

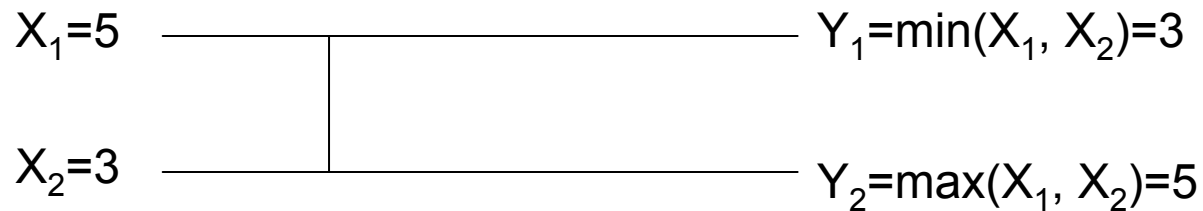
Redes de ordenación en paralelo.

Curso: Diseño y Análisis de Algoritmos.

Alumno: Mario Rincón Nigro

Comparadores

Un comparador es un circuito con 2 entradas y 2 salidas. Por conveniencia las entradas vienen a la izquierda y las salidas a la derecha.



Será cómodo suponer que las entradas se intercambian cuando $X_1 \geq X_2$. Un solo comparador puede ordenar 2 entradas. Supondremos en lo próximo que la tarea de un comparador la hace un solo procesador y que pueden funcionar cualquier cantidad de comparadores en paralelo

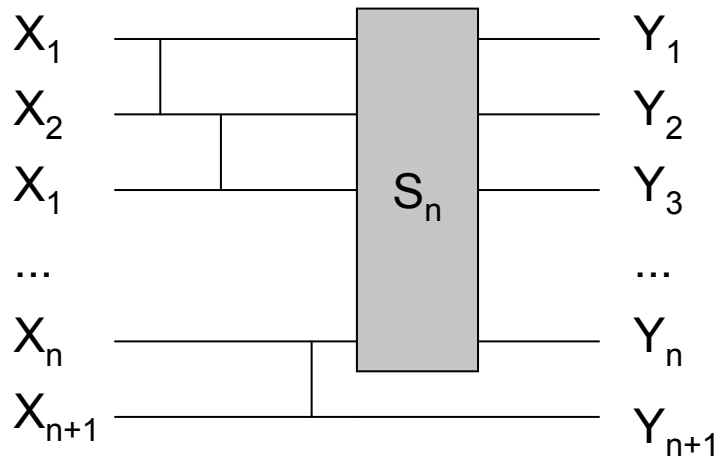


Redes de ordenación

Haciendo uso de los comparadores podemos diseñar redes que admitan n entradas, tales que si se les aplica como entrada un vector (X_1, X_2, \dots, X_n) de cómo salida un vector (Y_1, Y_2, \dots, Y_n) en el que $Y_1 \leq Y_2 \leq \dots \leq Y_n$.

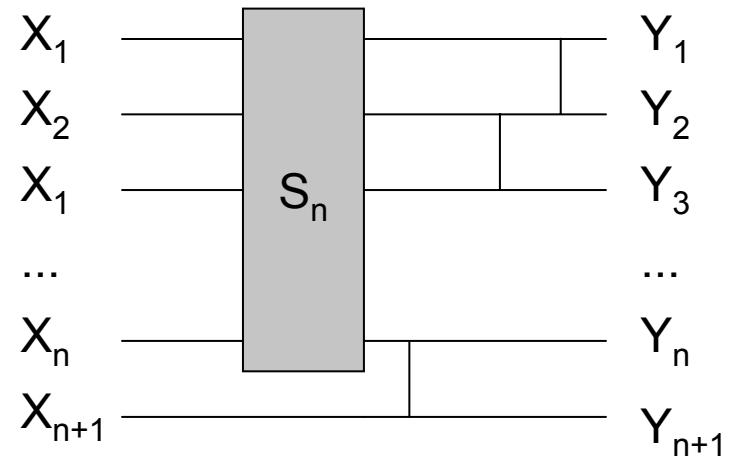
Llamaremos S_n a la red que ordena n entradas. Notese que S_1 no requiere ningún comparador y S_2 es un comparador. Una manera de construir redes mas grandes consiste en diseñar S_{n+1} en terminos de S_n

