שאלות

שאלה 1

כתוב פונקציה סטטית שמקבלת מחרוזת אשר מכילה רק את התווים "(,), [,], {, }" ומחזירה true אם"ם המחרוזת תקינה מתמטית.

דוגמא: {{[()]}}{()[]} → תקין

)({}[]() → לא תקין

שאלה 2

כתוב פונקציה המקבלת כקלט מחרוזת של תווים, שיש בה מספר עם נקודה עשרונית ובדוק, באמצעות מחסנית אחת, אם הספרות שאחרי הנקודה העשרונית מופיעות בסדר הפוך מאלה שלפני הנקודה העשרונית.

דוגמא: 2345.5432 → תקין

26.75 → לא תקין

שאלה 3

כתוב פונקציה המקבלת כקלט מחסנית של מספרים שלמים וממיינת את המחסנית בסדר עולה בעזרת מחסנית עזר ומחזירה כפלט את המחסנית הממויינת.

שאלה 4

כתוב פונקציה סטטית המקבלת מחרוזת ובודקת באמצעות תור אחד ומחסנית אחת האם המחרוזת הנתונה מהווה פלינדרום (סימטרית). הפונקציה מחזירה אמת אם המחרוזת מהווה פלינדרום, אחרת היא מחזירה שקר.

שאלה 5

הציעו כיצד ניתן לממש תור בעזרת 2 מחסניות
לאחר מכן ממשו זאת.

שאלה 6

שאלה בנושא מיון:
נתונה מחרוזת, ונתון טלפון עם 9 מקשים, על כל מקש מופיעות 3 אותיות, עבור האות הראשונה נצטרך להקיש פעם אחת על מנת להקליד אותה, עבור האות השניה פעמיים ועבור האות השלישית שלוש. לא ידוע באיזה סדר האותיות מסודרות על המקשים. מצאו את מספר ההקשות המינימלי עבור מחרוזת s.

שאלה 7

מחרוזת סימטרית כפולה str מוגדרת כך: $str = ww^r ww^r$ כאשר w היא מילה כלשהי ו w^r היא ההפכית שלה

לדוגמא - $w = abcd\ ,\ w^r = dcba$

קלט תקין - $str = abcddcbaabcddcba$

קלט לא תקין - $str = abcddcbarfggfr$

כתבו פונקציה הקובעת האם מחרוזת היא מחרוזת סימטרית כפולה.(תחזירו $True\ /\ False$)

פתרונות

פתרון שאלה 1

```
 // Function to check if given expression is balanced or not
public static boolean Question1(String s) {
    // take a empty stack of characters
    Stack<Character> st = new Stack<Character>();

    // traverse the input expression
    for(int i=0, n = s.length(); i<n ; i++) {

      char curr = s.charAt(i);

      // if current char in the expression is a opening brace,
      // push it to the stack
      if(curr == '(' || curr == '{' || curr == '[' ) {
        st.push(curr);
      }
      else { // Its } or ) or ]

        // return false if mismatch is found (i.e. if stack is
        // empty, the number of opening braces is less than number
        // of closing brace, so expression cannot be balanced)
        if(st.isEmpty())
          return false;

        // pop character from the stack
        char top = st.pop();
        if( top == '(' && curr != ')' ||
          top == '[' && curr != ']' ||
          top == '{' && curr != '}' )
          return false;
      }
    }
    // expression is balanced only if stack is empty at this point
    return st.isEmpty();
  }
```

פתרון שאלה 2

```
private static boolean Question2(double input) {
    Stack<Character> s = new Stack<Character>();
    String inputString = input + "";
    int index = inputString.indexOf('.') + 1;
    for(int i=0; inputString.charAt(i) != '.' ; i++)
      s.push(inputString.charAt(i));

    for(int i=index; i<inputString.length(); i++)
      if(s.pop() != inputString.charAt(i)) return false;

    return s.isEmpty();
  }
```

פתרון שאלה 3

```
public static Stack<Integer> sortstack(Stack<Integer>
                                        input)
    {
        Stack<Integer> tmpStack = new Stack<Integer>();
        while(!input.isEmpty())
        {
            // pop out the first element
            int tmp = input.pop();

            // while temporary stack is not empty and
            // top of stack is greater than temp
            while(!tmpStack.isEmpty() && tmpStack.peek()
                                              > tmp)
            {
                // pop from temporary stack and
                // push it to the input stack
            input.push(tmpStack.pop());
            }

            // push temp in temporary of stack
            tmpStack.push(tmp);
        }
        return tmpStack;
    }
```

