Tanmay Karmarkar 21143.

Search Algorithms

Problem Statement:

a) Write a python programname to store roll no's of students in array, who attended training program in random order. Write functions for searching of a particular student attended using Linear search, Sentinel Search, Binary search & Fibonacci Search

Objective:

1) Implement linear & sentinel search for randomly ordered array.
2) Implement Binary & Fibonacci search for sorted array.

S/W & H/W requirements:

Windows 10, python 3, Visual studio code, AMD RYZEN-5 5500U.

Theory: Searching is a technique of finding an element in list.

Standard Search algorithms:

1) Linear Search:
   Examine elements starting from first & terminate process when list is exhausted or comparision result is a success.
2) Sentinel Search:
   An extra record equal to search element is inserted at the end of array which speeds up search as it eliminates check for list exhaustion on every iteration.
3) Binary search:
   Works on sorted array only. Comparision result is one of the following:
   1) key = arr[i] then return success.
   2) key < arr[i] then call left half.
   3) key > arr[i] then call right half.

4) Fibonacci Search:
Works on sorted array only. Similar to Binary but partito
is a fibonacci number.

ALGORITHMS:

1. Algorithm for linear search:
1) Pass the list & roll no. to be searched as arguments.
2) Iterate through every element of the list & check if the element is equal to the required roll no.
3) If they are equal return index else print Not Found.
4)

2. Algorithm for sentinel search:
1. Pass the list & ~~roll no~~ roll no to be searched as arguments.
2. Put the element to be found at last index.
3. Traverse through the list with a counter.
4. If the element is found print index or else print not present.
5. Set the last element back to its original value.

3. Algorithm for binary search:
1. Pass list, top, bottom, roll no as arguments.
2. Sort the list.
3. Initialize 'mid' to average of top & bottom.
4. Compare element with mid.
5. If they are equal return mid.
6. If mid > roll no pass list, mid+1, top, roll no as arguments.
7. Else pass bottom, mid-1, roll no as arguments.
8. Else return -1.

4. Algorithm for Fibonacci Search:

1. Pass sorted list & roll no. to be found as arguments.
2. ~~Initialize~~ m = 0.
3. ~~Increament value~~ of m while a fibonacci number greater than ~~the length of list~~.
2. Find the smallest number greater than equal to n. Let this no be fibM. Let the two Fibonacci nos preceding it to be fibMm1 & fibMm2.
3. While the array has elements to be inspected:
   1. Compare x with the last element of the range covered by fibm
   2. If x matches, return index.
   3. Else if x is less than the element, move the three Fibonacci variables two fibonnacci down, indicating elimination of approx. rear two-third of the remaining array.
   4. Else x is > than element, move the 3 fibonacci variables one Fibonacci down. Reset offset to index. Together these indicate the elimination of approximately front one-third.
   3. Since there might be a single element remaining for comparision, check if fibMm1 is 1.

Pseudocode:

Linear Search:

```
y = int (input)
for i in rage (0,n):
    if (arr1[i] == y).
        return i
return -1.
```

Sentinel Search:

```
y = int(input)
last = arr1[n-1]
arr1[n-1] = y
i = 0
while(arr[i] != y):
    i += 1
arr1[n-1] = last
if((i < n-1) or (arr1[n-1] == y))
    print(i)
else:
    print (Not Present)
```

Binary Search:

```
arr1.sort()
if top ≥ bottom:
    mid = top + bottom//2
    if arr[mid] = y:
        return mid
elif arr1[mid] > y:
    return binary search(arr1, bottom, m-1, y)
else:
    return binsearch(arr1, m+1, top, y)
else:
    return -1:
```