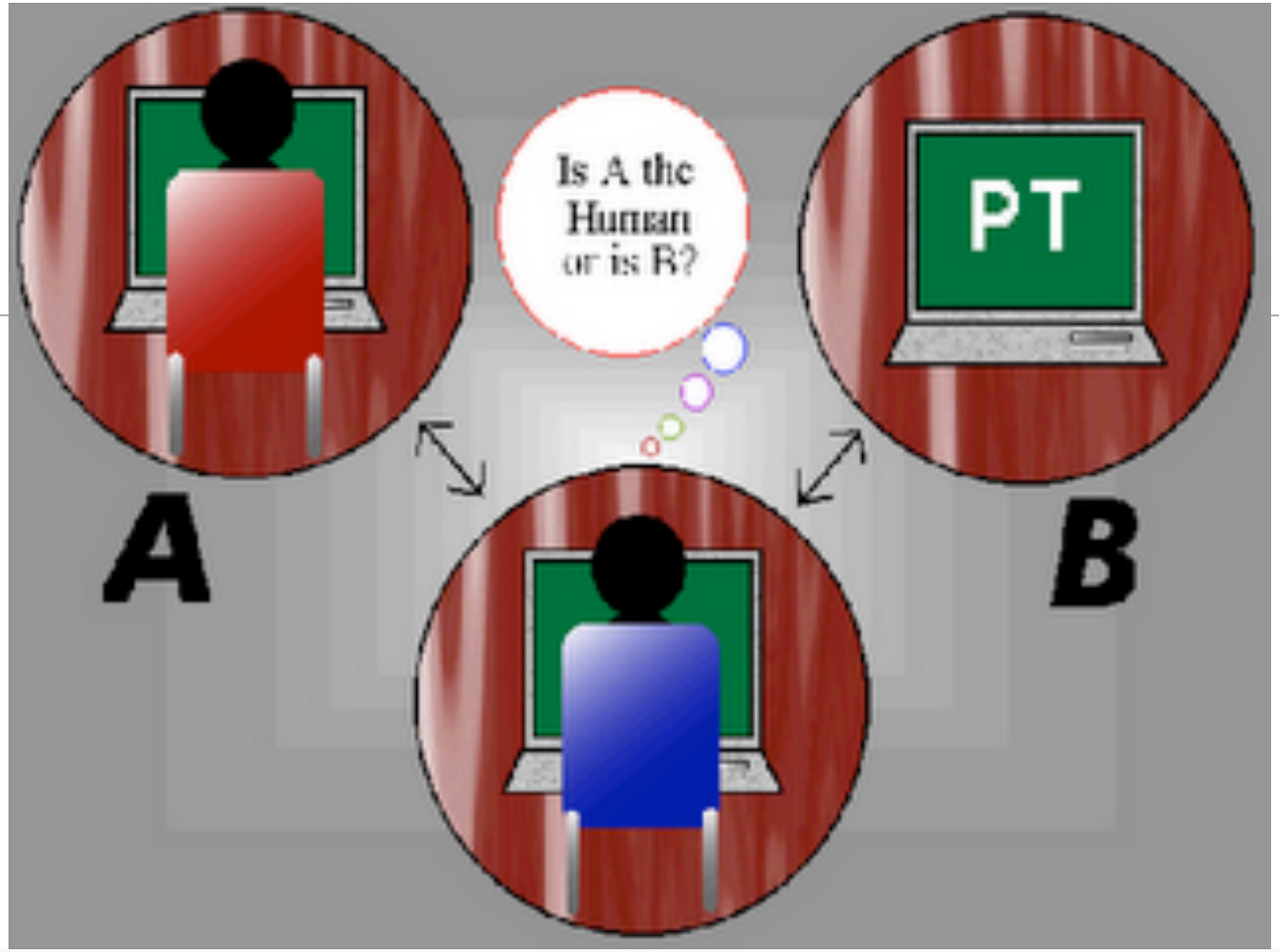
Análisis de algoritmos y Notación Asintótica

Prueba de Turing

Piensa una estrategia para, usando una operación aritmética, puedas detectar a la computadora. Supón que la computadora es honesta y el humano es deshonesto.



