Data Structures and Algorithms - Coursework 1

## Question 1: Given an array (A) of size n, where each element of the array is an integer in the range 1 to 100 inclusive, create and display a histogram that allows you to inspect visually the frequency distribution of the collection of integers, producing a chart

Algorithm in pseudocode

Algorithm: DisplayHistogram(int Array A)

**for** i := 0 **to** n **do**

**if** A[i] >= 1 **and** A[i] <= 10 **then**

X[0] := X[0] + 1

**Elseif** A[i]>= 11 **and** A[i]<= 20 **then**

X[1] := X[1] + 1

**Elseif** A[i]>= 21 **and** A[i]<= 30 **then**

X[2] := X[2] + 1

**Elseif** A[i]>= 31 **and** A[i]<= 40 **then**

X[3] := X[3] + 1

**Elseif** A[i]>= 41 **and** A[i]<= 50 **then**

X[4] := X[4] + 1

**Elseif** A[i]>= 51 **and** A[i]<= 60 **then**

X[5] := X[5] + 1

**Elseif** A[i]>= 61 **and** A[i]<= 70 **then**

X[6] := X[6] + 1

**Elseif** A[i]>= 71 **and** A[i]<= 80 **then**

X[7] := X[7] + 1

**Elseif** A[i]>= 81 **and** A[i]<= 90 **then**

X[8] := X[8] + 1

**Elseif** A[i]>= 91 **and Case** <= 100 **then**

X[9] := X[9] + 1

**endif**

**endfor**

counter := 1

**for** k **in** X **do**

**Case** based on counter

**Case** 1 **then**

print(“1-10 “)

**Case** 2 **then**

print(“11-20 “)

**Case** 3 **then**

print(“21-30 “)

**Case** 4 **then**

print(“31-40 “)

**Case** 5 **then**

print(“41-50 “)

**Case** 6 **then**

print(“51-60 “)

**Case** 7 **then**

print(“61-70 “)

**Case** 8 **then**

print(“71-80 “)

**Case** 9 **then**

print(“81-90 “)

**Case** 10 **then**

print(“91-100 “)

**Endcase**

counter := counter + 1

**for** j := 0 **to** k **do**

**print(“\*”)**

**endfor**

**endfor**

Run-Time Complexity

n = A.length m = X[k]

t(n) =

=

t(n) = +

t(n) = 20n + 10n

t(n) = 30n

O(n)

Source Code (Java)

/\*

\* Given an array (numbers) of size n, where each element of the array

\* is an integer in the range 1 to 100 inclusive, create and display a

\* histogram that allows you to inspect visually the frequency distribution

\* of the collection of integers, producing a chart

\*/

package histogram;

import java.util.Random;

/\*\*

\* 19/11/2016

\* @author Bijan G.A (100125463)

\*/

public class Histogram {

public static void displayHistogram(int[] numbers){

int[] frequency = new int[10];

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

if(numbers[i] >= 1 && numbers[i] <= 10){

frequency[0]++;

} else if(numbers[i] >= 11 && numbers[i] <= 20){

frequency[1]++;

} else if(numbers[i] >= 21 && numbers[i] <= 30){

frequency[2]++;

} else if(numbers[i] >= 31 && numbers[i] <= 40){

frequency[3]++;

} else if(numbers[i] >= 41 && numbers[i] <= 50){

frequency[4]++;

} else if(numbers[i] >= 51 && numbers[i] <= 60){

frequency[5]++;

} else if(numbers[i] >= 61 && numbers[i] <= 70){

frequency[6]++;

} else if(numbers[i] >= 71 && numbers[i] <= 80){

frequency[7]++;

} else if(numbers[i] >= 81 && numbers[i] <= 90){

frequency[8]++;

} else if(numbers[i] >= 91 && numbers[i] <= 100){

frequency[9]++;

}

}

int counter = 1;

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

switch(counter){

case 1:

System.out.print("1-10 ");

break;

case 2:

System.out.print("11-20 ");

break;

case 3:

System.out.print("21-30 ");

break;

case 4:

System.out.print("31-40 ");

break;

case 5:

System.out.print("41-50 ");

break;

case 6:

System.out.print("51-60 ");

break;

case 7:

System.out.print("61-70 ");

break;

case 8:

System.out.print("71-80 ");

break;

case 9:

System.out.print("81-90 ");

break;

case 10:

System.out.print("91-100 ");

break;

}

counter++;

for(int j = 0; j < frequency[i]; j++){

System.out.print("\*");

}

System.out.println("");

}

}

public static void main(String[] args) {

int[] numbers = new int[200];

Random random = new Random();

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

numbers[i] = random.nextInt(100) + 1;

}

displayHistogram(numbers);

}

}

Output

run:

1-10 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

11-20 \*\*\*\*\*\*\*\*\*\*\*

21-30 \*\*\*\*\*\*\*\*

31-40 \*\*\*\*\*\*\*\*\*\*\*\*\*

41-50 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

51-60 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

61-70 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

71-80 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

81-90 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

91-100 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BUILD SUCCESSFUL (total time: 0 seconds)

## Question 2: Sometimes, we wish to examine all permutations of a collection of n distinct elements. ( Coursework sheet, Description of the Assignment, 2). Design a recursive algorithm based on the above ideas to determine a list of (string representations of) all of the permutations of a given list of n distinct elements.

