

# Estructuras de Datos

Andrea Rueda

Pontificia Universidad Javeriana  
Departamento de Ingeniería de Sistemas

# Profesora

Ing. Andrea Rueda

Página personal:

<https://sites.google.com/site/asderina>

Contacto:

[rueda-andrea@javeriana.edu.co](mailto:rueda-andrea@javeriana.edu.co)

- correos **deben** incluir en el asunto “[**EDD**]”, si no, se asumen como **no recibidos**.
- mensajes por correo y Teams se contestan en horario laboral (Lun-Vie 8am-6pm)

# Participantes

- ¿Nombre?
- ¿Carrera y año de ingreso?
- ¿Experiencia en programación? ¿Lenguajes?
- ¿Algo puntual que espera y/o quiere aprender en este curso?

# Proyecto Educativo PUJ

- “La relación **profesor-estudiante** constituye elemento esencial de la Comunidad Educativa ... Ha de ser una relación **honesta**, **equitativa**, **respetuosa** y de **mutua exigencia**.”
- “En esta relación, el estudiante es el **principal artífice** de su formación.”
- “El profesor deberá **conocer** a sus estudiantes, sus posibilidades y limitaciones; **estimular la participación activa** de ellos en el proceso enseñanza-aprendizaje ...”

# San Francisco Javier

Patrono de nuestra Universidad

¿A qué te invita Francisco Javier hoy?

servicio riesgo entrega amistad  
compromiso apertura respeto  
disponibilidad aventura pasión

# Acreditación ABET



**Engineering  
Accreditation  
Commission**

# Sobre el curso...

¿Qué sabemos?

¿Qué nos han dicho o hemos oído?

¿Qué esperamos?

<https://bit.ly/3ghWa1m>

# Sobre el curso...

## Objetivo

- El objetivo del curso es proveer conceptos, técnicas y métodos formales de diseño, implementación y uso de estructuras de agrupamiento de datos. En particular, se espera introducir algoritmos y operaciones que hacen uso eficiente de estructuras de datos; así como formalizar una metodología de análisis, diseño, desarrollo y validación de algoritmos.



# Sobre el curso...

## Resultados de Aprendizaje Esperados

Al finalizar el curso el estudiante estará en capacidad de:

1. Comprender el concepto de eficiencia y complejidad de algoritmos básicos.
2. Definir estructuras de datos básicas (lineales, recurrentes y no lineales), con sus ventajas y desventajas.
3. Seleccionar la estructura de datos más adecuada para representar la información en un contexto informático.

# Sobre el curso...

## Resultados de Aprendizaje Esperados

Al finalizar el curso el estudiante estará en capacidad de:

4. Desarrollar aplicaciones que hagan uso eficiente de estructuras de datos adaptadas de forma adecuada al contexto.
5. Hacer uso de la terminal para la compilación, ejecución y depuración de programas.
6. Incorporar técnicas de programación genérica en la implementación de algoritmos en C++.

# Sobre el curso...

Un poco de matemática...

1 crédito  $\rightarrow$  48 horas de trabajo al semestre.

3 créditos  $\rightarrow$  144 horas de trabajo al semestre.  
 $\rightarrow$  9 horas de trabajo semanal (16 sem).

Distribución de horas de trabajo semanal:

4 horas trabajo guiado remoto  
+ 5 horas trabajo independiente

# Metodología

- Clases magistrales (presentación de temas).
- Aprendizaje colaborativo (trabajo en grupo).
- Talleres de aplicación de conceptos.
- Evaluación: parciales, proyecto final.
- Trabajo individual:
  - Tareas y ejercicios.
  - Preparación de clases.
  - Preparación de talleres.
  - Proyecto final.

# Evaluación

- Habilidad conceptual y analítica:
  - 3 parciales (20% cada uno)
- Habilidad práctica y trabajo en grupo:
  - Proyecto (20%)
  - Talleres (20%)  
(promedio de todos los elementos)

# Calificación

- Para cada elemento de evaluación (talleres, parciales, proyecto) se definirá una rúbrica de calificación.
- Rúbrica: instrumento estándar de evaluación. Conjunto de criterios usados para evaluar un nivel de desempeño.
- Se asignan unos niveles de calificación (entre 0.0 y 5.0) de acuerdo a los elementos particulares esperados, valores intermedios entre los niveles pueden indicar desarrollo parcial.

# Calificación

- Ejemplo:

**5.0 / 5.0:** El estudiante propuso un código que cubre lo pedido y el diseño de la solución es adecuado.

**3.5 / 5.0:** El estudiante propuso un código que cubre lo pedido, pero el diseño de la solución no tiene una calidad suficiente para ser un trabajo de ingeniería.