Explique porque esta función genera números aleatorios entre 0 y $10^{pot10}-1$

```
using namespace std;


long int giveMeRandomNPot10(int pot10){

        long int n=0,fact=1; //n es resultado y factor potencia de 10
        for(int i=0;i<pot10;i++){
            n += (rand() % 10) * fact;
            fact *=10;
        }
        return n;

}
```

Indique por qué este código verifica que los números aleatorios generados tienen distribución uniforme.

```
#define N_MAX 500000
int main(){
    long int count;
    long int vec[N_MAX];

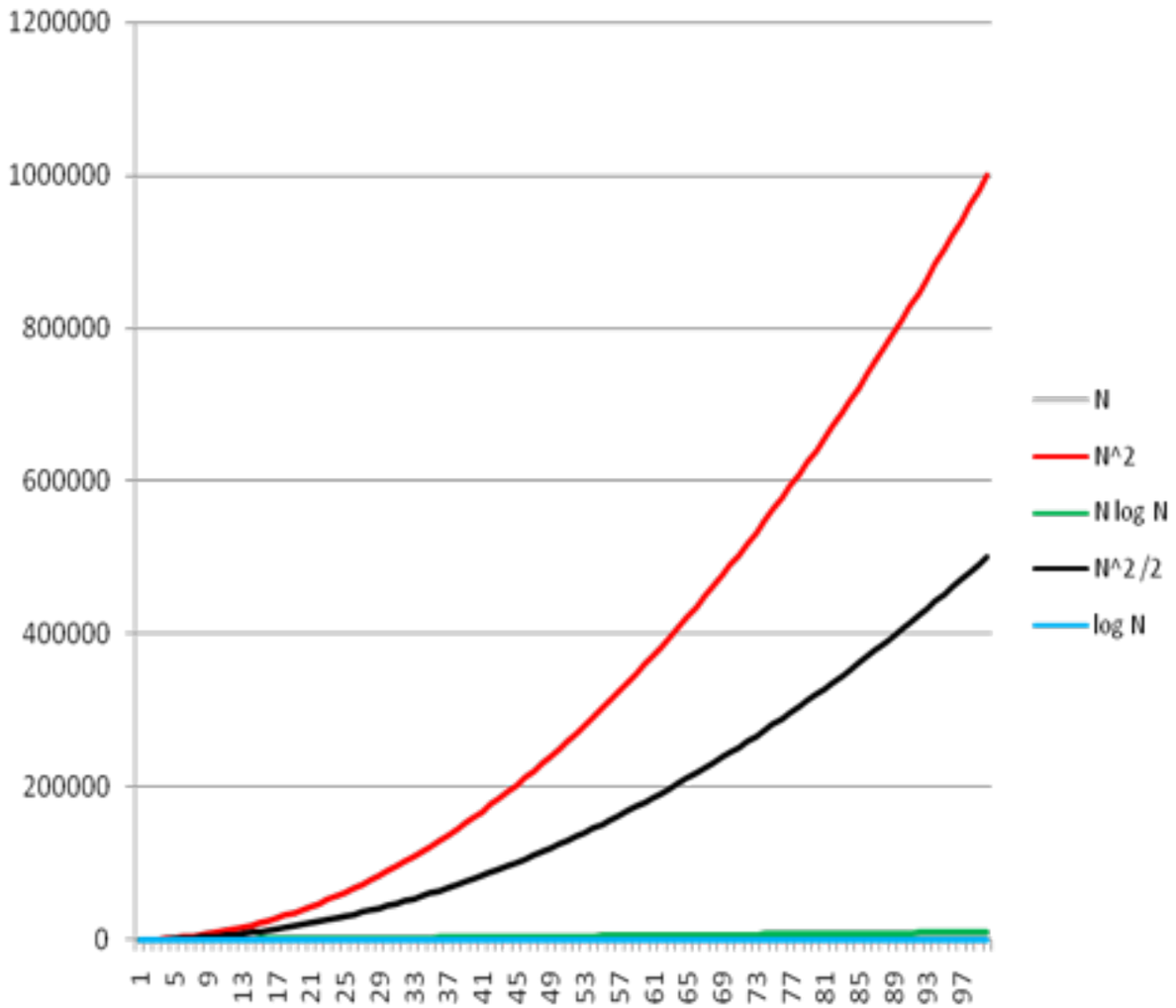
    for(int i=0;i<N_MAX;i++)
        vec[i] = 0; // inicializar contador a cero

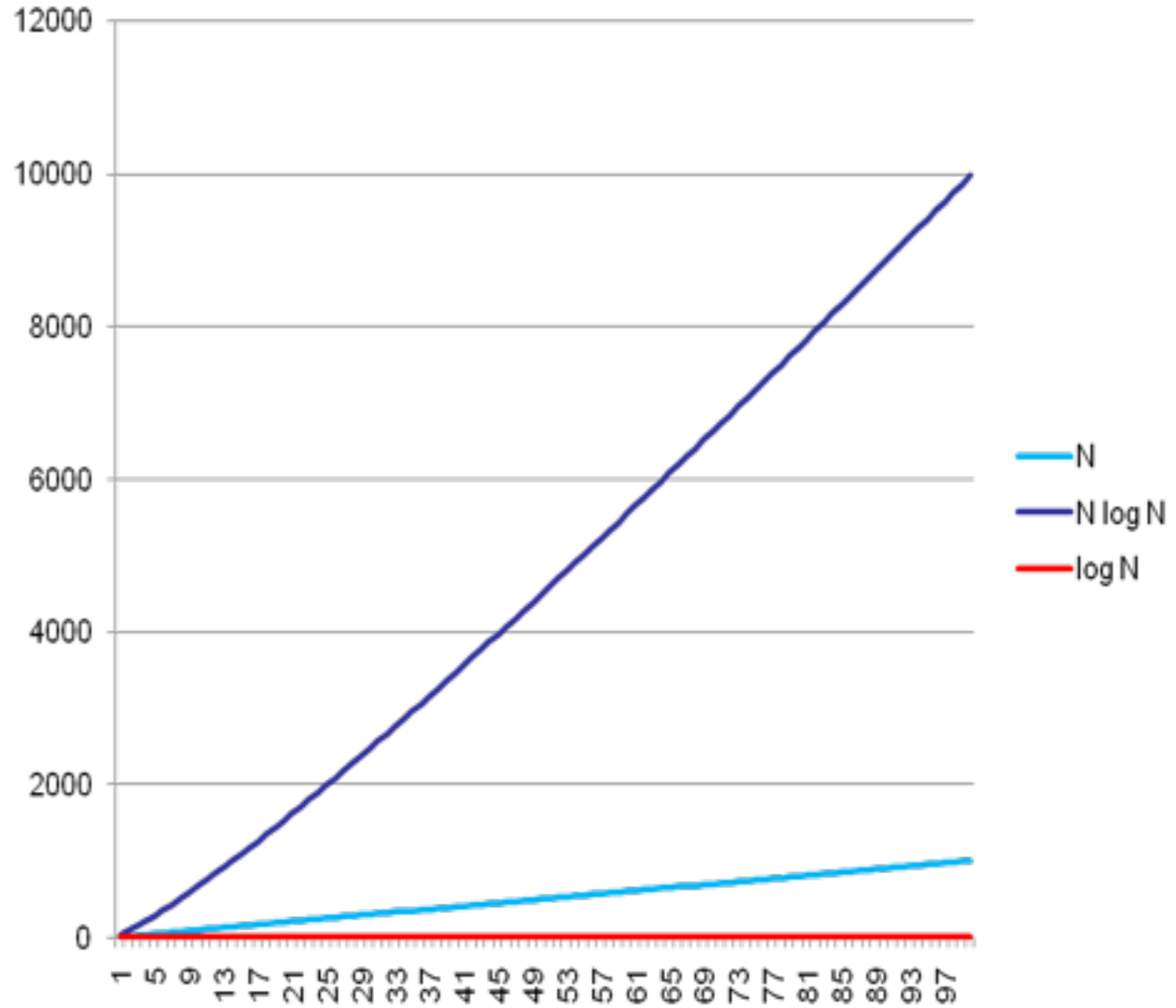
    for(count =0; count<100000000;count++){ //generar 100M muestras
        vec[giveMeRandomNPot10(6)%N_MAX]++; // ir contando
    }


    for(int i=0;i<N_MAX;i++) // manda a pantalla los contadores
        cout << vec[i] << endl;

    cout << "cont esperado / bin es: " << 100000000/N_MAX << endl;
}
```