Redes de ordenación



S_{n+1}

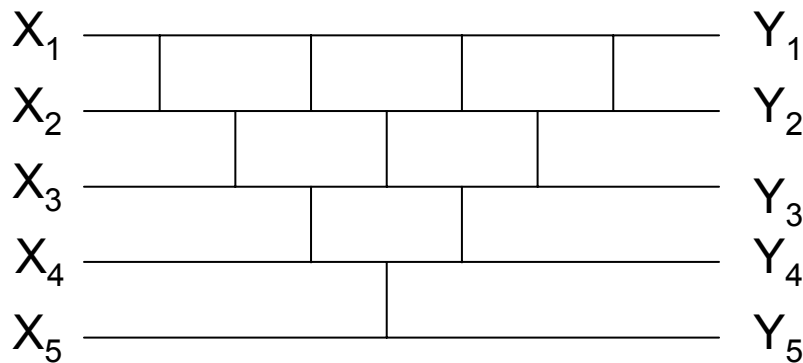
Selección. El más grande cae al fondo



S_{n+1}

Inserción. Ordena primeros n elementos e inserta en el lugar adecuado elemento $n+1$

Redes de ordenación



Red para
ordenar 5
elementos.
 S_5

Medidas de la calidad de una red:

Tamaño de la red: número de comparadores. Para S_n : $n(n-1)/2$.

Profundidad de la red: número máximo de comparadores por el que pasa una entrada para llegar a la salida. Para $S_n=2n-3$

Tamaño de $S_5=10$. Profundidad de $S_5=7$.



Redes de ordenación

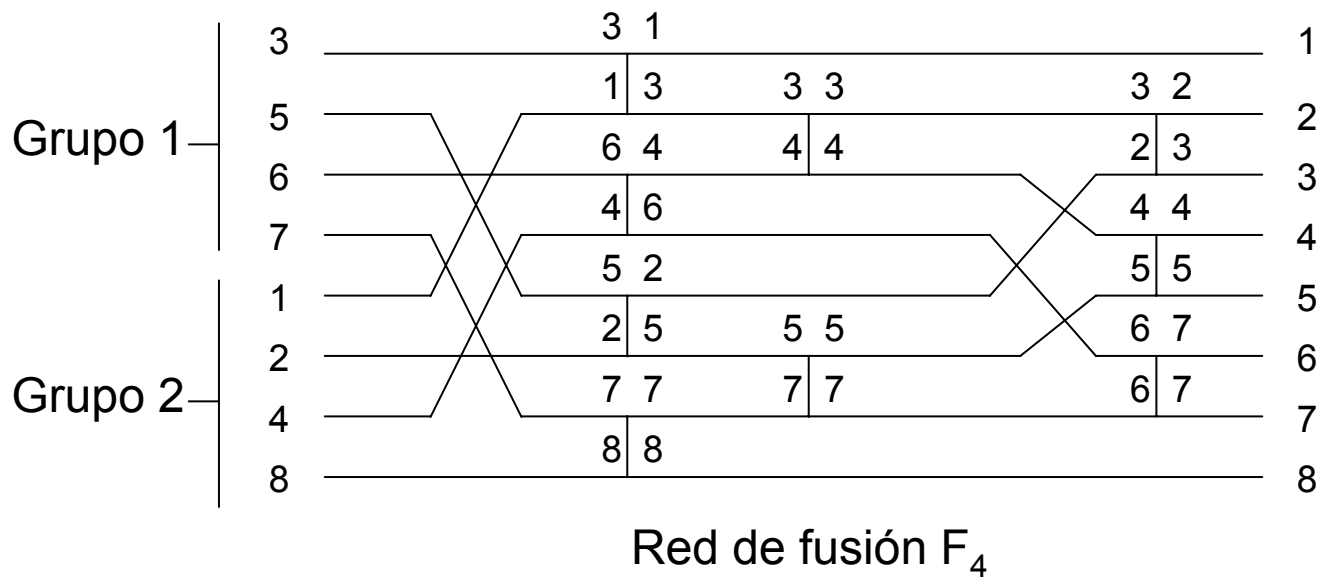
Si suponemos que cada comparador ordena sus dos entradas en tiempo constante, entonces el tiempo necesario por este tipo de red para ordenar n entradas es proporcional a la profundidad de la red, y por tanto esta en $\Theta(n)$.

