Diseño y Análisis de Algoritmos, 2023.
Tarea dos.

Maestría en Ciencias de la Computación, Cinvestav.

Entregar: lunes 2 de Octubre

Resuelve los siguientes problemas, muestra tu trabajo. Entregar las soluciones por escrito.

1. Usando el árbol de recursión, resuelve las siguientes recurrencias
   a) $T(n) = 3T(n/2) + n$
   b) $T(n) = 2T(n-3) + n/2$

2. Considera un tablero de ajedrez de tamaño $2^n \times 2^n$ al que se le ha quitado, arbitrariamente, alguna casilla.

   a) Argumenta por qué cualquier tablero, con las características descritas antes, se puede adoquinar (sin dejar huecos) con adoquines en forma de L (cada uno cubre tres casillas). Hint: Recursión.

   b) Describe y analiza un algoritmo que adoquine el tablero, la entrada a tu algoritmo será n y dos enteros de n bits representando el renglón y la columna de la casilla que se removió. Expresa la complejidad de tu algoritmo en términos de las llamadas recursivas. Hint: Recursión.

3. Los n discos de tu juego de las torres de Hanoi se han desordenado, de tal forma que los discos se encuentran apilados en desorden, y no de menor a mayor. Deseas ordenar los discos de menor a mayor para poder jugar. La única operación que te está permitido hacer es un *giro*: para cualquier k, $1 \leq k \leq n$, tomar los k discos de hasta arriba, y voltear la subpila entera. Mira la figura 1 que ilustra esta operación. Nota que se usa el mismo poste.

   a) Describe un algoritmo (recursivo)para ordenar una pila arbitraria de n discos usando la menor cantidad posible de giros. ¿Exactamente cuántos giros requiere tu algoritmo en el peor caso?

   b) Ahora supón que un lado de cada disco está sucio. Como no quieres que nadie note esto, quieres ordenarlos de tal forma que el lado sucio siempre quede hacia abajo. Describe un algoritmo para ordenar una pila arbitraria de n discos, de tal forma que el lado sucio siempre quede hacia abajo, usando la menor cantidad de giros posible. ¿Exactamente cuántos giros usa tu algoritmo en el peor caso?

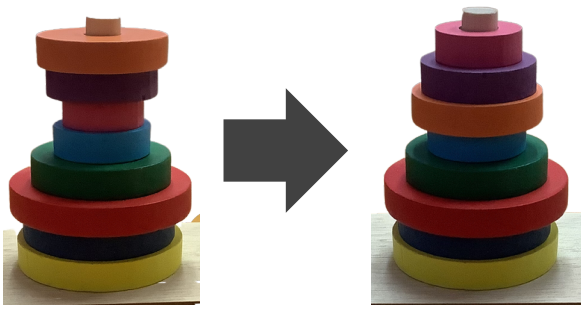

Figura 1: Resultado de la operación giro sobre los primeros $k = 3$ discos.

4. Considera el escenario en el que QuickSort se comporta como sigue: en el nivel dos de su árbol de recursión (las dos primeras llamadas recursivas) ocurre que el pivote divide al arreglo a la mitad, en el siguiente nivel el pivote cae en la última posición, en el siguiente nivel cae a la mitad, etcétera. Dibuja el árbol de recursión y acota por arriba la complejidad de QuickSort en tal escenario.

5. En este problema volveremos a jugar Torres de Hanoi. Los postes están etiquetados como $0, 1$ y 2, y el objetivo es mover los n discos del poste 0 al poste 2. Considera el siguiente algoritmo iterativo para resolver el problema. La función $p(n)$ se define como el menor entero k tal que $\frac{n}{2^k}$ no es un entero. Por ejemplo, $p(8) = 4$, pues $8/2^4$ no es entero pero $8/2^3$ sí lo es. Los primeros enteros en la secuencia $p(n)$ son: $1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 6, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1,2, 1, 7, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1,2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 6, 1, 2, 1, 3, 1, 2, 1, 4, 1, \ldots$.

```
1: procedure IHanoi(n)
2:     i = 1
3:     while p(i) ≤ n do
4:         if n − i es par then
5:             mueve el disco i hacia adelante        // 0 → 1 → 2 → 0
6:         else
7:             mueve el disco i hacia atrás           // 0 → 2 → 1 → 0
8:         i = i + 1
```

Demuestra que el algoritmo recursivo para resolver Torres de Hanoi, que vimos en clase, hace exactamente la misma secuencia de movimientos que IHanoi.