1. Dada un arreglo de n enteros, $A[0..n-1]$, tal que i, $0 \leq$ i n, se tiene que $|A[i] - A[i+1]| \leq 1$; si $A[0] = x$ y $A[n-1] = y$, se tiene que $x \leq y$. Diseñar un algoritmo de búsqueda eficiente, de orden logarítmico, que localice al índice j tal que $a[j] = z$, para un valor dado de $z, x \leq z \leq y$. **Justificar**.
   Respuesta
   Sea $[a] = A[i] - A[i+1]$ tenemos que $|a| \leq 1 \iff -1 \leq a \leq 1$ la cual nos quedaría de esta manera: $A[0...n-1]$tal que $\forall i, 0 \leq i \leq n-1$, obtenemos que $-1 \leq A[i] - A[i+1] \leq 1$, entonces $A[0] = x$ y $A[n-1] = y$, por ende tenemos que $x < y$ y con estas condiciones podemos decir que el arreglo A tiene una secuencia ascendente, debido a que sus elementos van incrementando de 1 en 1, esto gracias a la condición mencionada. Hagamos una ejecución para notar algunas cosas
   $A[0] = p$ luego $A[1] = p+1$ pero p es un elemento repetido donde $A[0] - A[1] = 0 \leq 1$, luego $A[1] = p+2$ y así sucesivamente hasta que $A[n-1] = p + (n+1)$ como podemos ver en la ejecución el arreglo de A, excepto los elementos iniciales y finales, siendo 1 la diferencia mínima entre estos elementos.

2. Suponga que se está usando un programa que manipula textos muy grandes, como un procesador de palabras. El programa toma como entrada un texto, representado como una secuencia de caracteres y produce alguna salida. Si en algún momento el programa encuentra un error del cual no puede recuperarse y además no puede indicar qué error es o dónde está, entonces la única acción que el programa toma es escribir ERROR y abortar el proceso. Supóngase que el error es local, esto es, se tiene una cadena en particular del texto la cual, por alguna extraña razón, al programa no le gusta. El error es independiente del contexto en el cual aparece la cadena “ofensiva”. Diseñar una estrategia logarítmica para localizar la fuente del error. **Respuesta**
   La estrategia a seguir es dividir el texto en mitades y usar esas mitades como entrada del programa, pero estas nos darían un error, entonces lo que hacemos es agarrar la parte que nos da error y aplicamos lo que dijimos anterior mente, es decir, dividimos el texto y usamos esas mitades como entrada del programa y después sucederá que una de las nuevas mitades de error, entonces volveremos a aplicarlo recursivamente hasta que no haya ningún error.

```
1  //PreCond: Que el texto no sea vacío
2  busaqueDeError(texto Arreglo de Cadenas
3                  parteIzq: Int
4                  parteDer: Int)
5              if (parteIzq == parteDer){
6                  return parteIzq
7              }
8              else {
9                  mitad=(parteIzq+parteDer)/2
10                 //caso en el que siga teniendo error el programa
11                 restulado = (texto[parteIzq+mitad])
12                 if(no es un error){
13                     return buscarError(texto.mitad+parteDer)
14                 }
15                 else{
16                     return buscarError(texto.mitad+parteIzq)
17                 }
18             }
19
```

3. Consideremos el siguiente juego, entre dos personas:
   El jugador J_A piensa un número entero en un rango. El jugador J_B intenta encontrar tal número haciendo preguntas de la forma:

   ¿ Es el número menor que x? o ¿ Es mayor que y ?

   El objetivo es realizar el menor número de preguntas. Se supone que nadie hace trampa.
   a) Diseñar una buena estrategia para el juego... cuando el jugador JA indica un rango específico, digamos

