

Laboratorio Nro. 2

Fuerza Bruta (*Brute force o Exhaustive search*)

Objetivos

Diseñar algoritmos usando la técnica de diseño de búsqueda por fuerza bruta

Consideraciones iniciales

Leer la Guía



Antes de comenzar a resolver el presente laboratorio, leer la “**Guía Metodológica para la realización y entrega de laboratorios de Estructura de Datos y Algoritmos**” que les orientará sobre los requisitos de entrega para este y todos los laboratorios, las rúbricas de calificación, el desarrollo de procedimientos, entre otros aspectos importantes.

Registrar Reclamos



En caso de tener **algún comentario** sobre la nota recibida en este u otro laboratorio, pueden **enviarlo** a través de <http://bit.ly/2g4TTKf>, el cual será atendido en la menor brevedad posible.

Traducción de Ejercicios

En el GitHub del docente, encontrarán la traducción al español de los enunciados de los Ejercicios en Línea.



Visualización de Calificaciones



A través de **Eafit Interactiva** encontrarán **un enlace** que les permitirá **ver un registro de las calificaciones** que **emite el docente** para cada taller de laboratorio y según las rubricas expuestas. **Véase sección 3, numeral 3.8.**

GitHub

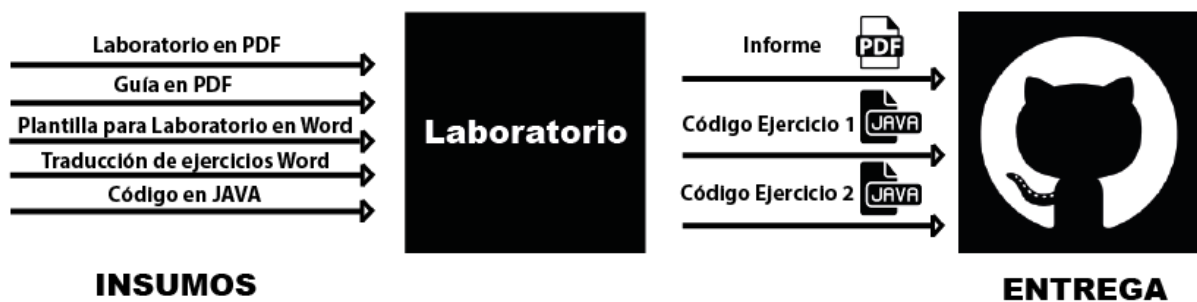


1. Crear un repositorio en su cuenta de GitHub con el nombre `st0247-suCodigoAqui`. 2. Crear una carpeta dentro de ese repositorio con el nombre `laboratorios`. 3. Dentro de la carpeta `laboratorio`, crear una carpeta con nombre `lab02`. 4. Dentro de la carpeta `lab02`, crear tres carpetas: `informe`, `codigo` y `ejercicioEnLinea`. 5. Subir el informe pdf a la carpeta `infome`, el código del ejercicio 1 a la carpeta `codigo` y el código del ejercicio en línea a la carpeta `ejercicioEnLinea`. Así:

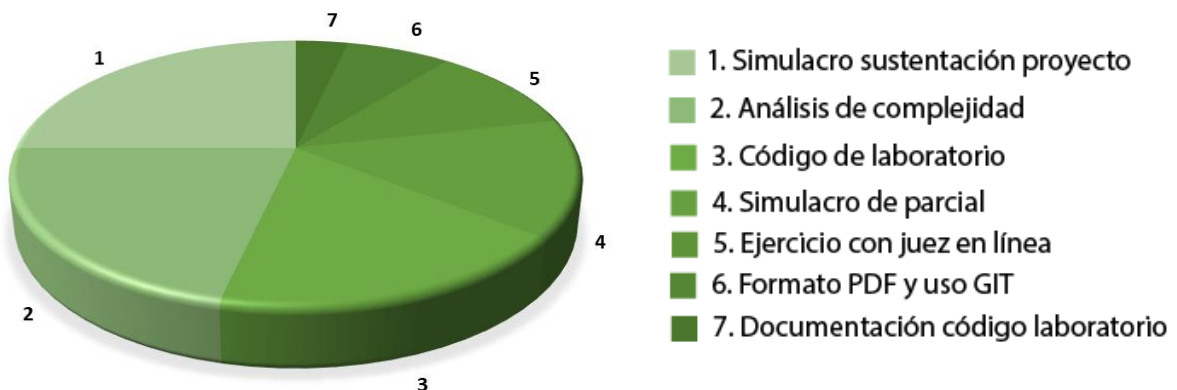
```
st0247-suCodigoAqui
  laboratorios
    lab01
      informe
      codigo
      ejercicioEnLinea
    lab02
      ...
```

Intercambio de archivos

Los archivos que **ustedes deben entregar** al docente son: **un archivo PDF** con el informe de laboratorio usando la plantilla definida, y **dos códigos**, uno con la solución al numeral 1 y otro al numeral 2 del presente. Todo lo anterior se entrega en **GitHub**.