פתרון שאלה 4

```
  public static boolean palindrom(String str) {

    Stack<Character> s = new Stack<Character>();
    Queue<Character> q = new LinkedList<Character>();

    for(int i=0; i<str.length(); i++) {
      s.push(str.charAt(i));
      q.add(str.charAt(i));
    }

    while(!s.isEmpty()) {
      if(!s.pop().equals(q.poll()) ) return false;
    }
    return true;


  }
}
```

פתרון שאלה 5

```
public class Question5 {

  public class TwoStacksQueue<T> {
    Stack<T> stack1;
    Stack<T> stack2;

    // Constructor to initialize queue
    public TwoStacksQueue() {
      stack1 = new Stack<T>();
      stack2 = new Stack<T>();
    }

    // Utility function to add an item to the queue
    public void enQueue(T element)  {
      stack1.add(element);
    }
    // Utility function to remove front element from the queue
    public T deQueue() {
      if(!stack2.isEmpty()) return stack2.pop();
      else {
        if(stack1.isEmpty()) throw new NoSuchElementException("No
elements present in Queue");
        while(!stack1.isEmpty())
          stack2.push(stack1.pop());

        return stack2.pop();
      }
    }

    // Utility function to return front element in the queue
    public T peek() {
      if(!stack2.isEmpty()) return stack2.peek();
      else {
        if(stack1.isEmpty()) throw new NoSuchElementException("No
elements present in Queue");
        while(!stack1.isEmpty())
```

```
          stack2.push(stack1.pop());
        return stack2.peek();
      }

    }
    public int size() { return stack1.size() + stack2.size(); }
    // Utility function to check if the queue is empty or not
    public boolean isEmpty() { return stack1.isEmpty() &&
stack2.isEmpty(); }
  }
  public static void main(String[] args) {
    // create a queue
    Question5.TwoStacksQueue<Integer> q = new Question5().new
TwoStacksQueue<Integer>();

    q.enQueue(1);
    q.enQueue(2);
    q.enQueue(3);

    System.out.println("Front element is: " + q.peek());
    System.out.println(q.deQueue());
    System.out.println("Front element is: " + q.peek());

    System.out.println("Queue size is " + q.size());

    System.out.println(q.deQueue());
    System.out.println(q.deQueue());
    System.out.println(q.deQueue());


    if (q.isEmpty())
      System.out.println("Queue Is Empty");
    else
      System.out.println("Queue Is Not Empty");
  }
}
```

פתרון שאלה 6

נספור את מספר המופעים של כל אות. למיין את התוצאה. להתחיל לסכום את המופעים כאשר 9 המקומות הראשונים במערך המופעים יוכפלו ב1, ה9 השניים יוכפלו ב2 והשאר ב3.

נחחזר את הסכום .

סיבוכיות: n = גודל המחרוזת, עוברים על המחרוזת פעם אחת וממיינים מערך בגודל קבוע, לכן נקבל $O(n)$.

פתרון שאלה 7

מקווה שאספיק גם לרשום לכם תשובה תכנותית אבל למקרה שלא הינה הפתרון שאני מציע:

נאתחל 2 מחסניות S_1 , S_2. נמצא את הגודל של w ע"י חלוקת המחרוזת ל4.

כדי לוודא שאכן נחזיר $true$ רק עבור מחרוזות שהן אכן סימטריות כפולות ולא מחרוזות כמו wm^Rmw^R או ww^Rmm^R כאשר $w \neq m$ (כלומר הם 2 מילים שונות) , נפעל כך:

1. נעבור על הרבע הראשון של המחרוזת(כלומר על w) ונוסיף ל 2 המחסניות S_1 , S_2 את w. ונקבל מצב ביניים:

$S_1 = [w]$, $S_2[w]$

2. נעבור על הרבע השני של המחרוזת(כלומר על w^R) ונבדוק עבור כל אותו שנקרא מהמחרוזת שהיא שווה לאות שנוציא ממחסנית $S_1(pop)$. במקביל נוסיף כל אות שקראנו אל המחסנית S_2. ונקבל מצב ביניים:

$S_1 = []$, $S_2\left[ww^R\right]$.

3. נעבור על הרבע השלישי של המחרוזת(כלומר על w) ונבדוק עבור כל אותו שנקרא מהמחרוזת שהיא שווה לאות שנוציא ממחסנית $S_2(pop)$. ונקבל מצב ביניים:

$S_1 = []$, $S_2[w]$.

4. נעבור על הרבע הרביעי של המחרוזת(כלומר על w^R) ונבדוק עבור כל אותו שנקרא מהמחרוזת שהיא שווה לאות שנוציא ממחסנית $S_2(pop)$. ונקבל לבסוף.

$S_1 = []$, $S_2[]$.

אם אחד מהשלבים לא מתקימים(בדיקה פשוטה לגבי האם המחסניות שצריכות ליהיות ריקות אכן ריקות) נחזיר false אחרת true.