Diseño y Análisis de Algoritmos ISIS1105

Daniel R. Barrero R.

Universidad de los Andes

January 24, 2025

Información básica

- **Instructor:** Daniel Barrero <dr.barrero2562>

Información básica

- **Instructor:** Daniel Barrero <dr.barrero2562>
    - **Horas de atención:** Jueves o Viernes 11 AM (agendar previamente por e-mail).
    - **Lugar de atención:** ML-761

Información básica

- **Instructor:** Daniel Barrero <dr.barrero2562>
  - **Horas de atención:** Jueves o Viernes 11 AM (agendar previamente por e-mail).
  - **Lugar de atención:** ML-761
- **Monitores:** Elkin Cuello <e.cuello>, Juan David Duarte <j.duartey>.

Información básica

Calificación del curso

Información básica

Calificación del curso

- Parcial 1 [20%]
- Parcial 2 [20%]
- Parcial 3 [20%]

Información básica

Calificación del curso

- Parcial 1 [20%]
- Parcial 2 [20%]
- Parcial 3 [20%]
- Proyecto (3 entregas) [20%]

Información básica

Calificación del curso

- Parcial 1 [20%]
- Parcial 2 [20%]
- Parcial 3 [20%]
- Proyecto (3 entregas) [20%]
- Talleres y Quices [20%]

Información básica

Calificación del curso

- Parcial 1 [20%]
- Parcial 2 [20%]
- Parcial 3 [20%]
- Proyecto (3 entregas) [20%]
- Talleres y Quices [20%]

Los proyectos se desarrollan en parejas, las demás actividades son individuales.

Política de aproximación de notas

Información básica

Calificación del curso

- Parcial 1 [20%]
- Parcial 2 [20%]
- Parcial 3 [20%]
- Proyecto (3 entregas) [20%]
- Talleres y Quices [20%]

Los proyectos se desarrollan en parejas, las demás actividades son individuales.

Política de aproximación de notas

- Para aprobar el curso es indispensable lograr una nota sin aproximar de 3.0 o superior.

Información básica

Calificación del curso

- Parcial 1 [20%]
- Parcial 2 [20%]
- Parcial 3 [20%]
- Proyecto (3 entregas) [20%]
- Talleres y Quices [20%]

Los proyectos se desarrollan en parejas, las demás actividades son individuales.

Política de aproximación de notas

- Para aprobar el curso es indispensable lograr una nota sin aproximar de 3.0 o superior.
- La mejor nota del curso será aproximada a 5.0

Información básica

Calificación del curso

- Parcial 1 [20%]
- Parcial 2 [20%]
- Parcial 3 [20%]
- Proyecto (3 entregas) [20%]
- Talleres y Quices [20%]

Los proyectos se desarrollan en parejas, las demás actividades son individuales.

Política de aproximación de notas

- Para aprobar el curso es indispensable lograr una nota sin aproximar de 3.0 o superior.
- La mejor nota del curso será aproximada a 5.0
- No se hace aproximación de las demás notas finales.

Información básica

Calificación del curso

- Parcial 1 [20%]
- Parcial 2 [20%]
- Parcial 3 [20%]
- Proyecto (3 entregas) [20%]
- Talleres y Quices [20%]

Los proyectos se desarrollan en parejas, las demás actividades son individuales.

Política de aproximación de notas

- Para aprobar el curso es indispensable lograr una nota sin aproximar de 3.0 o superior.
- La mejor nota del curso será aproximada a 5.0
- No se hace aproximación de las demás notas finales.

Bibliografía

- Cormen et al. *Introduction to algorithms*. MIT Press, 2009.
- Bohórquez, Cardoso. *Análisis de algoritmos*. Universidad de los Andes, 1992.

Bibliografía

- Cormen et al. *Introduction to algorithms*. MIT Press, 2009.
- Bohórquez, Cardoso. *Análisis de algoritmos*. Universidad de los Andes, 1992.
- Brassard, Bratley. *Algorithmics: theory and practice*. Prentice-Hall, 1988.