Porcentajes y criterios de evaluación para el laboratorio



Resolver Ejercicios

1. Códigos para entregar en GitHub:



En la vida real, la documentación del software hace parte de muchos estándares de calidad como CMMI e ISO/IEC 9126



Véase Guía *en Sección 3, numeral 3.4*



Código de laboratorio en *GitHub*. Véase Guía en *Sección 4, numeral 4.24*



Documentación en *HTML*



No se reciben archivos en *.RAR* ni en *.ZIP*



En la vida real, la resolución de acertijos como las *N reinas* es una pregunta frecuente en entrevistas técnicas de grandes como compañías como Google, Microsoft y Facebook. Léase más en <http://bit.ly/2hH0xlx>

1.1 Teniendo en cuenta lo anterior Implementen el algoritmo de fuerza bruta para encontrar TODAS las soluciones de las N Reinas.

1.2 Construyan ejemplos usando JUnit para probar su implementación de las N Reinas usando fuerza bruta.

Como muestra, use los ejemplos trabajados en el taller de clase sobre fuerza bruta para el problema de las 4 reinas



NOTA: Si utilizan Python o C++, utilice una librería equivalente para pruebas unitarias en dichos lenguajes.



PISTA 1: Véase Guía, **Sección 4, numeral 4.14** “Cómo hacer pruebas unitarias en BlueJ usando JUnit”



NOTA 2: Todos los ejercicios del numeral 1 deben ser documentados en formato HTML. Véase Guía en **Sección 4, numeral 4.1** “Cómo escribir la documentación HTML de un código usando JavaDoc”

2) Ejercicios en línea sin documentación HTML en GitHub



Véase Guía en **Sección 3, numeral 3.3**



No entregar
documentación **HTML**



Entregar un archivo
en **.JAVA**



No se reciben archivos
en **.PDF**



Resolver los problemas
de **CodingBat** usando
Recursión



Código del ejercicio en línea
en **GitHub**. Véase Guía en
Sección 4, numeral 4.24



NOTA: Recuerden que, si toman la respuesta de alguna fuente, deben referenciar según el tipo de cita correspondiente. Véase *Guía en Sección 4, numerales 4.16 y 4.17*

2.1 Resuelvan el siguiente problema:

El problema de la n reinas es conocido por cualquier persona que haya estudiado *backtracking*.

En este problema usted debe contar el número de formas de ubicar n -reinas en un tablero de $n \times n$ de tal forma que no haya ni un par de reinas que se ataquen. Para hacer el problema un poco más duro (¿o más fácil?), resulta que hay unos cuadros malos en los que no se pueden poner reinas.

Por favor, tenga en cuenta que los cuadros malos no se pueden usar para bloquear un ataque de una reina, es decir, no es un muro, es sólo un cuadro malo. También considere que, incluso si dos soluciones se vuelven la misma después de hacer unas rotaciones al tablero, se deben contar como diferentes.

Por consiguiente, existen 92 soluciones en el problema tradicional de las 8 reinas.

Entrada

La entrada consiste de al menos 10 casos de prueba. Cada caso contiene un entero n ($3 < n < 15$) en la primera línea. Posteriormente, las siguientes líneas representan el tablero donde los cuadros en blanco son representados por puntos (.) y los cuadros malos son representados por asteriscos (*). El último caso es seguido de un número 0, que no debe ser procesado.

Salida

Para cada caso de prueba, imprima el número del caso y el número de soluciones

Entrada de ejemplo

8

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Entrada de ejemplo

Case 1: 92

Case 2: 1



PISTA 1: Este algoritmo será muy lento si se hace con fuerza bruta. Use una técnica de diseños de algoritmos que sea más eficiente para resolver este problema



PISTA 2: Primero defina una estructura de datos para representar los huecos



PISTA 3: Véase Guía en **Sección 4, numeral 4.13** “*Cómo usar Scanner o BufferedReader*”

3 Simulacro de preguntas de sustentación de Proyectos



Véase Guía en **Sección 3,**
Numeral 3.5



Entregar informe de
laboratorio en **PDF**



Usen la **plantilla** para
responder laboratorios



No apliquen Normas
Icontec para esto



En la vida real, las técnicas usadas para resolver muchos
problemas en Ingeniería de Sistemas son las mismas que las
existentes para las N reinas

3.1 Teniendo en cuenta lo anterior, respondan: Para resolver el problema de las N Reinas, fuera de fuerza bruta, ¿qué otras técnicas computacionales existen?