Principio cero-uno

Lema. Una red de ordenación con n entradas ordena correctamente cualquier conjunto de valores dado si y solo si ordena correctamente todos los 2^n vectores de entrada que constan únicamente de ceros y unos.

Redes de fusión en paralelo

Para todo entero positivo n , una red de fusión F_n es una red formada por comparadores, con dos grupos de n entradas y un solo grupo de $2n$ salidas. Siempre y cuando ambos grupos de estados estén ya ordenados, las salidas estarán también ordenadas. Por sencillez supondremos a n potencia de 2.

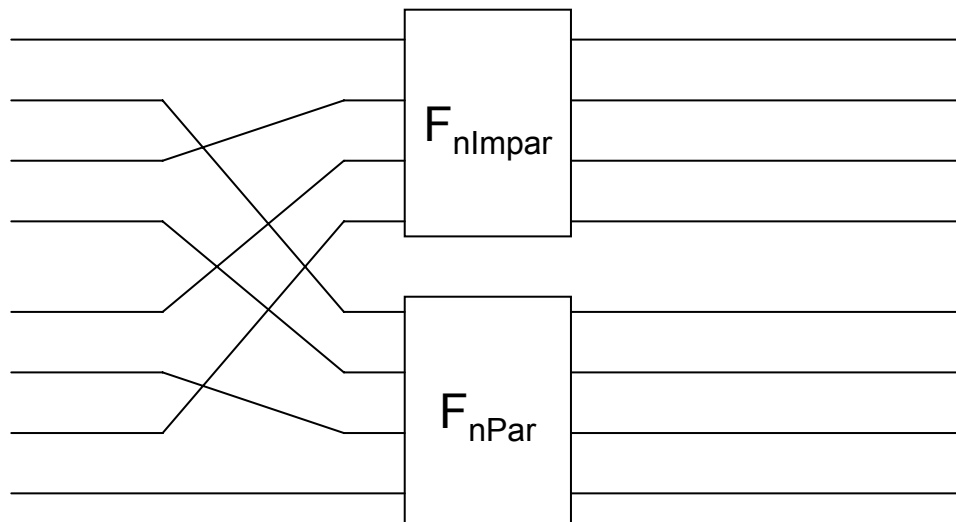


Redes de fusión en paralelo

Obsérvese que F_1 es un solo comparador. Partiendo de esto creamos redes de fusión F_2 y F_4 y así sucesivamente, diseñando siempre a F_{2n} en términos de F_n . La construcción se lleva a cabo de la siguiente manera. Supongamos que los dos grupos de entradas son $(W_1, W_2, W_3, \dots, W_n)$ y $(X_1, X_2, X_3, \dots, X_n)$ y que las salidas son $(Y_1, Y_2, Y_3, \dots, Y_{2n})$, fusionamos entonces las entradas de posición impar $(W_1, W_3, \dots, W_{n-1})$ del primer grupo con las impares del segundo grupo $(X_1, X_3, \dots, X_{n-1})$, mediante una red de fusión F_n , y luego las entradas en posición par del primer grupo (W_2, W_4, \dots, W_n) con las entradas pares del segundo grupo (X_2, X_4, \dots, X_n) mediante otra red. Llamaremos a las salidas de estas fusiones $(V_1, V_2, \dots, V_{2n})$. Permutaremos esta salida de tal manera que de arriba hasta abajo se vean $(V_1, V_{n+1}, V_2, V_{n+2}, \dots, V_n, V_{2n})$.

