DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Nro. 1

13/04/2009 Algoritmos y Estructuras de Datos III

Integrante	LU	Correo electrónico
Dinota, Matías	076/07	matiasgd@gmail.com
Huel, Federico Ariel	329/07	federico.huel@gmail.com
Leveroni, Luciano	360/07	lucianolev@gmail.com
Mosteiro, Agustín	125/07	agustinmosteiro@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Ejercicio 1: La conjetura de Goldbach

Introducción

La idea de este ejercicio es intentar demostrar de manera empírica la conjetura de Goldbach. Esta misma predica que todo número par positivo mayor que dos se puede escribir como la suma de dos números primos. Con este objetivo se procedió a la implementación de un algoritmo iterativo que, dado un número par mayor que dos, retorna dos números primos que satisfacen dicha conjetura.

En principio, se probaron distintos casos aleatorios para encontrar un patrón común. A partir de esto, se realizó la siguiente conjetura: "Sea p el mayor primo comprendido en el rango $3..n-2$ entonces p y $n-p$ son dos números primos cuya suma es n". De ser cierta esta conjetura, bastaría con hallar el primo más cercano a $n-2$ para obtener, mediante operaciones baásicas, el resultado esperado. Sin embargo, al poco tiempo se encontró un contraejemplo ($128 = 119 + 9$, pero 9 no es primo) que falseaba la conjetura. A pesar de esto, la idea sirvió para encontrar una solución al problema. En la sección siguiente veremos el algoritmo en cuestión.

Algoritmo

A continuación se encuentra el pseudocódigo relativo al algoritmo propuesto que resuelve el problema, seguido de una breve explicación acerca del funcionamiento y correctitud del mismo.

```
esPrimo(n)
  si n = 2
    devolver true
  si n es divisible por 2
    devolver false
  para todo i par desde 3 hasta raiz de n
    si n es divisible por i
      devolver false
  devolver true

encontrarPrimos(n)
  si n = 4
    devolver (2,2)
  para todo i par desde 3 hasta n/2
    si esPrimo(i) y esPrimo(n-i)
      devolver (i,n-i)
  devolver (0,0)
```

En primer lugar, se encuentrá la definición de la función *esPrimo* que, tal como su nombre indica, retorna *true* en caso de que el número pasado como parámentro sea un número primo y *false* en caso contrario. El algoritmo en cuestión consiste en un método simple pero efectivo: En primer lugar, hay 2 casos base, el primero retorna *true* en caso de tratarse del número 2 y el segundo evalua si el numero es par (divisible por 2), retornando *false* en este caso, ya que todo numero par mayor que 2 no es primo. Luego, el ciclo principal evalua si el parámetro n es divisible por algún número impar i con $3 \leq i \leq \sqrt{n}$, retornando *false* en tal caso. La correctitud de tal procedimiento se desprende directamente del teorema que predica que si n es un número primo entonces existe al menos un divisor d tal que $2 \leq d < \sqrt{n}$.

Con respecto al algoritmo relativo al problema en sí, la idea consiste en recorrer todos los números impares comprendidos entre 3 y $n-3$ de a pares cuya suma equivalgan a n. Los pares $i, n-i$ satisfacen trivialmente esta condición, ya que $i + (n-i) = n$. Del mismo modo, es fácil ver que para generar todos los números del rango mencionado, basta con variar i tal que $3 \leq i \leq n/2$. De esta forma se ve claramente que esPrimo(i) comprende el rango $3..n/2$, y $esPrimo(n-i)$ el

rango $n/2..n-3$. De este modo, cuando se satisface que $esPrimo(i)\&esPrimo(n-i) == true$ se puede afirmar que se ha encontrado el par de números que cumplen con la conjetura: dos números primos cuya suma es n. En caso de que no hallar dicha tupla detro del rango propuesto, se puede concluir que la conjetura de Goldbach era falsa, retornando la tupla (0,0) en este caso.

Complejidad

A continuación se analizara la complejidad de peor caso sobre los modelos uniforme y logaritmico.

Nota: En el siguiente análisis se hara referencia a funciones propias de la implementación de los algoritmos. Puede ver el código fuente del mismo en la sección *Anexos*.

Modelo Uniforme:

En este modelo el análisis no esta centrado en el tamaño de los operandos por lo que el tiempo de ejecución de cada operación se considera constante.