**3.0 / 5.0:** El estudiante propuso un código que cubre lo pedido, pero no hizo el diseño de la solución.

**0.0 / 5.0:** El estudiante no presentó código ni diseño.

# Fechas importantes

- Parciales:
  - Parcial 1: viernes 26 de febrero (sem. 5).
  - Parcial 2: viernes 16 de abril (sem. 11).
  - Parcial 3: viernes 28 de mayo (sem. 17).
- Proyecto:
  - Entrega 0 (inicial): viernes 12 de febrero (sem. 3).
  - Primera entrega: viernes 5 de marzo (sem. 6).
  - Segunda entrega: viernes 23 de abril (sem. 12).
  - Entrega final: viernes 4 de junio (sem. 18).



# Talleres

- Taller 1: Complejidad  
miércoles 3 de febrero (sem. 2).
- Taller 2: Estructuras lineales  
viernes 19 de febrero (sem. 4).
- Taller 3: Árboles de búsqueda  
viernes 19 de marzo (sem. 8).
- Taller 4: Árboles de partición del espacio  
viernes 9 de abril (sem. 10).
- Taller 5: Grafos  
viernes 21 de mayo (sem. 16).

# Grupos

- Talleres y proyecto final se realizarán en grupos de **estrictamente 3** personas.
  - Necesario definir los grupos desde la primera semana y se deben mantener así a lo largo de todo el semestre.

# Grupos

- Talleres y proyecto final se realizarán en grupos de **estrictamente 3** personas.
  - Necesario definir los grupos desde la primera semana y se deben mantener así a lo largo de todo el semestre.

## **Tarea #1:**

Antes del **martes 2 de febrero**, enviar a través de la actividad de UVirtual **Definición grupo**, los nombres de los integrantes del grupo de trabajo (un sólo envío por grupo).

# Proyecto

- Proyecto de curso en 3 entregas.
- Cada entrega busca aplicar los conocimientos adquiridos hasta el momento en un problema de la vida real.
- A desarrollarse en los mismos grupos de talleres.
- Enunciado disponible en la página del curso.
- Cada entrega tendrá una sustentación por grupos, en un horario asignado.

# Recursos

- Página del curso en Uvirtual:  
[uvirtual.javeriana.edu.co](http://uvirtual.javeriana.edu.co)
  - Programa del curso, planeación de sesiones, notas.
  - Diapositivas contenidos, videos sesiones.
  - Enunciado, envío de talleres.
  - Enunciado, desarrollo de parciales.
  - Enunciado, envío entregas proyecto final.
  - Enlace Blackboard Collaborate (sesiones remotas).

# Sesiones remotas

- Conexión preferiblemente por cable (para mejor ancho de banda).
- Evitar compartir ancho de banda con otras aplicaciones (Spotify, YouTube, Netflix, WhatsApp Web).
- Mantener el micrófono en silencio. Sólo activarlo al intervenir (levantar la mano).
- Encender la cámara al inicio de la clase (15 minutos – saludo). Luego apagarla para mejor ancho de banda.

# Sesiones remotas

- Ingresar con algo de anticipación (15 minutos) para probar conexión, sonido y video.
- Las sesiones inician a las 9:10am.
- Las diferentes partes de la sesión serán grabadas y publicadas posteriormente.
- Se harán pausas activas durante la sesión, para evitar la fatiga :).

# Lenguaje

## The 2020 Top Ten Programming Languages

Rank	Language	Type	Score
1	Python ▼	  	100.0
2	Java ▼	  	95.3
3	C ▼	  	94.6
4	C++ ▼	  	87.0
5	JavaScript ▼		79.5
6	R ▼		78.6
7	Arduino ▼		73.2
8	Go ▼	 	73.1
9	Swift ▼	 	70.5
10	Matlab ▼		68.4



# Lenguaje

- C++: lenguaje de programación de propósito general, desarrollado por Bjarne Stroustrup (1979, rev. 1984, versión comercial 1985).
- Estandarizado por ISO en revisiones:  
C++98, C++03, **C++11**, C++14, C++17.

# Lenguaje

- C++ Standard: lenguaje y librería estándar.
- C++ y C:
  - La mayoría del código C puede compilarse en C++.
  - Algunos elementos de C pueden ser inválidos en C++, o comportarse de forma diferente.

<https://isocpp.org/tour>

# Restricciones

- Se usa sólo un compilador, **no un IDE**.
  - **NO:** Netbeans, Eclipse, Visual Studio, Code::Blocks, Dev-C++, etc...
  - El código fuente se escribe en cualquier editor de texto básico.
  - Luego se compila en una terminal, consola o línea de comandos usando g++ (versión 4.6 en adelante).
  - Ejemplo: <https://repl.it/>, MinGW
  - Esto permite entender el proceso que realiza el IDE “por debajo”. Adicionalmente se reduce la cantidad de archivos relacionados.

# Restricciones

- Código fuente debe compilarse y ejecutarse correctamente en la línea de comandos.
  - Durante el desarrollo, el código fuente puede escribirse, compilarse y probarse en cualquier S.O.
  - Esto facilita implementaciones portables, estándar, no dependientes de librerías especiales de cada S.O.

