“Linear Search Iterative”.

1. Where is linear searching used?
a) When the list has only a few elements
b) When performing a single search in an unordered list
c) Used all the time
d) When the list has only a few elements and When performing a single search in an unordered list
View Answer
Answer: d
Explanation: It is practical to implement linear search in the situations mentioned in When the list has only a few elements and When performing a single search in an unordered list, but for larger elements the complexity becomes larger and it makes sense to sort the list and employ binary search or hashing.

2. Select the code snippet which performs unordered linear search iteratively?
a)

```
int unorderedLinearSearch(int arr[], int size, int data)
{
    int index;
    for(int i = 0; i < size; i++)
    {
        if(arr[i] == data)
        {
            index = i;
            break;
        }
    }
    return index;
}
```

b)

Note: Join free Sanfoundry classes at Telegram or Youtube


```
int unorderedLinearSearch(int arr[], int size, int data)
{
    int index;
    for(int i = 0; i < size; i++)
    {
```

```
        if(arr[i] == data)
        {
            break;
        }
    }
    return index;
}
```

c)

Take Data Structure II Practice Tests - Chapterwise!
Start the Test Now: Chapter 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

```
int unorderedLinearSearch(int arr[], int size, int data)
{
    int index;
    for(int i = 0; i <= size; i++)
    {
        if(arr[i] == data)
        {
            index = i;
            break;
        }
    }
    return index;
}
```

d)

```
int unorderedLinearSearch(int arr[], int size, int data)
{
    int index;
    for(int i = 0; i < size-1; i++)
    {
        if(arr[i] == data)
        {
            index = i;
            break;
        }
    }
    return index;
}
```

View Answer

Answer: a
Explanation: Unordered term refers to the given array, that is, the elements need not be ordered. To search for an element in such an array, we need to loop through the elements until the desired element is found.

3. What is the best case for linear search?
a) O(nlogn)
b) O(logn)
c) O(n)
d) O(1)
View Answer
Answer: d
Explanation: The element is at the head of the array, hence O(1).

4. What is the worst case for linear search?
a) O(nlogn)
b) O(logn)
c) O(n)
d) O(1)
View Answer
Answer: c
Explanation: Worst case is when the desired element is at the tail of the array or not present at all, in this case you have to traverse till the end of the array, hence the complexity is O(n).

5. Select the code snippet which performs ordered linear search iteratively?
a)

```
public int linearSearch(int arr[],int key,int size)
{
       int index = -1;
            int i = 0;
       while(size > 0)
       {
             if(data[i] == key)
             {
                    index = i;
             }
             if(data[i] > key))
             {
                   index = i;
```

```
                break;
            }
            i++;
        }
        return index;
}
```

b)

```
public int linearSearch(int arr[],int key,int size)
{
        int index = -1;
             int i = 0;
        while(size > 0)
        {
              if(data[i] == key)
              {
                      index = i;
              }
              if(data[i] > key))
              {
                   break;
              }
              i++;
        }
        return index;
}
```

c)

```
public int linearSearch(int arr[],int key,int size)
{
        int index = -1;
             int i = 0;
        while(size > 0)
        {
              if(data[i] == key)
              {
                    break;
              }
              if(data[i] > key))
              {
                  index = i;
```

```
            }
            i++;
        }
        return index;
}
```

d)

```
public int linearSearch(int arr[],int key,int size)
{
       int index = -1;
            int i = 0;
       while(size > 0)
       {
             if(data[i] == key)
             {
                   break;
             }
             if(data[i] > key))
             {
                 break;
                 index=i;
             }
             i++;
        }
        return index;
}
```

View Answer

Answer: b

Explanation: The term ordered refers to the items in the array being sorted(here we assume ascending order). So traverse through the array until the element, if at any time the value at i exceeds key value, it means the element is not present in the array. This provides a slightly better efficiency than unordered linear search.