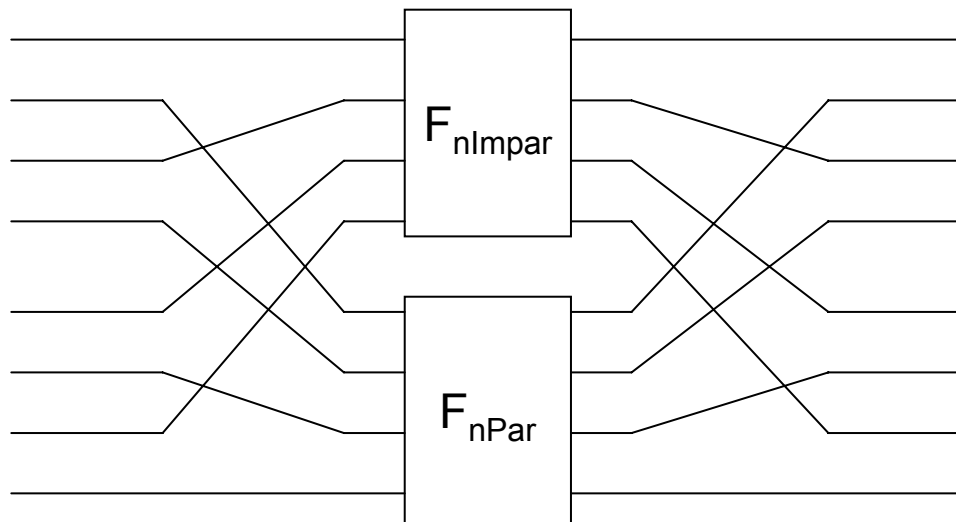
Redes de fusión en paralelo

Por último se instalan comparadores entre las que ahora son las salidas 2 y 3, 4 y 5, ... $2n-2$ y $2n-1$. Para las redes de fusión en paralelo también es válido el principio cero-uno.



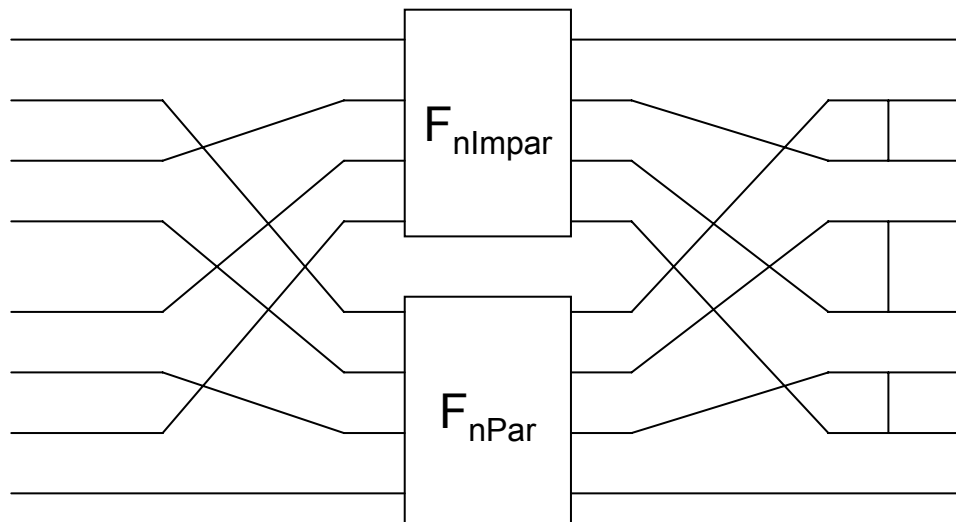
Redes de fusión en paralelo

Por último se instalan comparadores entre las que ahora son las salidas 2 y 3, 4 y 5, ... $2n-2$ y $2n-1$. Para las redes de fusión en paralelo también es válido el principio cero-uno.