En la función esPrimo todas las operaciones y asignaciones utilizadas se logran en tiempo constante por lo mencinado anteriormente, de esta manera la complejidad de esta función depende del ciclo que contiene. Como itera sobre i entre 3 (valor de inicialización) y $\sqrt{n}+1$, y ademas en cada vuelta del ciclo la variable es aumentada en 2, claramente se aprecia que en el peor caso (es decir que nunca se cumpla la condición n mod i = 0) el ciclo se recorre aproximadamente $\sqrt{n}/2$ veces. De esta manera concluimos que $T(n) \in O(\sqrt{n})$.

En cuanto a la complejidad de las operaciones y asignaciones realizadas en la función encontrarPrimos el caso es el mismo al de esPrimo debido al modelo de computo. La unica operación dentro del pseudocodigo de encontrarPrimos de complejidad superior a constante es la analizada anteriormente y dicha operacion se encuentra dentro del único ciclo de la función por lo que concentraremos el análisis de complejidad aqui. En este caso iteramos sobre el mismo valor de inicio pero el limite del ciclo esta dado por $n/2$. Tambien aqui la variable i es aumentada en 2 en cada iteración lo que determina que en el peor caso se iterará aproximada mente $(n/2)/2 = n/4$ veces. Como dentro del ciclo se llama a esPrimo (complejidad $\sqrt{n}$) la complejidad final de la función encontrarPrimos es $T(n) \in O(n * \sqrt{n})$.

Modelo Logarítmico:

En este modelo de computo si nos concentraremos en el tamaño de la entrada y el costo de las operaciones en función de la cantidad de bits de sus operandos.

En la función esPrimo se puede apreciar que las operaciónes que mayor complejidad conllevan son $\sqrt{n}$ y n mod i. Como ambas son llamadas dentro del único ciclo de la función, la complejidad estará determinada por dicho ciclo.

Entonces tenemos

$$\begin{aligned} T(n) &= \sum [\log(\sqrt{n}+1) + \log(n)^2 + \log(n)^2 + \log(n \, mod \, i) + \log(i)] \\ &\leq \sum [\log(n) + 2\log(n)^2 + \log(n) + \log(n)] \\ &= \sqrt{n}(3\log(n) + 2\log(n)^2) \Longrightarrow T(n) \in O(\sqrt{n} * \log(n)^2) \end{aligned}$$

Con respecto a la función encontrarPrimos sucede lo mismo con respecto a que las operaciones de mayor complejidad son realizadas dentro del ciclo por lo que en este caso la complejidad total de la función también estará determinada por dicho ciclo.

La función de complejidad es

$$\begin{aligned} T(n) &= \sum [\log(i) + \sqrt{i}\log(i)^2 + \sqrt{n-i}\log(n-i)^2 + \log(n) + \log(n) + \log(i)] \\ &\leq \sum [\log(n) + \sqrt{n}\log(n)^2 + \sqrt{n}\log(n)^2 + \log(n) + \log(n) + \log(n)] \end{aligned}$$

$$= n(4\log(n) + 2\sqrt{n}\log(n)^2) \Longrightarrow T(n) \in O(n^{3/2} * \log(n)^2)$$

Por último analizaremos también la complejidad en función del tamaño de la entrada para los dos modelos de computo vistos. Como la función encontrarPrimos toma unicamente un natural n, si llamamos t al tamaño de entrada tenemos que $t = \log(n) \Longrightarrow 2^t = n$. De esta manera, reemplazando n en las complejidades calculadas previamente obtendremos el costo de los dos algoritmos.

- Modelo Uniforme:

$$T(n) \in O(n\sqrt{n}) = O(n^{3/2}) = O((2^t)^{3/2}) = O(2^{3t/2})$$

- Modelo Logarítmico:

