Legajo:……. Nombre y Apellido: ……

Primer Parcial de Estructura de Datos y Algoritmos

Ejer 1	Ejer 2	Ejer 3	Nota
/4	/4	/2	/10

Duración: 2 horas y 10 minutos

Condición Mínima de Aprobación. Deben cumplir estas 2 condiciones:

- Sumar **no menos de cuatro** puntos
- Sumar por lo menos 3 puntos entre el ejercicio 1 y 2.

Muy Importante

Al terminar el examen deberían subir los siguientes 2 grupos de archivos, según lo explicado en los ejercicios:

1) Clases Java: Corredores.java, StringEval.java según lo pedido. Si hay código auxiliar, entregarlo también.

2) Para todos los ejercicios que no consistan en implementar código Java y pidan calcular complejidades, dibujar matrices, completar cuadros, hacer seguimientos, etc. pueden optar por alguna de estas estrategias:
   a. O completar este documento y subirlo también
   b. O directamente resolverlo en hojas de papel y sacarle fotos (formato jpg, png o pdf) y subir todas las imágenes.

Se considera una entrega aquella que corresponda a “un upload a Campus” para esta actividad, donde pondrán todos los archivos que deseen entregar.

Si bien pueden subir múltiples veces, sólo se corrige la última entrega realizada dentro del horario que dura la actividad.

Legajo:……. Nombre y Apellido: ……

Ejercicio 1

Se corrió la maratón de Buenos Aires y tenemos los datos de tiempo de finalización de cada participante. Como parte del staff del comité organizador, tenemos que poder responder muchas preguntas de la prensa.

En particular, a los periodistas les interesa saber **cuántos participantes terminaron en un rango específico de tiempos (intervalo cerrado)**. No todos los periodistas quieren saber el mismo rango, por lo que tomamos notas de sus pedidos para poder **responderles a todos juntos**.

Para esto, queremos armar la función **tiemposEntre** que, recibiendo los tiempos de los corredores y los pedidos de los periodistas, devuelva todas las respuestas a esos pedidos. El arreglo que devuelve debe ser la respuesta a cada pedido, respetando el orden de los mismos. El método a implementar es:

int[] tiemposEntre(int[] tiempos, Pedido[] p)

Todos los corredores parten al mismo tiempo y los tiempos se registran en minutos de acuerdo al orden de llegada por lo que el **arreglo de tiempos estará ordenado y puede tener repetidos**.

La clase Pedido luce así:

```
class Pedido {
   public int desde;
   public int hasta;

   public Pedido(int desde, int hasta) {
      this.desde = desde;
      this.hasta = hasta;
   }
}
```

Las variables *desde* y *hasta* representan los tiempos a considerar en el pedido, incluyendo ambos extremos.

Por ejemplo, para el siguiente código:

Legajo:……. Nombre y Apellido: ……

```
public static void main(String[] args) {
    Corredores c = new Corredores();

    Pedido[] pedidos = new Pedido[] {
        new Pedido(200, 240),
        new Pedido(180, 210),
        new Pedido(220, 280),
        new Pedido(0, 200),
        new Pedido(290, 10000)
    };

    int[] tiempos = new int[] {
        192,
        200,
        210,
        221,
        229,
        232,
        240,
        240,
        243,
        247,
        280,
        285
    };

    int [] respuestas = c.tiemposEntre(tiempos, pedidos);
    for(int i=0; i< respuestas.length; i++) {
       System.out.println(respuestas[i]);
    }

  }
```

Debe imprimir lo siguiente:

```
7
3
8
2
0
```

Corresponde a:

- 7 tiempos entre 200 y 240 (incluyendo los extremos)
- 3 tiempos entre 180 y 210 (incluyendo los extremos)
- 8 tiempos entre 220 y 280 (incluyendo los extremos)
- 2 tiempos entre 0 y 200 (incluyendo los extremos)
- 0 tiempos entre 290 y 1000 (incluyendo los extremos)

Legajo:……. Nombre y Apellido: ……

Descargar de campus el archivo **Corredores.java**

Se pide:

Implementar el método **public *int[] tiemposEntre(int[] tiempos, Pedido[] p)***

de la clase *Corredores*.

Considerar:
- La cantidad de corredores puede estar entre 1 y 500,000 (**N**)

- Los tiempos pueden variar entre 1 y 1,000

- La cantidad de pedidos puede ser de 1 a 1,000,000. (**M**)

Se pide que la complejidad del método tiemposEntre sea **O (M log N)**

Ejercicio 2

Se pide implementar un evaluador de expresiones de **Strings posfijo** donde:

- **Los caracteres válidos de los operandos son a-z y A-Z.**

- Los **operadores binarios** a considerar son estos 5 (ninguno da error ni excepción)

(+) concatenación
AAAAA BBBBB + **devolvería** AAAAABBBBB

(-) borrado de substring, borra la primera ocurrencia del segundo operando en el primero
AAAAABBCCBB BB - **devolvería** AAAAACCBB

(*) intercalado de caracteres
AAA BBBBB * **devolvería** ABABABBB

Legajo:……. Nombre y Apellido: ……

(/) borrado de caracteres que borra cada uno de los caracteres del segundo operando en todas las apariciones del primero operando
AAAAABBBCCCDDDAAA AB / **devolvería** CCCDDD

(^) Intercalado especial: Para el segundo operando op2 se generan todos los prefijos de longitud 1, 2, .. hasta N siendo N la longitud del mismo. Sean $op2_1$, $op2_2$, … $op2_N$ dichos prefijos. El operador concatena el primer operando con el primer prefijo, y eso otra vez con el primer operando y con el segundo prefijo, …. hasta que finalmente concatena el primer operando con el último prefijo. En el ejemplo el segundo operando op2 es ABCD, por lo que sus prefijos son A, AB, ABC y ABCD. El primer operando op1 es EE. Por eso, el resultado en este caso es:
EE ABCD ^ **devolvería EEAEEABEEABCEEABCD**

La clase a implementar es:

```
public class StringEval {
   public String evaluate(String expression){
      // TODO: completar
   }
}
```

Se debe entregar el archivo .java con la implementación.

Ejemplos:

Postfija:
AA BB CC DEF ^ * AE / + BC -
Resultado: **AABCDCCDCCDF**

Postfija:
HOLA QUE + TAL COMO ^ ESTAS / BIEN * + BIEN -
Resultado: **HOLAQUELBCILECNOLCOMLCOMO**

Legajo:……. Nombre y Apellido: ……

Ejercicio 3

Dado el siguiente algoritmo:

```
public static int unalgoritmo(int[] array) {
    int i = 0;
    int j = 0;
    int m = 0;
    int c = 0;
    while (i < array.length) {
      if (array[i] == array[j]) {
        c++;
      }
      j++;
      if (j >= array.length) {
        if (c > m) {
          m = c;
        }
        c = 0;
        i++;
        j = i;
      }
    }
    return m;
  }
```

Determinar:

- ¿Qué hace el algoritmo **unalgoritmo** en palabras y en una oración)? - por ejemplo: encuentra el elemento más pequeño en un arreglo.
- ¿Qué orden (O grande) de complejidad temporal tiene? ¿Qué orden (O grande) de complejidad espacial tiene?
- Implementar el algoritmo

```
public static int mialgoritmo(int[] array)
```

que haga lo mismo que **unalgoritmo** pero que tenga un **orden de complejidad temporal mejor**. Calcular dicha complejidad temporal ¿Cómo es su **complejidad espacial**?