Yahoo algorithm

1. TwoSum

Given an array of integers, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

You may assume that each input would have exactly one solution.

Input: numbers={2, 7, 11, 15}, target=9

Output: index1=1, index2=2

```
public class TwoSum {
    public static int[] twoSum(int[] numbers, int target) {
        int[] num = numbers.clone();
        Arrays.sort(num);
         int i = 0;
         int j = num.length-1;
         ArrayList<Integer> index = new ArrayList<Integer>();
         int[] result = new int[2];
        while(i<j) {
            int s = num[i]+num[j];
            if(s==target){
                for(int k=0;k<num.length;k++){
                    if(num[i]==numbers[k]){
                        index.add(k+1);
                    } else if(num[j]==numbers[k]){
                        index.add(k+1);
                    }
                }
                i++;
                j--;
            } else if(s<target){
                i++;
            } else if(s>target){
                j--;
            }
        }
        result[0]=index.get(0);
        result[1]=index.get(1);
        return result;
    }

    public static void main(String[] args) {
         int[] numbers = {0, 2, 4, 0};
```

```
            int target = 0;

            int[] index2 = new int[2];

            index2 = twoSum(numbers, target);

            System.out.println("index1=" + index2[0] + ", index2=" + index2[1]);
        }
}

//HashMap
public static int[] twoSum(int[] numbers, int target) {
        HashMap<Integer, Integer> num = new HashMap<Integer, Integer>();
        int[] result = new int[2];
        for(int i=0;i<numbers.length;i++){
            num.put(numbers[i], i);
        }
        for(int i=0;i<numbers.length;i++){
            int re = target-numbers[i];
            if(num.get(re)!=null && num.get(re)!=i){
                result[0]=i+1;
                result[1]=num.get(re)+1;
            }
        }

        if(result[0]>result[1]){
            int temp = result[0];
            result[0]=result[1];
            result[1]=temp;
        }
        return result;
    }
```

2. Lowest Common Ancestor in a Binary Tree

```
public class Node {
    public int data;
    public Node right;
    public Node left;

    public Node(int data) {
        this.data = data;
    }
}
public static Node lowestCommonAncestor(Node root, Node a, Node b) {
```

```
        if (root == null) {
            return null;
        }

        if (root.equals(a) || root.equals(b)) {
            // if at least one matched, no need to continue
            // this is the LCA for this root
            return root;
        }

        Node l = lowestCommonAncestor(root.left, a, b);
        Node r = lowestCommonAncestor(root.right, a, b);

        if (l != null && r != null) {
            return root;    // nodes are each on a seaparate branch
        }

        // either one node is on one branch,
        // or none was found in any of the branches
        return l != null ? l : r;
    }

    //parent pointer
    int height(Node n){
       int h=-1;
       while(n!=null){h++;n=n.parent;}
       return h;
    }
    Node LCA(Node n1, Node n2){
       int discrepancy=height(n1)-height(n2);
       while(discrepancy>0) {n1=n1.parent;discrepancy--;}
       while(discrepancy<0) {n2=n2.parent;discrepancy++;}
       while(n1!=n2){n1=n1.parent();n2=n2.parent();}
       return n1;
    }
    /*
    Node LCA(Node n1, Node n2, Node CA){
      while(true){
       if(n1.val<CA.val & n2.val<CA.val) CA=CA.left;
       else if (n1.val>CA.val & n2.val>CA.val) CA=CA.right;
       else return CA;
      }
    }*/
```

3. Reverse linked list

```
//iterative
public reverseListIteratively (Node head){
      if (head == NULL || head.next == NULL)
            return;    //empty or just one node in list

      Node Second = head.next;

      //store third node before we change
      Node Third = Second.next;

      //Second's next pointer
      Second.next = head;    //second now points to head
      head.next = NULL;    //change head pointer to NULL

      //only two nodes, which we already reversed
      if (Third == NULL)
            return;

      Node CurrentNode = Third;

      Node PreviousNode = Second;

      while (CurrentNode != NULL) {
            Node NextNode = CurrentNode.next;

            CurrentNode.next = PreviousNode;

/*    repeat the process, but have to reset
        the PreviousNode and CurrentNode
*/

            PreviousNode = CurrentNode;
            CurrentNode = NextNode;
      }

      head    = PreviousNode; //reset the head node
}

//recursive
public void recursiveReverse(Node currentNode ) {
       //check for empty list
      if(currentNode == NULL)
            return;
```

```
/* if we are at the TAIL node:
       recursive base case:
 */
      if(currentNode.next == NULL) {
      //set HEAD to current TAIL since we are reversing list
            head = currentNode;
            return; //since this is the base case
      }

      recursiveReverse(currentNode.next);
      currentNode.next.next = currentNode;
      currentNode.next = null; //set "old" next pointer to NULL
}
```