$$T(n) \in O(n^{3/2}\log(n)^2) = O((2^t)^{3/2}(\log(2^t))^2) = O((2^{3t/2}t^2)$$

Análisis de resultados

Con el fin de analizar la eficiencia del algoritmo propuesto se realizaron diversas pruebas. En primer lugar, se hicieron experimentos orientados a estudiar el tiempo de ejecución del mismo en relación al parámetro ingresado (de ahora ne adelante, Prueba 1). Para tal propósito se procedió a la creación de un programa de prueba (ver pseudocódigo debajo) que calcula el tiempo de ejecución de la función *encontrarPrimos* para valores entre 4 y $2^{31} - 4$. Al tratarse de una cantidad de números tan grande, como veremos más adelante, la ejecución real de dicha prueba demoraría en exceso (años). Por tal motivo se decidió tomar números pares separados por una distancia aleatoria entre $10^4..2*10^4$. La razón del uso de un parámetro aleatorio radica en poder conseguir una buena muestra representativa del rango propuesto anteriormente y evitando así casos patológicos.

```
Prueba 1
--------
mientras i <= 2^31-4
  T = tiempo que tarda la operacion encontrarPrimos(i) en microsegundos
  i = i + 10000 + numero aleatorio par entre 0 y 10000
  guardar en Ej1-Complejidad.txt la linea "i T"
```

La segunda prueba tiene como objetivo mostrar que la cantidad de operaciones que realiza el algoritmo es significativa menor que la cota teorica propuestas, aún en los peores casos. Con este fin, se creó una aplicación que obtiene los máximos de la primer componente de la tupla solución (número primo mas chico) para 20 ejecuciones del algoritmo. Al igual que en la primer prueba, se optó por un enfoque aleatorio, tomando muestras de nḿeros a intervalos aleatorios entre 10^5 y $2 * 10^5$.

```
Prueba 2
--------
para j desde 0 hasta 20
  peor_caso_primo1 = 0
  mientras i <= 2^31-4
    <primo1, primo2> = encontrarPrimos(i)

    si primo1 > peor_caso_primo1
      peor_caso = i
      peor_caso_primo1 = primo1
      peor_caso_primo2 = primo2
```

```
    i = i + 10000 + numero aleatorio par entre 0 y 10000

  guardar en Ej1-MayoresPrimos.txt la linea "peor_caso peor_caso_primo1 peor_caso_primo2"
```

Por último, con el fin de observar más en detalle el comportamiento del algoritmo se decidió extraer los peores casos de la Prueba 1 y estudiarlos en profundidad (Prueba 3). Para realizar esta tarea, se implementó una sencilla aplicación (ver pseudocódigo debajo) que tomalrededor de 200 secuencias consecutivas de 1000 números aleatorios comprendidas entre 4 y $2^{31} - 4$ utilizando el mismo criterio que en el caso anterior. Luego, obtiene el máximo tiempo de ejecución entre todos los números de cada secuencia.

```
Prueba 3
--------
j = 1
mientras i <= 2^31-4
  maximo_tiempo = 0
  T = tiempo que tarda la operacion encontrarPrimos(i) en microsegundos

  si T > maximo_tiempo
    maximo_tiempo = T
    peor_caso = i

  si j = 1000
    guardar en Ej1-Peorescasos.txt la linea "peor_caso maximo_tiempo"

  j = j + 1
  i = i + 10000 + numero aleatorio par entre 0 y 10000
```

A continuación se presentan los resultados relativos a la primera aplicación (Prueba 1).

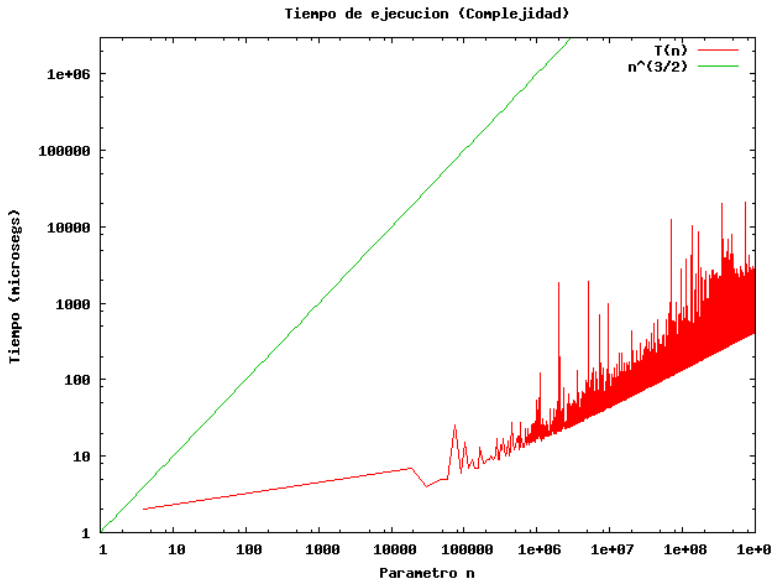

Figura 1.1

Este gráfico muestra la comparación entre las mediciones de tiempo de la primera prueba realizada y la complejidad teórica expuesta anteriormente. Llamaremos f a la función que relaciona el tiempo de ejecución con el tamaño de entrada. Como se puede apreciar, el crecimiento de f es