Complejidad - Funciones de crecimiento (4)





Complejidad

- Cuidar hacer la diferencia entre problemas donde la entrada sea:
    - Un número **N de objetos,** queremos saber qué pasa cuando tenemos 1 reina o 1000 reinas.
    - Un **número N**, en un algoritmo donde las operaciones se harán sobre este número (en este ejemplo la complejidad es el número de bits que tome representar a **N**)

Complejidad en el espacio

- Caracterización de una implementación en términos de la memoria que requiere (espacio)

- Función ***M(n)*** del “tamaño” de los datos **(n)**, que permite evaluar la cantidad de memoria requerida.

Complejidad en el tiempo

Complejidad en el tiempo

- Función ***T(n)*** que describe, si es posible, **el comportamiento del algoritmo en cuanto a su tiempo de ejecución**, en términos de ***n***, el tamaño de datos de entrada al algoritmo.

Complejidad en el tiempo

- Función ***T(n)*** que describe, si es posible, **el comportamiento del algoritmo en cuanto a su tiempo de ejecución**, en términos de ***n***, el tamaño de datos de entrada al algoritmo.

- Si se puede, se expresa la función ***T(n)*** de manera exacta en términos de las operaciones elementales (operaciones aritméticas, operaciones de asignación), y no en términos de tiempo absoluto ...

Complejidad en el tiempo

- Función ***T(n)*** que describe, si es posible, **el comportamiento del algoritmo en cuanto a su tiempo de ejecución**, en términos de ***n***, el tamaño de datos de entrada al algoritmo.
- Si se puede, se expresa la función ***T(n)*** de manera exacta en términos de las operaciones elementales (operaciones aritméticas, operaciones de asignación), y no en términos de tiempo absoluto ...
- Nos interesa en particular:
    - ver lo que pasa para **grandes números de objetos**
    - ver la **escalabilidad** del programa (que pasa al atender a 10 clientes, 11 clientes, etc.)
    - **comparar** independientemente del hardware.

Complejidad en el tiempo

Complejidad en el tiempo

- Se considera la complejidad en términos de **operaciones elementales**.

Complejidad en el tiempo

- Se considera la complejidad en términos de **operaciones elementales**.

- `void Ex1(int n){`

Complejidad en el tiempo

- Se considera la complejidad en términos de **operaciones elementales**.

- `void Ex1(int n){`
  - `int i,j;`

Complejidad en el tiempo

- Se considera la complejidad en términos de **operaciones elementales**.

- `void Ex1(int n){`
  - `int i,j;`
  - `int sum=0;`

Complejidad en el tiempo

- Se considera la complejidad en términos de **operaciones elementales**.

- ```
  void Ex1(int n){
    int i,j;
    int sum=0;
    for ( i=1; i<n; i++ ){
  ```
  ```

Complejidad en el tiempo

- Se considera la complejidad en términos de **operaciones elementales**.

