Centro de
Investigación en
Computación

Divide and Conquer

Divide y Vencerás

Ejemplos

Tema 5

Course

Analysis and design of algorithms

Instructor

Acosta Bermejo Raúl et al.

Lecture notes

Instituto
Politécnico
Nacional

Table of contents (outline)

Tabla de contenido

Ejemplos

Ejemplos extra

Introduction

Introducción

In science and philosophy, a **paradigm** is a distinct set of concepts or thought patterns, including theories, research methods, postulates, and standards for what constitutes legitimate contributions to a field.

- Divide and Conquer (D&C) is an algorithm design **paradigm** based on multi-branched recursion.
- A D&C algorithm works by:
  - **Recursively breaking down a problem** into two or more sub-problems of the same (or related) type (divide), until these become simple enough to be solved directly (conquer).
  - The solutions to the sub-problems are then **combined** to give a solution to the original problem.
- Understanding and designing D&C algorithms is a complex skill that requires a good understanding of the **nature of the underlying problem** to be solved.
- Similar to proving a theorem by **induction**, it is often necessary to replace the original problem with a more general or complicated problem in order to initialize the recursion, and there is no systematic method for finding the proper generalization.
- Wikipedia
  - https://en.wikipedia.org/wiki/Divide_and_conquer_algorithms

Introduction

Introducción

- The name D&C is sometimes applied also to algorithms that reduce each problem to **only** one sub-problem, such as the binary search algorithm.
- Therefore, some authors consider that the name D&C should be used only when each problem may generate two or more subproblems.
- The name **Decrease and conquer** has been proposed instead for the single-subproblem class.
  - An important application is in optimization, where if the search space is reduced ("pruned") by a constant factor at each step, the overall algorithm has the same asymptotic complexity as the pruning step, with the constant depending on the pruning factor (by summing the geometric series); this is known as prune and search.

5.0 Basic problems

Problemas básicos y clásicos

Basic problems

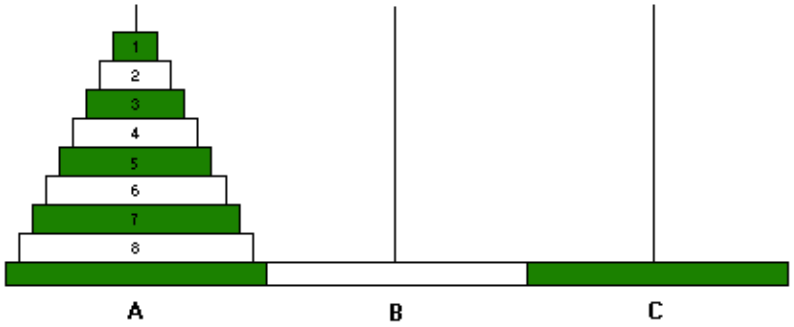
Revisión

Lista de problemas

- Juegos
  - Torres de Hanoi.
- Ordenamientos
  - Mergesort
  - Quicksort

Classic problems

Tower of Hanoi



The Tower of Hanoi puzzle

- Es un **rompecabezas** o juego matemático inventado en 1883 por el matemático francés Édouard Lucas.
- Este juego se trata de ocho discos de radio creciente que se apilan insertándose en una de las tres estacas de un tablero.
- El objetivo del juego es crear la pila en otra de las estacas siguiendo ciertas reglas:
  - Sólo se puede mover un disco cada vez.
  - Un disco de mayor tamaño no puede descansar sobre uno más pequeño que él mismo.
  - Sólo puedes desplazar el disco que se encuentre arriba en cada varilla.
- El número de movimientos necesarios para transferir *n* discos del poste A al poste C es: $2^n - 1$.

Tipo de solución: Basada en la(s) fórmulas matemáticas

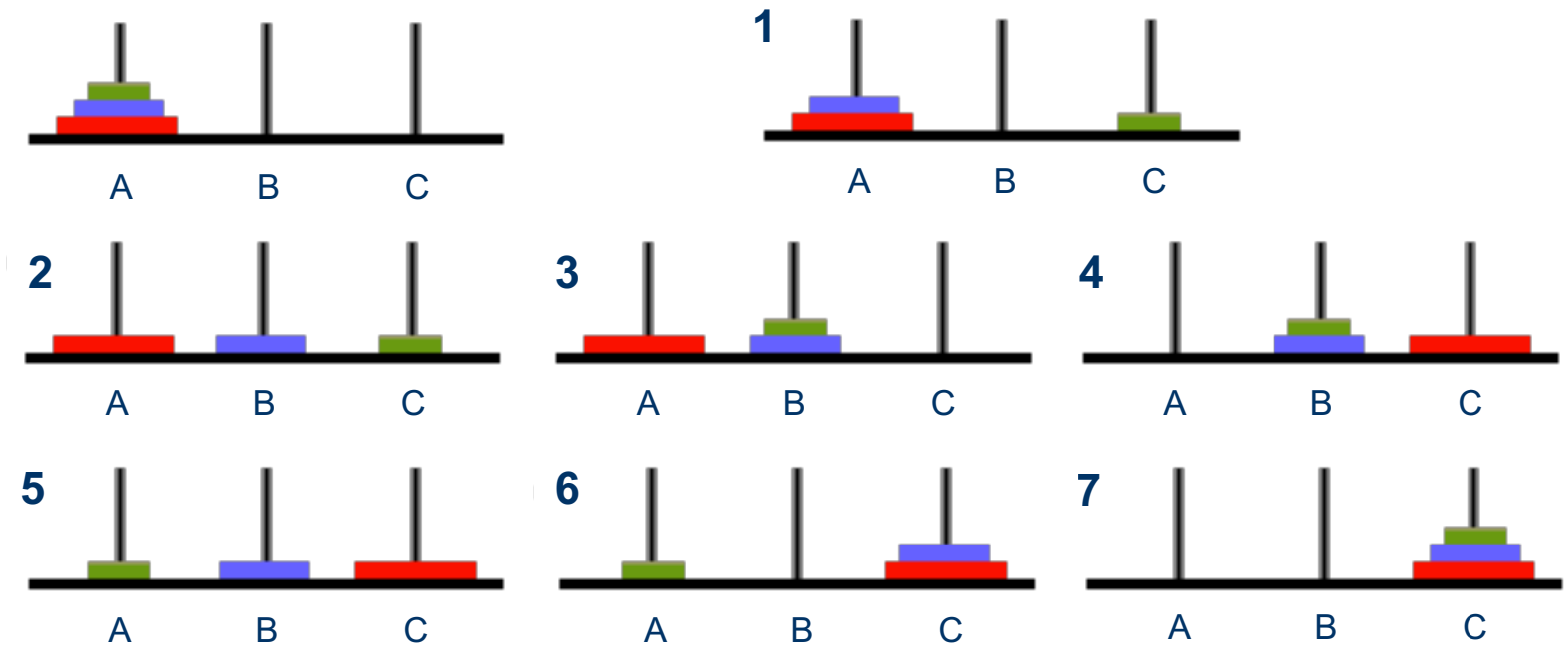
Classic problems

Tower of Hanoi

Tower of Hanoi

Movimientos para 3 discos.

¿Cuántos movimientos son necesarios?
Dar fórmula pero para
N discos y M estacas.



Classic problems

Tower of Hanoi

Algorithm (iterative, Pseudo-code)

1. Calculate the total number of moves required
   - "pow(2, n) - 1" here n is number of disks.
2. If number of disks (i.e. n) is even then
   - interchange destination pole and auxiliary pole.
3. **for** i = 1 to total number of moves:
   - **if** i%3 == 1 Legal movement of top disk between source pole and destination pole.
   - **if** i%3 == 2 Legal movement top disk between source pole and auxiliary pole.
   - **if** i%3 == 0 Legal movement top disk between auxiliary pole and destination pole.

https://en.wikipedia.org/wiki/Tower_of_Hanoi#Iterative_solution

https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/

Classic problems

Tower of Hanoi

Algorithm (recursive)

We use the following function to solve the problem. It has four arguments:

1. Number of disks: N.
2. Three pegs: source (Src), intermediary (Aux) and destination (Dst).

from_rod, to_rod, aux_rod

```
hanoi (N, Src, Aux, Dst)
  if N is 0
    exit
  else
     hanoi (N-1, Src, Dst, Aux)
     Move from Src to Dst
     hanoi (N -1 , Aux, Src, Dst)
```

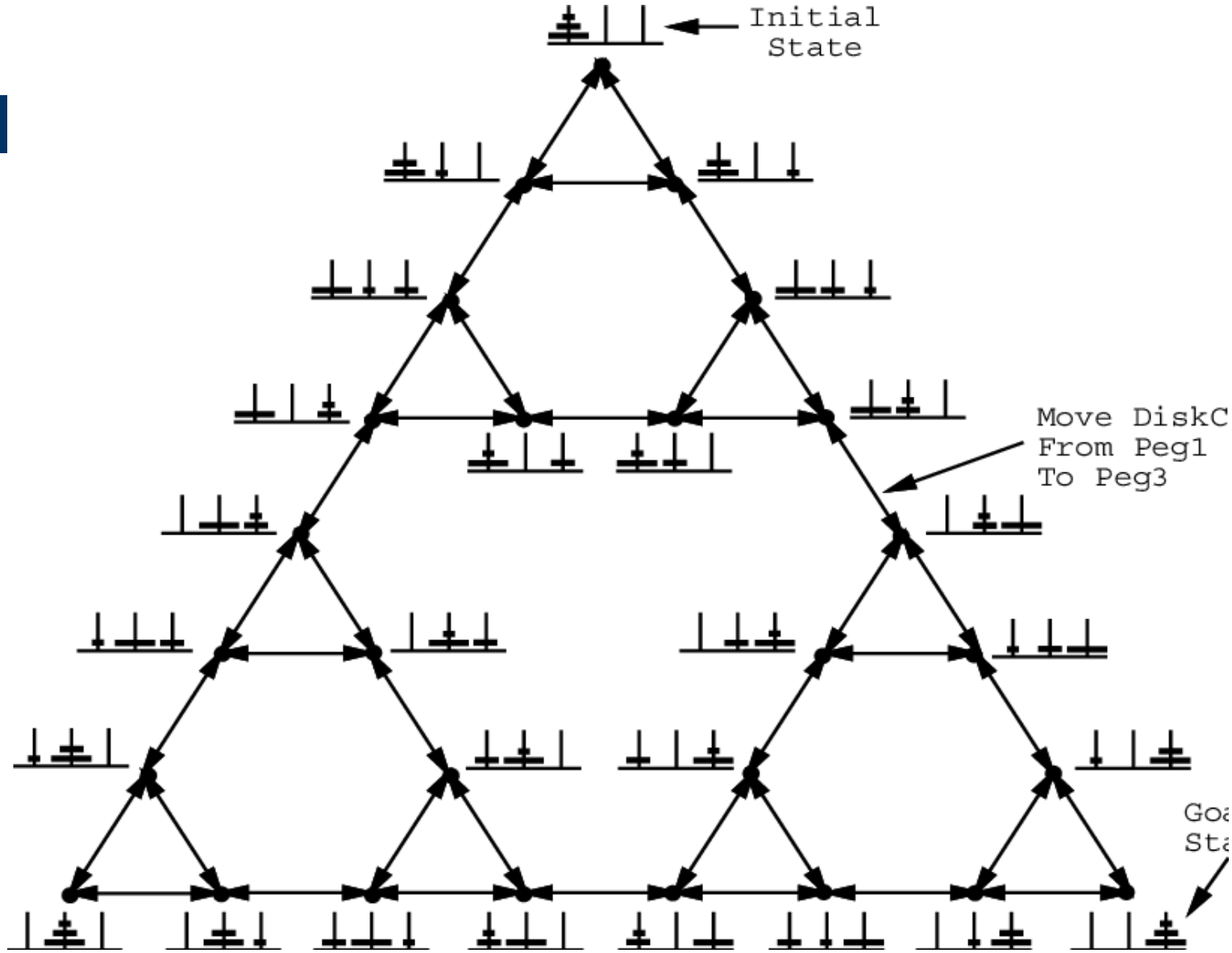
Condición para romper la recursividad al inicio