6. What is the best case and worst case complexity of ordered linear search?

a) O(nlogn), O(logn)

b) O(logn), O(nlogn)

c) O(n), O(1)

d) O(1), O(n)

View Answer

Answer: d
Explanation: Although ordered linear search is better than unordered when the element is not present in the array, the best and worst cases still remain the same, with the key element being found at first position or at last position.

7. Choose the code snippet which uses recursion for linear search.
a)

```
public void linSearch(int[] arr, int first, int last, int key)
{
         if(first == last)
        {
                  System.out.print("-1");
         }
         else
        {
                  if(arr[first] == key)
                {
                          System.out.print(first);
                  }
                  else
                {
                          linSearch(arr, first+1, last, key);
                  }
         }
}
```

b)

```
      public void linSearch(int[] arr, int first, int last, int key)
      {
                  if(first == last)
                {
                          System.out.print("-1");
                  }
                  else
                {
                          if(arr[first] == key)
                        {
                                  System.out.print(first);
                          }
                          else
                        {
```

```
                                    linSearch(arr, first+1, last-1, key);
                            }
                    }
        }
```

c)

```
public void linSearch(int[] arr, int first, int last, int key)
{
          if(first == last)
        {
                    System.out.print("-1");
          }
          else
        {
                    if(arr[first] == key)
                {
                            System.out.print(last);
                    }
                    else
                {
                            linSearch(arr, first+1, last, key);
                    }
          }
}
```

d)

```
public void linSearch(int[] arr, int first, int last, int key)
{
          if(first == last)
        {
                    System.out.print("-1");
          }
          else
        {
                    if(arr[first] == key)
                {
                            System.out.print(first);
                    }
                    else
                {
                            linSearch(arr, first+1, last+1, key);
```

```
                }
        }
}
```

View Answer

Answer: a

Explanation: Every time check the key with the array value at first index, if it is not equal then call the function again with an incremented first index.

8. What does the following piece of code do?

```
for (int i = 0; i < arr.length-1; i++)
{
    for (int j = i+1; j < arr.length; j++)
    {
        if( (arr[i].equals(arr[j])) && (i != j) )
        {
            System.out.println(arr[i]);
        }
    }
}
```

a) Print the duplicate elements in the array

b) Print the element with maximum frequency

c) Print the unique elements in the array

d) Prints the element with minimum frequnecy

View Answer

Answer: a

Explanation: The print statement is executed only when the items are equal and their indices are not.

9. Select the code snippet which prints the element with maximum frequency.

a)

```
public int findPopular(int[] a)
{
        if (a == null || a.length == 0)
                return 0;
        Arrays.sort(a);
        int previous = a[0];
        int popular = a[0];
```

```
         int count = 1;
         int maxCount = 1;
         for (int i = 1; i < a.length; i++)
        {
                  if (a[i] == previous)
                  count++;
                  else
                {
                           if (count > maxCount)
                        {
                                    popular = a[i-1];
                                    maxCount = count;
                           }
                  previous = a[i];
                  count = 1;
                  }
         }
         return count > maxCount ? a[a.length-1] : popular;
}
```

b)

```
public int findPopular(int[] a)
{
         if (a == null || a.length == 0)
                  return 0;
         Arrays.sort(a);
         int previous = a[0];
         int popular = a[0];
         int count = 1;
         int maxCount = 1;
         for (int i = 1; i < a.length; i++)
        {
                  if (a[i] == previous)
                           count++;
                  else
                {
                           if (count > maxCount)
                        {
                                    popular = a[i];
                                    maxCount = count;
                           }
                           previous = a[i];
```

```
                        count = 1;
                }
        }
        return count > maxCount ? a[a.length-1] : popular;
}
```

c)

```
public int findPopular(int[] a)
{
        if (a == null || a.length == 0)
                return 0;
        Arrays.sort(a);
        int previous = a[0];
        int popular = a[0];
        int count = 1;
        int maxCount = 1;
        for (int i = 1; i < a.length; i++)
       {
                if (a[i+1] == previous)
                        count++;
                else
               {
                        if (count > maxCount)
                     {
                                popular = a[i-1];
                                maxCount = count;
                        }
                        previous = a[i];
                        count = 1;
                }
        }
        return count > maxCount ? a[a.length-1] : popular;
}
```