Algorithm in pseudocode

Algorithm: Permutations(List A, int n, String head, List Prems)

**Global String str = “”;**

**If** A.length := 0 **then**

**return** A

**Endif**

**For** i in A **then**

**str := str +** i

**Endfor**

**If** str.length == 1 **then**

**Return** A

**Endif**

**If** n+1 > str.length **then**

**return** Prems

**endif**

**If** str.length := 2 **then**

**String** container **= "";**

Prems **add** head

**If** str.charAt(n) == str.charAt(0) **then**

Container := container + str.charAt(1)

**Else then**

Container := container + str.charAt(0)

**Endif**

Perms.last\_element := Perms.last\_element + str.charAt(n)

Perms.last\_element := Perms.last\_element + container

Permutations(A, n + 1, head, prems)

**Else**

Head := head + str.charAt(n)

List new\_list = A - A[n]

List X = Permutations(new\_list, n, head, Perms)

**If** head.length > 1 **then**

Permutations(A , n+1, head.substring(0, head.length() - 1), X)

**Else then**

Permutations(A, n + 1, “”, X)

**Endif**

**Endif**

**Return** perms

Source Code (Java)

/\*

\* Sometimes, we wish to examine all permutations of a collection of n distinct

\* elements. ( Coursework sheet, Description of the Assignment, 2).

\* Design a recursive algorithm based on the above ideas to determine a

\* list of (string representations of) all of the permutations of a given

\* list of n distinct elements.

\*/

package permutations;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* 21/11/2016

\* @author Bijan

\*/

public class Permutations {

/\*

\* A function to compute all the possible permutations of a list of elemets

\* Returns - A list of all possible permutations

\* Parameters ->

\* -List list: A list of random elements

\* -int n: The index of the element that is the head

\* -String head: The head of a possible permutation

\* -List<String> perms: A list to be returned with all the possible permutations

\*/

public static List list\_of\_perms(List list, int index, String head, List<String> perms){

// If there is no content in the given list, there is no permutation

if(list.size() == 0){

System.out.println("List does not contain any elements");

return list;

}

// Making a string out of the elements of the list given

String str = "";

for(int i = 0; i < list.size(); i++){

str += list.get(i);

}

if(str.length() == 1){

return list;

}

// If the current index is more than the number of elements, return perms

if(str.length() < index + 1){

return perms;

} else {

/\* If there are 2 characters in the String made out of the given

\* list, add the character that we are given the index of to the last,

\* element of the list permsand then add the other one and add 1 to

\* the index

\*/

if(str.length() == 2){

perms.add(head);

String container = "";

if(str.charAt(index) == str.charAt(0)){

container += str.charAt(1);

} else {

container += str.charAt(0);

}

int last\_index\_of\_perms = perms.size() - 1;

String last\_element\_of\_perms = perms.get(last\_index\_of\_perms);

last\_element\_of\_perms += str.charAt(index);

last\_element\_of\_perms += container;

perms.set(last\_index\_of\_perms, last\_element\_of\_perms);

list\_of\_perms(list, index + 1, head, perms);

} else {

/\* If there are more than 2 characters in the String made out of

\* the given list, make the element that we are given the index of

\* the head, and make a new list without the mentioned element

\*/

head += str.charAt(index) ;

ArrayList new\_list = new ArrayList<String>();

String new\_str = "";

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

if(str.charAt(index) == str.charAt(i)){

continue;

} else {

new\_list.add(str.charAt(i));

}

}

List X = list\_of\_perms(new\_list, 0, head, perms);

if(head.length() > 1){

list\_of\_perms(list, index + 1, head.substring(0, head.length() - 1), X);

} else {

list\_of\_perms(list, index + 1, "", X);

}

}

}

return perms;

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

// TESTING

List list = new ArrayList<String>();

list.add("a");

list.add("b");

list.add("c");

// list.add("d");

// list.add("e");

// list.add("f");

// list.add(1);

// list.add(2);

// list.add(3);

// list.add(4);

// list.add(5);

// list.add(6);

List perms = new ArrayList<String>();

List permutations = list\_of\_perms(list, 0, "", perms);

// Printing the results

System.out.print("All permutations: (");

for(int i = 0; i < permutations.size(); i++){

System.out.print(permutations.get(i));

if(i != permutations.size() - 1){

System.out.print(", ");

} else {

System.out.println(")");

}

}

System.out.println("Total number of permutations: " + permutations.size());

}

}

Output

run:

All permutations: (abc, acb, bac, bca, cab, cba)

Total number of permutations: 6

BUILD SUCCESSFUL (total time: 0 seconds)

## Question 3: Draw diagrams to illustrate your answers to this exercise. Insert the following integers in the order given into a BST. Do not perform any re-balancing operations. 37,25,45,22,39,19,30,50,27,34

Binary Search Tree



## Insert 28 into the resulting tree and illustrate the operations necessary to re-balance the tree.



**RIGHT ROTATION (Right subtree of 25)**



**LEFT ROTATION (Left subtree of 37)**



**RIGHT ROTATION (Left subtree of 27)**



## Now delete 34 from the resulting tree and again illustrate the operations necessary to re-balance the tree.