En el main:

```
hanoi(N, 'A', 'C', 'B')
```

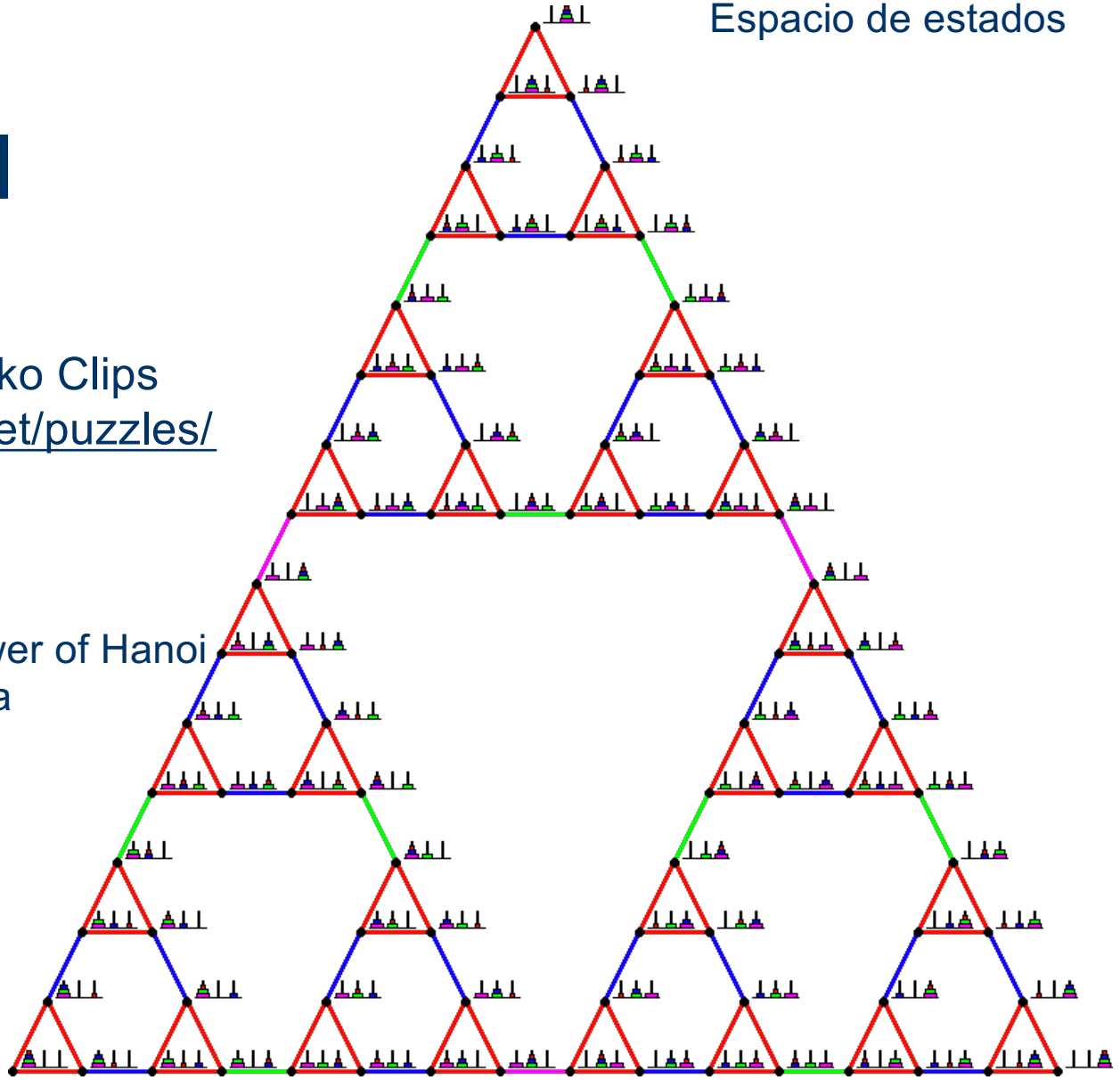
Toda solución
recursiva tiene una iterativa
y viceversa
Habilidad: reconocer cual es más
Fácil de diseñar

Introduction



Introduction

Espacio de soluciones
Espacio de estados



Variantes

- Rudenko Disk, Rudenko Clips
- https://www.jaapsch.net/puzzles/hanoi.htm

Artículos

- Shortest paths in the Tower of Hanoi graph and finite automata
  Dan Romik, 2008.
- Twin Towers of Hanoi
  ZoranŠunić, 2012.

5.1 Mergesorting algorithms

Ordenamiento por mezcla

Mergesort algorithm

Ordenamiento por mezcla

Some problems become easier once elements are sorted.

- Identify statistical outliers.
- Binary search in a database.
- Remove duplicates in a mailing list.

There are a lot of obvious applications.

Non-obvious applications.

- Convex hull.
- Closest pair of points.
- Interval scheduling / interval partitioning.
- Minimum spanning trees (Kruskal's algorithm).
- ...

Mergesort algorithm

Ordenamiento por mezcla

Lists of sorting algorithms (most known)

1. Bubble sort
2. Binary tree sort
3. Bucket sort
4. Heapsort
5. Insertion sort
6. Merge sort Divide & Conquer
7. Quicksort
8. Radix sort
9. Shell sort
10. Selection sort

En la wikipedia hay 24 (*https://en.wikipedia.org/wiki/Sorting_algorithm*).
https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento.

Mergesort algorithm

Ordenamiento por mezcla

- Fue desarrollado en 1945 por John Von Neumann.
- Mergesort is a divide and conquer algorithm.
- Mergesort parallelizes well due to use of previous algorithm.
- It has an average and worst-case performance of O(n log n).
- Many implementationes:
  - https://en.wikibooks.org/wiki/Algorithm_Implementation/Sorting/Merge_sort
  - https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/overview-of-merge-sort.

Ver slides de Algo de ordenamiento

Fue un matemático húngaro-estadounidense. 1903-1957

Mergesort algorithm

Ordenamiento por mezcla

John Von Neumann
1903-1957

- Fue un matemático húngaro-estadounidense.
- Realizó contribuciones fundamentales en:
  - Física cuántica
  - Análisis funcional
  - Teoría de conjuntos y T. De juegos
  - Ciencias de la computación (autómatas celulares)
  - Arquitectura de computadoras: Maq. Von Neumann
  - Economía
  - Análisis numérico
  - Cibernética
  - Hidrodinámica
  - Estadística
  - Y muchos otros campos.

Fue un genio dotado de Memoria fotográfica
Poliglota que a los 8 años dominó el calculo.

Trabajo en la Univ. Princeton (junto con Einstein)
Proyecto Manhattan
2ª guerra mundial, método implosión
1ª bomba atómica

5.3 Counting inversions

Conteo de inversiones

Descripción del problema

Solución usando D&C

Cálculo de la complejidad

5.3 Counting inversions

Conteo de inversiones

Problem

Music site tries to match your song preferences with others:

- You rank *n* songs.
- To find people with similar tastes (consults database).

Similarity metric: number of inversions between two rankings.

- My rank: 1, 2, ..., *n*.
- Your rank: a_1, a_2, ..., a_n.
- Songs *i* and *j* are inverted if $i < j$, but $a_i > a_j$.

	C	D	A	B	E
Me	1	2	3	4	5
You	1	**3**	4	**2**	5

Me is order?

5.3 Counting inversions

Conteo de inversiones

- Inversion: *i* and *j* are inverted if $i < j$, but $a_i > a_j$.
- How far (or close) the array is from being sorted.
- If array is already sorted then inversion count is 0.
- If array is sorted in reverse order that inversion count is the maximum.

You

1	3	4	2	5

2 inversiones: (4,2) y (3,2)

Brute force

Check all $\Theta(n^2)$ pairs.

Ya vimos la serie:
1+2+3+4+5+ ...

$$\sum_{k=1}^{n} k = \frac{n(n-1)}{2} = an^2 - bn \Rightarrow O(n^2)$$

5.3 Counting inversions

Conteo de inversiones

Strategy D&C

- Divide: separate list into two halves *A* and *B*.
- Conquer: recursively count inversions in each list.
- Combine: count inversions (*a*, *b*) with $a \in A$ and $b \in B$.
- Return sum of three counts.

1	5	4	8	10	2	6	9	3	7

Left half A

1	5	4	8	10

(5,4)

Right half B

2	6	9	3	7

(6,3), (9,3), (9,7)

Between halves

(4,2), (4,3), (5,2), (5,3), (8,2), (8,3), (8,6), (8,7), (10,2), (10,3), (10,6), (10,7), (10,9)

output 1 + 3 + 13 = 17

5.3 Counting inversions

Conteo de inversiones

Question

How to count inversions (a, b) with $a \in A$ and $b \in B$?

Answer

Easy if A and B are sorted!

how to combine two subproblems?

Warmup algorithm

- Sort A and B.
- For each element $b \in B$.
  - Binary search in A to find how elements in A are greater than b.

List A				
7	10	18	3	14
3	7	10	14	18

Sort

List B				
17	23	2	11	16
2	11	16	17	23

binary search to count inversions (a, b) with a ∈ A and b ∈ B

5.3 Counting inversions

Conteo de inversiones

Binary search

- It finds the position of a target value within a sorted array.
- Worst-case O(log n)
- Best case performance O(1)
- There is also an **iterative** implementation.