Bibliografía

- Cormen et al. *Introduction to algorithms*. MIT Press, 2009.
- Bohórquez, Cardoso. *Análisis de algoritmos*. Universidad de los Andes, 1992.
- Brassard, Bratley. *Algorithmics: theory and practice*. Prentice-Hall, 1988. ☺☺

Bibliografía

- Cormen et al. *Introduction to algorithms*. MIT Press, 2009.
- Bohórquez, Cardoso. *Análisis de algoritmos*. Universidad de los Andes, 1992.
- Brassard, Bratley. *Algorithmics: theory and practice*. Prentice-Hall, 1988. ☺☺

Otras referencias

- Kocay, Kreher. *Graphs, Algorithms, and Optimization*. CRC Press, 2017.
- Maurer, Ralston. *Discrete algorithmic mathematics*. CRC Press, 2005.

Why algorithmics?

Example: Write a program to compute the n-th Fibonacci number.

Why algorithmics?

Example: Write a program to compute the n-th Fibonacci number.

```
public static int fib1(int n) {
    if(n <= 1) {
        return 1;
    }
    return fib1(n-2) + fib1(n-1);
}
```

Why algorithmics?

Example: Write a program to compute the n-th Fibonacci number.

```
public static int fib1(int n) {
    if(n <= 1) {
        return 1;
    }
    return fib1(n-2) + fib1(n-1);
}

public static int fib2(int n) {
    if(n <= 1) {
        return 1;
    }
    int result = 1;
    int prev = 1;
    for(int i = 2; i <=n; i++) {
        result = result+prev;
        prev = result-prev;
    }
    return result;
}
```

Why algorithmics?

Example: Let M be a row-and-column sorted square matrix. Write an algorithm to find a given integer.

Why algorithmics?

Example: Let M be a row-and-column sorted square matrix. Write an algorithm to find a given integer.

```
public static boolean naiveSearch(int[][] mm, int y) {
    int height = mm.length;
    int width = mm[0].length;
    for(int row = 0; row < height; row++) {
        for(int col = 0; col < width; col++) {
            if(mm[row][col] == y) {
                return true;
            }
        }
    }
    return false;
}
```

Why algorithmics?

Example: Let M be a row-and-column sorted square matrix. Write an algorithm to find a given integer.

Why algorithmics?

Example: Let M be a row-and-column sorted square matrix. Write an algorithm to find a given integer.

```java
public static boolean searchInSorted(int[][] mm, int y) {
    int height = mm.length;
    int width = mm[0].length;
    int row = 0;
    int col = width-1;
    while(row < height && col >= 0) {
        if(y == mm[row][col]) {
            return true;
        }
        else if(y > mm[row][col]) {
            row++;
        }
        else {
            col--;
        }
    }
    return false;
}
```

Why algorithmics?

Example: Write an algorithm to sort an array of integers.

Why algorithmics?

Example: Write an algorithm to sort an array of integers.

```
public static void insertionSort(int[] aa) {
    int n = aa.length;
    for(int i=1; i<n; i++) {
        int x = aa[i];
        int j = i-1;
        while(j>=0 && x<aa[j]) {
            aa[j+1] = aa[j];
            j--;
                    }
        aa[j+1] = x;
    }
}
```

Why algorithmics?

Example: Write an algorithm to sort an array of integers.

```
public static void insertionSort(int[] aa) {
    int n = aa.length;
    for(int i=1; i<n; i++) {
        int x = aa[i];
        int j = i-1;
        while(j>=0 && x<aa[j]) {
            aa[j+1] = aa[j];
            j--;
                    }
        aa[j+1] = x;
    }
}
```

- What about *merge sort*?

Complexity

Let $\mathcal{P}$ be a program. Its *time complexity* is the function

$$\tau_{\mathcal{P}} : \mathbf{N} \to \mathbf{R}^{>0}$$

given by $\tau_{\mathcal{P}}(n) = \max\{t(\mathcal{P}, I) \mid |I| = n\}$, where $t(\mathcal{P}, I)$ is the time in seconds in which $\mathcal{P}$ runs input I.

Complexity

Let $\mathcal{P}$ be a program. Its *time complexity* is the function

$$\tau_{\mathcal{P}} : \mathbf{N} \to \mathbf{R}^{>0}$$

given by $\tau_{\mathcal{P}}(n) = \max\{t(\mathcal{P}, I) \mid |I| = n\}$, where $t(\mathcal{P}, I)$ is the time in seconds in which $\mathcal{P}$ runs input I.