# Apoyo

Monitor del curso:

- Sergio Andrés Mejía Tovar  
[sergio.mejia@javeriana.edu.co](mailto:sergio.mejia@javeriana.edu.co)
  - Apoyo con preguntas puntuales
  - Verificación de talleres

# Recomendaciones

- Dedicación, trabajo honesto y sincero.
- Rigor y formalidad propios del trabajo en Ingeniería.
- Siempre usar citaciones y referencias pertinentes de los medios consultados.
- Intuición, recursividad, inquietud por aprender.
- Aprovechar los medios de contacto ante cualquier inquietud, sugerencia, problema, ...

¿Preguntas?

¿Sugerencias?

¿Comentarios?

...

# Introducción



Ir a:

<https://bit.ly/3gfysCA>

Al terminar, ir a:

<https://bit.ly/3gdgjpa>

¿De qué trata este curso?

# ¿De qué trata este curso?

- Definición de tipos abstractos de datos (TAD) de mayor complejidad.
- Integración entre el proceso de ingeniería: análisis, diseño, implementación.
  - no es sólo programar a prueba y error, el proceso previo de análisis y diseño es importante también.

# ¿De qué trata este curso?

- Introducción de unos pocos elementos nuevos de programación:
  - poner los elementos ya vistos en PC (PA) en un contexto más amplio.
  - por lo cual el curso NO tiene la misma dinámica de PC (PA).
- Descubrimiento del proceso interno que permite llevar el código fuente a un programa ejecutable.

¿Qué se estudia en este curso?

# Estructuras de Datos

- Formas particulares y eficientes de organizar y almacenar los datos:
  - Algunas suficientemente generales para cualquier aplicación.
  - Algunas particularmente específicas para un problema puntual.
- Manejo de grandes volúmenes de datos.
- Claves para algoritmos eficientes de procesamiento.

# ¿Conocemos algunas estructuras?

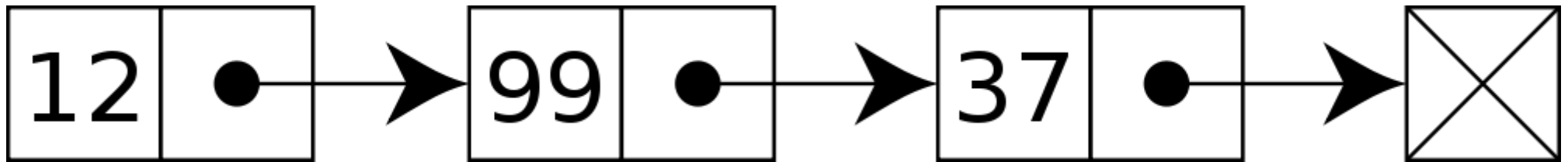
- Variables.
- Vectores.
- Matrices.
- Estructuras (registros).
- Listas.
- Pilas.
- Colas.





# ¿Qué otras estructuras existen?

- Estructuras lineales.



[en.wikipedia.org/wiki/Linked\\_list](https://en.wikipedia.org/wiki/Linked_list)

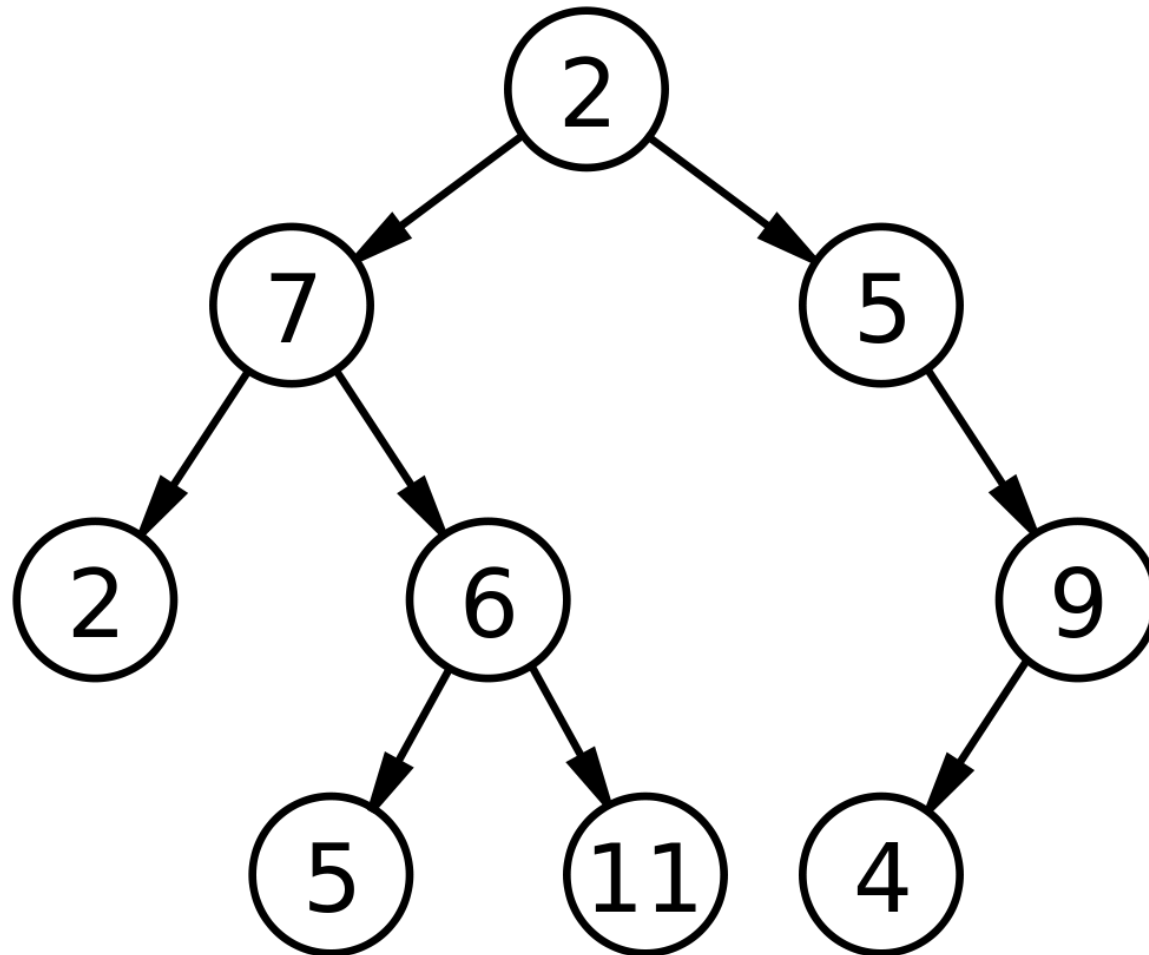
# ¿Qué otras estructuras existen?