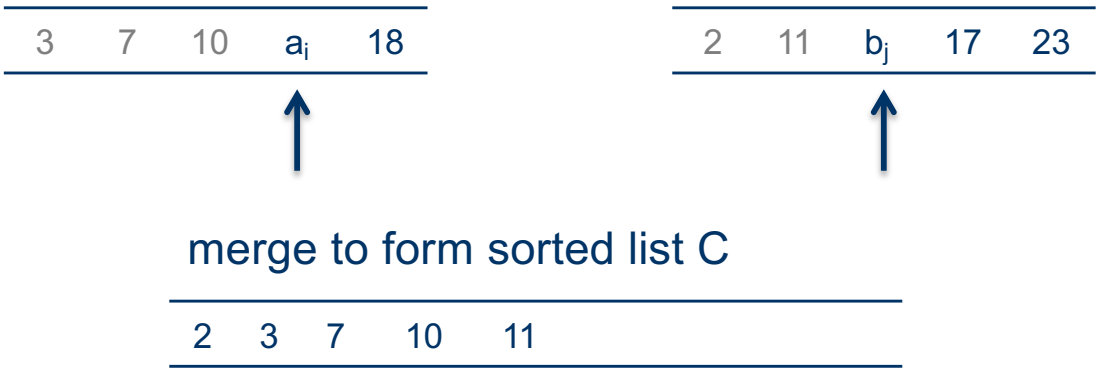
```
int binary_search(int A[], int key, int imin, int imax)
{
  if (imax < imin)
    return KEY_NOT_FOUND;
  else{
     int imid = midpoint(imin, imax);

     if( A[imid] > key )
        return binary_search(A, key, imin, imid - 1);
     else if ( A[imid] < key )
       return binary_search(A, key, imid + 1, imax);
     else
       return imid;
   }
}
```

5.3 Counting inversions

Conteo de inversiones

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
- If $a_i < b_j$, then a_i is not inverted with any element left in B.
- If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.



5.3 Counting inversions

Conteo de inversiones

Input. List L.
Output. Number of inversions in L and sorted list of elements L'.

sort-and-count(L)

if list L has one element

return (0, L)

DIVIDE the list into two halves A and B

(rA , A') ← sort-and-count(A)
(rB , B') ← sort-and-count(B)
(rAB , L') ← merge-and-count(A', B')

return (rA + rB + rAB , L')

Repaso de C
Como regreso 2 o más valores?

5.3 Counting inversions

Conteo de inversiones

Proposition

The sort-and-count algorithm counts the number of inversions in a permutation of size n in $O(n \log n)$ time.

Proof

The worst-case running time $T(n)$ satisfies the recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil) + \Theta(n) & \text{Otherwise} \end{cases}$$

If we apply Master theorem which case applies?

5.3 Counting inversions

Conteo de inversiones

Using the Master theorem and taking the recursive case we have:

$T(n) = 2T(\lceil n/2 \rceil) + \Theta(n)$

If a = 2, b = 2, considering f(n) = n, k=1

Comparing a vs b^k: $2 = 2^1$

Which is the Case 3.

$T(n) = \Theta(n^k \log n) = \Theta(n^1 \log n) = \Theta(n \log n)$

5.4 Finding the closest pair of points

Encontrando la pareja de puntos más cercanos

Descripción del problema

Solución usando D&C

Cálculo de la complejidad

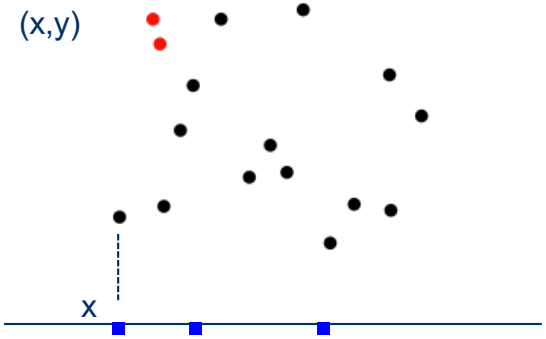
5.4 Finding the closest pair of points

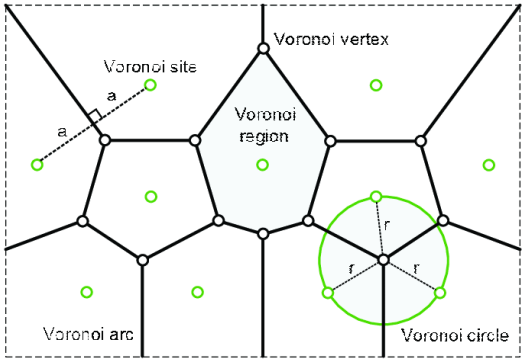
Encontrando la pareja de puntos más cercanos

Closest pair problem

Given n points in the plane, find a pair of points with the smallest Euclidean distance between them.

Special case of nearest neighbor, Euclidean MST, Voronoi.





Brute force. Check all pairs with $\Theta(n^2)$ distance calculations.

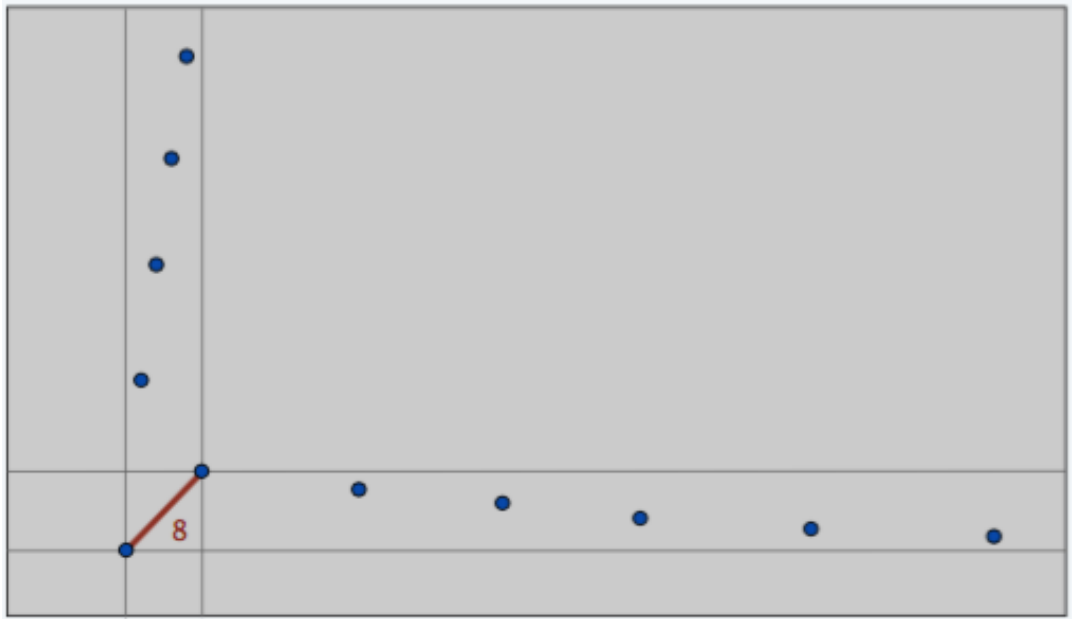
1D version. Easy $O(n \log n)$ algorithm if points are on a line.

5.4 Finding the closests pair of points

Encontrando la pareja de puntos más cercanos

Sorting solution

- Sort by *x*-coordinate and consider nearby points.
- Sort by *y*-coordinate and consider nearby points.

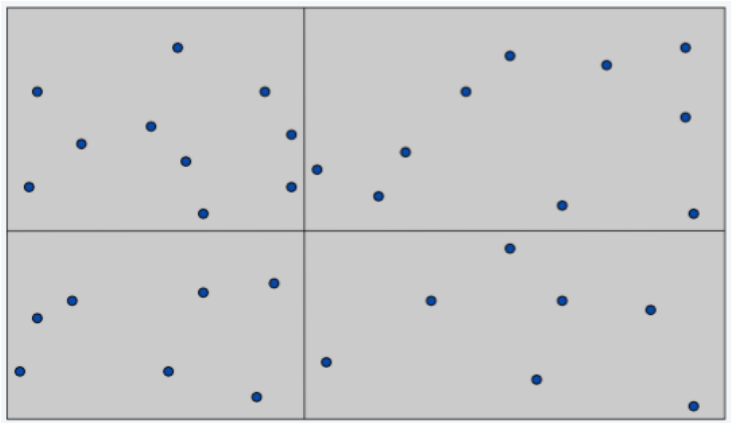


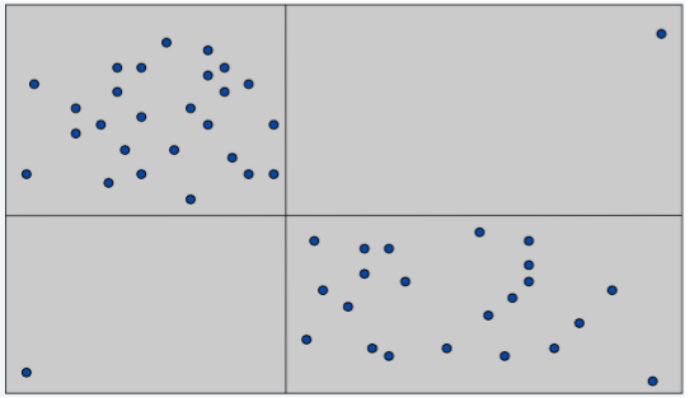
5.4 Finding the closests pair of points

Encontrando la pareja de puntos más cercanos

Divide. Subdivide region into 4 quadrants.

Obstacle. Impossible to ensure *n* / 4 points in each piece.



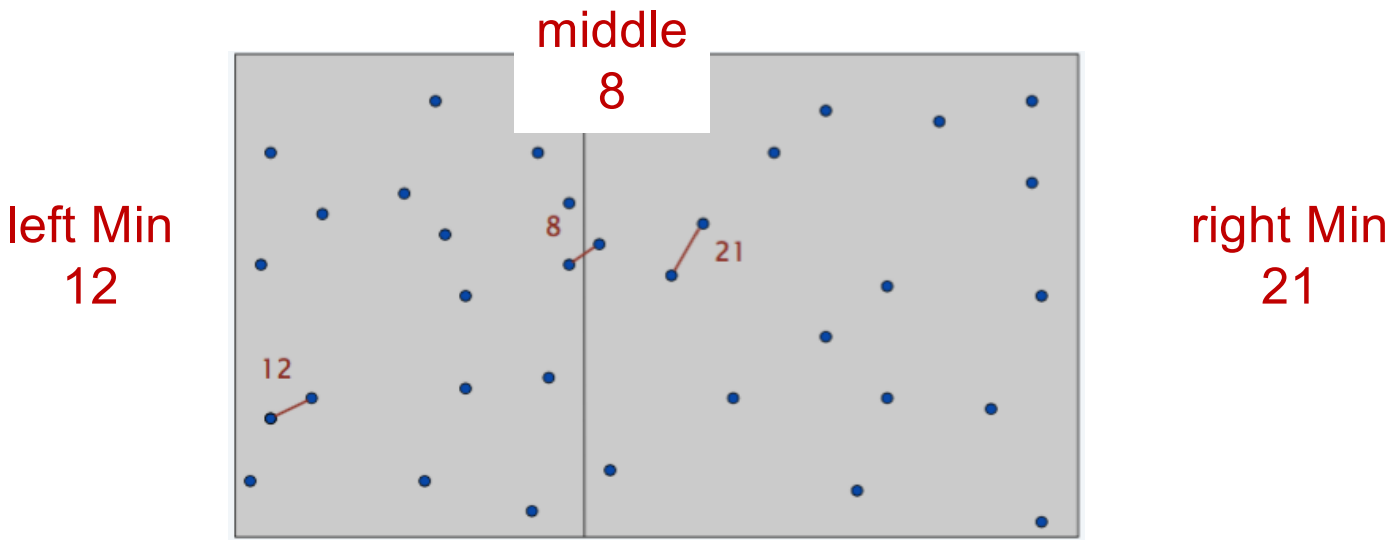


5.4 Finding the closests pair of points

Encontrando la pareja de puntos más cercanos

D&C Algo

1. Divide: draw vertical line *L* so that *n* / 2 points on each side.
2. Conquer: find closest pair in each side recursively.
3. Combine: find closest pair with one point in each side.
4. Return best of 3 solutions.

