EXPERIMENT NO. 2

NAME: ADITYA B. KULKARNI
REGISTRATION NO : 241070908
SUBJECT : DAA (LAB)
SY BTECH COMPUTER ENGINEERING

AIM: Write a program for Linear search and Binary search.

Theory :

Linear search, also known as sequential search, is one of the most straightforward searching algorithms used in computer science. It operates by examining each element in a list or array one by one until the desired value is found or the end of the list is reached.

Characteristics

1. Simplicity: The linear search algorithm is simple to understand and easy to implement. It does not require any complex data structures or sorting of the data.
2. Versatility: It can be applied to both sorted and unsorted lists, making it a flexible choice for various searching scenarios.
3. Sequential Access: The algorithm accesses elements in a sequential manner. It starts from the first element and moves through to the last, which can lead to inefficiencies for large datasets.

Process

The process of linear search involves the following steps:

1. Initialization: Start from the first element of the array or list.
2. Comparison: Compare the current element with the target value.
3. Element Found: If a match is found, return the index of that element.
4. Next Element: If no match is found, move to the next element and repeat the comparison.
5. End of List: If the end of the list is reached without finding the target, return an indication that the target is not present (often -1).

Binary search is an efficient algorithm for finding a target value within a sorted array or list. It operates by repeatedly dividing the search interval in half. If the target value is less than the middle element, the search continues in the lower half; otherwise, it continues in the upper half. This halving process significantly reduces the number of comparisons needed to find the target.

Characteristics

1. **Efficiency**: Binary search has a time complexity of **O(log n)**, which makes it significantly faster than linear search, especially for large datasets, provided the data is sorted.
2. **Requirement for Sorted Data**: Binary search can only be applied to sorted lists or arrays, which means the data must be organized in a specific order (either ascending or descending).
3. **Divide and Conquer**: The algorithm utilizes a divide-and-conquer approach by eliminating half of the remaining elements with each comparison, leading to faster search times.

Process

The process of binary search involves the following steps:

1. **Initialization**: Set two pointers, one at the beginning (low) and one at the end (high) of the array.
2. **Middle Element**: Calculate the middle index of the current search range (using the formula **middle=low+high / 2)**and retrieve the middle element.
3. **Comparison**: Compare the middle element with the target value.
    - **If Equal**: If the middle element matches the target, return its index.
    - **If Less**: If the middle element is less than the target, adjust the low pointer to search the upper half (i.e., set low to middle + 1).
    - **If Greater**: If the middle element is greater than the target, adjust the high pointer to search the lower half (i.e., set high to middle - 1).
4. **Repeat**: Continue the process until the low pointer exceeds the high pointer.
5. **End of Search**: If the low pointer exceeds the high pointer, return an indication that the target is not present (often -1)

ALGORITHM AND PSEUDOCODE FOR LINEAR SEARCH :

Algorithm Linear search

Step 1 Start at 0^{th} index of array & compare the key value with the element at current index

Step 2 If matched return the current index

Step 3 If not matched, move to the next index & repeat step 1

Step 4 continue until a match is found or the end of the array is reached

Step 5 If no match is found, print a msg indicating that the element is not present in the array.

Pseudocode

```
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.
```

ALGORITHM AND PSEUDOCODE FOR BINARY SEARCH :

Algorithm Binary Search

step 1. Select middle item of the array & compare it with the key value.

step 2. If matched, return the index of middle item

step 3. If not matched determine if key is greater or less than middle value.
- if greater search right sub array
- if less search left sub-array.

step 4. Repeat until sub array size is 1

step 5. If no match, indicate unsuccessfull search.