d)

```
public int findPopular(int[] a)
{
        if (a == null || a.length == 0)
                return 0;
        Arrays.sort(a);
        int previous = a[0];
```

```
        int popular = a[0];
        int count = 1;
        int maxCount = 1;
        for (int i = 1; i < a.length; i++)
       {
                if (a[i+2] == previous)
                        count++;
                else
              {
                        if (count > maxCount)
                    {
                                popular = a[i-1];
                                maxCount = count;
                        }
                        previous = a[i];
                        count = 1;
                }
        }
        return count > maxCount ? a[a.length-1] : popular;
}
```

View Answer

Answer: a

Explanation: Traverse through the array and see if it is equal to the previous element, since the array is sorted this method works with a time complexity of O(nlogn), without sorting a Brute force technique must be applied for which the time complexity will be $O(n^2)$.

10. Which of the following is a disadvantage of linear search?

a) Requires more space

b) Greater time complexities compared to other searching algorithms

c) Not easy to understand

d) Not easy to implement

View Answer

Answer: b

Explanation: The complexity of linear search as the name suggests is O(n) which is much greater than other searching techniques like binary search(O(logn)). Linear search is easy to implement and understand than other searching techniques.

"Binary Search Iterative".

1. What is the advantage of recursive approach than an iterative approach?
a) Consumes less memory
b) Less code and easy to implement
c) Consumes more memory
d) More code has to be written
View Answer
Answer: b
Explanation: A recursive approach is easier to understand and contains fewer lines of code.

2. Choose the appropriate code that does binary search using recursion.
a)

```
public static int recursive(int arr[], int low, int high, int key)
{
        int mid = low + (high - low)/2;
        if(arr[mid] == key)
        {
                return mid;
        }
        else if(arr[mid] < key)
        {
                return recursive(arr,mid+1,high,key);
        }
        else
        {
                return recursive(arr,low,mid-1,key);
        }
}
```

b)


```
public static int recursive(int arr[], int low, int high, int key)
{
        int mid = low + (high + low)/2;
        if(arr[mid] == key)
        {
                return mid;
        }
        else if(arr[mid] < key)
```

```
        {
                return recursive(arr,mid-1,high,key);
        }
        else
        {
                return recursive(arr,low,mid+1,key);
        }
}
```

c)

Check this: Programming MCQs | Computer Science MCQs

```
public static int recursive(int arr[], int low, int high, int key)
{
        int mid = low + (high - low)/2;
        if(arr[mid] == key)
        {
                return mid;
        }
        else if(arr[mid] < key)
        {
                return recursive(arr,mid,high,key);
        }
        else
        {
                return recursive(arr,low,mid-1,key);
        }
}
```

d)

```
public static int recursive(int arr[], int low, int high, int key)
{
        int mid = low + ((high - low)/2)+1;
        if(arr[mid] == key)
        {
                return mid;
        }
        else if(arr[mid] < key)
        {
                return recursive(arr,mid,high,key);
        }
        else
```

```
        {
                return recursive(arr,low,mid-1,key);
        }
}
```

View Answer

Answer: a

Explanation: Calculate the ‘mid’ value, and check if that is the key, if not, call the function again with 2 sub arrays, one with till mid-1 and the other starting from mid+1.

3. Given an input arr = {2,5,7,99,899}; key = 899; What is the level of recursion?

a) 5

b) 2

c) 3

d) 4

View Answer

Answer: c

Explanation: level 1: mid = 7

level 2: mid = 99

level 3: mid = 899(this is the key).

4. Given an array arr = {45,77,89,90,94,99,100} and key = 99; what are the mid values(corresponding array elements) in the first and second levels of recursion?

a) 90 and 99

b) 90 and 94

c) 89 and 99

d) 89 and 94

View Answer

Answer: a

Explanation: At first level key = 90

At second level key= 99

Here 90 and 99 are mid values.