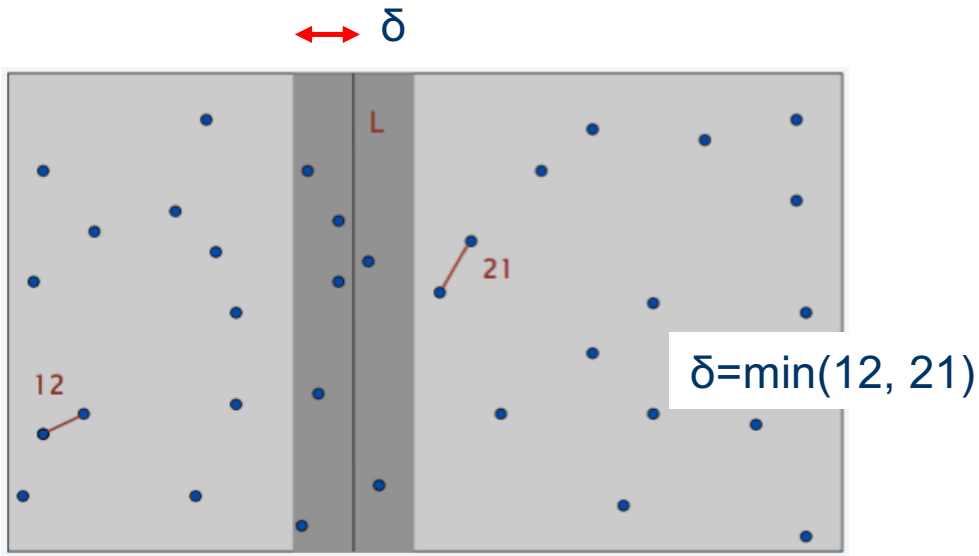


5.4 Finding the closests pair of points

Encontrando la pareja de puntos más cercanos

How to find closest pair with one point in each side?

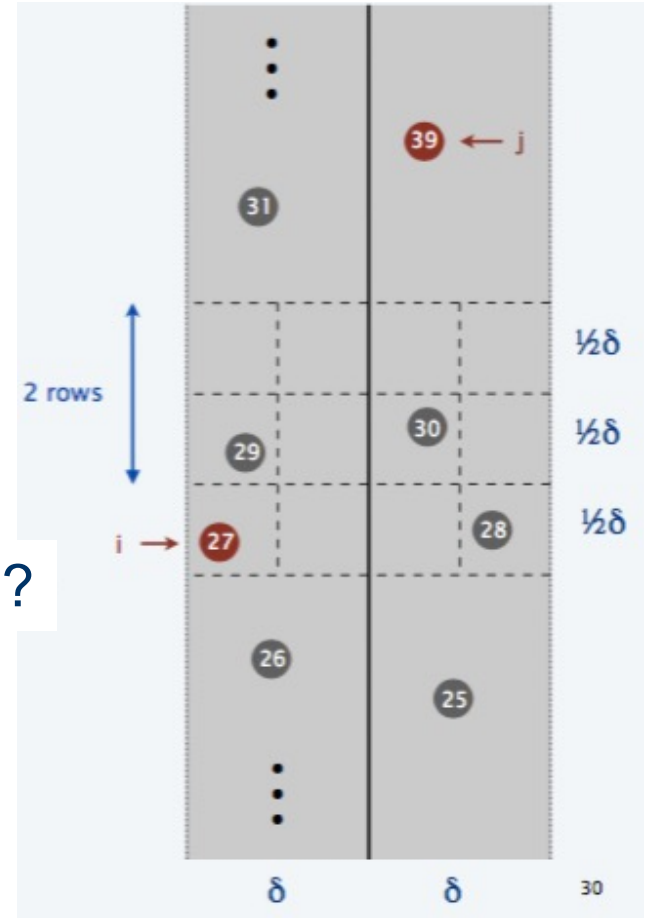
- Assuming that distance < δ (delta minúscula).
- Observation: only need to consider points within δ of line *L*.
- Sort points in 2 δ-strip by their *y*-coordinate.
- Only check distances of those within N positions (n=11?) in sorted list.



5.4 Finding the closests pair of points

Encontrando la pareja de puntos más cercanos

Only check distances of those within N positions in sorted list.



n=11?

5.4 Finding the closests pair of points

Encontrando la pareja de puntos más cercanos

CLOSEST-PAIR (p_1, p_2, ..., p_n)

if n<2
Caso base

- Compute separation line *L* such that half the points are on each side of the line (todos los puntos eje x). O(n log n)
- $\delta_1 \leftarrow$ CLOSEST-PAIR (points in left half).
- $\delta_2 \leftarrow$ CLOSEST-PAIR (points in right half). } 2 T (n/2)
- $\delta \leftarrow \min \{ \delta_1 , \delta 2 \}$. O(1)
- Delete all points further than δ from line *L*. O(n)
- Sort remaining points by *y*-coordinate. O(n log n)
- Scan points in *y*-order and compare distance between each point and next neighbors (How many? N). If any of these distances is less than δ_3, update δ. O(n)

RETURN δ.

Ecuación
De
Recurrencia

5.4 Finding the closests pair of points

Encontrando la pareja de puntos más cercanos

Theorem

The divide-and-conquer algorithm for finding the closest pair of points in the plane can be implemented in $O(n \log n)$ time.

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil) + O(n \log n) & \text{Otherwise} \end{cases}$$

If we apply Master theorem which case applies?

5.4 Finding the closests pair of points

Encontrando la pareja de puntos más cercanos

Using the Master theorem and taking the recursive case we have:

$T(n) = 2T(\lceil n/2 \rceil) + \Theta(n \log n)$

If a = 2, b = 2, considering f(n) = n log n

We have to use the general theorem:

- It applies case 3.
- If $f(n) = \Omega(n^{\log_b a+\varepsilon})$ then $T(n) = \Theta(f(n))$.
- We have $n^{\log_2 2+\varepsilon} = n^{1+\varepsilon}$, the lim. grows.
- $T(n) = \Theta(n \log n)$

Caso 3: El denominador crece más lento y el límite tiende a **aumentar** (infinito).

$$\lim_{x\to\infty}\left(\frac{f(n)}{n^{\log_b a+\epsilon}}\right)$$

5.5 Integer multiplication

Multiplicación de enteros

Planteamiento del problema

Y solución

5.5 Integer multiplication

Multiplicación de enteros

Linear algebra

It is the branch of mathematics concerning *vector spaces* and linear mappings between such spaces. It includes the study of lines, planes, and subspaces, but is also concerned with properties common to all vector spaces.

Many fields

- Linear transformations
  - Like other algebraic structures.
  - Mappings between vector spaces that preserve the vector-space structure
  - Eigenvalues and eigenvectors
- Matrix theory

La palabra alemana *eigen*, que se traduce en español como *propio*, se usó por primera vez en este contexto por **David Hilbert** en 1904.

Momento histórico

El programa de Hilbert

En 1920 propuso de forma explícita un proyecto de investigación que quería que la matemática fuese formulada sobre unas bases sólidas y completamente lógicas. Creía que, en principio, esto podía lograrse, mostrando que:

- Toda la matemática se construye de un sistema finito de axiomas escogidos correctamente; y
- Que tal sistema axiomático se puede probar consistente.

Momento histórico

El grupo Bourbaki

Nicolas Bourbaki

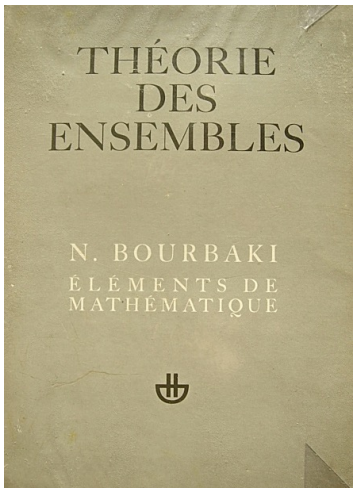
Es el nombre colectivo de un grupo de matemáticos franceses que, en los años 1930, se propusieron revisar los fundamentos de la matemática con una exigencia de rigor mucho mayor que la que entonces era moneda corriente en esta ciencia.

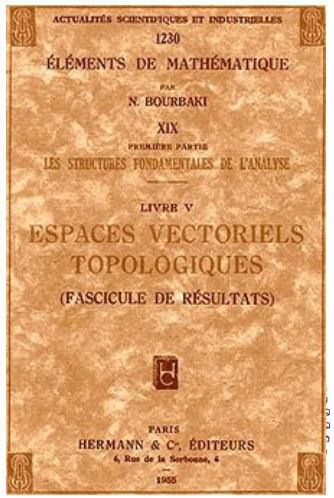
Elementos de matemática

- Teoría de conjuntos
- Algebra
- Topología
- Grupos y álgebras
- 10 volúmenes.

Enciclopedia

Henri Cartan
André Weil
Claude Chevalley
...
Jean Delsarte
Varias generaciones
Hasta 40 simultaneos





Integer adition

Suma de enteros

Addition. Given two *n*-bit integers *a* and *b*, compute *a* + *b*.

Subtraction. Given two *n*-bit integers *a* and *b*, compute *a* – *b*.

Grade-school algorithm. Θ(*n*) bit operations.

1	1	1	1	1	1	0	1	acarreo
	1	1	0	1	0	1	0	1 a
+	0	1	1	1	1	1	0	1 + b
1	0	1	0	1	0	0	1	0

Remark. Grade-school addition and subtraction algorithms are asymptotically optimal.

5.5 Integer multiplication

Multiplicación de enteros

Multiplication. Given two n-bit integers a and b, compute $a \times b$.

Grade-school algorithm. $\Theta(n^2)$ bit operations.