mucho menor que la complejidad teórica, por lo cual el algoritmo resulta mucho más eficiente que lo esperado. Sin embargo, a pesar de su comportamiento altamente irregular, se puede apreciar que el tiempo de ejecución aumenta a medida que crece el tamaño de la entrada. La razón de la discrepancia entre ambas funciones reside un hecho que denominaremos "el lema de la aproximación asintotica natural en el anillo conmutativo modelado implicitamente por los numeros primos y sus respectivas operaciones". La razón de la discrepancia entre ambas funciones reside en el hecho de que el algoritmo implementado encuentra dos números primos que cumplen con la conjetura en una muy baja cantidad de operaciones en relación al tamanño de la entrada. Si bien se desconoce el motivo de este fenómeno, hemos probado empíricamente que esto se cumple como se puede notar en el siguiente gráfico (Prueba 2).

GRAFICO

Como se observa en la Figura 1.2, para los peores casos la diferencia entre cada uno de los primos solución es muy alta, muy cercana al parámetro de entrada n. Esto explica que la complejidad teórica no se corresponda con el tiempo de ejecución.

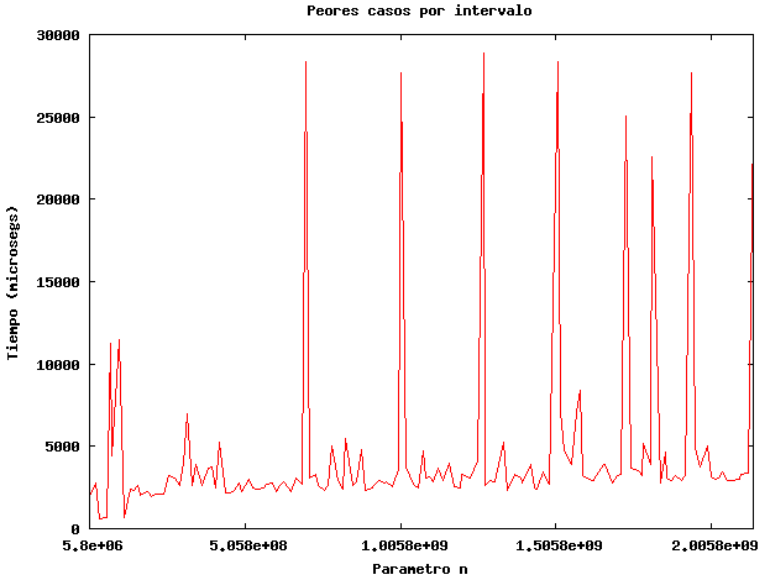

Figura 1.3

La idea de este gráfico es mostrar cúan irregular es el tiempo de ejecución del algoritmo. A pesar de haber hecho el esfuerzo de buscar los peores casos (mediante la aplicación de la Prueba 3), el tiempo que demoran los mismos difieren entre ellos considerablemente. Esto muestra que los peores casos no tienen una ralación directa con el tamaño de la entrada sino mas bien con la distribución desconocida de los números primos pertenecientes al intervalo estudiado.

Conclusiones

TODO

Ejercicio 2: Resolucion de Puzzle

Introducción

TODO

Algoritmo

TODO

Complejidad

TODO

Análisis de resultados

TODO

Conclusiones

TODO

Ejercicio 3: Cálculo de la mediana entre dos vectores

Introducción

EL problema que plantea este ejercicio es calcular la mediana entre dos vectores de números enteros, es decir, el elemento que, una vez concatenados y ordenados los vectores, deja a ambos lados la misma cantidad de elementos. Según lo especificado en el problema, ambos vectores tienen la misma longitud y están ordenados.

En principio, la solución trivial del problema fue descartada ya que el costo de realizarla era excesivo para los fines del ejercicio. Esta solución consistía en concatenar ambos vectores, ordenarlos y posteriormente obtener la mediana a partir del elemento medio de dicha secuencia.

