Esercizio 4

Implementare una versione iterativa della DFS usando uno stack.

Esercizio 4

```
public class IterativeDFS {
  private boolean[] marked;

  public void dfsUsingStack(Graph G, int v){
      marked = new boolean[G.V()];

      for (int i = 0; i<G.V();i++) {
          marked[i] = false;
      }

      Stack<Integer> stack = new Stack<Integer>();
      stack.push(v);

      while(!stack.isEmpty()) {
          int vertex = stack.pop();
          if (!marked[vertex]){
              System.out.println(vertex);
              marked[vertex] = true;
          }

          for (int w: G.adj[vertex]) {
              if (!marked[w]) {
                  stack.push(w);
              }
          }
      }
  }
}
```

Esercizio 5

Implementare un algoritmo per eseguire lo swap di due variabili di tipo intero, senza utilizzare variabili temporanee.

Esercizio 5

```java
public class Swap {

    public void swap(Integer a, Integer b) {
        a = a + b;

        // Dal momento che a = (a + b) -> b = (a + b) - b = a
        b = a - b;

        // Dal momento a = (a + b) e b = a -> a = (a + b) - a = b
        a = a - b;
    }

}
```

Esercizio 6

Implementare un algoritmo ricorsivo che data una Stringa verifichi che essa sia palindroma. L'algoritmo deve avere complessità temporale sublineare.

Esercizio 6

```
public class Palindroma {

  public static boolean palindroma(String s) {
      if (s.length() < 2)
          return true;
      if (s.charAt(0) == s.charAt(s.length() - 1))
          return palindroma(s.substring(1, s.length() - 1));
      else
          return false;
  }

}
```

Esercizio 7

Implementare un algoritmo che date due Stringhe verifichi che la seconda stringa sia un anagramma della prima. La complessità temporale dell'algoritmo deve essere minore di $O(n^2)$

Esempio: "calendario" e "locandiera"

Esercizio 7

```
public class Anagram {

public static boolean isAnagram(String word, String anagram){

    if (word.length != anagram.length){
        return false;
    }
    char[] charFromWord = word.toCharArray();
    char[] charFromAnagram = anagram.toCharArray();

    // I due array vengono ordinati
    Arrays.sort(charFromWord);
    Arrays.sort(charFromAnagram);

    // verifico che i due array ordinati contengano gli stessi
    // caratteri nelle stesse posizioni
    for (int i = 0; i<charFromWord.length; i++){
          if (charFromWord[i] != charFromAnagram[i]){
                return false;
          }
    }

    return true;

}

}
```

Esercizio 8

Dato un array di n interi, A tale che

$$\forall\, i = 0, \dots, n-1 \; \mathrm{A[i]} \in \{0,1, \dots, k\} \; con \; k < n$$

Implementare un algoritmo con complessità temporale minore di O(n log n) che ordini l'array A.

Esercizio 8

```java
public class CountingSort{

public static void countingSort(int[] input, int k) {
  // Definisco un array di contatori con k+1 posizioni
  int counter[] = new int[k + 1];

  // Conteggio le occorrenze di ogni elemento dell'array
  for (int i : input) {
      // Utilizzo l'elemento dell'array orignario
      // come indice per incrementare il conteggio
      counter[i]++;
  }

  // ordino l'array utilizzando le occorrenze di ogni elemento
  int n = 0;
  for (int i = 0; i < counter.length; i++) {
      while (counter[i] > 0) {
          input[n++] = i;
          counter[i]--;
      }
  }
}

}
```

Esercizio 9

Implementare un algoritmo che ricevuto in ingresso un array di interi di n elementi distinti e un intero x, restituisca:

- **true** nel caso in cui all'interno dell'array esistano due elementi la cui somma sia pari a x, indicando gli indici di tali elementi;
- **false** altrimenti.

L'algoritmo deve avere complessità temporale O(n).

Esercizio 9

```
class Main{
  public boolean findPair(int[] A, int x)
  {
      // utilizzo una tabella di hash
      Map<Integer, Integer> map = new HashMap<Integer, Integer>();


      for (int i = 0; i < A.length; i++)
      {
          // verifico se la coppia (A[i], x-A[i]) esiste
          if (map.containsKey(x - A[i]))
          {
              System.out.println("La somma degli elementi di posizione " +
                                 map.get(x - A[i]) + " e " + i +" è "+ x);
              return true;
          }

          // memorizzo l'indice dell'elemento corrente nell'array
          map.put(A[i], i);
      }

      System.out.println("Non esistono coppie la cui somma sia "+x);
      return false;
  }

}
```