- Estructuras lineales.



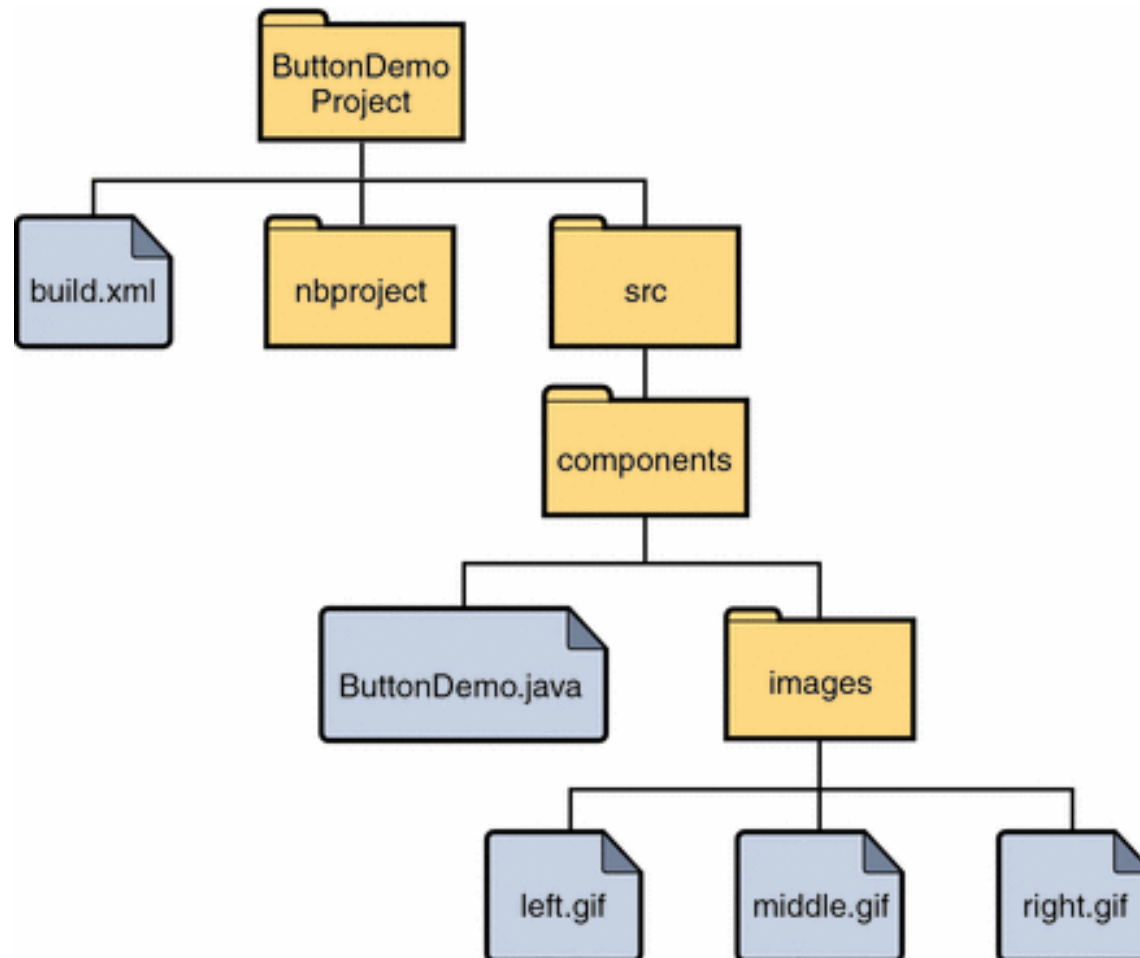
# ¿Qué otras estructuras existen?