Descartada esta opción,

TODO

Algoritmo

A continuación se mostrará el pseudocódigo del algoritmo implementado seguido de una demostración de su correctitud.

```
pseudocodigo
```

Como se puede apreciar el algoritmo sigue los pasos de un algoritmo clásico de *divide and conquer* dividiendo el problema en arreglos de menor tamaño al ir desccartando elementos mayores y menores a la mediana. De esta manera, si logramos mostrar que en cada llamado recursivo la mediana original es la misma que la de los dos nuevos arreglos, entonces podremos asegurar que el algoritmo es correcto. Para mostrar esto primero veremos que en cada paso la mediana sigue estando dentro de alguno de los arreglos, y luego que los números menores o iguales a la mediana descartados son la misma cantidad que los números mayores o iguales a la misma que se descartan.

Demostración de que la mediana se encuentra en el próximo llamado recursivo

Dividiremos la demostración en dos casos, cuando la longitud de los arreglos iniciales es impar y cuando es par.

Sean X e Y los arreglos iniciales, entonces si llamo A a $concatenar(X, Y)$ tengo que $ordenar(A) = [a_1, a_2, .., a_n, .., a_{2n}]$, como la longitud de A es par, su mediana es a_n. Se puede ver entonces que la mediana tiene $n-1$ elementos menores o iguales a ella ($A[1..n-1]$) y n elementos mayores o iguales ($A[n+1..2n]$).

1. Longitud impar:

   Los arreglos que tomará la función para el llamado recursivo estan dados por la comparación entre las medianas de los mismos siendo $x_{(n+1)/2}$ la mediana de X e $y_{(n+1)/2}$ la mediana de Y.

   (a) $x_{(n+1)/2} \geq y_{(n+1)/2}$:
   En este caso el algoritmo toma como nuevos arreglos a $X[1..(n+1)/2]$ e $Y[(n+1)/2..n]$. Supongo ahora, que m (a partir de ahora nos referiremos asi a la mediana) no está en los nuevos arreglos, es decir, pertenece a $X((n+1)/2..n]$ o a $Y[1..(n+1)/2)$.

   i. Si $m \in X((n+1)/2..n] \implies m \geq x_{(n+1)/2}$. Si $m = x_{(n+1)/2}$ entonces SI pertenece a los nuevos arreglos pues $x_{(n+1)/2}$ pertenece. Si no son iguales entonces $m > x_{(n+1)/2} \implies m > x \ \forall x \in X[1..(n+1)/2]$. Ademas si $m > x_{(n+1)/2} \geq y_{(n+1)/2} \implies m > y \ \forall y \in Y[1..(n+1)/2]$. Como $X[1..(n+1)/2]$ y $Y[1..(n+1)/2]$ tienen $(n+1)/2$ elementos cada uno entonces m es mayor a $2((n+1)/2) = n+1$ elementos lo que es absurdo porque la mediana solo puede tener tener $n-1$ elementos menores por lo visto anteriormente.

   ii. Si $m \in Y[1..(n+1)/2) \implies m \leq y_{(n+1)/2}$. Si $m = y_{(n+1)/2}$ entonces SI pertenece a los nuevos arreglos pues $y_{(n+1)/2}$ pertenece. Si no son iguales entonces $m < y_{(n+1)/2} \implies m < y \ \forall y \in Y[(n+1)/2..n]$. Ademas si $m < y_{(n+1)/2} \leq x_{(n+1)/2} \implies m < x \ \forall x \in X[(n+1)/2..n]$. Como $X[(n+1)/2..n]$ y $Y[(n+1)/2..n]$ tienen $(n+1)/2$ elementos cada uno entonces m es menor a $2((n+1)/2) = n+1$ elementos lo que es absurdo porque la mediana solo puede tener tener n elementos mayores por lo visto anteriormente.

   De esta manera queda demostrado que si n es impar y $x_{(n+1)/2} \geq y_{(n+1)/2}$ entonces la mediana pertenece a alguno de los nuevos arreglos pasados como parámetro.