								1	1	0	1	0	1	0	1
							x	0	1	1	1	1	1	**0**	**1**
								1	1	0	1	0	1	0	1
							0	0	0	0	0	0	0	0	
						1	1	0	1	0	1	0	1		
					1	1	0	1	0	1	0	1			
				1	1	0	1	0	1	0	1				
			1	1	0	1	0	1	0	1					
		1	1	0	1	0	1	0	1						
	0	**0**	**0**	**0**	**0**	**0**	**0**	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1

a
x b

Same values

5.5 Integer multiplication

Multiplicación de enteros

Kolmogorov
Matemático sovietico
1903-1987

Conjecture. Andrey Kolmogorov 1952 (other sources 1956)
Grade-school algorithm is optimal $\Omega(n^2)$.

Theorem. Anatolii A. Karatsuba 1960 Conjecture is wrong.
It could multiply two n-digit numbers in $O(n^{\log_2 3})$.

$$1.584 \quad \frac{\log x}{\log 2}$$

Karatsuba
Matemático ruso
1937-2008

Preliminar list of algorithms

1. Karatsuba's fast multiplication method.
2. Strassen algorithm.
3. Coppersmith–Winograd algorithm.

https://en.wikipedia.org/wiki/Coppersmith-Winograd_algorithm

5.5 Integer multiplication

Multiplicación de enteros

To multiply two *n*-bit integers *x* and *y*:

- Divide *x* and *y* into low- and high-order bits.
- Multiply four *1⁄2n*-bit integers, recursively.
- Add and shift to obtain result.

Remember:

number = $2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0$ = 128+64+32+16+8+4+2+1

Example x * y = 141 * 125

La multiplicación es un corrimiento

8 13 7 13

x =1000 1101 *y* =0111 1101

a b c d

128

x = 141 = 128+8+4+1 = val*8 + 13 (val = 2^4 =16) = 16*8+13

5.5 Integer multiplication

Multiplicación de enteros

To multiply two *n*-bit integers *x* and *y*:

- Divide *x* and *y* into low- and high-order bits.
- Multiply four *1⁄2n*-bit integers, recursively.
- Add and shift to obtain result.

x =1000 1101 y =0111 1101

a b c d

En lugar de una multiplicación grande son 4 chicas

$$(2^m a + b) * (2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

1 2 3 4

$m = n/2$

$a = x / 2^m$ $b = x \bmod 2^m$

$c = y / 2^m$ $d = y \bmod 2^m$

use bit shifting to compute 4 terms

5.5 Integer multiplication

Multiplicación de enteros

```
MULTIPLY(x, y, n)
    if (n = 1)
        return x * y
    else
        m ← n / 2
        a ← x / 2^m,      b ← x mod 2^m
        c ← y / 2^m,      d ← y mod 2^m
        e ← MULTIPLY(a, c, m)          1
        g ← MULTIPLY(b, c, m)          2
        h ← MULTIPLY(a, d, m)          3
        f ← MULTIPLY(b, d, m)          4
       return 2^2m e + 2^m (g + h) + f
```

5.5 Integer multiplication

Multiplicación de enteros

Proposition

The divide-and-conquer multiplication algorithm requires $\Theta(n^2)$ bit operations to multiply two n-bit integers.

Proof

Apply case 1 of the master theorem to the recurrence:

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add/ shifts}} \Rightarrow T(n) = \Theta(n^2)$$

5.5 Integer multiplication

Multiplicación de enteros

Karatsuba trick

To compute middle term $bc + ad$, use identity:

$$bc + ad = ac + bd - (a - b)(c - d)$$

Aquí se usa un truco Algebraico para reducir el número de multiplicaciones

$$\begin{array}{cccccccccccc} & & & & & 1 & & 2 & & 3 & & 4 \\ (2^m & a & + & b) & (2^m & c & + & d) & = & 2^{2m} & ac & + & 2^m & (bc & + & ad) & + & bd \end{array}$$

$$= 2^{2m}\, ac + 2^m\, (ac + bd - (a - b)(c - d)) + bd$$

$$\begin{array}{cccccc} & & & & & \\ & 1 & & 1 & 3 & & 2 & & 3 \end{array}$$

Bottom line. Only three multiplication of n / 2-bit integers.

5.5 Integer multiplication

Multiplicación de enteros

KARATSUBA-MULTIPLY(x, y, n)

  if (n = 1)

    return $x * y$

  else

    $m \leftarrow n / 2$

    $a \leftarrow x / 2^m$, $b \leftarrow x \bmod 2^m$

    $c \leftarrow y / 2^m$, $d \leftarrow y \bmod 2^m$

    $e \leftarrow$ KARATSUBA-MULTIPLY(a, c, m) **1**

    $f \leftarrow$ KARATSUBA-MULTIPLY(b, d, m) **3**

    $g \leftarrow$ KARATSUBA-MULTIPLY(a-b, c-d, m) **2**

return $2^{2m} e + 2^m (e + f - g) + f$

5.5 Integer multiplication

Multiplicación de enteros

Karatsuba analysis

Proposition. Karatsuba's algorithm requires $O(n^{1.585})$ bit operations to multiply two n-bit integers.

Proof

Apply case 1 of the master theorem to the recurrence:

$$T(n) = \mathbf{3}\ T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n^{\lg_2 3}) = O(n^{1.585})$$

Practice

Faster when the multiplications are longer than 320-640 bits.

5.5 Integer multiplication

Multiplicación de enteros

Integer multiplication. Given two n-bit integers, compute their product.

integer arithmetic problems with the same complexity as integer multiplication

Problem	arithmetic
integer division	a/b, a mod b
integer square	a^2
integer square root	$\sqrt{a}$

5.5 Integer multiplication

Multiplicación de enteros

History of asymptotic complexity of integer multiplication:

Year	Algorithm	Order of growth
?	brute force	$\Theta(n^2)$
1962	Karatsuba-Ofman	$\Theta(n^{1.585})$
1963	Toom-3, Toom-4	$\Theta(n^{1.465})$, $\Theta(n^{1.404})$
1966	Toom-Cook	$\Theta(n^{1+\varepsilon})$
1971	Schönhage–Strassen	$\Theta(n \log n \log \log n)$
2007	Fürer	$n \log n \, 2^{O(\log^* n)}$
?		$\Theta(n)$

Remark. GNU Multiple Precision Library uses one of five different algorithm depending on size of operands.

5.5 Matrix multiplication

Multiplicación de matrices

Planteamiento del problema

Y solución

Matrix multiplication

Multiplicación de matrices

Dot product

Given two length n vectors a and b, compute $c = a \cdot b$.

Grade-school. $\Theta(n)$ arithmetic operations.

$$a \cdot b = \sum_{i=1}^{n} a_i \, b_i$$

a = [.70 .20 .10]

b = [.30 .40 .30]

$a \cdot b$ = (.70 × .30) + (.20 × .40) + (.10 × .30) = .32

Remark. Grade-school dot product algorithm is asymptotically optimal.

Matrix multiplication

Multiplicación de matrices

Matrix multiplication

Given two n-by-n matrices A and B, compute $C = AB$.

$$C_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

Grade-school. Which is the complexity? arithmetic operations.

$\Theta(n^3)$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

Question

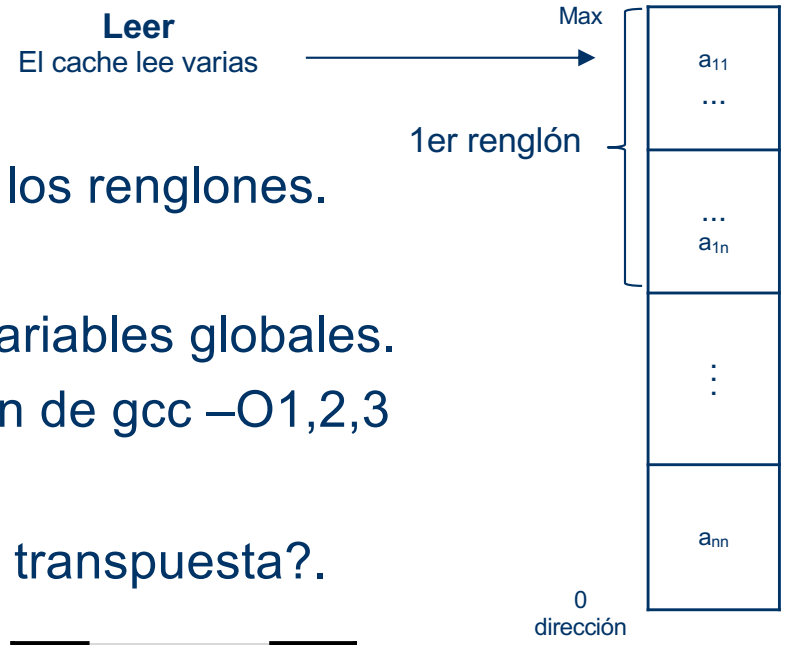
Is grade-school matrix multiplication algorithm asymptotically optimal?

Matrix multiplication

Multiplicación de matrices

Práctica

- Verificar el uso de la memoria cache en los renglones.
  - Medir tiempos.
  - Verificar el tamaño del espacio de variables globales.
  - Verificar las opciones de compilación de gcc –O1,2,3
- Usar la transpuesta de la matriz:
  - Cuanto tiempo tarda el cálculo de la transpuesta?.



$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

2^{32} = 4 GB

Matrix multiplication

Multiplicación de matrices

Strassen
Matemático Alemán
1936-80 años

Algoritmo de Strassen

- En la disciplina matemática del álgebra lineal, el algoritmo de Strassen, llamado así por Volker Strassen, es un algoritmo usado para la multiplicación de matrices.
- Es asintóticamente más rápido que el algoritmo de multiplicación de matrices estándar, pero más lento que el algoritmo más rápido conocido, y es útil en la práctica para matrices grandes.
- Volker Strassen publicó el algoritmo de Strassen en 1969. Fue el primero en señalar que el enfoque estándar no es óptimo. Su artículo comenzó la búsqueda de algoritmos aún más rápidos, como:
  - El complejo algoritmo de Coppersmith–Winograd de Shmuel Winograd en 2010.
  - Utiliza 20 multiplicaciones binarias, pero utiliza 155 sumas binarias en lugar de las 18 del algoritmo de Strassen. Fue publicado en 2000.

