is ordered and you know the count of elements in list. But While sorting the list, you can access a single element at a time through a pointer to that node i.e. either a previous node or next node.

4. What is the principle of working of Binary search?

Ans.

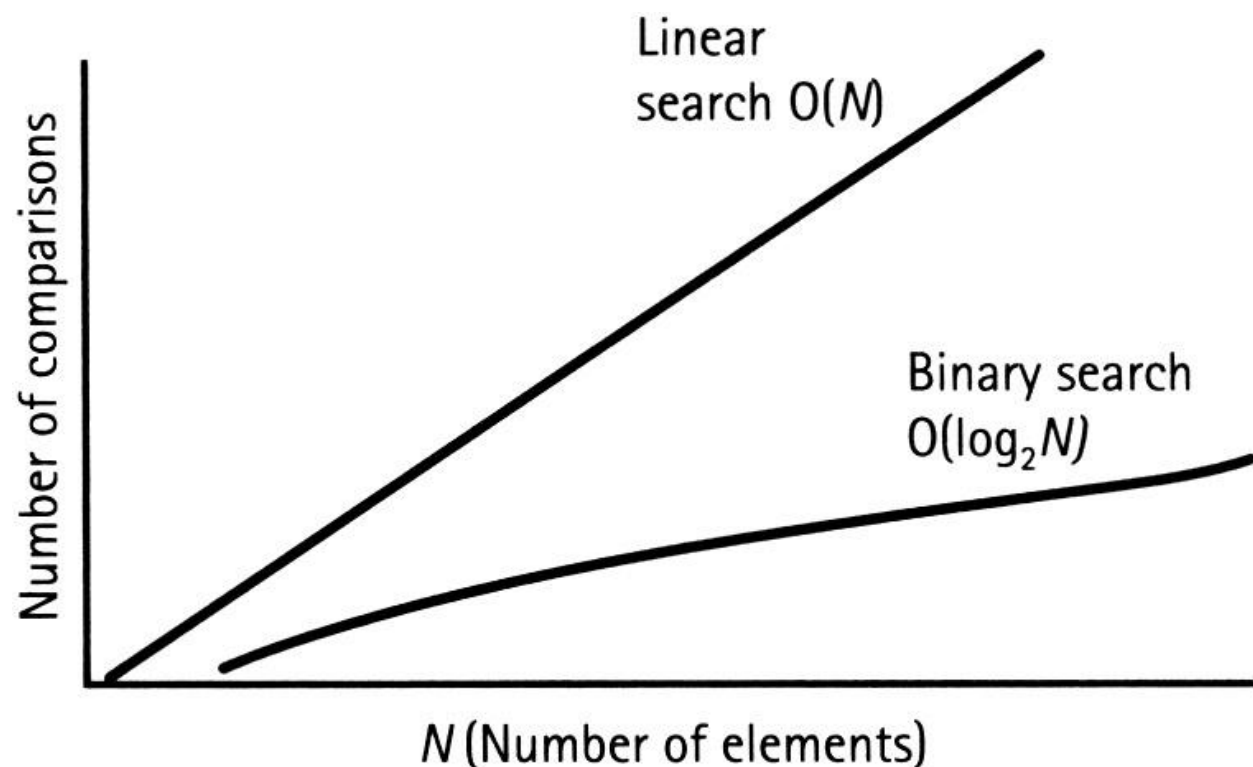
Divide and Conquer

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.

### 5. Which searching algorithm is efficient one?

Ans.

Binary search is a more efficient search algorithm which relies on the elements in the list being sorted. We apply the same search process to progressively smaller sub-lists of the original list, starting with the whole list and approximately halving the search area every time.



### 6. Under what circumstances, binary search cannot be applied to a list of elements?

Ans.



In case the list of elements is not sorted, there's no way to use binary search because the median value of the list can be anywhere and when the list is split into two parts, the element that you were searching for could be cut off.

The main problem that binary search takes  **$O(n)$  time in Linked List** due to fact that in linked list we are not able to do indexing which led traversing of each element in Linked list take  $O(n)$  time. In this paper a method is implemented through which binary search can be done with time complexity of  $O(\log_2 n)$ .