Pseudocode

```
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 FOR LINEAR SEARCH:

Time complexity for linear search

1. Best case : $O(1)$

→ Target value is located at very first index of array.

$$T(n) = 1 \rightarrow O(1)$$

2. Average case : $O(n)$

→ we assume target value is equally likely to be at any position in the array. we need to examine half of the elements ($n/2$ comparison) since constants are ignored in Big-O notation so time complexity is $O(n)$

$$T(n) = n/2 \rightarrow O(n)$$

3. Worst case : $O(n)$

→ either target value is present in last or not present at all.

$$T(n) = n \rightarrow O(n)$$

TIME COMPLEXITY FOR BINARY SEARCH:

Time complexity for Binary Search.

1. Best case. $T(n) = O(1)$

→ target value is located at middle index of array on first comparison.

$T(n) = 1 \rightarrow O(1)$

2. Average case $T(n) = O(\log n)$

→ will have to perform several iterations to find the target. Each comparison eliminates half of the remaining elements if array has n elements, max comparisons required can be

$T(n) = \log_2 n$

In Big O notation we drop the base & constant factors so.

$O(\log n)$

3. Worst cas $T(n) = O(\log n)$

→ The target value is not present in the array algorithm will continue until only one element remains.

max no. of comparisons $\log_2 n$

$T(n) = \log_2 n \rightarrow O(\log n)$

• Using master theorem

$a = 1$ (1 subproblem)

$b = 2$ (subproblem $n/2$)

$d = 0$ (cost of dividing combining $O(1)$)

$b^d = 2^0 = 1$

$a = 1$ & $b^d = 1$

$a = b^d$

$a = b^d = T(n) = O(n^d \log n)$

$d = 0$

$T(n) = O(n^0 \log n)$

$= O(\log n)$

CODE LINEAR SEARCH :

```
def linear_search(a, n, key):
   count = 0
   for i in range(n):
      if(a[i] == key):
         print("The element is found at position", (i+1))
         count = count + 1
   if(count == 0):
      print("The element is not present in the array")

a = [14, 56, 77, 32, 84, 9, 10]
n = len(a)
key = 32
linear_search(a, n, key)
key = 3
linear_search(a, n, key)
```

CODE BINARY SEARCH :

```
def binary_search(a, low, high, key):
   mid = (low + high) // 2
   if (low <= high):
      if(a[mid] == key):
         print("The element is present at index:", mid)
      elif(key < a[mid]):
         binary_search(a, low, mid-1, key)
      elif (a[mid] < key):
         binary_search(a, mid+1, high, key)
   if(low > high):
      print("Unsuccessful Search")

a = [6, 12, 14, 18, 22, 39, 55, 182]
n = len(a)
low = 0
high = n-1
key = 22
binary_search(a, low, high, key)
key = 54
binary_search(a, low, high, key)
```

POSITIVE TEST CASES LINEAR SEARCH :

```
OUTPUT    TERMINAL    DEBUG CONSOLE    PROBLEMS    PORTS

PS C:\Users\adity\OneDrive\Desktop\DAA lab> & "c:/Users/adi
Users/adity/OneDrive/Desktop/DAA lab/.venv/linear_2.py"
The element is found at position 4
The element is found at position 3
The element is found at position 2
The element is found at position 7
The element is found at position 1
PS C:\Users\adity\OneDrive\Desktop\DAA lab>
```

NEGATIVE TEST CASES LINEAR SEARCH :

```
OUTPUT    TERMINAL    DEBUG CONSOLE    PROBLEMS    PORTS

PS C:\Users\adity\OneDrive\Desktop\DAA lab> ^C
PS C:\Users\adity\OneDrive\Desktop\DAA lab> & "c:/Users/a
Users/adity/OneDrive/Desktop/DAA lab/.venv/linear_2.py"
The element is not present in the array
The element is not present in the array
The element is not present in the array
The element is not present in the array
The element is not present in the array
PS C:\Users\adity\OneDrive\Desktop\DAA lab>
```

POSITVE TEST CASES BINARY SEARCH :

```
OUTPUT    TERMINAL    DEBUG CONSOLE    PROBLEMS    PORTS

PS C:\Users\adity\OneDrive\Desktop\DAA lab> ^C
PS C:\Users\adity\OneDrive\Desktop\DAA lab> & "c:/Users/adity/On
Users/adity/OneDrive/Desktop/DAA lab/.venv/binary_2.py"
The element is present at index: 4
The element is present at index: 2
The element is present at index: 5
The element is present at index: 7
The element is present at index: 0
PS C:\Users\adity\OneDrive\Desktop\DAA lab>
```

NEGATIVE TEST CASES BINARY SEARCH :

```
OUTPUT    TERMINAL    DEBUG CONSOLE    PROBLEMS    PORTS

PS C:\Users\adity\OneDrive\Desktop\DAA lab> & "c:/Users/
Users/adity/OneDrive/Desktop/DAA lab/.venv/binary_2.py"
Unsuccessful Search
Unsuccessful Search
Unsuccessful Search
Unsuccessful Search
Unsuccessful Search
PS C:\Users\adity\OneDrive\Desktop\DAA lab>
```

CONCLUSION – Hence we have learnt and implemented LINEAR AND BINARY SEARCH Algorithms