Matrix multiplication

Multiplicación de matrices

Idea del algoritmo de Strassen

- Dividir cada matriz en cuatro cuadrantes: A x B = C

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

$$C_{11} = A_{11}\,B_{11} + A_{12}\,B_{21}$$
$$C_{12} = A_{11}\,B_{12} + A_{12}\,B_{22}$$
$$C_{21} = A_{21}\,B_{11} + A_{22}\,B_{21}$$
$$C_{22} = A_{21}\,B_{12} + A_{22}\,B_{22}$$

- Se realizan 8 multiplicaciones de matrices de *N*/2 x *N*/2

Matrix multiplication

Multiplicación de matrices

- La ecuación de recurrencia es:

$$T(n) = 8T\left(\frac{n}{2}\right) + O(n^2)$$

- Al resolver esta ecuación la complejidad sigue siendo $O(n^3)$.
- Estrategia de Strassen:

  Disminuir el número de subproblemas: a sólo 7.

$$M_1 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$
$$M_2 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$
$$M_3 = (A_{11} - A_{21}) \cdot (B_{11} + B_{12})$$
$$M_4 = (A_{11} + A_{12}) \cdot B_{22}$$
$$M_5 = A_{11} \cdot (B_{12} - B_{22})$$
$$M_6 = A_{22} \cdot (B_{21} - B_{11})$$
$$M_7 = (A_{21} + A_{22}) \cdot B_{11}$$

Matrix multiplication

Multiplicación de matrices

- De tal forma que los cuadrantes se calculan como:

$$C_{11} = M_1 + M_2 - M_4 + M_6$$
$$C_{12} = M_4 + M_5$$
$$C_{21} = M_6 + M_7$$
$$C_{22} = M_2 - M_3 + M_5 - M_7$$

- La nueva ecuación recurrencia es:

$$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$$

- La complejidad queda como:

$$T(n) = O\left(n^{\log_2 7}\right) = O(n^{2.81})$$

Matrix multiplication

Multiplicación de matrices

Algorithm

Assume n is a power of 2

keep track of indices of submatrices (don't copy matrix entries)

STRASSEN (n, A, B)

IF (n = 1) RETURN A x B.
Partition A and B into 2-by-2 block matrices.

$P1 \leftarrow$ **STRASSEN** (n / 2, $A11$, ($B12 - B22$))
$P2 \leftarrow$ **STRASSEN** (n / 2, ($A11 + A12$), $B22$)
$P3 \leftarrow$ **STRASSEN** (n / 2, ($A21 + A22$), $B11$)
$P4 \leftarrow$ **STRASSEN** (n / 2, $A22$, ($B21 - B11$))
$P5 \leftarrow$ **STRASSEN** (n / 2, ($A11 + A22$) x ($B11 + B22$))
$P6 \leftarrow$ **STRASSEN** (n / 2, ($A12 - A22$) x ($B21 + B22$))
$P7 \leftarrow$ **STRASSEN** (n / 2, ($A11 - A21$) x ($B11 + B12$))

$C11 = P5 + P4 - P2 + P6$
$C12 = P1 + P2$
$C21 = P3 + P4$
$C22 = P1 + P5 - P3 - P7$
RETURN C

Matrix multiplication

Multiplicación de matrices

Question

What if n is not a power of 2?

Answer

Could pad matrices with zeros.

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 10 & 11 & 12 & 0 \\ 13 & 14 & 15 & 0 \\ 16 & 17 & 18 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 84 & 90 & 96 & 0 \\ 201 & 216 & 231 & 0 \\ 318 & 342 & 366 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Matrix multiplication

Multiplicación de matrices

Conclusiones

En la práctica, algoritmo de Strassen funciona mejor que el algoritmo simple cuando *N* es “grande”.

- El punto a partir del cual el algoritmo de Strassen es más eficiente depende de la implementación específica y del hardware.
- Se ha estimado que el algoritmo de Strassen es más rápido para:
  - Matrices con anchura desde 32 a 128 para implementaciones optimizadas, y
  - N= 60.000 o más para implementaciones básicas.

Links

- https://es.wikipedia.org/wiki/Algoritmo_de_Strassen.

5.6 Convolutions and the FFT

Transformada Rápida de Fourier

5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Theorem Fourier

- The theorem was developed by the French mathematician J.B. Fourier around 1800.
- Any (sufficiently smooth) periodic function can be expressed as the sum of a series of sines and cosines: **Fourier series**.
- Another example of application is the **Z-transform** which reduces to a Fourier series for the important case |z|=1.
- Fourier series are also central to the original proof of the Nyquist–Shannon sampling theorem.

Jean-Baptiste Joseph Fourier 1768–1830.

5.6 Convolutions and the FFT

Transformada Rápida de Fourier

- Fourier made important contributions to the study of **trigonometric series**.
- After preliminary investigations by Leonhard Euler, Jean le Rond d'Alembert, and Daniel Bernoulli.
- Fourier introduced the series for solving the **heat equation** in a metal plate (published in 1807 Mémoire sur la propagation de la chaleur dans les corps solides).
- **Trigonometric series** are of the form:

$$a_0 + \sum_{n=1}^{\infty} (a_n \cos(nx) + b_n \sin(nx))$$

It is called a Fourier series if the terms a_n and b_n have the form:

$$a_n = 1/\pi \int_0^{2\pi} f(x) \cos(nx)\, dx \quad n=0,1,2,3$$

$$b_n = 1/\pi \int_0^{2\pi} f(x) \sin(nx)\, dx \quad n=1,2,3$$

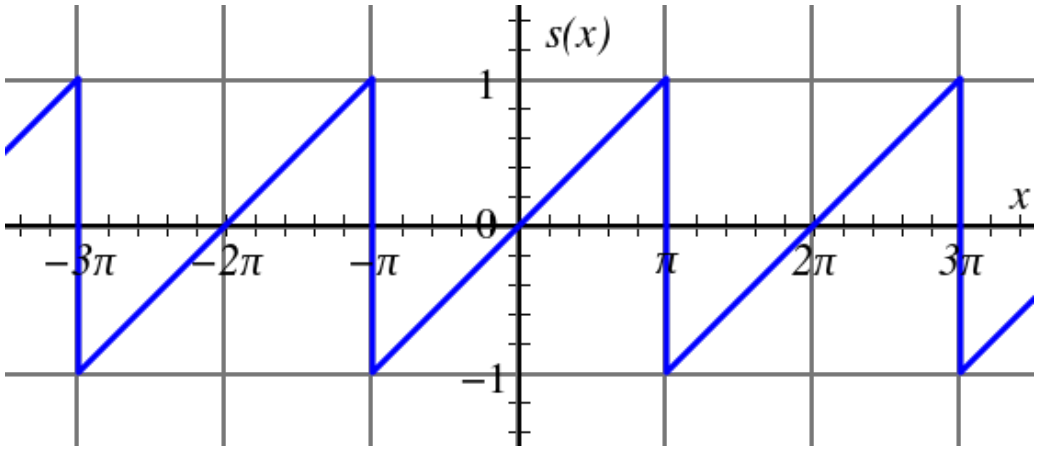
5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Una función periódica con periodo T_0 puede ser expresada como una serie de Fourier, de la siguiente manera:

$$y(t) = a_0/2 + \sum_{n=1}^{\infty} (a_n \cos(2\pi_0 t) + b_n \sin(2\pi_0 t))$$

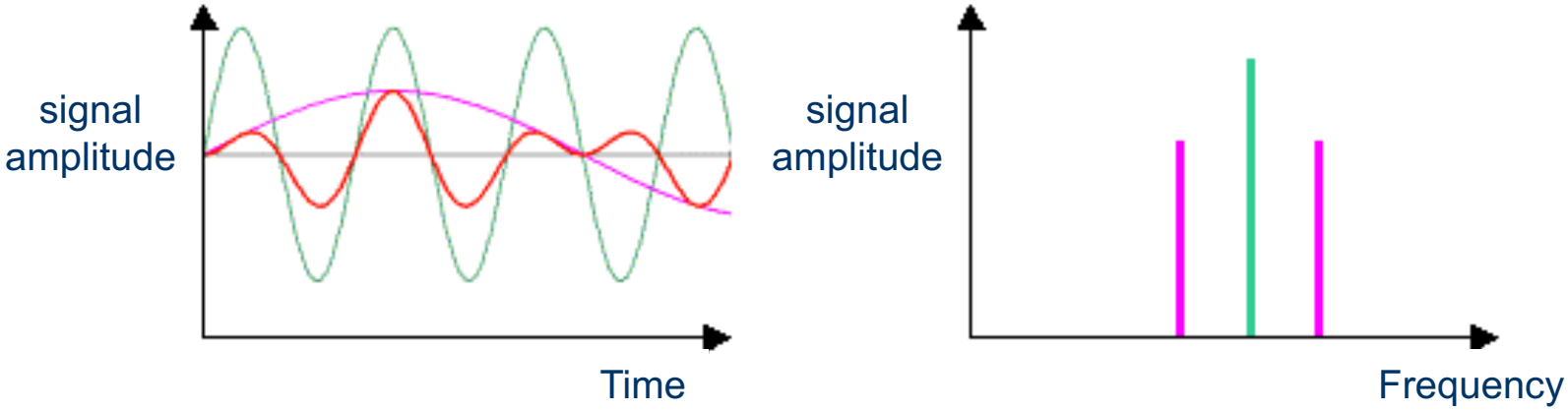
Example $s(x) = x/\pi$ for $-\pi < x < \pi$



5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Time domain vs. frequency domain



Time domain

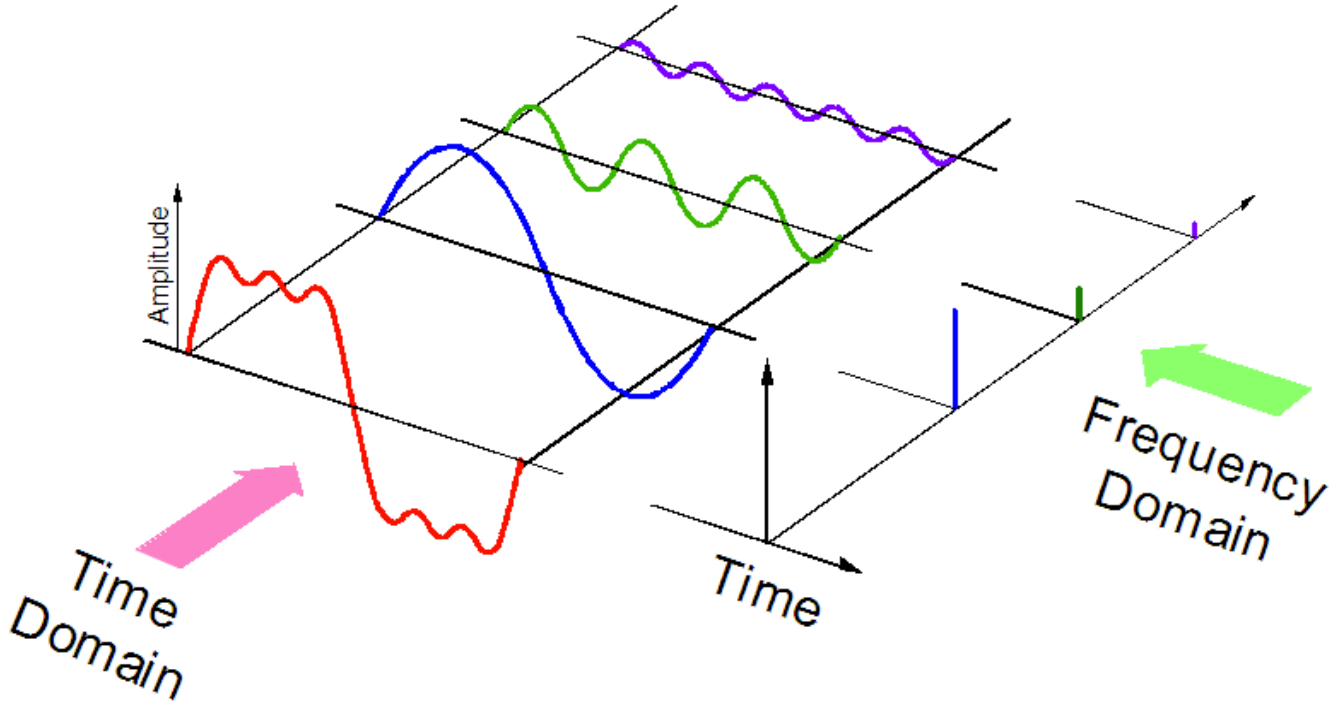
Frequency domain

Se representan señales senoidales (amp y frec)

5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Time domain vs. frequency domain



5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Applications of FFT

- Optics, acoustics, quantum physics, telecommunications, radar, control systems, signal processing, speech recognition, data compression, image processing, seismology, mass spectrometry, ...
- Digital media. [DVD, JPEG, MP3, H.264].
- Medical diagnostics. [MRI, CT, PET scans, ultrasound].
- Numerical solutions to Poisson's equation.
- Integer and polynomial multiplication.
- Shor's quantum factoring algorithm.
- ...