   (b) $x_{(n+1)/2} < y_{(n+1)/2}$:
   En este caso el algoritmo toma como nuevos arreglos a $X[(n+1)/2..n]$ e $Y[1..(n+1)/2]$. Supongo ahora, que m no está en los nuevos arreglos, es decir, pertenece a $X[1..(n+1)/2)$ o a $Y((n+1)/2..n]$.

   i. Si $m \in X[1..(n+1)/2) \implies m \leq x_{(n+1)/2}$. Si $m = x_{(n+1)/2}$ entonces SI pertenece a los nuevos arreglos pues $x_{(n+1)/2}$ pertenece. Si no son iguales entonces $m < x_{(n+1)/2} \implies m < x \ \forall x \in X[(n+1)/2..n]$. Ademas si $m < x_{(n+1)/2} < y_{(n+1)/2} \implies m < y \ \forall y \in Y[(n+1)/2..n]$. Como $X[(n+1)/2..n]$ y $Y[(n+1)/2..n]$ tienen $(n+1)/2$ elementos cada uno entonces m es menor a $2((n+1)/2) = n+1$ elementos lo que es absurdo porque la mediana solo puede tener tener $n-1$ elementos menores por lo visto anteriormente.

   ii. Si $m \in Y((n+1)/2..n] \implies m \geq y_{(n+1)/2}$. Si $m = y_{(n+1)/2}$ entonces SI pertenece a los nuevos arreglos pues $y_{(n+1)/2}$ pertenece. Si no son iguales entonces $m > y_{(n+1)/2} \implies m > y \ \forall y \in Y[1..(n+1)/2]$. Ademas si $m > y_{(n+1)/2} > x_{(n+1)/2} \implies m > x \ \forall x \in X[1..(n+1)/2]$. Como $X[(n+1)/2..n]$ y $Y[(n+1)/2..n]$ tienen $(n+1)/2$ elementos cada uno entonces m es mayor a $2((n+1)/2) = n+1$ elementos lo que es absurdo porque la mediana solo puede tener tener n elementos mayores por lo visto anteriormente.

De esta manera queda demostrado que si n es impar y $x_{(n+1)/2} \geq y_{(n+1)/2}$ entonces la mediana pertenece a alguno de los nuevos arreglos pasados como parámetro.

2. Longitud par

   Los arreglos que tomará la función para el llamado recursivo estan dados por la comparación entre $x_{n/2}$ e $y_{n/2+1}$.

   (a) $x_{n/2} \geq y_{n/2+1}$: En este caso el algoritmo toma como nuevos arreglos a $X[1..n/2]$ e $Y[n/2+1..n]$. Supongo ahora, que m no está en los nuevos arreglos, es decir, pertenece a $X(n/2..n]$ o a $Y[1..n/2+1)$.

      i. Si $m \in X(n/2..n] \implies m \geq x_{n/2}$. Si $m = x_{n/2}$ entonces SI pertenece a los nuevos arreglos pues $x_{n/2}$ pertenece. Si no son iguales entonces $m > x_{n/2} \implies m > x \ \forall x \in X[1..n/2]$. Ademas si $m > x_{n/2} \geq y_{n/2+1} \implies m > y \ \forall y \in Y[1..n/2+1]$. Como $X[1..n/2]$ y $Y[1..n/2+1]$ tienen $n/2$ y $n/2+1$ elementos respectivamente entonces m es mayor a $n/2+n/2+1 = n+1$ elementos lo que es absurdo porque la mediana solo puede tener tener $n-1$ elementos menores por lo visto anteriormente.

      ii. Si $m \in Y[1..n/2+1) \implies m \leq y_{n/2+1}$. Si $m = y_{n/2+1}$ entonces SI pertenece a los nuevos arreglos pues $y_{n/2+1}$ pertenece. Si no son iguales entonces $m < y_{n/2+1} \implies m < y \ \forall y \in Y[n/2+1..n]$. Ademas si $m < y_{n/2+1} \leq x_{n/2} \implies m < x \ \forall x \in X[n/2..n]$. Como $X[n/2..n]$ y $Y[n/2+1..n]$ tienen $n/2+1$ y $n/2$ elementos respectivamente entonces m es menor a $n/2+1+n/2 = n+1$ elementos lo que es absurdo porque la mediana solo puede tener tener n elementos mayores por lo visto anteriormente.

   De esta manera queda demostrado que si n es par y $x_{n/2} \geq y_{n/2+1}$ entonces la mediana pertenece a alguno de los nuevos arreglos pasados como parámetro.