- Estructuras recurrentes.



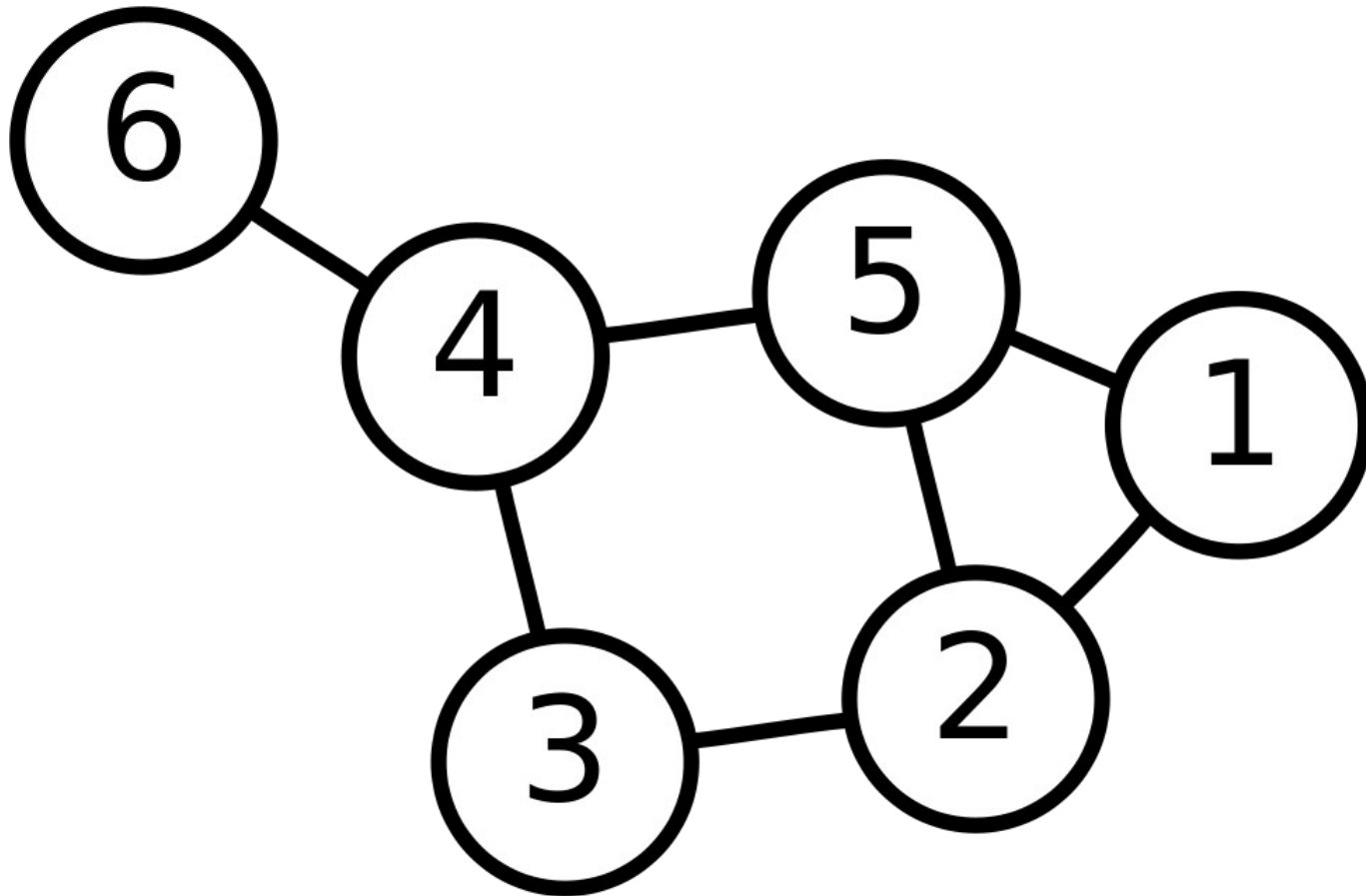
# ¿Qué otras estructuras existen?

- Estructuras recurrentes.