5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Transformada Z

Convierte una señal real o compleja definida en el dominio del tiempo discreto en una representación en el dominio de la frecuencia compleja.

Transformada de Laplace

Es un tipo de transformada integral frecuentemente usada para la resolución de ecuaciones diferenciales ordinarias. Por lo tanto utiliza señales continuas (reales).

- El nombre de Transformada Z procede de la variable del dominio, al igual que se podría llamar "Transformada S" a la Transformada de Laplace.
- La TZ es a las señales de tiempo discreto lo mismo que Laplace a las señales de tiempo continuo. serie de Taylor
- La serie de Laurent de una función compleja f(z) es la representación de la misma función en la forma de una serie de potencias, la cual tiene términos de grado negativo.
- **Transformada de Hilbert** de una función real se obtiene mediante la convolución de 2 señales y se puede interpretar como la salida de un sistema Linear Time-Invariant (LTI).

5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Serie
The sum of the terms of an infinite sequence.

Divergent
A value may not always be given to such an infinite sum.

Convergent
If the partial sum of the first terms tends to a limit when the number of terms increases indefinitely.

Series

Trigonometric
- Fourier

Power
- Taylor (derivadas)
- Laurent

The study of infinite series is a major part of **mathematical analysis**

Transformadas

Converting between Domains

Time ↓ Frequency

Inversa ↑

Z (señales de tiempo discreto)
Laplace (señales de tiempo continuo)
Hilbert (convoluciones)

5.6 Convolutions and the FFT

Transformada Rápida de Fourier

FFT (Fast Fourier Transform)

Fast way to convert between time-domain and frequency-domain.

Alternate viewpoint.

Fast way to **multiply and evaluate polynomials** (we take this approach) using euler's identity.

Euler's identity

$e^{ix} = \cos(x) + i \sin(x)$

Sinusoids

Sum of sine and cosines = sum of complex exponentials.

5.6 Convolutions and the FFT

Transformada Rápida de Fourier

- El algoritmo de FFT para una variable usa:

$$F(\mu) = 1/N \sum_{x=0}^{N-1} f(x)\, e^{-j2\pi\mu x / N}$$

- Una TF de varias variables puede ser calculada por aplicación sucesiva de la TF de una variable.
- FFT inversa (frecuencia a tiempo).

  Resulta que todo algoritmo que se implemente para calcular la FFT discreta con modificaciones simples en sus entradas, puede ser utilizado para el cálculo de la inversa.

5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Ahorro

- En el número de operaciones es significativo.
- Por ejemplo para imágenes de 1024 x 1024 pixels, N = 1024, se tendría:

  N^2 = 1,048,576 operaciones complejas.

  $N^2 \log_2 N$ = 10,240 operaciones complejas.

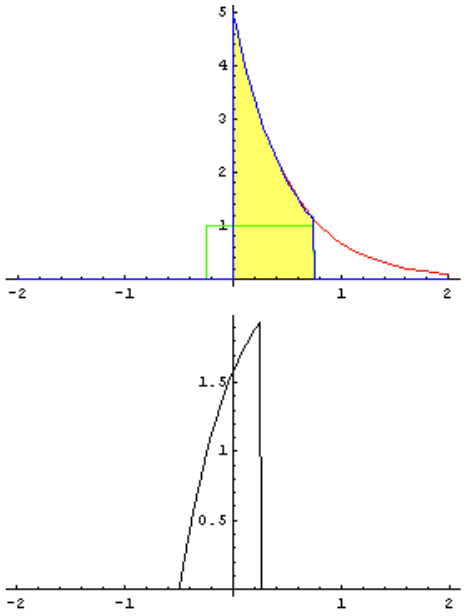
- Reducción de 102.4 a 1, el tiempo de cómputo.

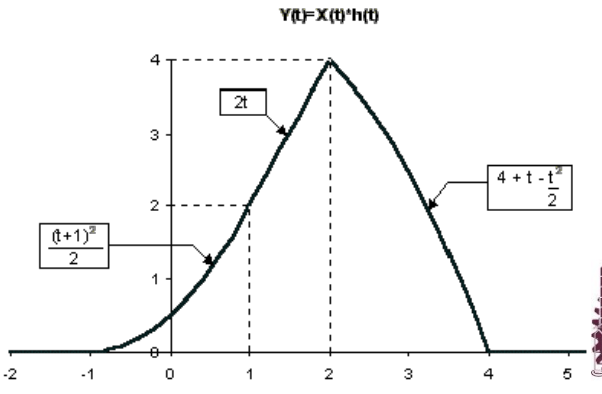
5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Convolutions

- La convolución entre dos funciones es un concepto físico importante en muchas ramas de la ciencia.
- Como muchas relaciones matemáticas importantes, no es sencillo comprender sus alcances e implicaciones.
- Es un **operador matemático** que transforma dos funciones f y g en una 3ª que en cierto sentido representa la magnitud en la que se superponen f y una versión trasladada e invertida de g.





5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Aplicación (Ingeniería)

- Si una función puede ser descompuesta en impulsos, y
- Si la respuesta a cualquier impulso es el impulso unitario desplazado y escalado, y
- Si la suma de los componentes (impulsos) de salida da la salida total.
- Entonces si se conoce la respuesta al impulso: **Se conoce todo sobre el sistema**.

- Para sistemas lineales e invariantes en el tiempo, la integral de convolución permite determinar la respuesta del sistema ante cualquier entrada, conociendo la respuesta del sistema ante una única entrada particular, el impulso.

FIN del mini-repaso!

5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Polynomial

$$A(x)=a_0 +a_1x+a_2x^2 +...+a_{n-1}x^{n-1}$$
$$B(x)=b_0 +b_1x+b_2x^2 +...+b_{n-1}x^{n-1}$$

Polynomials
coefficient representation

Add $O(n)$ arithmetic operations.

$$A(x)+B(x) = (a_0 +b_0)+(a_1+b_1)x+...+(a_{n-1}+b_{n-1})x_{n-1}$$

Evaluate $O(n)$ using Horner's method

$$A(x) = a_0 +(x(a_1+x(a_2 +...+x(a_{n-2} +x(a_{n-1}))...))$$

Multiply (convolve). $O(n^2)$ using brute force

$$A(x) \cdot B(x) = \sum_{i=0}^{2n-2} c_i\, x^i, \text{ where } c_i = \sum_{j=0}^{i} a_j\, b_{i-j}$$

5.6 Convolutions and the FFT

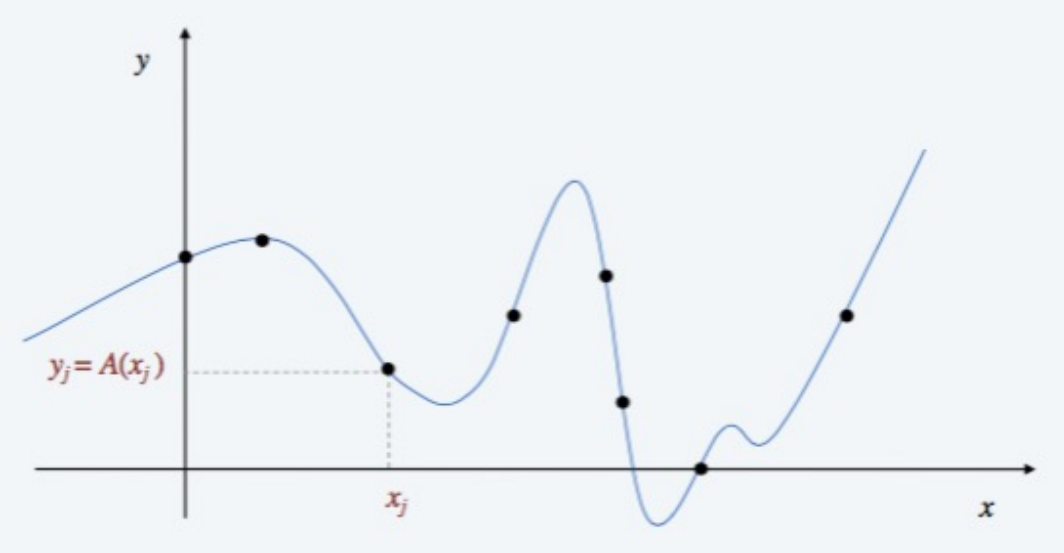
Transformada Rápida de Fourier

Fundamental theorem of algebra

A degree n polynomial with complex coefficients has exactly n complex roots.

Corollary

A degree $n - 1$ polynomial $A(x)$ is uniquely specified by its evaluation at n distinct values of x.



5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Polynomial [coefficient representation]

$A(x)$: $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$

$B(x)$: $(x_0, z_0), \ldots, (x_{n-1}, z_{n-1})$

Polynomials
point-value representation

Add $O(n)$ arithmetic operations.

$A(x)+B(x)$: $(x_0, y_0+z_0), \ldots, (x_{n-1}, y_{n-1}+z_{n-1})$

Multiply (convolve) $O(n)$ but need 2n-1 points

$A(x) \cdot B(x)$: $(x_0, y_0 \cdot z_0), \ldots, (x_{n-1}, y_{n-1} \cdot z_{n-1})$

Evaluate $O(n^2)$ using Lagrange formula

$$A(x)= \sum_{k=0} y_k \left(\prod(x-x_j)\right) / \prod(x_k-x_j)$$

5.6 Convolutions and the FFT

Transformada Rápida de Fourier

Converting between two representations

Tradeoff

Fast evaluation or fast multiplication. We want both!

representation	multiply	evaluate
coefficient	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n^2)$

Goal

Efficient conversion between two representations ⇒ all ops fast.