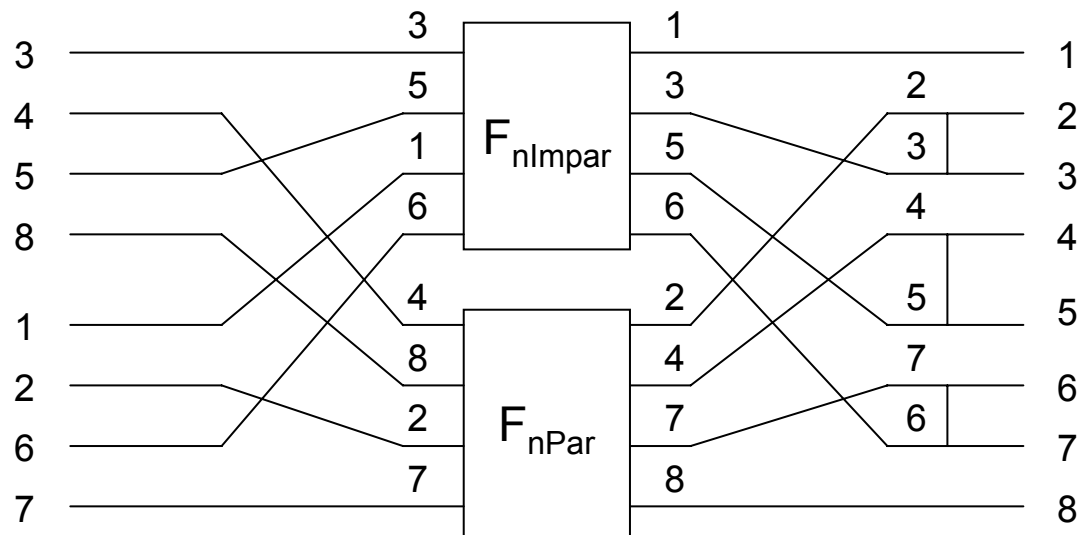
Redes de fusión en paralelo

Por último se instalan comparadores entre las que ahora son las salidas 2 y 3, 4 y 5, ... $2n-2$ y $2n-1$. Para las redes de fusión en paralelo también es válido el principio cero-uno.



Redes de fusión en paralelo

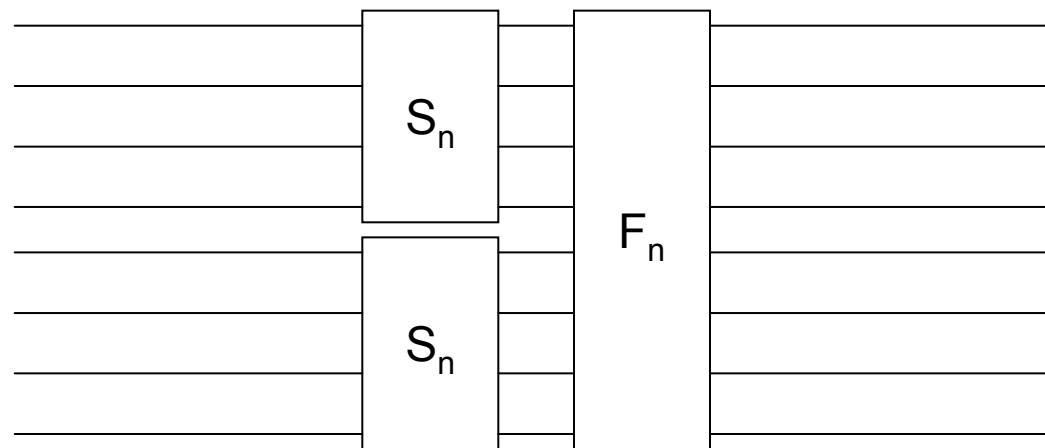
Por último se instalan comparadores entre las que ahora son las salidas 2 y 3, 4 y 5, ... $2n-1$ y $2n-1$. Para las redes de fusión en paralelo también es válido el principio cero-uno.