- ```
  void Ex1(int n){
    int i,j;
    int sum=0;
    for ( i=1; i<n; i++ ){
      for ( j=1; j<i; j++){
  ```
  ```

Complejidad en el tiempo

- Se considera la complejidad en términos de **operaciones elementales**.

- ```
  void Ex1(int n){
    int i,j;
    int sum=0;
    for ( i=1; i<n; i++ ){
      for ( j=1; j<i; j++){
        ++sum;
  ```
  ```

Complejidad en el tiempo

- Se considera la complejidad en términos de **operaciones elementales**.

- ```
  void Ex1(int n){
    int i,j;
    int sum=0;
    for ( i=1; i<n; i++ ){
      for ( j=1; j<i; j++){
        ++sum;
      }
  ```
  ```

Complejidad en el tiempo

- Se considera la complejidad en términos de **operaciones elementales**.

- ```
  void Ex1(int n){
    int i,j;
    int sum=0;
    for ( i=1; i<n; i++ ){
      for ( j=1; j<i; j++){
        ++sum;
      }
  }
  ```
  ```

- `void Ex1(int n){`

- void Ex1(int n){
  - int i,j;

- void Ex1(int n){
  - int i,j;
  - int sum=0;

```
• void Ex1(int n){
   •int i,j;
   •int sum=0;
   •for ( i=1; i<n; i++ ){
```

```
• void Ex1(int n){
   •int i,j;
   •int sum=0;
   •for ( i=1; i<n; i++ ){
      •for ( j=1; j<i; j++){
```

```
• void Ex1(int n){
   •int i,j;
   •int sum=0;
   •for ( i=1; i<n; i++ ){
      •for ( j=1; j<i; j++){
         •++sum;
```

```
• void Ex1(int n){
   •int i,j;
   •int sum=0;
   •for ( i=1; i<n; i++ ){
      •for ( j=1; j<i; j++){
         •++sum;
      •}
```

```
• void Ex1(int n){
   •int i,j;
   •int sum=0;
   •for ( i=1; i<n; i++ ){
      •for ( j=1; j<i; j++){
         •++sum;
      •}
   •}
```

```
• void Ex1(int n){
   •int i,j;
   •int sum=0;                    ——————————tiempo constante: 1
   •for ( i=1; i<n; i++ ){          1 + n + (n-1)
      •for ( j=1; j<i; j++){
         •++sum;                   1 + i + (i-1) + (i-1) = 3i-1
      •}
   •}
```

```
• void Ex1(int n){
   •int i,j;
   •int sum=0;                    ——————————tiempo constante: 1
   •for ( i=1; i<n; i++ ){          1 + n + (n-1)
      •for ( j=1; j<i; j++){
         •++sum;                                  1 + i + (i-1) + (i-1) = 3i-1
      •}
   •}
```

$$T(n) = 2 + n + (n-1) + \sum_{i=1}^{n-1}(3i-1)$$

—

—

```
• void Ex1(int n){
   •int i,j;
   •int sum=0;                    ——————————tiempo constante: 1
   •for ( i=1; i<n; i++ ){          1 + n + (n-1)
      •for ( j=1; j<i; j++){
         •++sum;                                  1 + i + (i-1) + (i-1) = 3i-1
      •}
   •}
```

$$T(n) = 2 + n + (n-1) + \sum_{i=1}^{n-1}(3i-1)$$

$$T(n) = 1 + 2n + 3\sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} 1$$

```
• void Ex1(int n){
   •int i,j;
   •int sum=0;                          tiempo constante: 1
   •for ( i=1; i<n; i++ ){               1 + n + (n-1)
      •for ( j=1; j<i; j++){
         •++sum;                        1 + i + (i-1) + (i-1) = 3i-1
      •}
   •}
```

$$T(n) = 2 + n + (n-1) + \sum_{i=1}^{n-1}(3i-1)$$

$$T(n) = 1 + 2n + 3\sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} 1$$

$$T(n) = 1 + 2n + 3\left(\frac{n(n-1)}{2}\right) - (n-1)$$