   (b) $x_{n/2} < y_{n/2+1}$: En este caso el algoritmo toma como nuevos arreglos a $X[n/2..n]$ e $Y[1..n/2+1]$. Supongo ahora, que m no está en los nuevos arreglos, es decir, pertenece a $X[1..n/2)$ o a $Y(n/2+1..n]$.

      i. Si $m \in X[1..n/2) \implies m \leq x_{n/2}$. Si $m = x_{n/2}$ entonces SI pertenece a los nuevos arreglos pues $x_{n/2}$ pertenece. Si no son iguales entonces $m < x_{n/2} \implies m < x \ \forall x \in X[n/2..n]$. Ademas si $m < x_{n/2} \leq y_{n/2+1} \implies m < y \ \forall y \in Y[n/2+1..n]$. Como $X[n/2..n]$ y $Y[n/2+1..n]$ tienen $n/2+1$ y $n/2$ elementos respectivamente entonces m es menor a $n/2+1+n/2 = n+1$ elementos lo que es absurdo porque la mediana solo puede tener tener n elementos mayores por lo visto anteriormente.

      ii. Si $m \in Y(n/2+1..n] \implies m \geq y_{n/2+1}$. Si $m = y_{n/2+1}$ entonces SI pertenece a los nuevos arreglos pues $y_{n/2+1}$ pertenece. Si no son iguales entonces $m > y_{n/2+1} \implies m > y \ \forall y \in Y[1..n/2+1]$. Ademas si $m > y_{n/2+1} > x_{n/2} \implies m > x \ \forall x \in X[1..n/2]$. Como $X[1..n/2]$ y $Y[1..n/2+1]$ tienen $n/2$ y $n/2+1$ elementos respectivamente entonces m es mayor a $n/2+n/2+1 = n+1$ elementos lo que es absurdo porque la mediana solo puede tener tener $n-1$ elementos menores por lo visto anteriormente.

   De esta manera queda demostrado que si n es par y $x_{n/2} \geq y_{n/2+1}$ entonces la mediana pertenece a alguno de los nuevos arreglos pasados como parámetro.

En conclusión queda demostrado que siempre que se realiza un llamado recursivo, la mediana pertenece a uno de los dos arreglos pasados como parámetro.

Demostración de que se descartan la misma cantidad de elementos mayores o iguales y menores o iguales que la mediana

Sean X e Y los arreglos pasados como parámetro y n su longituud.

1. Si la longitud de los arreglos es impar, los elemenos que se descartan dependen de la comparación entre $x_{(n+1)/2}$ e $y_{(n+1)/2}$.

   (a) Si $x_{(n+1)/2} \geq y_{(n+1)/2}$ entonces se descarta $X((n+1)/2..n]$ e $Y[1..(n+1)/2)$. Como la mediana es menor o igual que $x_{(n+1)/2}$ por estar incluída y los elementos de $X((n+1)/2..n]$ son mayores o iguales que $x_{(n+1)/2}$ entonces se descartan $(n-1)/2$ elementos mayores o iguales que la mediana. Ademas como la mediana es mayor o igual que $y_{(n+1)/2}$ por estar incluída y los elementos de $Y(1..(n+1)/2]$ son menores o iguales que $y_{(n+1)/2}$ entonces se descartan $(n-1)/2$ elementos menores o iguales que la mediana. De esta manera queda demostrado que para el caso en que n es impar y $x_{(n+1)/2}$ es mayor o igual a $y_{(n+1)/2}$ se descartan la misma cantidad de elementos menores o igual que mayores o iguales que la mediana.

   (b) Si $x_{(n+1)/2} < y_{(n+1)/2}$ entonces se descarta $Y((n+1)/2..n]$ e $X[1..(n+1)/2)$. Como la mediana es mayor o igual que $x_{(n+1)/2}$ por estar incluída y los elementos de $X[1..(n+1)/2)$ son menores o iguales que $x_{(n+1)/2}$ entonces se descartan $(n-1)/2$ elementos menores o iguales que la mediana. Ademas como la mediana es menor o igual que $y_{(n+1)/2}$ por estar incluída y los elementos de $Y((n+1)/2..n]$ son mayores o iguales que $y_{(n+1)/2}$ entonces se descartan $(n-1)/2$ elementos mayores o iguales que la mediana. De esta manera queda demostrado que para el caso en que n es impar y $x_{(n+1)/2}$ es menor o igual a $y_{(n+1)/2}$ se descartan la misma cantidad de elementos menores o igual que mayores o iguales que la mediana.

2. Si la longitud de los arreglos es par, los elementos que se descartan dependen de la comparación entre $x_{n/2}$ e $y_{n/2+1}$.