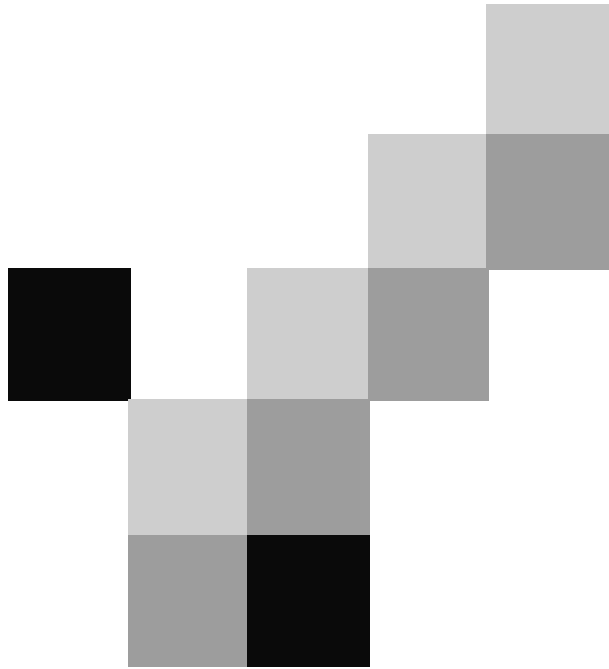


Redes de ordenación mejoradas

Podemos construir otro tipo de redes de ordenación mediante la técnica divide y vencerás de la siguiente manera. Se usan dos redes S_n para ordenar las n primeras entradas y las n últimas, luego se utiliza una red de fusión F_n para completar la ordenación. El enfoque divide y vencerás se detendrá cuando llegue a S_2 , que es un solo comparador.

El tamaño de estas redes está en $\Theta(n \log^2 n)$ y su profundidad en $\Theta(\log^2 n)$. Son conocidas desde 1964 y se deben a Batcher.





Argumentos del adversario.

Curso: Diseño y Análisis de Algoritmos.

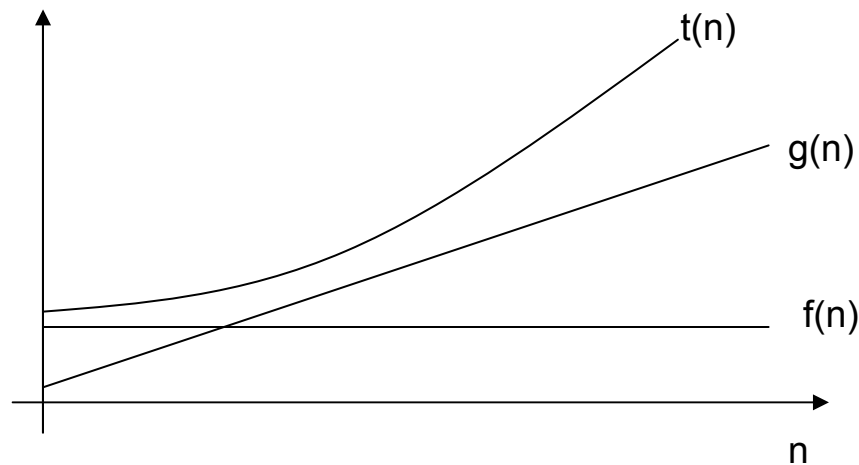
Alumno: Mario Rincón Nigro

Argumentos del adversario

Técnica de análisis de complejidad.

