Vasanthan_T_DSA_Practice-6

18/11/2024

1. **Bubble Sort**

Given an array, arr[]. Sort the array using bubble sort algorithm.

Examples :

Input: arr[] = [4, 1, 3, 9, 7]

Output: [1, 3, 4, 7, 9]

Input: arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Code:

```
import java.util.*;
public class BubbleSort{
   public static void bubble(int arr[]) {
      // code here
      int n=arr.length;
      for(int i=0;i<n-1;i++){
          boolean swap=false;
         for(int j=0;j<n-1-i;j++){
            if(arr[j]>arr[j+1]){
               arr[j]=arr[j]^arr[j+1];
               arr[j+1]=arr[j]^arr[j+1];
               arr[j]=arr[j]^arr[j+1];
               swap=true;
            }

         }
         if(swap==false) break;
```

```
    }
        for(int i=0;i<n;i++){
            System.out.print(arr[i]+" ");
        }
  }
        public static void main(String[] args){
            int[] arr={10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
            bubble(arr);
        }
}
```

Output:

```
D:\code\JavaCodes>javac BubbleSort.java

D:\code\JavaCodes>java BubbleSort.java
1 2 3 4 5 6 7 8 9 10
D:\code\JavaCodes>
```

Time Complexity: O(n^2)

Space Complexity: O(n)

2. Non Repeating Character

Given a string s consisting of lowercase Latin Letters. Return the first non-repeating character in s. If there is no non-repeating character, return '$'.
Note: When you return '$' driver code will output -1.

Examples:

Input: s = "geeksforgeeks"

Output: 'f'

Explanation: In the given string, 'f' is the first character in the string which does not repeat.

Input: s = "racecar"
Output: 'e'
Explanation: In the given string, 'e' is the only character in the string which does not repeat.

Code:

```
import java.util.*;
```

```
class NonRepeatingchars{
  public static char Char(String s) {
      // Your code here
      int[] hash=new int[26];
      char ans='$';
      for(char i:s.toCharArray()){
         hash[i-'a']++;
      }
      for(int i=0;i<s.length();i++){
         if(hash[s.charAt(i)-'a']==1){
            return s.charAt(i);
         }
      }
      return ans;


   }
         public static void main(String[] ar){
                  String s="geeksforgeeks";
                  System.out.println(Char(s));
         }
}
```

Output:

```
D:\code\JavaCodes>javac NonRepeatingchars.java

D:\code\JavaCodes>java NonRepeatingchars.java
f
```

Time Complexity:O(n)

Space Complexity: O(1)

3. **K largest element**

Given an array **arr[]** of positive integers and an integer **k**, Your task is to return **k largest elements** in decreasing order.

Examples

Input: arr[] = [12, 5, 787, 1, 23], k = 2

Output: [787, 23]

Explanation: 1st largest element in the array is 787 and second largest is 23.

Code:

```
import java.util.*;
public class KLargest{

    static List<Integer> largest(int arr[], int k) {
        // write code here
        List<Integer> ans=new ArrayList<>();
        PriorityQueue<Integer> pq=new PriorityQueue<>(Comparator.reverseOrder());
        for(int i:arr){
            pq.add(i);
        }
        for(int i=0;i<k;i++){
            ans.add(pq.poll());
        }
        return ans;

    }
            public static void main(String[] ar){
                    int[] arr={1, 23, 12, 9, 30, 2, 50};
                    int k=3;
                    System.out.println(largest(arr,k));
            }
}
```

Output:

```
D:\code\JavaCodes>javac KLargest.java

D:\code\JavaCodes>java KLargest.java
[50, 30, 23]
```

Space complexity:O(n)

Time Complexity: O(n log n)

4. **Form the Largest Number**

Given an array of strings arr[] representing non-negative integers, arrange them so that after concatenating them in order, it results in the largest possible number. Since the result may be very large, return it as a string.

Note: There are no leading zeros in each array element.

Examples:

Input: arr[] = ["3", "30", "34", "5", "9"]

Output: "9534330"

Explanation: Given numbers are {"3", "30", "34", "5", "9"}, the arrangement "9534330" gives the largest value.

Input: arr[] = ["54", "546", "548", "60"]

Output: "6054854654"

Explanation: Given numbers are {"54", "546", "548", "60"}, the arrangement "6054854654" gives the largest value.