5. What is the worst case complexity of binary search using recursion?

a) O(nlogn)

b) O(logn)

c) O(n)

d) $O(n^2)$

View Answer

Answer: b

Explanation: Using the divide and conquer master theorem.

6. What is the average case time complexity of binary search using recursion?

a) O(nlogn)

b) O(logn)

c) O(n)

d) $O(n^2)$
View Answer
Answer: b
Explanation: T(n) = T(n/2) + 1, Using the divide and conquer master theorem.

7. Which of the following is not an application of binary search?
a) To find the lower/upper bound in an ordered sequence
b) Union of intervals
c) Debugging
d) To search in unordered list
View Answer
Answer: d
Explanation: In Binary search, the elements in the list should be sorted. It is applicable only for ordered list. Hence Binary search in unordered list is not an application.

8. Choose among the following code for an iterative binary search.
a)

```
public static int iterative(int arr[], int key)
{
        int low = 0;
        int mid = 0;
        int high = arr.length-1;
        while(low <= high)
        {
                mid = low + (high + low)/2;
                if(arr[mid] == key)
                {
                        return mid;
                }
                else if(arr[mid] < key)
                {
                        low = mid - 1;
                }
                else
                {
                        high = mid + 1;
                }
        }
        return -1;
}
```

b)

```
public static int iterative(int arr[], int key)
```

```
{
        int low = 0;
        int mid = 0;
        int high = arr.length-1;
        while(low <= high)
        {
                mid = low + (high - low)/2;
                if(arr[mid] == key)
                {
                        return mid;
                }
                else if(arr[mid] < key)
                {
                        low = mid + 1;
                }
                else
                {
                        high = mid - 1;
                }
        }
        return -1;
}
```

c)

```
public static int iterative(int arr[], int key)
{
        int low = 0;
        int mid = 0;
        int high = arr.length-1;
        while(low <= high)
        {
                mid = low + (high + low)/2;
                if(arr[mid] == key)
                {
                        return mid;
                }
                else if(arr[mid] < key)
                {
                        low = mid + 1;
                }
                else
                {
```

```
                        high = mid - 1;
                }
        }
        return -1;
}
```

d)

```
public static int iterative(int arr[], int key)
{
        int low = 0;
        int mid = 0;
        int high = arr.length-1;
        while(low <= high)
        {
                mid = low + (high - low)/2;
                if(arr[mid] == key)
                {
                        return mid;
                }
                else if(arr[mid] < key)
                {
                        low = mid - 1;
                }
                else
                {
                        high = mid + 1;
                }
        }
        return -1;
}
```

View Answer

Answer: b

Explanation: Find the ‘mid’, check if it equals the key, if not, continue the iterations until low <= high.

9. Binary Search can be categorized into which of the following?

a) Brute Force technique

b) Divide and conquer

c) Greedy algorithm

d) Dynamic programming

View Answer

Answer: b
Explanation: Since ‘mid’ is calculated for every iteration or recursion, we are diving the array into half and then try to solve the problem.

10. Given an array arr = {5,6,77,88,99} and key = 88; How many iterations are done until the element is found?
a) 1
b) 3
c) 4
d) 2
View Answer
Answer: d
Explanation: Iteration1 : mid = 77; Iteration2 : mid = 88;

11. Given an array arr = {45,77,89,90,94,99,100} and key = 100; What are the mid values(corresponding array elements) generated in the first and second iterations?
a) 90 and 99
b) 90 and 100
c) 89 and 94
d) 94 and 99
View Answer
Answer: a
Explanation: Trace the input with the binary search iterative code.

12. What is the time complexity of binary search with iteration?
a) O(nlogn)
b) O(logn)
c) O(n)
d) $O(n^2)$
View Answer
Answer: b
Explanation: T(n) = T(n/2) + theta(1)
Using the divide and conquer master theorem, we get the time complexity as O(logn).