$a_0 , a_1, ..., a_{n-1}$ $\longleftrightarrow$ $(x_0 , y_0), ..., (x_{n-1}, y_{n-1})$

coefcient representation — point-value representation

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Ejemplo extra fuera del temario

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

- Vamos a resolver el problema de la siguiente forma:
  - Solución 1: Fuerza bruta
  - Solución 2: Mejora a la fuerza bruta
  - Solución 3: Usando D&C
  - Solución 4: Último algoritmo eficiente

- Existen varios problemas que se han resuelto poco a poco (a pasos).
- Este procedimiento de encontrar una solución y mejorarla se le conoce como **refinamiento**.
- En ocasiones el refinamiento da pistas para proponer una nueva solución disruptiva.

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Problema

- Dados enteros $A_1, \ldots, A_n$ (posiblemente negativos), encontrar el valor máximo de:

$$\sum_{k=i}^{j} A_k$$

- Si todos los números son negativos, la subsecuencia de suma máxima es 0.
- Ejemplo:

  Secuencia: -2,11,-4,13,-5,-2

  Respuesta: 20

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Análisis

- El espacio de soluciones (posibles) esta dado por la fórmula:

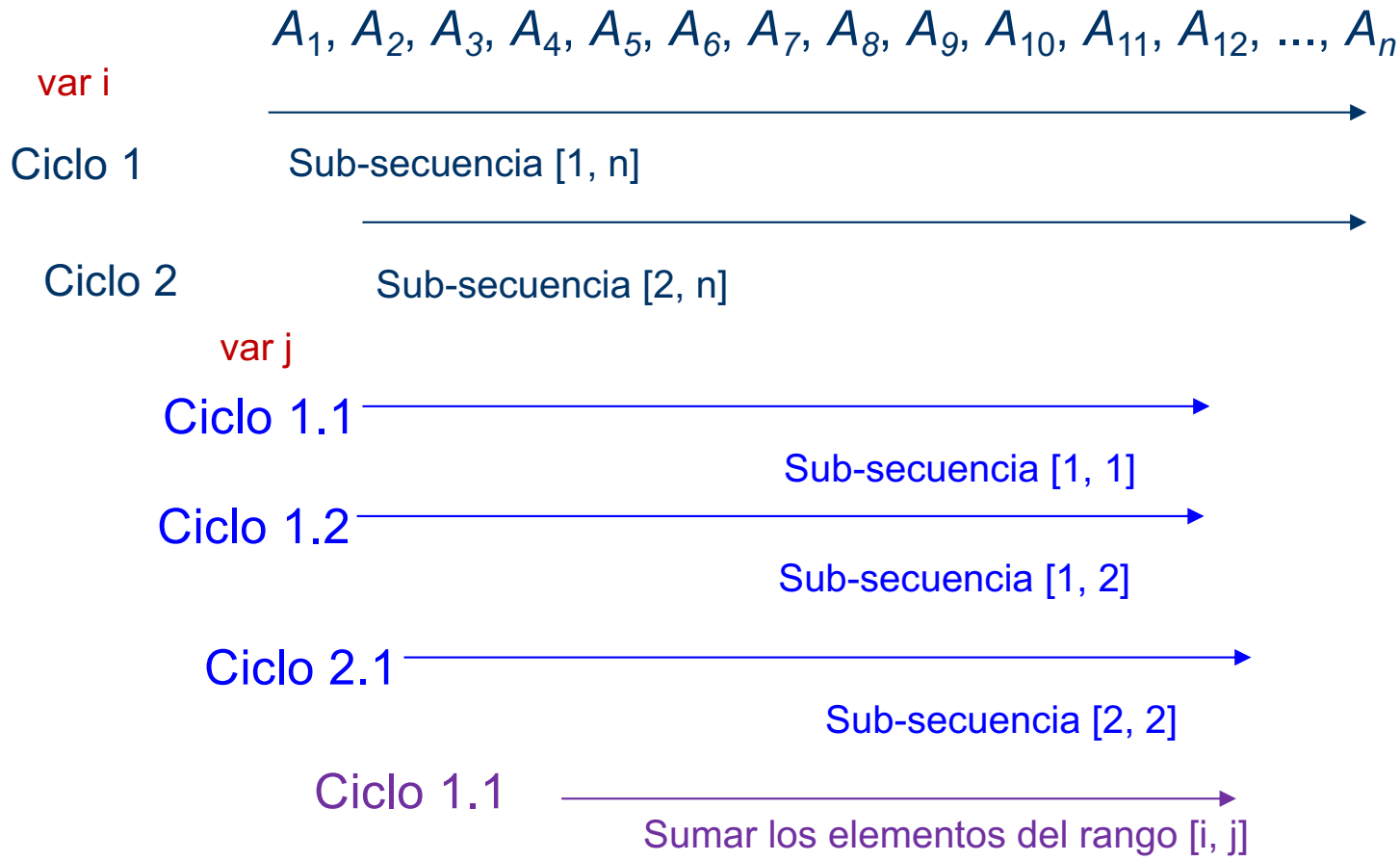
$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

- Asi que la complejidad de solo generar el espacio sería de inicio $O(n^2)$.
- Pero además a este cálculo se le agregan las operaciones “extras” que se hagan.

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 1 Fuerza bruta



5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 1 Fuerza bruta

- Calcular la suma de todas las subsecuencias.
- Quedarse con la suma mayor.
- Código:

```
int maxSum = 0;
for( i=0; i<a.length; i++)
{
  for( j=i; j<a.length; j++)
   {
     int thisSum = 0;
     for (k=i; k<=j; k++)
       thisSum += a[k];
     if (thisSum > maxSum)
       maxSum = thisSum;
   }
}
```

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-2} \sum_{k=i}^{j} 1 = (n^3 + 3n^2 + 2n) / 6$$

$$O(n^3)$$

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 2 Mejora a Fuerza bruta

- Nótese que de la ecuación:

$$\sum_{k=i}^{j} A_k = A_j + \sum_{k=i}^{j-1} A_k$$

- Por lo tanto, el tercer ciclo **for** se puede eliminar.
  - Es decir, al recorrer el segundo ciclo se hace al mismo tiempo la suma de los elementos.

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 2 Mejora a Fuerza bruta

- Código:

```
int maxSum = 0;
for( i=0; i<a.length; i++)
{
  int thisSum = 0;
   for (j=i; j<=a.length; j++)
   {
     thisSum += a[j];
     if (thisSum > maxSum)
       maxSum = thisSum;
   }
}
```

Se tienen sólo 2 ciclos ”for” de la misma longitud

$O(n^2)$

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 3 D&C

Idea principal es:

- Dividir el problema en dos sub-problemas del mismo tamaño.
- Resolver recursivamente.
- Mezclar las soluciones.
- Obtener solución final.

Subsecuencia de suma máxima puede estar en tres partes:

- Primera mitad
- Segunda mitad
- Cruza por el medio ambas mitades

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 3 D&C

Ejemplo:

Secuencia inicial
4 -3 5 -2 -1 2 6 -2

Primera mitad	Segunda mitad
4 -3 5 -2	-1 2 6 -2

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 3 D&C

Ejemplo:

Primera mitad	Segunda mitad
4 -3 5 -2	-1 2 6 -2

Suma máxima primera mitad: 6

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 3 D&C

Ejemplo:

Primera mitad	Segunda mitad
4 -3 5 -2	-1 2 6 -2

Suma máxima segunda mitad: 8

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 3 D&C

Ejemplo:

Primera mitad	Segunda mitad
4 -3 5 **-2**	**-1** 2 6 -2

- Suma máxima incluyendo último primera mitad: 6-2=4
- Idem primer elemento segunda mitad: 8-1=7
- Total: 4 + 7 = 11
  Mayor que máximo en ambas mitades.

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 3 Algoritmo

- Dividir secuencia en dos (izquierda, derecha)
- Resolver recursivamente las mitades (6 y 8)
  - Caso base: secuencia de largo 1
- Calcular suma máxima centro
  - = borde izquierdo + borde derecho = 6+8 = 14
- retornar max { izquierda, derecha, centro }

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 3 Complejidad

- Dos llamadas recursivas de tamaño $n/2$
- Suma máxima centro: $O(n)$
- Ecuación de recurrencia:

$$T(n) = 2\,T(n/2) + O(n)$$

- Complejidad

  $O(n \log n)$

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 4 Complejidad

Observaciones

- No es necesario conocer donde esta la mejor subsecuencia.
- La mejor subsecuencia no puede comenzar en un número negativo.
  - Cualquier subsecuencia negativa no puede ser prefijo de la subsecuencia óptima.

Inducción (reforzada)

- Se conoce la mejor subsecuencia entre 1 y *j*
- Se conoce la mejor subsecuencia que termina en *j*

Observese que esto no es D&C

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 4 Algoritmo

- Se almacenan ambos valores (inicialmente 0).
- Se incrementa *j* en 1.
- Se actualiza mejor subsecuencia si es necesario.
- Si subsecuencia que termina en j es < 0 se puede descartar, volver su valor a 0.

0	1	2	3	4	5	6	7
4	-3	5	-2	-1	2	6	-2

i j+1 j+2

Suma subsec =	4			6				11
		1		6	4	3	5	11	9

5.7 Maximum Subsequence Sum

Subsecuencia de Suma Máxima

Solución 4 Código

```
int maxSum = 0;
int thisSum = 0;

for( j=0; j<a.length; j++) {
      thisSum += a[j];
      if (thisSum > maxSum)
           maxSum = thisSum;
      else if (thisSum < 0)
           thisSum = 0;
 }
```

Complejidad

O(n)

Several topics

Varios temas

Notas históricas

Problemas abiertos

Tareas

URLs

A recordar

Recommended reading and Web sites

Lecturas recomendadas

Remember that this can change at any time (o estar fuera de linea)

Sorting algorithms

1. Animations:
    1. http://www.sorting-algorithms.com/.
    2. http://sorting.at/
    3. https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html
2. Tower .
    1. http://www.rodoval.com/heureka/hanoi/

Open problemas

Problemas sin resolver

Número primo de Mersenne

- Un número M es un número de Mersenne si es una unidad menor que una potencia de 2. $M_n = 2^n-1$.
- Un número primo de Mersenne es un número de Mersenne que es primo, es decir, $M_n=2^n-1$, con n primo (no es una condición suficiente que n sea primo para que M_n lo sea).

Links

- https://es.wikipedia.org/wiki/N%C3%BAmero_primo_de_Mersenne
- Premios: http://www.mersenne.org/

El más grande conocido es
$M_{74,207,281}$

Homeworks

Tareas

Resolver los problemas de:

1. Libro Kleinberg, Capítulo 4.
   4 problemas
2. Libro Cormen, Section 2.3.1.
   7 ejercicios y 4 problemas

The end

Contacto

Raúl Acosta Bermejo

http:www.cic.ipn.mx
http://www.ciseg.cic.ipn.mx/

racostab@ipn.mx
racosta@cic.ipn.mx

57-29-60-00
Ext. 56652