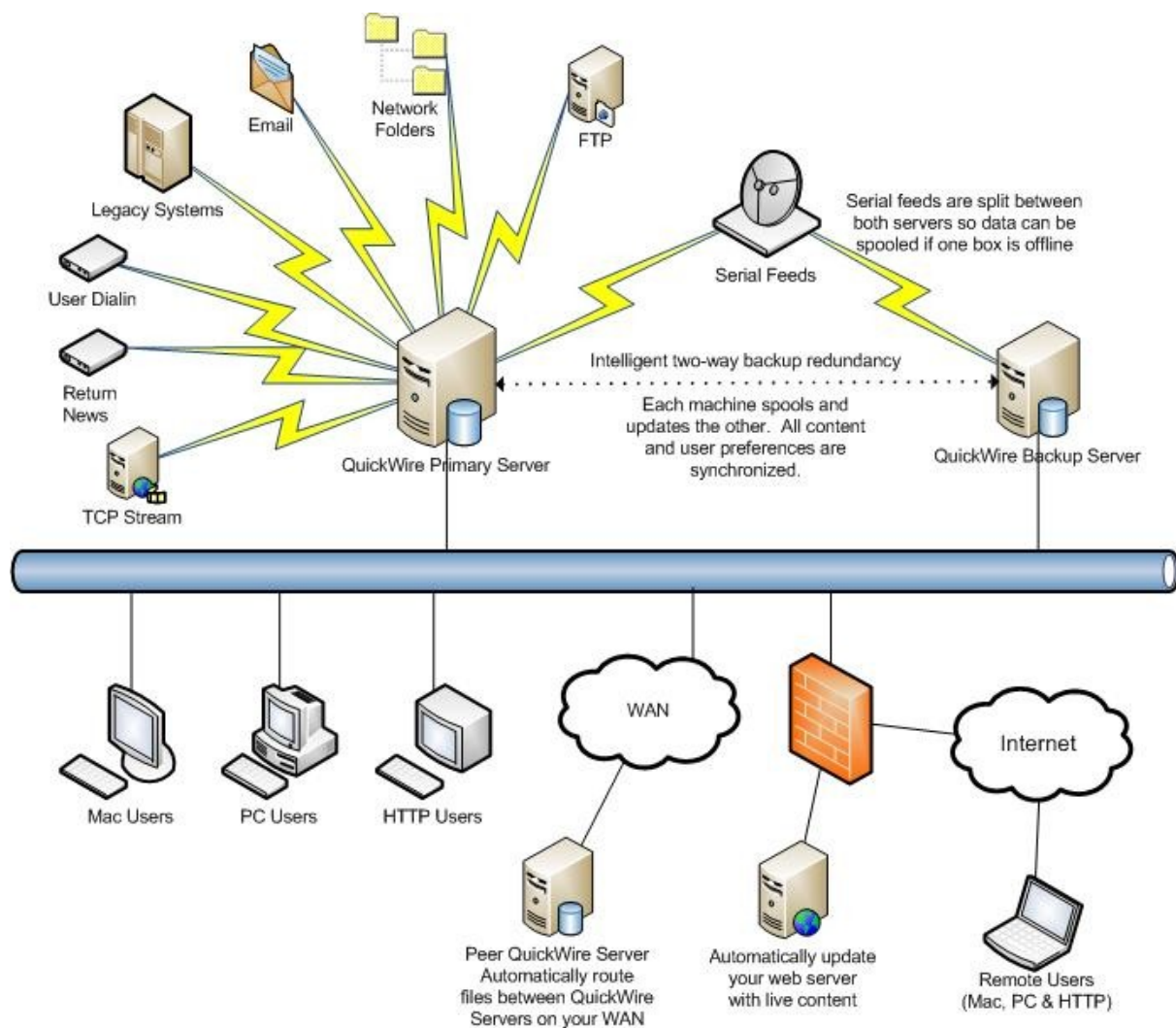
# ¿Qué otras estructuras existen?

- Estructuras no lineales.



# ¿Qué otras estructuras existen?

- Estructuras no lineales.



# Fundamentos de complejidad

# Algoritmo

- Procedimiento computacional (secuencia de pasos) bien definidos que toman un valor (o conjunto de valores) como entrada y produce un valor (o conjunto de valores) como salida.
- Elementos clave: exactitud, eficiencia.



¿Es eficiente un algoritmo?

# ¿Es eficiente un algoritmo?

- Análisis de los recursos que el algoritmo requerirá.
- Complejidad en espacio:
  - Memoria.
  - Ancho de banda de comunicación.
  - Almacenamiento en disco.
- Complejidad en tiempo:
  - **Tiempo computacional.**

# ¿Cómo medir la complejidad?

- Análisis empírico.
- Análisis teórico (asintótico).

# Análisis empírico

- ¿Es un algoritmo rápido? ¿Qué significa rápido?
- Incorporar sentencias de medición de tiempo
- Ejecutar el algoritmo varias veces:
  - con las mismas entradas, promediar tiempos o escoger el mejor tiempo.
  - con diferentes entradas, por cada una promediar tiempos o escoger el mejor tiempo.

# Análisis empírico

## Limitaciones:

- Requiere implementación y ejecución del algoritmo varias veces.
- Resultados sólo sobre un conjunto limitado de entradas (puede no ser indicativo para otras entradas).
- Para comparar algoritmos, se requiere el mismo entorno (mismo procesador, misma memoria...).