- Why does the function take values in $\mathbf{R}^{>0}$ and not integers?

Complexity

Let $\mathcal{P}$ be a program. Its *time complexity* is the function

$$\tau_{\mathcal{P}} : \mathbf{N} \to \mathbf{R}^{>0}$$

given by $\tau_{\mathcal{P}}(n) = \max\{t(\mathcal{P}, I) \mid |I| = n\}$, where $t(\mathcal{P}, I)$ is the time in seconds in which $\mathcal{P}$ runs input I.

- Why does the function take values in $\mathbf{R}^{>0}$ and not integers?
- What do we mean by $|I| = n$?

Complexity : big-O notation

Let $f, g : \mathbf{N} \to \mathbf{R}^{>0}$. We say that $f(n)$ *is* $O(g(n))$ if

$$\exists c \in \mathbf{R}^{>0}, n_0 \in \mathbf{N} : \ \forall n \geq n_0 : \ f(n) \leq cg(n).$$

More conveniently, we will write $f(n) \in O(g(n))$.

Questions:

- $n^2 \in O(n^3)$?

Complexity : big-O notation

Let $f, g : \mathbf{N} \to \mathbf{R}^{>0}$. We say that $f(n)$ *is* $O(g(n))$ if

$$\exists c \in \mathbf{R}^{>0}, n_0 \in \mathbf{N} : \ \forall n \geq n_0 : \ f(n) \leq cg(n).$$

More conveniently, we will write $f(n) \in O(g(n))$.

Questions:

- $n^2 \in O(n^3)$?
- $n^3 \in O(n^2)$?

Complexity : big-O notation

Let $f, g : \mathbf{N} \to \mathbf{R}^{>0}$. We say that $f(n)$ *is* $O(g(n))$ if

$$\exists c \in \mathbf{R}^{>0}, n_0 \in \mathbf{N} : \ \forall n \geq n_0 : \ f(n) \leq cg(n).$$

More conveniently, we will write $f(n) \in O(g(n))$.

Questions:

- $n^2 \in O(n^3)$?
- $n^3 \in O(n^2)$?
- $n^2 + n \in O(n^2)$?

Complexity : big-O notation

Let $f, g : \mathbf{N} \to \mathbf{R}^{>0}$. We say that $f(n)$ *is* $O(g(n))$ if

$$\exists c \in \mathbf{R}^{>0}, n_0 \in \mathbf{N} : \ \forall n \geq n_0 : \ f(n) \leq cg(n).$$

More conveniently, we will write $f(n) \in O(g(n))$.

Questions:

- $n^2 \in O(n^3)$?
- $n^3 \in O(n^2)$?
- $n^2 + n \in O(n^2)$?
- $2^n \in O(n^2)$?

Complexity : big-O notation

Let $f, g : \mathbf{N} \to \mathbf{R}^{>0}$. We say that $f(n)$ *is* $O(g(n))$ if

$$\exists c \in \mathbf{R}^{>0}, n_0 \in \mathbf{N} : \ \forall n \geq n_0 : \ f(n) \leq cg(n).$$

More conveniently, we will write $f(n) \in O(g(n))$.

Questions:

- $n^2 \in O(n^3)$?
- $n^3 \in O(n^2)$?
- $n^2 + n \in O(n^2)$?
- $2^n \in O(n^2)$?
- $n^5 \in O(2^n)$?

Complexity : big-O notation and Algorithms

If a program $\mathcal{P}$ has time complexity $\tau_{\mathcal{P}} \in O(f(n))$, we say $\mathcal{P}$ *has* complexity *of the order of* $f(n)$.

Complexity : big-O notation and Algorithms

If a program $\mathcal{P}$ has time complexity $\tau_{\mathcal{P}} \in O(f(n))$, we say $\mathcal{P}$ *has* complexity *<u>of the order of</u>* $f(n)$.

Example: *insertionSort* is of order n^2, and *mergeSort* is of order $n \log n$ (proof later). *naiveSearch* is of order n^2, and *searchInSorted* is of order n.