```
• void Ex1(int n){
   •int i,j;
   •int sum=0;                    ——————— tiempo constante: 1
   •for ( i=1; i<n; i++ ){          1 + n + (n-1)
      •for ( j=1; j<i; j++){
         •++sum;                   1 + i + (i-1) + (i-1) = 3i-1
      •}
   •}
```

$$T(n) = 2 + n + (n-1) + \sum_{i=1}^{n-1}(3i-1)$$

$$T(n) = 1 + 2n + 3\sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} 1$$

$$T(n) = 1 + 2n + 3\left(\frac{n(n-1)}{2}\right) - (n-1)$$

```
• void Ex1(int n){
   •int i,j;
   •int sum=0;                    ——————————tiempo constante: 1
   •for ( i=1; i<n; i++ ){          1 + n + (n-1)
      •for ( j=1; j<i; j++){
         •++sum;                                  1 + i + (i-1) + (i-1) = 3i-1
      •}
   •}
```

$$T(n) = 2 + n + (n-1) + \sum_{i=1}^{n-1}(3i-1)$$

$$T(n) = 1 + 2n + 3\sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} 1$$

$$T(n) = 1 + 2n + 3\left(\frac{n(n-1)}{2}\right) - (n-1)$$

- complejidad del orden de n^2

Complejidad en el tiempo

```
void Ex2(int n){
   int i,j;
   int sum=0, summ=0, limit=n*n;
   for ( i=1; i<limit; i++ )
      ++ sum;
for ( j=1; j<sum; j++)
   ++summ;
}
```

- para valores grandes de ***n***, ***T(n)*** *se comporta como* n^2