# Análisis teórico

- Tiempo de ejecución **T**: número de operaciones primitivas o “pasos” ejecutados.
    - Acceso a memoria, asignación, comparación, operación aritmética, ...
  - Para un algoritmo, usualmente el tiempo de ejecución es función del tamaño de la entrada.
    - Número de elementos de entrada.
- $T(n)$** , con  $n$  tamaño de la entrada.

# Análisis teórico

- Ejemplo: búsqueda lineal.
- ¿Mejor caso?
- ¿Peor caso?
- ¿Caso promedio?

# Análisis teórico

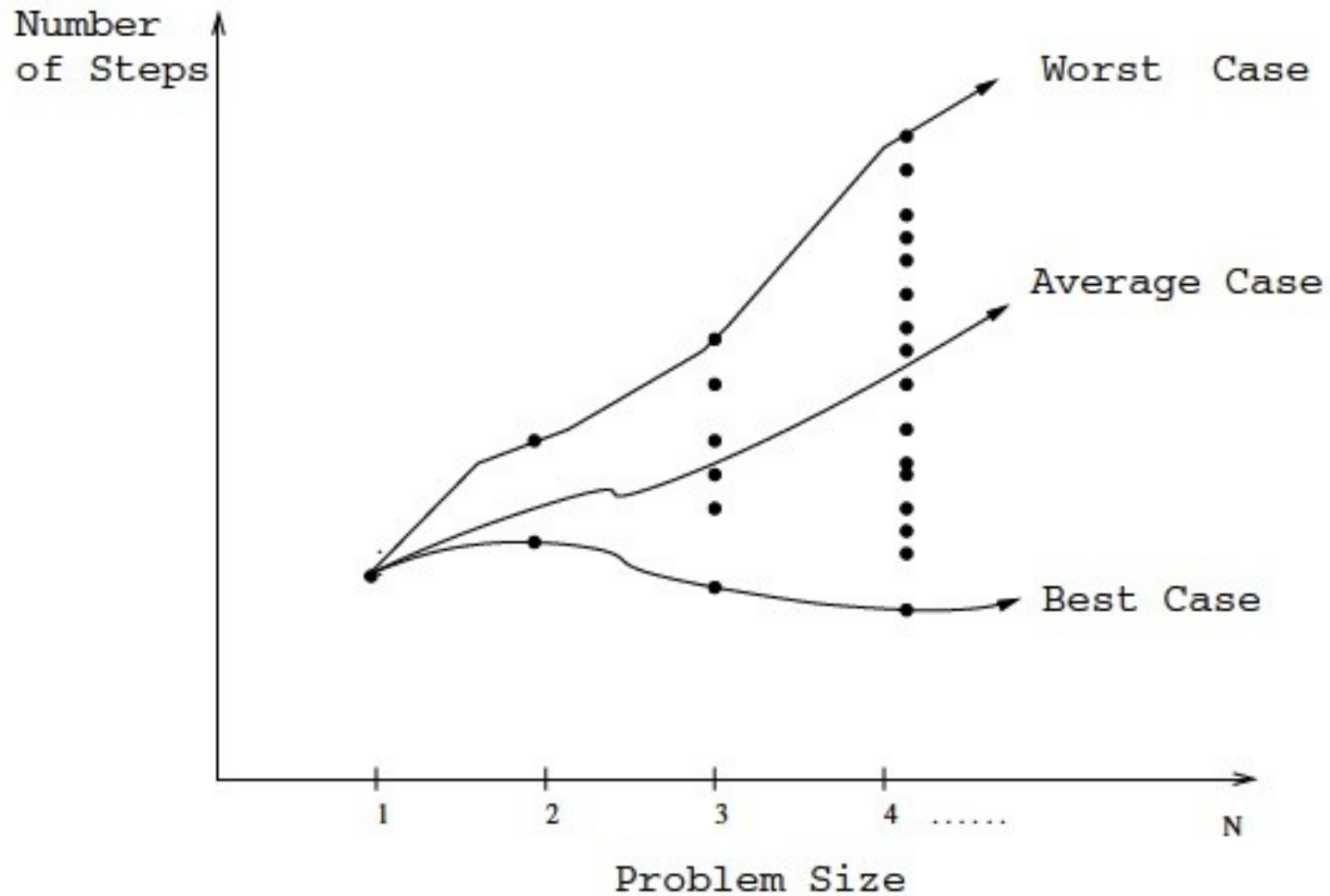
- Ejemplo: búsqueda lineal.
- Mejor caso: elemento al inicio de la lista.
- Peor caso: elemento al final de la lista.
- Caso promedio: elemento en la mitad de la lista.



# Análisis teórico

- Ejemplo: búsqueda lineal.
- Mejor caso: elemento al inicio de la lista.  
 $T(n) = 1$
- Peor caso: elemento al final de la lista.  
 $T(n) = n$
- Caso promedio: elemento en la mitad de la lista.  
 $T(n) = 1/2 n$

# Análisis teórico



# Análisis teórico

- Ejemplo: búsqueda lineal.