PISTA : Léase <http://bit.ly/2hvFCb2>

3.2 Tomen los tiempos de ejecución del programa implementado en el numeral 1.1 y completen la siguiente tabla. Si se demora más de 5 minutos, coloque “se demora más de 50 minutos”, no sigan esperando, podría tomar siglos en dar la respuesta, literalmente.

Valor de N	Tiempo de ejecución
4	
8	
16	
32	
N	$O(?)$



PISTA: Véase Guía en **Sección 4, numeral 4.6** “Cómo usar la escala logarítmica en Microsoft Excel 2013”

3.3 Escriban una explicación entre 3 y 6 líneas de texto del código del ejercicio en línea del numeral 2.1. Digan cómo funciona, cómo está implementado y destaquen las estructuras de datos y algoritmos usados

3.4 Expliquen con sus propias palabras la estructura de datos que utiliza para resolver el problema del numeral 2.1 y cómo funciona el algoritmo.



NOTA: Recuerden que debe explicar su implementación en el informe PDF

3.5 Calculen la complejidad del ejercicio en línea del numeral 2.1 y agréguela al informe PDF



PISTA: Véase **Guía en Sección 4, numeral 4.11** “Cómo escribir la complejidad de un ejercicio en línea”

3.6 Expliquen con sus palabras las variables (qué es ‘n’, qué es ‘m’, etc.) del cálculo de complejidad del numeral 3.5

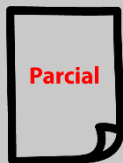
4) Simulacro de Parcial en el informe PDF



PISTA 1: Véase *Guía en Sección 4, Numeral 4.18* “Respuestas del Quiz”



PISTA 2: Lean las diapositivas tituladas “*Data Structures II: Brute Force*”, encontrarán la mayoría de las respuestas



Para este simulacro, agreguen ***sus respuestas*** en el informe PDF.



El día del Parcial no tendrán computador, JAVA o acceso a internet..

1. El problema de encontrar el **subarreglo máximo** consiste en encontrar un subarreglo contiguo dentro de un arreglo de números cuya suma sea la máxima.

Como un ejemplo, para la secuencia de números -2,1,-3,4,-1,2,1,-5,4; el subarreglo contiguo con suma máxima es 4,-1,2,1 y su suma es 6.

El siguiente algoritmo es una solución con fuerza bruta del problema. El algoritmo consiste en buscar cada posible subarreglo contiguo y luego encontrar la suma de cada uno de ellos.

```
01 int subarregloMax(int[] a) {  
02   int maximo = 0;  
03   for (int i = 0; i < a.length; i++) {  
04     int actual = 0;  
05     for (int j = i; j < a.length; j++) {  
06       actual = actual + a[j];  
07       if (_____)  
08         maximo = actual;  
09     }  
}
```

DOCENTE MAURICIO TORO BERMÚDEZ

Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627

Correo: mtorobe@eafit.edu.co

```
10 }  
11 return maximo;  
12 }
```

De acuerdo a lo anterior, resuelva lo siguiente:

- a) (10%) Complete el espacio en línea 07

- b) (10%) ¿Cuál es la complejidad asintótica, para el peor de los casos, del algoritmo? $O(\text{_____})$

2. La función ordenar es un algoritmo de ordenamiento, de menor a mayor, por fuerza bruta. Dicho algoritmo calcula todas las permutaciones posibles de un arreglo *arr* hasta encontrar una permutación donde los elementos están ordenados (es decir, cuando *estaOrdenado* retorna verdadero).

Tenga en cuenta que ordenar imprime en la pantalla el arreglo *arr* ordenado, pero no necesariamente deja el arreglo a ordenado por la forma en que está diseñado el algoritmo. Aunque esto último no es deseable, ese no es el problema de este parcial.

```
static boolean estaOrdenado(int[] a) {  
    for (int i = 0; i < a.length - 1; i++)  
        if (a[i] > a[i + 1])  
            return false;  
    return true;  
}  
static void cambiar(int[] arr, int i, int k){  
    int t = arr[i];  
    arr[i] = arr[k];  
    arr[k] = t;  
}  
static void ordenar(int[] arr, int k){  
    for(int i = k; i < arr.length; i++){  
        cambiar(arr, i, k);  
        if (estaOrdenado(arr))  
            System.out.println(  
                Arrays.toString(arr));  
    }
```

```
ordenar(_____, _____);  
cambiar(arr, k, i);  
}  
}
```

De acuerdo a lo anterior, resuelva lo siguiente:

- a) (10%) Complete los espacios vacíos en el llamado recursivo del método ordenar
_____, _____
- b) (10 %) Complete la complejidad, en el peor de los casos, del método ordenar
 $O(\text{_____})$

5. [Ejercicio Opcional] Lectura recomendada



"Quienes se preparan para el ejercicio de una profesión requieren la adquisición de competencias que necesariamente se sustentan en procesos comunicativos. Así cuando se entrevista a un ingeniero recién egresado para un empleo, una buena parte de sus posibilidades radica en su capacidad de comunicación; pero se ha observado que esta es una de sus principales debilidades..."