```
void Ex2(int n){
   int i,j;
   int sum=0, summ=0, limit=n*n;
   for ( i=1; i<limit; i++ )
      ++ sum;
   for ( j=1; j<sum; j++)
      ++summ;
   }
```

$1 + 1 + 1 + 1$

$1 + n^2 + 2(n^2-1)$

$1 + (n^2-1) + 2(n^2-2)$

$$T(n) = 6n^2 - 1$$

- complejidad del orden de n^2

Complejidad en el tiempo

- Muchas veces no se puede establecer una forma exacta o aún aproximada de ***T(n)***, entonces hay que expresarlo **estadísticamente**.
  - en **promedio**.
  - en el **peor caso.**
  - en el mejor caso, (un poco tramposo)

Tiempo polinomial como definición de eficiencia

Tiempo polinomial como definición de eficiencia

- En los años 1960s se llegó a un consenso sobre cómo cuantificar la noción de tiempo de ejecución *¨razonable¨.*

Tiempo polinomial como definición de eficiencia

- En los años 1960s se llegó a un consenso sobre cómo cuantificar la noción de tiempo de ejecución *¨razonable¨.*
- Los espacios de búsqueda naturales para problemas combinatorios tienden a crecer de forma exponencial al tamaño de N, si N aumenta en 1, las posibilidades aumentan multiplicativamente.

Tiempo polinomial como definición de eficiencia

- En los años 1960s se llegó a un consenso sobre cómo cuantificar la noción de tiempo de ejecución *¨razonable¨.*
- Los espacios de búsqueda naturales para problemas combinatorios tienden a crecer de forma exponencial al tamaño de N, si N aumenta en 1, las posibilidades aumentan multiplicativamente.
- Queremos algoritmos con mejores comportamientos de escalamiento, si N crece un factor de 2, el tiempo de ejecución será más lento por un factor constante C.

Tiempo polinomial como definición de eficiencia

- En los años 1960s se llegó a un consenso sobre cómo cuantificar la noción de tiempo de ejecución *¨razonable¨.*

- Los espacios de búsqueda naturales para problemas combinatorios tienden a crecer de forma exponencial al tamaño de N, si N aumenta en 1, las posibilidades aumentan multiplicativamente.

- Queremos algoritmos con mejores comportamientos de escalamiento, si N crece un factor de 2, el tiempo de ejecución será más lento por un factor constante C.

- Aritméticamente podemos expresar este comportamiento como: para dos constantes absolutas $c>0$, $d>0$ tales que, para cada instancia de entrada N, el tiempo de ejecución está acotado por cN^d pasos computacionales.

Tiempo polinomial como definición de eficiencia

- En los años 1960s se llegó a un consenso sobre cómo cuantificar la noción de tiempo de ejecución *¨razonable¨.*

- Los espacios de búsqueda naturales para problemas combinatorios tienden a crecer de forma exponencial al tamaño de N, si N aumenta en 1, las posibilidades aumentan multiplicativamente.

- Queremos algoritmos con mejores comportamientos de escalamiento, si N crece un factor de 2, el tiempo de ejecución será más lento por un factor constante C.

- Aritméticamente podemos expresar este comportamiento como: para dos constantes absolutas $c>0$, $d>0$ tales que, para cada instancia de entrada N, el tiempo de ejecución está acotado por cN^d pasos computacionales.

- En este caso decimos que el algoritmo tiene *tiempo de* ***ejecución polinomial.***

Tiempo polinomial como definición de eficiencia

$$N \rightarrow 2N$$

Tiempo polinomial como definición de eficiencia

$$N \to 2N \qquad cN^d \to c(2N)^d = c \cdot 2^d N^d$$

Tiempo polinomial como definición de eficiencia

$$N \to 2N \qquad cN^d \to c(2N)^d = c \cdot 2^d N^d$$

Definicion 1: Un algoritmo es *eficiente* si tiene un tiempo de ejecución polinomial

Tiempo polinomial como definición de eficiencia

$$N \to 2N \qquad cN^d \to c(2N)^d = c \cdot 2^d N^d$$

Definicion 1: Un algoritmo es *eficiente* si tiene un tiempo de ejecución polinomial

Muy diferente a, por ejemplo:

Tiempo polinomial como definición de eficiencia

$$N \to 2N \qquad cN^d \to c(2N)^d = c \cdot 2^d N^d$$

Definicion 1: Un algoritmo es *eficiente* si tiene un tiempo de ejecución polinomial

Muy diferente a, por ejemplo:

$$N \to 2N$$

Tiempo polinomial como definición de eficiencia

$$N \to 2N \qquad cN^d \to c(2N)^d = c \cdot 2^d N^d$$

Definicion 1: Un algoritmo es *eficiente* si tiene un tiempo de ejecución polinomial

Muy diferente a, por ejemplo:

$$N \to 2N \qquad c2^N \to c2^{(2N)} = c2^N 2^N$$

Tiempo polinomial como definición de eficiencia

- Con un procesador que ejecuta *un millón de instrucciones* de alto nivel *por segundo:*

Tiempo polinomial como definición de eficiencia

- Con un procesador que ejecuta *un millón de instrucciones* de alto nivel *por segundo:*

	n	$n\log_2 n$	n^2	n^3	1.5^n	2^n	n!
n=10	< 1s	< 1s	<1s	<1s	<1s	<1s	4s
n=30	< 1s	< 1s	<1s	<1s	<1s	18 min	10^{25} años
n=50	< 1s	<1s	<1s	<1s	11 min	36 años	--
n=100	< 1s	<1s	<1s	1 s	12,892 años	10^{17} años	--
n=1000	< 1s	< 1s	1 s	18 min	--	--	--
n=10000	< 1s	< 1s	2 min	12 días	--	--	--
n=100000	< 1s	2 s	3 horas	32 años	--	--	--
n=1000000	1 s	20 s	12 días	31,710 años	--	--	--

Orden de crecimiento asintótico

- Identificar el comportamiento de un algoritmo en el peor, en el mejor o en el caso promedio.

- Tener una idea del comportamiento de un algoritmo cuando se escala.

- Identificar clases de algoritmos con comportamientos comparables

- Expresiones como $1.62n^2 + 3.5n + 8$ dependen de parámetros de representación intermedia y del hardware utilizado.