- Mejor caso: límite inferior.

$$T(n) = 1 \rightarrow \mathbf{\Omega()}$$

- Peor caso: límite superior.

$$T(n) = n \rightarrow \mathbf{O()}$$

- Caso promedio: límite ajustado.

$$T(n) = 1/2 n \rightarrow \mathbf{\Theta()}$$

# Análisis teórico

- Ejemplo: búsqueda lineal.

- Mejor caso: límite inferior.

$$T(n) = 1 \rightarrow \Omega()$$

- Peor caso: límite superior.

$$T(n) = n \rightarrow \mathbf{O}()$$

- Caso promedio: límite ajustado.

$$T(n) = 1/2 n \rightarrow \Theta()$$

# Ejemplo

- Análisis de complejidad de algoritmos:

```
double exp(double a, int n)
{
    double res = 1;
    int i = 0;
    while(i < n) {
        res = res * a;
        i = i + 1;
    }
    return(res);
}
```

```
double exp(double a, int n)
{
    if(n == 0)
        return(1);
    else
        return(exp(a, n-1) * a);
}
```

# Ejemplo

- Análisis de complejidad de algoritmos:


```
double exp(double a, int n) {  
    double res = 1;  
    int i = 0;  
    while(i < n) {  
        res = res * a;  
        i = i + 1;  
    }  
    return(res);  
}
```

# Ejemplo

- Análisis de complejidad de algoritmos:

```
double exp(double a, int n) {  
    double res = 1;    // 1  
    int i = 0;         // 1  
    while(i < n) {     // 1 * (n+1)  
        res = res * a; // 2 * n  
        i = i + 1;     // 2 * n  
    }  
    return(res);       // 1  
}
```

Cantidad de  
operaciones  
primitivas en  
cada línea  
del código



# Ejemplo

- Análisis de complejidad de algoritmos:

```
double exp(double a, int n) {  
    double res = 1;    // 1  
    int i = 0;         // 1  
    while(i < n) {     // 1 * (n+1)  
        res = res * a; // 2 * n  
        i = i + 1;     // 2 * n  
    }  
    return(res);       // 1  
}
```

$$T_{\text{exp}}(n) = 1+1+(n+1)+n(2+2)+1 = 5n+4$$



# Ejemplo

- Análisis de complejidad de algoritmos:

```
double exp(double a, int n) {  
    double res = 1;    // 1  
    int i = 0;         // 1  
    while(i < n) {     // 1 * (n+1)  
        res = res * a; // 2 * n  
        i = i + 1;     // 2 * n  
    }  
    return(res);      // 1  
}
```

$$T_{\text{exp}}(n) = 1+1+(n+1)+n(2+2)+1 = 5n+4 \rightarrow O(n)$$

# Ejemplo

- Análisis de complejidad de algoritmos:

```
double exp(double a, int n) {  
    if(n == 0)  
        return(1);  
    else  
        return(exp(a, n-1) * a);  
}
```

# Ejemplo

- Análisis de complejidad de algoritmos:

```
double exp(double a, int n) {  
    if(n == 0) // 1  
        return(1); // 1  
    else  
        return(exp(a, n-1) * a); // 1+1+T(n-1)  
}
```

$$T_{\text{exp}}(n) = 3 + T_{\text{exp}}(n-1)$$

$$T_{\text{exp}}(n) = 3 + 3 + T_{\text{exp}}(n-2)$$

$$T_{\text{exp}}(n) = 3 + 3 + \dots + 2$$

$$T_{\text{exp}}(n) = 3n + 2$$

# Ejemplo

- Análisis de complejidad de algoritmos:

```
double exp(double a, int n) {  
    if(n == 0)                // 1  
        return(1);           // 1  
    else  
        return(exp(a, n-1) * a); // 1+1+T(n-1)  
}
```

$$T_{\text{exp}}(n) = 3 + T_{\text{exp}}(n-1)$$

$$T_{\text{exp}}(n) = 3 + 3 + T_{\text{exp}}(n-2)$$

$$T_{\text{exp}}(n) = 3 + 3 + \dots + 2$$

$$T_{\text{exp}}(n) = 3n + 2 \rightarrow O(n)$$

# Tarea inicial

Investigar:

- ¿Qué son parámetros de línea de comandos, y cómo se usan en un programa en C++11?
- ¿Cómo se mide el tiempo de operaciones o bloques de instrucciones en C++11?
- ¿Cómo se compila un programa escrito en C++11 en la consola (terminal, línea de comandos)?
- ¿Cómo se ejecuta un programa C++11 ya compilado en la consola (terminal, línea de comandos)?

# Referencias

- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. Introduction to Algorithms, 3<sup>rd</sup> edition. MIT Press, 2009.
- [vaxxxa.github.io/talks/introduction.to.algorithms-computational.complexity/](https://vaxxxa.github.io/talks/introduction.to.algorithms-computational.complexity/)