Tecnológico de Monterrey, Campus Monterrey
Departamento de Ciencias Computacionales
ANALISIS Y DISEÑO DE ALGORITMOS - Ing. Román Martínez M.
PRIMER EXAMEN DE PROGRAMACIÓN - 8 de Marzo 2016

Nombre: _____Matrícula: _____

Instrucciones:

- Para resolver este examen puedes utilizar el compilador de C++ de tu preferencia para programar tus respuestas.
- Se te provee de un archivo llamado `ex1em16.cpp` en donde se encuentra cierto código que utilizarás y en ese mismo archivo programarás tus respuestas.
- Identifica tu archivo con un comentario inicial con tu nombre y matrícula, y documenta el código con los comentarios que consideres importantes y necesarios para explicar lo que programas.
- El archivo final de entrega como respuesta de tu examen, deberá subirse a la página del curso en el espacio correspondiente.
- Está estrictamente PROHIBIDO el acceso a códigos previamente realizados y cualquier tipo de comunicación electrónica o presencial. Cualquier situación deshonesta será penalizada con una calificación de DA.

Revisa el código fuente que se encuentra en el archivo `ex1em16.cpp`. Podrás observar que se encuentran definidos algunos arreglos que utilizarás y que ya está la implementación del algoritmo del Mergesort y parcialmente el de Gilbert y Moore para encontrar el ABB óptimo. Usa este código para resolver los problemas II y III que se presentan a continuación. El problema I no requiere de ningún código adicional. Para probar el programa, deberás seguir las instrucciones que se indican y responder las preguntas asociadas a cada problema. El ALGORITMARIO contiene únicamente el algoritmo de la búsqueda binaria por si te fuera útil.

PROBLEMA 1. *(30 puntos – Tiempo sugerido para solucionar el problema: 20 min).*

Dado un arreglo **ordenado** ascendentemente de enteros (todos distintos), implementa un algoritmo de complejidad O(log n) en su peor caso, que sirva para encontrar un índice i tal que a[i] = i, suponiendo que tal índice existe.

Responde en base a los resultados de tu programa:

- ¿Cuál es el índice que da como resultado el algoritmo con el arreglo datos1? 1
- ¿Cuál es el índice que da como resultado el algoritmo con el arreglo datos2 que se te envió por correo electrónico? No tengo ese arreglo
- ¿Cuál es la técnica de diseño de algoritmos que utilizaste y porqué representa una ventaja utilizarla?

  Variante de DyV y la ventaja es que es log n, es decir, es lo mejor que hay

PROBLEMA 2. *(35 puntos – Tiempo sugerido para solucionar el problema: 30 min).*

¿Es mejor dividir en 3 partes que en 2 al realizar el ordenamiento por el Mergesort?
Modifica el algoritmo que ya se encuentra en el archivo `ex1em16.cpp` para que ahora en vez de dividir el arreglo de datos en dos mitades, lo haga en 3 partes de un tamaño igual o similar. Una vez ordenada cada tercera parte, deberá utilizarse el mismo algoritmo de unión (Une sin modificaciones) para construir la solución, primero uniendo las primeras dos partes, y el resultado, unirlo con la tercera parte.

Responde lo siguiente:

- ¿Funciona tu algoritmo para ordenar el arreglo datos3? si
- ¿Cómo afecta en el orden de complejidad este cambio en el algoritmo? ¿lo mejora o lo empeora? Explica y justifica brevemente.

aunque sigue siendo n log n, empeora

PROBLEMA 3. *(35 puntos – Tiempo sugerido para solucionar el problema: 30 min).*

En el archivo `ex1em16.cpp` podrás encontrar la implementación del algoritmo de **Gilbert y Moore** para encontrar el ABB óptimo. Sin embargo, hace falta la implementación de las funciones `minimo` y `sumatoria` para que puedan utilizarse para obtener resultados. Implementa estas funciones y prueba el algoritmo con el arreglo llamado `datos4` que se te enviará por correo electrónico y que contiene las probabilidades de búsqueda de las llaves a insertar en el ABB, y responde lo siguiente:

