**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech. Sem V

**Course: Design and Analysis of Algorithms**

**List of Experiments**

w.e.f. 1th Jul 2020

**Faculty:** Abhay Kolhe

LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.02**

**A.1 Aim:**

Implementation of Linear Search and Binary Search Technique.

**A.2 Prerequisite:**

1 Concepts of C/C++/Python Programming

2. Knowledge of Array Handling.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Design & develop a program searching a key in given data collection.
2. Identify the applications of Searching techniques.
3. Compare the performance of Linear Search and Binary Search techniques.

**A.4 Theory:**

**A.4.1.**

**Linear search** is a search algorithm, also known as **sequential search**, that is suitable for searching a list of data for a particular value.

It operates by checking every element of a list one at a time in sequence until a match is found. Linear search runs in O(n). If the data are distributed randomly, the expected number of comparisons that will be necessary is:

****

where *n* is the number of elements in the list and *k* is the number of times that the value being searched for appears in the list. The best case is that the value is equal to the first element tested, in which case only 1 comparison is needed. The worst case is that the value is not in the list (or it appears only once at the end of the list), in which case *n* comparisons are needed.

The simplicity of the linear search means that if just a few elements are to be searched it is less trouble than more complex methods that require preparation such as sorting the list to be searched or more complex data structures, especially when entries may be subject to frequent revision. Another possibility is when certain values are much more likely to be searched for than others and it can be arranged that such values will be amongst the first considered in the list.

**Binary search:**

A **binary search algorithm** (or **binary chop**) is a technique for locating a particular value in a sorted list. The method makes progressively better guesses, and closes in on the location of the sought value by selecting the middle element in the span (which, because the list is in sorted order, is the median value), comparing its value to the target value, and determining if it is greater than, less than, or equal to the target value. A guessed index whose value turns out to be too high becomes the new upper bound of the span, and if its value is too low that index becomes the new lower bound. Only the sign of the difference is inspected: there is no attempt at an interpolation search based on the size of the difference. Pursuing this strategy iteratively, the method reduces the search span by a factor of two each time, and soon finds the target value or else determines that it is not in the list at all. A binary search is an example of a divide and conquer search algorithm.

**A.5 Procedure/Algorithm:**

**A.5.1:**

**Linear Search technique**

For each item in the list:

Check to see if the item you're looking for matches the item in the list.

If it matches.

Return the location where you found it (the index).

If it does not match.

Continue searching until you reach the end of the list.

If we get here, we know the item does not exist in the list.

Return -1

**Binary Search Technique**

int find (const apvector &list, double target)

*// pre: list is sorted in ascending order*

*//post: ITERATIVE binary search will return the index of the target element, else -1*

{

int mid;

int first = 0;

int last = list.length( ) -1;

while ( first <= last )

{

mid = (first + last) / 2;

if ( list[mid] == target )

return mid;

if ( list[mid] > target )

last = mid - 1;

else

first = mid + 1;

}

return -1;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. | Name: |
| Class : | Batch : |
| Date of Experiment: | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

***(Paste your c/c++ code completed during the 2 hours of practical in the lab here)***

**B.2 Input and Output:**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Input Data:**

**Output Data:**

**B.3 Observations and learning:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

Q.1 Identify the applications of Linear Search Technique.

Q.2 Identify the applications of Binary Search Technique.

Q.3 Comment on the performance of Linear Search and Binary Search Technique, after

filling up the following table.

|  |  |
| --- | --- |
| **Array Size(n)** | **Data** |
| 4 |  |
| 8 |  |
| 16 |  |
| 32 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Array Size**  **(n)** | **Binary Search** | | | | | | **Linear Search** | | | | | | |
| **Best Case** | | **Avg. Case** | | **Worst Case** | | **Best Case** | | **Avg. Case** | | **Worst Case** | | |
| **Key** | **NoC+** | **Key** | **NoC+** | **Key** | **NoC+** | **Key** | **NoC+** | **Key** | **NoC+** | **Key** | **NoC+** | |
| 4 |  |  |  |  |  |  |  |  |  |  |  | |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  | |  |
| 16 |  |  |  |  |  |  |  |  |  |  |  | |  |
| 32 |  |  |  |  |  |  |  |  |  |  |  | |  |

NoC+  Number of Comparisons

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*