Permite, dado un problema, demostrar la validez de una cota inferior para el tiempo requerido en el peor caso por parte de cualquier algoritmo que resuelva tal problema correctamente.

Como cota inferior entenderemos la mayor función $g(n)$, tal que $t(n) = \Omega(g(n))$. Recordemos que $t(n) = \Omega(g(n))$ si existen reales c y n_0 tales que $t(n) \geq cg(n)$ para todo $n \geq n_0$





Argumentos del adversario

La idea detrás de esta técnica es poner en marcha el algoritmo con una entrada inicialmente no especificada, salvo por su tamaño. Un ente hipotético “malevolente”, al que conoceremos como el adversario, será quien especifique a cada paso la entrada. El objetivo de el adversario será mantener incierto al algoritmo sobre la solución del problema, el mayor tiempo posible.

El adversario es consecuente al escoger la entrada en el sentido en que si el algoritmo es correcto y la examina un numero suficiente de veces, aceptara entonces que ha conseguido la solución una vez que termine.



Argumentos del adversario

Si el algoritmo no examina suficientemente la entrada y afirma en algún momento conocer la solución, el adversario mostrará una posible entrada para la cual el algoritmo este errado.

Ejemplo: Juego de las 4 preguntas: Su amigo escoge un número entre 1 y 16, y le pide que lo adivine usando no mas de 4 preguntas del tipo “¿Es el número menor que x ?”.

La estrategia mas eficiente que usted puede usar es dividir el intervalo $[1, 16]$ por la mitad a cada pregunta. La manera honesta de jugar por parte de su amigo es escoger el número antes de que usted haga la primera pregunta. Veamos qué pasaría si pretendiéramos ganar el juego con solo 3 preguntas cuando el adversario escoge la entrada.



Juego de las 4 preguntas

{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 }

Algoritmo: Pregunta 1: ¿Es el número menor que 9?.

Adversario: Si.

{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 }

Algoritmo: Pregunta 2: ¿Es el número menor que 5?.

Adversario: No.

{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 }

Algoritmo: Pregunta 3: ¿Es el número menor que 7?.

Adversario: Si.

{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 }

Algoritmo: El número es 5.

Adversario: Error, el número que había escogido es el 6.



Juego de las 4 preguntas

El adversario nos ha probado entonces que se necesita al menos una pregunta más.

{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 }

Algoritmo: Pregunta 4: ¿Es el número menor que 6?.

Adversario: Si.

{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 }

Algoritmo: El número es 5.

Adversario: Correcto.

Búsqueda del máximo en un vector

Consideremos el problema de encontrar el máximo en un vector. El siguiente algoritmo resuelve el problema en $O(n)$.

maximo(T: Arreglo[n] de Entero): Entero	
1	$m = T[1]$
2	[si $T[i] > m$ entonces $m = T[i]$] $i = 2, n$
3	devolver m

El algoritmo lleva a cabo $n-1$ comparaciones.

¿Es posible diseñar un algoritmo basado en comparaciones mas eficiente?.



Búsqueda del máximo en un vector

Al comparar 2 elementos $T[i]$ y $T[j]$, diremos que el menor entre los dos ha perdido en una comparación.

