Nombre: Tomás Maturana

Rut: 19.186.973-7

Control 1

P1)

1) Supongamos que se preprocesa el conjunto, ordenándolo crecientemente según k, entregando este arreglo como input.

   - Para reducir B a A, basta con crear un contador "count", entregarle el input a B, luego sumar 1 a count en cada iteración de B y finalmente entregar count como el output. Esto funciona a priori si B hace una búsqueda lineal en el input, hasta encontrar un k' mayor o igual a x, luego devolviendo el valor leído justo antes de encontrar k'.

   - Para reducir A a B, se crea una variable "val" que guarda el valor del par encontrado por A en la penúltima iteración. Luego de entregar el input a A y que se encuentra k' mayor o igual a x mediante búsqueda lineal, se entrega val como output.

2) La complejidad será $O(\log_2 n)$, ya que para ambos algoritmos, el proceso con mayor complejidad será la búsqueda en un arreglo ordenado de n elementos; lo demás son procesos con complejidad constante que se suman a la complejidad de la búsqueda, resultando sólo la complejidad de la búsqueda.

P2)

1) Ya que se debe comparar el ángulo formado por p y p' con el ángulo alpha ya dado, se tendrán que hacer n-1 comparaciones. Esto ya que p es uno de los n elementos y se debe calcular alpha prima con cada uno de ellos, excepto él mismo. Luego, existen n-1 alpha prima que comparar con alpha; en efecto, se deben hacer <u>n-1 comparaciones</u>. Por el contrario, si no se comparara con uno de esos alpha, el adversario podría decir que justo ese alpha prima no comparado es menor que todos los demás.

2) Se puede esperar que sea de una complejidad de al menos O(n+r*f(n)). Esto ya que el algoritmo completo quedaría con complejidad O(complejidadConsulta + n), luego complejidadConsulta >= n+r*f(n). Si no fuera así, tendríamos un algoritmo de costo menor a O(n+r*f(n)) para la cápsula convexa.

P3)

1) Nuestro algoritmo debería lograr construir el arreglo C en $O(n \log_m n)$. Consideramos memoria de tamaño M, bloques de tamaño B y que M>B.
   Para esto, se llama recursivamente al algoritmo, que va dividiendo el arreglo B[] en no más de (M/B -1) subarreglos, hasta que estos subarreglos quepan en memoria. Estando en memoria se hace el cálculo de B[i] y se escriben en C[] ordenados de menor a mayor. Luego,

la cantidad de niveles del árbol de recursión para B[] será $O(\log_{M/B} N/M)$, resultando en $O(n \log_m n)$ I/Os.
Una vez terminada esa etapa, se hace lo mismo, pero ahora con A[i] y C[] a la vez… se llama recursivamente, dividiendo A[] y C[] en no más de (M/B -1)/2 subarreglos hasta que quepan en memoria; luego se substituye C[i] por A[C[i]] para todo i. A penas se acaba un subarreglo de C[]), se lee el siguiente subarreglo de C[]. Si no se encuentra el valor de A[C[i]], se lee el siguiente subarreglo de A[]. Ya que ambos están ordenados de menor a mayor, siempre se encontrará el valor de A[C[i]] en los subarreglos en memoria o en alguno de los siguientes a leer. (Esto es asumiendo que el valor C[i] está entre los N valores de i.) Todo esto será $O(n \log_m n + n \log_m n)$ I/Os, ya que se hará lo mismo que para B[], pero esta vez a cada arreglo.
Finalmente, la complejidad del algoritmo queda en $O(n \log_m n + n \log_m n + n \log_m n)$ I/Os, que es lo mismo que $O(n \log_m n)$ I/Os.

2) El problema de este algoritmo es si se usan discos magnéticos, ya que si se ocupan muchos bloques, aumenta la cantidad de seeks a posiciones aleatorias. Aún así, si no se ocupa la cantidad de bloques máxima, sigue siendo mejor que O(N).