de 1 a N.
En este caso, ¿Cuál resulta ser la complejidad del algoritmo? **Justificar**.
Respuesta
Para este problema tenemos como cota mínima o inicial al 0 ya que es el mínimo número entero, pero no tenemos definido la cota superior por lo cual no tenemos idea de hasta dónde extendernos ni algún referente de X. Al no tener espacio de búsqueda delimitado es imposible aplicar búsqueda binaria directamente y tampoco podemos usar interpolación al no conocer X. El inicio del problema es no tener delimitado el espacio de búsqueda y para hacerlo es importante que sea de la manera más pequeña posible para no tener que hacer tantas comparaciones.
Ejemplo.
Supongamos que el número por adivinar es 100,000
Caso 1: Delimitar el espacio de búsqueda a 1,000,000
Preguntar si 1,000,000 es menor a X
Aplicar búsqueda binaria desde 0 a 500,000 hasta llegar a 100
0...100,000...200,000...300,000...400,000...500,000.....1,000,000
0-250,000
0-125,000
0-62,500
= Aquí lo encontramos entre los últimos dos
Caso 2: Delimitar el espacio de búsqueda a 125,000
Se ahorrarían dos preguntas iniciando en si X es menor a 500,000?
0-250
Ya que el objetivo es el menor número de preguntas posible la idea más conveniente es es usar la búsqueda exponencial con una modificada.
Sabemos que el número X se encontrará en algún punto, la idea es ir avanzando con búsqueda exponencial hasta hallar una cota superior inicial y después una búsqueda binaria. Sólo que la condición para salir de la búsqueda exponencial es si el límite superior es menor al tamaño de la entrada. Sabemos que encontraremos X, entonces nuestra búsqueda acabará cuando encontremos a X, no es necesario regresar algo que indique que la búsqueda ha fallado.

b) Diseñar una buena estrategia para el juego... cuando el jugador JA no indica el rango del número que pensó. ¿Cuál es la complejidad del algoritmo propuesto? **Justificar**.
Respuesta
1. Establecer límite inferior =0 y límite superior 2^8 (valor arbitrario)
2. Preguntar si X es menor del límite superior
a. Si este es el caso hemos encontrado el espacio de búsqueda y comenzamos búsqueda binaria sobre:
$A[límite_inf....límite_sup]$
b. Si no actualizamos valores:
$límite_inf = límite_sup$
$límite_sup = límite_sup * 2$

4. Consideremos el Algoritmo de Búsqueda por Interpolación.
(a) Presentar un ejemplo de al menos 300 datos para el cual el algoritmo termine la búsqueda (exitosa o no) en pocas iteraciones. **Ejemplificar**.
Respuesta
Propongamos un arreglo A tal que $[4n+1|1 \leq n \leq 300]$. Ahora escogeremos el número 229 para encontrarlo en el arreglo, aplicando en algoritmo nos queda algo asi:
izq=1, der= 300
número a buscar = x = 229

$$ecc = izq + \left[\frac{(x - A[izq](der - izq))}{A[der] - A[izq]}\right] + 1$$
$$= 0 + \left[\frac{(229-5)(300-1)}{1201-5}\right] + 1$$
$$= \left[\frac{(224)(299)}{1196}\right] + 1$$
$$= \left[\frac{66976}{1196}\right] + 1 = 56 + 1 = 57$$

Ahora verificamos esto:
$4n + 1 = 4(57) + 1 = 28 + 1 = 229$ y esto nos demuestra que en tan solo una iteración se pudo obtener.
(b) Dar un ejemplo de, al menos, 300 datos para el cual el algoritmo termine la búsqueda (exitosa o no) en muchas iteraciones. **Ejemplificar**.
Justificar la respuesta, en ambos casos.
Respuesta
Ahora supongamos un arreglo A, que tiene una secuencia de $[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21....]$ así hasta lograr 300 elementos y queremos buscar el elementos 19 que esta en la posición 9, empezamos comparando los elementos. Antes de ver un ejemplo tenemos que dejar unas cosas claras, como i= 1 y tenemos la condición de $i < n$ y $A[i] \leq Buscado$, se debe de incrementar el valor de de $i = i^2$ hasta encontrar el elemento que buscamos, luego debemos de dividir $i/2$ al final del arreglo, es decir, A[] $-Busqueda(A[i], i/2, min(i, n-1))$. Ahora con esto tenemos el arreglo A en la posición 0 $A[0] = 1$ y buscamos el número 19 y no es el número que buscamos, entonces seguimos con con el siguiente elemento $A[1] = 3$ y tampoco es el número que buscamos, seguimos incrementando hasta que i sea menor o igual a i, pero esto no se podrá hacer debido a que i^2=16, entonces tendremos que dividir $i/2$ y vemos que el elemento anterior es 17 en la posición 8 y el elemento después es el 21 en la posición 10, entonces, ahora sustituimos los valores

$$mid = (10 + 8)/2$$

$$= 18/2 = 9$$

que es la posición en la que esta el elemento que estamos buscando que es 19. En este ejercicio la cable es i, debido a que no sabemos si la estructura que estamos utilizando es ilimitada o demasiado grande y con este método podemos llegar a la respuesta en un tiempo más rápido.