   (a) Si $x_{n/2} \geq y_{n/2+1}$ entonces se descarta $X(n/2..n]$ e $Y[1..n/2+1)$. Como la mediana es menor o igual que $x_{n/2}$ por estar incluída y los elementos de $X(n/2..n]$ son mayores o iguales que $x_{n/2}$ entonces se descartan $n/2$ elementos mayores o iguales que la mediana. Ademas como la mediana es mayor o igual que $y_{n/2+1}$ por estar incluída y los elementos de $Y[1..n/2+1)$ son menores o iguales que $y_{n/2+1}$ entonces se descartan $n/2$ elementos menores o iguales que la mediana. De esta manera queda demostrado que para el caso en que n es par y $x_{n/2}$ es mayor o igual a $y_{n/2+1}$ se descartan la misma cantidad de elementos menores o igual que mayores o iguales que la mediana.

   (b) Si $x_{n/2} < y_{n/2+1}$ entonces se descarta $Y(n/2+1..n]$ e $X[1..n/2)$. Como la mediana es menor que $x_{n/2}$ por estar incluída y los elementos de $X[1..n/2)$ son mayores que $x_{n/2}$ entonces se descartan $n/2-1$ elementos menores o iguales que la mediana. Ademas como la mediana es mayor o igual que $y_{n/2+}$ por estar incluída y los elementos de $Y(n/2+1..n]$ son mayores o iguales que $y_{n/2+1}$ entonces se descartan $n/2-1$ elementos mayores o iguales que la mediana. De esta manera queda demostrado que para el caso en que n es par y $x_{n/2}$ es menor o igual a $y_{n/2+1}$ se descartan la misma cantidad de elementos menores o igual que mayores o iguales que la mediana.

Finalmente queda demostrado que en todos los casos se descartan la misma cantidad de elementos mayores o iguales y menores o iguales que la mediana.

Juntando ambas demostraciones podemos afirmar que en cada llamado recursivo la mediana original es la misma que la mediana entre la concatenación de los dos nuevos arreglos. Debido a esto, al aplicar sucesivos llamados a la función se desembocara en un caso base el cual nos devolverá el resultado buscado.

Complejidad

En la siguiente sección se calculará la complejidad en el peor caso del algoritmo presentado basándose en el modelo de cómputo uniforme.

La función presenta dos casos base que se dan cuando n (variable que indica la longitud de los arreglos) toma los valores 1 ó 2. En ambos casos el algoritmo sólo realiza comparaciones entre elementos y retorna un resultado, sin entrar en ningún ciclo.

Si n es mayor a 2 primero se realizan asignaciones y operaciones simples (tiempo constante) para luego efectuar un llamado recursivo. En el caso en que el tamaño de los arreglos es par, se toma $n = n/2$ como nuevo parámetro de longitud. En el caso impar, la longitud de los arreglos será $n + 1/2$. De esta manera, cuando n es potencia de 2, el algoritmo ejecuta $\log(n)$ llamados recursivos hasta llegar a un caso base. De lo contrario se producen $\log(n) + 1$ llamadas (se refiere a la parte entera de $\log(n)$).

Por lo tanto la función que determina la complejidad es:

$$T(n) = c + T(n/2) = c + c + T(n/4) = .. = \sum_{i=1}^{\log(n)+1} c = c \sum_{i=1}^{\log(n)+1} 1 = c(\log(n) + 1)$$

$$\implies T(n) \in O(\log(n))$$

Por último se estudiará la complejidad en relación al tamaño de la entrada. Sea t el tamaño de la entrada, X e Y los arreglos pasados como parámetro y n su tamaño.

$$t = \log(n) + \sum_{i=1}^{n} \log(X_i) + \sum_{i=1}^{n} \log(Y_i) > \log(n) + \sum_{i=1}^{n} 1 + \sum_{i=1}^{n} 1$$

$$= \log(n) + 2n > \log(n)$$

$$\implies \text{como } T(n) \in O(\log(n)) \text{ y } \log(n) < t \implies T(t) \in O(t)$$

Análisis de resultados

TODO

Conclusiones

TODO

Referencias

- Articulo sobre el formato JPG de Wikipedia: *http://es.wikipedia.org/wiki/JPG*.
- Manuales de Intel de la Arquitectura IA32, Volúmenes 2A y 2B.