Tomado de <http://bit.ly/2gJKzJD>



Véase Guía en **Sección 3, numeral 3.6 y 4.20** de la Guía Metodológica, "Lectura recomendada" y "Ejemplo para realización de actividades de las Lecturas Recomendadas", respectivamente

Posterior a la lectura del texto *Anany Levitin, Introduction to the Design & Analysis of Algorithms Chapter 3: Brute Force and Exhaustive Search. Páginas 97 – 120.*, realicen las siguientes actividades que les permitirán sumar puntos adicionales:

- a) Escriban un resumen de la lectura que tenga una longitud de 100 a 150 palabras



PISTA 1: En el siguiente enlace, unos consejos de cómo hacer un buen resumen <http://bit.ly/2knU3Pv>



PISTA 2: [Aquí](#) le explican cómo contar el número de palabras en Microsoft Word

- b) Hagan un mapa conceptual que destaque los principales elementos teóricos.



PISTA: Para que hagan el mapa conceptual se recomiendan herramientas como las que encuentran en <https://cacoo.com/> o <https://www.mindmup.com/#m:new-a-1437527273469>



NOTA 1: Si desean una lectura adicional en inglés, consideren la siguiente: “Skiena, *The algorithm design manual* (2nd edition), Section 14.4. 2010”, que pueden encontrarla en biblioteca.



NOTA 2: Estas respuestas también deben incluirlas en el informe PDF

6. [Ejercicio Opcional] Trabajo en Equipo y Progreso Gradual



El trabajo en equipo es una exigencia actual del mercado. "Mientras algunos medios retratan la programación como un trabajo solitario, la realidad es que requiere de mucha comunicación y trabajo con otros. Si trabajas para una compañía, serás parte de un equipo de desarrollo y esperarán que te comuniques y trabajes bien con otras personas"

Tomado de <http://bit.ly/1B6hUDp>



Véase *Guía* en **Sección 3, numeral 3.7** y **Sección 4, numerales 4.21, 4.22 y 4.23** de la *Guía Metodológica*

- a) Entreguen copia de todas las actas de reunión usando el tablero Kanban, con fecha, hora e integrantes que participaron



PISTA: Véase *Guía en Sección 4, Numeral 4.21* “Ejemplo de cómo hacer actas de trabajo en equipo usando Tablero Kanban”

- b) Entreguen el reporte de *git*, *svn* o *mercurial* con los cambios en el código y quién hizo cada cambio, con fecha, hora e integrantes que participaron



PISTA: Véase *Guía en Sección 4, Numeral 4.23* “Cómo generar el historial de cambios en el código de un repositorio que está en *svn*”

- c) Entreguen el reporte de cambios del informe de laboratorio que se genera *Google docs* o herramientas similares



PISTA: Véase Guía en Sección 4, Numeral 4.22 “**Cómo ver el historial de revisión de un archivo en Google Docs**”



NOTA: Estas respuestas también deben incluirlas en el informe PDF

Resumen de ejercicios a resolver

1.1 Teniendo en cuenta lo anterior Implementen el algoritmo de fuerza bruta para encontrar TODAS las soluciones de las N Reinas.

1.2 Construyan ejemplos usando JUnit para probar su implementación de las N Reinas usando fuerza bruta.

2.1 Resuelvan el problema de las N Reinas con algunos cuadros malos

3.1 Para resolver el problema de las N Reinas, fuera de fuerza bruta, ¿qué otras técnicas computacionales existen?

3.2 Tomen los tiempos de ejecución del programa implementado en el numeral 1.1 y completen la siguiente tabla. Si se demora más de 5 minutos, coloque “*se demora más de 50 minutos*”, no sigan esperando, podría tomar siglos en dar la respuesta, literalmente.

3.3 Escriban una explicación entre 3 y 6 líneas de texto del código del ejercicio en línea del numeral 2.1. Digan cómo funciona, cómo está implementado y destaquen las estructuras de datos y algoritmos usados


3.4 Expliquen con sus propias palabras la estructura de datos que utiliza para resolver el problema del numeral 2.1 y cómo funciona el algoritmo.

3.5 Calculen la complejidad del ejercicio en línea del numeral 2.1 y agréguela al informe PDF

3.6 Expliquen con sus palabras las variables (*qué es ‘n’, qué es ‘m’, etc.*) del cálculo de complejidad del numeral 3.5

4. Simulacro de Parcial

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

	<p>UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS</p>	<p>Cód. ST0247</p> <p>Estructuras de Datos 2</p>
---	---	--

5. Lectura recomendada **[Ejercicio Opcional]**

6. Trabajo en Equipo y Progreso Gradual **[Ejercicio Opcional]**

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co