Code:

```
import java.util.*;

class Largest{

   public static String printLargest(String[] arr) {

      // code here

      String ans="";

      Arrays.sort(arr,(a,b)->(b+a).compareTo(a+b));

      if(arr[0].equals("0")) return "0";

      for(int i=0;i<arr.length;i++){

         ans+=arr[i];

      }
```

```
        return ans;
    }
            public static void main(String[] ar){
                    String[] arr={"3", "30", "34", "5", "9"};
                    System.out.println(printLargest(arr));
            }
}
```

Output:

```
D:\code\JavaCodes>javac Largest.java

D:\code\JavaCodes>java Largest.java
9534330
```

Time Complexity:O(n logn)

Space Complexity: O(n)

5.Quick Sort

Implement Quick Sort, a Divide and Conquer algorithm, to sort an array, **arr**[] in ascending order. Given an array, **arr**[], with starting index **low** and ending index **high**, complete the functions **partition()** and **quickSort()**. Use the last element as the pivot so that all elements less than or equal to the pivot come before it, and elements greater than the pivot follow it.

Note: The **low** and **high** are inclusive.

Examples:

Input: arr[] = [4, 1, 3, 9, 7]

Output: [1, 3, 4, 7, 9]
Explanation: After sorting, all elements are arranged in ascending order.

Code:

```
import java.util.*;

public class QuickSort{

    // Function to sort an array using quick sort algorithm.

    public static int[] Sort(int arr[], int low, int high) {

        // code here4
```

```
    if(low<high){
        int pi=partition(arr,low,high);
        Sort(arr,low,pi-1);
        Sort(arr,pi+1,high);
    }

    return arr;

}

public static int partition(int arr[], int low, int high) {
    // your code here
    int pivot=arr[high];
    int i=low-1;
    for(int j=low;j<high;j++){
        if(arr[j]<pivot){
            i++;
            int temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
        }
    }
    int temp=arr[i+1];
    arr[i+1]=arr[high];
    arr[high]=temp;
    return i+1;
}
        public static void main(String[] args){
                int[] arr = {4, 1, 3, 9, 7};
                int n=arr.length;
                Sort(arr,0,n-1);
```

```
            for(int i=0;i<n;i++){
                System.out.print(arr[i]);
            }



        }
}
```

Output:

```
D:\code\JavaCodes>javac QuickSort.java

D:\code\JavaCodes>java QuickSort.java
13479
```

Time Complexity: O(n logn)

Space Complexity: O(n)

6. Edit Distance

Given two strings s1 and s2. Return the minimum number of operations required to convert s1 to s2.

The possible operations are permitted:

Insert a character at any position of the string.

Remove any character from the string.

Replace any character from the string with any other character.

Examples:

Input: s1 = "geek", s2 = "gesek"

Output: 1

Explanation: One operation is required, inserting 's' between two 'e'.

Code:

```
import java.util.*;
public class EditDistance{
   public static int Distance(String s1, String s2) {
      int n=s1.length();
      int m=s2.length();

     int[][] dp=new int[n+1][m+1];
     for(int i=0;i<=n;i++){
        for(int j=0;j<=m;j++){
           dp[i][j]=-1;
        }
     }
     return helper(s1,s2,n,m,dp);

   }
   public static int helper(String s1, String s2,int n, int m, int[][] dp){
      if(m==0) return n;
      if(n==0) return m;
      if(dp[n][m]!=-1) return dp[n][m];
      if(s1.charAt(n-1)==s2.charAt(m-1)){
         dp[n][m]=helper(s1,s2,n-1,m-1,dp);
      }
      else{
      int insert=helper(s1,s2,n,m-1,dp);
      int delete=helper(s1,s2,n-1,m,dp);
      int replace=helper(s1,s2,n-1,m-1,dp);
      dp[n][m]=1+Math.min(insert,Math.min(delete,replace));
      }
      return dp[n][m];

   }
```

```
public static void main(String[] ar){
        String s1 = "GEEXSFRGEEKKS";
    String s2 = "GEEKSFORGEEKS";
        System.out.println(Distance(s1,s2));
}
```

Output:

```
D:\code\JavaCodes>javac editDistance.java

D:\code\JavaCodes>java editDistance.java
3
```

Time Complexity: O(n*m)

Space Complexity: O(n*m)