4. BFS & DFS

```
Class Main {
      public void bfs()
      {
            // BFS uses Queue data structure
            Queue queue = new LinkedList();
            queue.add(this.rootNode);
            printNode(this.rootNode);
            rootNode.visited = true;
            while(!queue.isEmpty()) {
                  Node node = (Node)queue.remove();
                  Node child=null;
                  while((child=getUnvisitedChildNode(node))!=null) {
                        child.visited=true;
                        printNode(child);
                        queue.add(child);
                  }
            }
            // Clear visited property of nodes
            clearNodes();
      }

      public void dfs() {
            // DFS uses Stack data structure
            Stack stack = new Stack();
            stack.push(this.rootNode);
            rootNode.visited=true;
            printNode(rootNode);
            while(!stack.isEmpty()) {
```

```
                Node node = (Node)s.peek();
                Node child = getUnvisitedChildNode(n);
                if(child != null) {
                    child.visited = true;
                    printNode(child);
                    s.push(child);
                }
                else {
                    s.pop();
                }
            }
            // Clear visited property of nodes
            clearNodes();
        }
}


Class Node {
    Char data;
    Public Node(char c) {
        this.data=c;
    }
}
```

5. Print Matrix Diagonally

```
public class B {
    public static void printDiagnolly(int[][] matrix, int rowSize, int colSize){
        for(int row = 0; row < rowSize; row++){
            int col = 0, rowCursor = row;
            while(rowCursor >= 0){
                System.out.print(matrix[rowCursor--][col++]+" ");
            }
            System.out.println("");
        }

        for(int col = 1; col < colSize; col++){
            int row = rowSize-1, colCursor = col;
            while(colCursor < colSize){
                    System.out.print(matrix[row--][colCursor++]+" ");
            }
            System.out.println("");
        }
    }
```

```
public static void printDiagnolly2(int[][] matrix, int rowSize, int colSize){
    for(int col = 0; col < colSize; col++){
        int row = 0, colCursor = col;
        while(colCursor >= 0 && row<rowSize){
            System.out.print(matrix[row++][colCursor--]+" ");
        }
        System.out.println("");
    }

    for(int row = 1; row < rowSize; row++){
        int col = colSize-1, rowCursor = row;
        while(rowCursor < rowSize){
                System.out.print(matrix[rowCursor++][col--]+" ");
        }
        System.out.println("");
    }
}

public static void main(String[] args) {
    int[][] matrix = new int[][]{
    { 1, 2, 3, 4, 2},
    { 5, 6, 7, 8, 2},
    { 9,10,11,12, 2},
    {13,14,15,16, 2}
    };
    int rowSize = matrix.length;
    int colSize = matrix[0].length;
    //printDiagnolly(matrix, rowSize, colSize);
    printDiagnolly2(matrix, rowSize, colSize);
}
```

6. Remove Duplicates from Sorted Array & II

```
// Create an array with all unique elements
public static int[] removeDuplicates(int[] A) {
    if (A.length < 2)
        return A;

    int j = 0;
    int i = 1;

    while (i < A.length) {
        if (A[i] == A[j]) {
            i++;
        } else {
```

```
                j++;
                A[j] = A[i];
                i++;
            }
        }

        int[] B = Arrays.copyOf(A, j + 1);

        return B;
    }

    public static void main(String[] args) {
        int[] arr = { 1, 2, 2, 3, 3 };
        arr = removeDuplicates(arr);
        System.out.println(arr.length);
    }
    //II
    public class Solution {
        public int removeDuplicates(int[] A) {
            if (A.length <= 2)
                return A.length;

            int prev = 1; // point to previous
            int curr = 2; // point to current

            while (curr < A.length) {
                if (A[curr] == A[prev] && A[curr] == A[prev - 1]) {
                    curr++;
                } else {
                    prev++;
                    A[prev] = A[curr];
                    curr++;
                }
            }

            return prev + 1;
        }
    }
```

7. Generate prime numbers

```java
package com.javacodegeeks.snippets.basics;

public class GeneratePrimeNumbersWithForLoop {

    public static void main(String[] args) {

        int max = 100;

        System.out.println("Generate Prime numbers between 1 and " + max);

        // loop through the numbers one by one
        for (int i = 1; i<max; i++) {

            boolean isPrimeNumber = true;

            // check to see if the number is prime
            for (int j = 2; j < i; j++) {
                if (i % j == 0) {
                    isPrimeNumber = false;
                    break; // exit the inner for loop
                }
            }

            // print the number if prime
            if (isPrimeNumber) {
                System.out.print(i + " ");
            }
        }

    }

}
```