Design and Analysis of Algorithms (Week1 Solution)
PCS-505

Note - You can find all the problem statements and their solutions on following link - DAA Link

Solution - 1

For *linear search*, we just need to scan the array from the beginning till the end, index 1 to index n, and check if the entry at that position equal to v or not. The pseudocode can be written as follows…

```
LINEAR-SEARCH (A, v)
1  for i = 1 to A.length
2      if A[i] == v
3          return i
4  return NIL
```

Solution - 2

A binary search divides a range of values into halves, and continues to narrow down the field of search until the unknown value is found. It is the classic example of a "divide and conquer" algorithm.

 # Iterative Approach

```
binarySearch(array, left, right, key):
        while left <= right:
                mid = left + (right - left) // 2
                if array[mid] == key:
                        return mid
                elif array[mid] < key:
                        left = mid + 1
                else:
                        right = mid - 1
        return -1
```

 # Recursive Approach

```
binarySearch (array, left, right, key):
  if left <= right:
    mid = left + (right - left) // 2
    if array[mid] == key:
      return mid
    elif array[mid] < key:
      return binarySearch(array, mid + 1, right, key)
    else:
      return binarySearch(array, left, mid-1, key)
  else:
    return -1
```

Solution - 3

For an array arr[] of size n and block (to be jumped) size m, search at the indexes arr[0], arr[m], arr[2m].....arr[km] and so on. Once the interval **(arr[km] < key < arr[(k+1)m])** is found, perform a linear search operation from the index **km** to find the element **key**.

jumpSearch(array, n, key):

```
jump = math.sqrt(n)
steps = jump
prev = 0
while array[int(min(steps, n)-1)] < key:
    prev = steps
    steps += jump
    if prev >= n:
        return -1

while array[int(prev)] < key:
    prev += 1
    if prev == min(steps, n):
        return -1

if array[int(prev)] == key:
    return prev

return -1
```

Solution - 4

For an array arr[n], search at the indexes **arr[0], arr[1], arr[2], arr[4],.....,arr[2k]** and so on. Once the interval **(arr[2k] < key < arr[2(k+1)])** is found, perform a linear search or binary search operation from the index 2k to find the element key.

exponentialSearch(array, n, key):

```
if array[0] == key:
    return 0
i = 1
while i < n and array[i] <= key:
    i = i * 2
return linearSearch (array, int(i/2), n, key)
return binarySearch(array, int(i/2), min(i, n-1), key)
```