- ¿Cuál es el promedio de búsqueda de estas llaves en el ABB óptimo? 2.435
- PUNTOS EXTRAS: ¿Cuál es el índice de la llave que será la raíz en el ABB óptimo? _____
- ¿Cuál es el promedio de búsqueda en el ABB óptimo que se formaría con la secuencia de la tercera a la séptima llave? 1.43
- ¿Cuál es el orden de complejidad del algoritmo que implementaste para la función `mínimo`? n
- ¿Cuál es el orden de complejidad del algoritmo que implementaste para la función `sumatoria`? n

Responde: ¿Qué resultado esperas en este examen? _____ ¿es congruente con tu aprendizaje y esfuerzo?

ALGORITMARIO

Algoritmo recursivo en pseudocódigo de la búsqueda binaria

```
function busca (inicio, fin: index) : index
if (inicio > fin) then return 0;
else
   mitad = (inicio + fin) div 2;
   if (x == arreglo[mitad]) then return mitad;
   else if (x < arreglo[mitad]) then
            return(busca(inicio, mitad-1));
        else return(busca(mitad+1,fin));
```

ANEXO – código de inicio para el examen

```
#include <cstdlib>
#include <iostream>

using namespace std;

const int TAM = 500;
const int N = 21;

// DEFINICIONES DE DATOS NECESARIOS PARA LAS PRUEBAS

// Problema 1
// Escribe aqui la implementación del problema 1

// Problema 2

// Función para unir dos subarreglos ya ordenados
void une(int datos[], int inicio, int fin, int mitad)
{
    int i, j, k, aux[TAM];
    i = inicio;
    k = inicio;
    j = mitad + 1;
    while (i <= mitad && j <= fin)
    {
        if (datos[i] < datos[j])
        {
            aux[k] = datos[i];
            k++;
            i++;
        }
        else
        {
            aux[k] = datos[j];
            k++;
            j++;
        }
    }
    while (i <= mitad)
    {
        aux[k] = datos[i];
        k++;
        i++;
    }
    while (j <= fin)
    {
        aux[k] = datos[j];
        k++;
        j++;
    }
    for (i = inicio; i < k; i++)
    {
        datos[i] = aux[i];
    }
}

// Función recursiva para hacer el ordenamiento
void mergesort(int datos[], int inicio, int fin)
{
    int mitad;
    if (inicio < fin)
    {
        mitad = (fin-inicio)/2;
        mergesort(datos,inicio,inicio+mitad);
        mergesort(datos,inicio+mitad+1,fin);
        une(datos,inicio,fin,inicio+mitad);
    }
}
```

```
//Problema 3
// Algoritmo de Gilbert y Moore

float minimo(int i, int j, float A[N][N])
{
	// implementación a resolver en el problema 3
}

float sumatoria(int i, int j, float p[])
{
	// implementación a resolver en el problema 3
}


float gilbertymoore (float p[], int n)
{
	float A[N][N], R[N][N];
	// se considera que se usarán la columnas 0 a n y los renglones 1 a n+1 en la matriz
	int j;

	for (int i = 1; i<= n; i++)
	{
		A[i][i-1] = 0; A[i][i] = p[i];
		R[i][i] = i; R[i][i-1] = 0;
	}
	A[n+1][n] = 0;
	R[n+1][n] = 0;

	for (int diag = 1; diag <= n-1; diag++)
	  for (int i = 1; i <= n-diag; i++)
		{
			j = i+diag;
			A[i][j] = minimo(i, j, A) + sumatoria(i, j, p);
			// calcula el valor mínimo entre los diversos valores de:
			// A[i,k-1] + A[k+1, j] para k desde i hasta j.
			// La sumatoria calcula la suma de las probabilidades de la llave i hasta j
		}
	return A[1][n];
}


int main()
{
    //Escribe aqui las llamadas para probar y responder preguntas del examen

	system("pause");
}
```