Ejercicio I

El programas se ejecuta ingresando un ./main x, con x > 0. Este x indica el número que se desea buscar dentro del arreglo.

El programa inicia tomando x como parámetro, creando un n aleatorio equivalente al largo que tendrá el arrelgo A (El cuál toma valores desde 1 hasta 500). También crea la variable inc que tendrá, un incremento de los números del arreglo y unos punteros.

Se inicia con el primer valor del arreglo A en 0 y se ejecuta un for para añadir los números aleatorios. Estos se irán creando con un incremento (El número de la posición siguiente siempre será mayor por "inc" del anterior, esto para asegurarnos de que el arreglo será no decreciente).

La función isXinA busca donde podría estar aproximadamente el valor 'x' en el arreglo. Para esto toma su posición aproximada del método posAprox, y según esto empieza a realizar la búsqueda en los márgenes de esa posición. El algoritmo irá revisando en caso de que no se exceda de los márgenes del arreglo. En caso de que lo encuentre retornará un true, caso contrario un false.

BbXinA realiza la busqueda binaria contando cuántas veces le tomó al algorimo encontrar el número deseado. Lo que hace la BB es ir acortando la busqueda en la mitad, hasta llegar a dar con el número deseado. En caso que no lo encuentre retornará false.

Ejercicio II

El programas se ejecuta ingresando un ./main n, con n > 0. Este n indica la cantidad de nodos que se le desean ingresar al programa.

El programa incia creando las tres columnas y las tres pilas respectivamente, al iniciar, se rellena la columna Q1 con números aleatorios, tanto como de nivel de urgencias (1 al 3), como de la edad de los pacientes (1 al 90).

La manera en la que se me ocurrió realizar el intercambio de pacientes entre las pilas fue la siguiente: El algoritmo recorre la cola original y comienza detectando solo los pacientes con prioridad 3, cuando lo detecta, crea un nuevo nodo con la información de este para agregarlo a la cola 2 (Q2), y cuando termina de agregarlo, elimina el nodo de la cola 1 (Q1) para que así sea más fácil realizar la búsqueda (Ya que así habrán menos nodos para recorrer en la cola).

El modo de eliminar no pide la edad del paciente, solo pide el nivel de prioridad que debe tener el nodo. Esto se debe a que el algoritmo irá recorriendo de izquierda a derecha hasta encontrar un nodo con prioridad 2 (La edad del paciente en este caso es despreciable), una vez que lo encuentre realizará la copia, pero como siempre detectará el primero de la lista (Sin importar si se repite), al realizar la búsqueda para eliminarlo encontrará siempre la primera coincidencia (La misma que es agregada a Q2).

Cuando termina de rellenar la cola de prioridad 2, empieza a rellenar la cola de prioridad 3 según las condiciones de edad dadas. Para esto se crean varios punteros que recorren la cola, y por cada coincidencia de edad la va agregando a Q3. Sé que el hecho de realizar varios punteros no es un modo muy "Sofisticado" de resolverlo, pero sí es bastante intuitivo y entendible. Además de que no encontré una mejor manera de que se agregaran los nodos con las condiciones de edad de manera variable.

Una vez que se agregaron todos los nodos de prioridad 3, el algoritmo elimina todos los nodos que tengan esta prioridad para que así sea más fácil y rápida la búsqueda con los pacientes de prioridad 1. Así, en Q1 solo quedarán pacientes de esta prioridad, por lo que esto agiliza el hecho que se deban agregar a las seis pilas.

A la hora de buscar un método más eficiente que el if para realizar la búsqueda sin preguntar, llegué a este link en internet (https://elbauldelprogramador.com/por-que-un-switch-es-mas-rapido-que-su-homologo-if-then-else/), el cual dice que en términos de eficiencie el Switch es superior al if.

Dado que encontré esto, realicé la implementación del switch dentro del algoritmo. Dado que el switch no admite condiciones dentro de rangos, tuve que ingresar varios switch. Cada uno de estos switch representa el rango de edad en el que se desean ingresar a los pacientes. Luego de eso, el algorimo solo recorre la cola (La cual ahora solo tiene pacientes de prioridad 1) y los va ingresando en las pilas correspondientes.

En este caso, descarté la idea de ir eliminando los nodos una vez utilizados, ya que como se recorre de izquiera a derecha la cola, no se realiza un búsqueda nuevamente dentro de esta. Además cuando se pasen los nodos de Q1 a $P_{i,}$, estos no volverán a ser utilzados (Se finaliza el programa).

Eliminar los nodos solo implicaría un consumo extra de rendimiento, y como este no volverá a ser utilizado es innecesario.