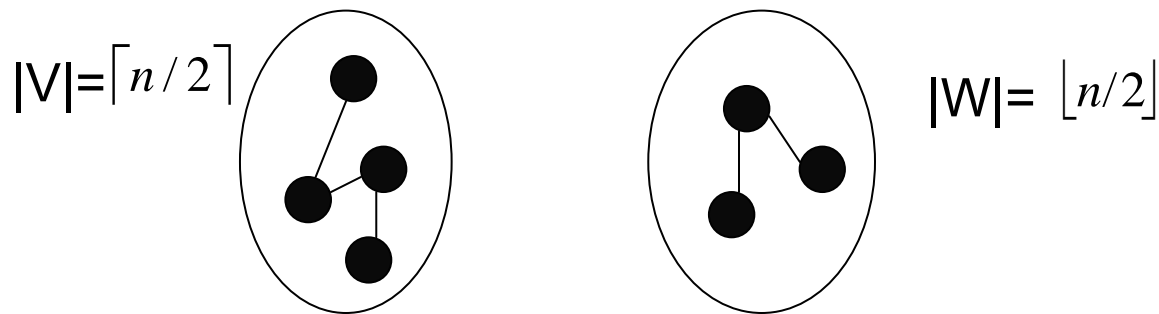
La estrategia que usa cualquier algoritmo basado en comparaciones para resolver esta problema es comparar cada elemento con el ganador de una comparación anterior. Si nuestro algoritmo hiciera solo $n-2$ comparaciones para obtener el máximo elemento del arreglo, eso significaría que debe haber al menos un elemento cuyo valor es desconocido y el adversario siempre podría mostrar un arreglo de entrada tal que ese elemento ganaría en una comparación con el máximo hallado por el algoritmo.

No se puede mejorar el tiempo $O(n)$ de cualquier algoritmo que resuelva el problema mediante comparaciones.

Comprobación de la conectividad de un grafo

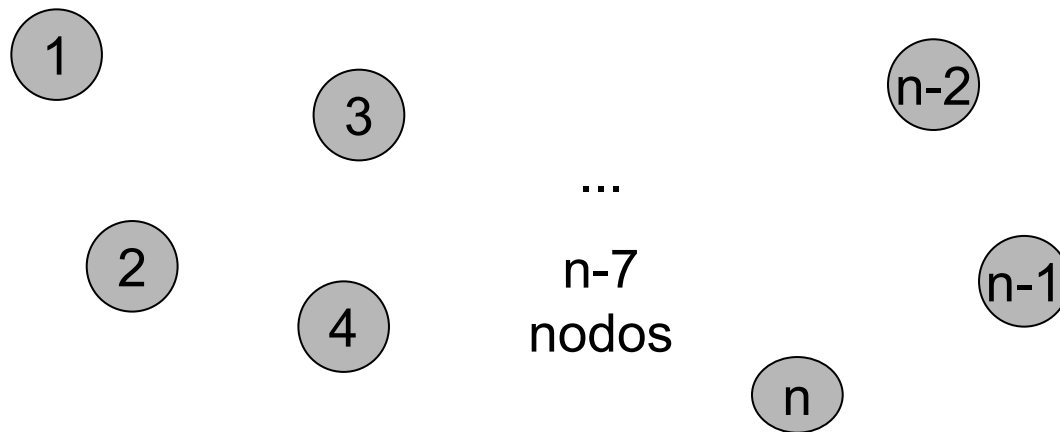
Sea un algoritmo que resuelve este problema para un grafo no dirigido con n vértices. Las únicas preguntas que se admiten son del tipo: “¿Existe un arco entre los vértices j y k ?”. Este problema lo resuelve la búsqueda en profundidad en $O(n^2)$ en el peor de los casos para un grafo implementado con matriz de adyacencia.

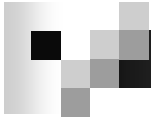
Un primer argumento que el adversario podría construir sería el siguiente: descomponer el conjunto de vertices en 2 subconjuntos V y W , y responder sí cada vez que se pregunte si son adyacentes 2 nodos del mismo subconjunto y no en caso contrario.



Comprobación de la conectividad de un grafo

El argumento anterior no toma en cuenta si los subgrafos de los subconjuntos son conexos o no. Solo da una cota inferior necesaria mas no suficiente para un algoritmo que resuelva el problema correctamente. Un argumento un poco mas sofisticado prueba que ningún algoritmo que consulte al menos $(n-1)n/2$ aristas distintas puede resolver el problema correctamente.





Preguntas