Tarea 7

Layla Tame A01192934

**Divide y Vencerás – Problema 1 Examen EM16**

#include <cstdlib>

#include <iostream>

using namespace std;

const int TAM = 500;

const int N = 21;

// DEFINICIONES DE DATOS NECESARIOS PARA LAS PRUEBAS

int arr1[10] = {-1,0,1,3,4,6,7,8,9,10};

int arr2[20] = {-4,-3,-2,-1,0,2,3,4,6,7,8,10,12,14,15,16,19,20,21,22};

// Problema 1

int binSearch(int arr[], int iStart, int iFinish)

{

int iIndex;

if (iFinish >= iStart)

{

int iMiddle = (iStart + iFinish)/ 2;

if (arr[iMiddle] == iMiddle)

{

return iMiddle;

}

else if (arr[iMiddle] > iMiddle)

{

iIndex = binSearch(arr, iStart, iMiddle - 1);

}

else

{

iIndex = binSearch(arr, iMiddle + 1, iFinish);

}

}

else

{

iIndex = -1;

}

return iIndex;

}

int main()

{

cout << "Problema 1.1: " << binSearch(arr1, 0, 9) << endl;

cout << "Problema 1.2: " << binSearch(arr1, 0, 19) << endl;

}

Responde en base a los resultados de tu programa:

\*Como no se tenían los arreglos de datos, se inventaron dos diferentes.

1. **¿Cuál es el índice que da como resultado el algoritmo con el arreglo datos1**? 4
2. **¿Cuál es el índice que da como resultado el algoritmo con el arreglo datos2 que se te envió por correo electrónico?** 12
3. **¿Cuál es la técnica de diseño de algoritmos que utilizaste y porqué representa una ventaja utilizarla?** Utilicé divide y vencerás usando búsqueda binaria y la ventaja es que el orden del algoritmo es de O(log n) que es el mejor orden que se puede tener.

**Merge Sort – Problema 2 Examen EM16**

#include <cstdlib>

#include <iostream>

Using namespace std;

int arr3[16] = {3,5,8,1,9,4,2,13,11,20,16,0,15,7,22,14};

void mergesortMod(int arr[], int inicio, int fin)

{

int primArr, segArr;

if (inicio < fin)

{

primArr = (fin-inicio)/3;

segArr = fin - primArr;

mergesortMod(arr,inicio,inicio+primArr);

mergesortMod(arr,inicio+primArr+1,segArr);

mergesortMod(arr,segArr+1,fin);

une(arr,inicio,segArr,inicio+primArr);

une(arr,inicio,fin,segArr);

}

}

int main()

{

mergesortMod(arr3, 0, 15);

for(int i = 0; i < 16; i++)

{

cout << arr3[i] << " ";

}

cout << endl;

}

1. **¿Funciona tu algoritmo para ordenar el arreglo datos3?** Sí
2. **¿Cómo afecta en el orden de complejidad este cambio en el algoritmo? ¿lo mejora o lo empeora? Explica y justifica brevemente.** El orden de complejidad del algoritmo empeora, ya que la llamada recursiva a la función de mergesort se hace una vez más que en el mergesort normal.

**Gilbert and Moore – Problema 3 Examen EM16**

//

// main.cpp

// problema1

//

// Created by Layla Tame on 3/13/19.

// Copyright © 2019 Layla Tame. All rights reserved.

//

//Problema 1. Divide y Venceras

#include <cstdlib>

#include <iostream>

#include <float.h>

using namespace std;

float arr4[16] = {0, 0.2, 0.4, 0.6, 0.8, 0, 1.0, 1.2, 1.4, 1.6, 0, 1.8};

//Problema 3

// Algoritmo de Gilbert y Moore

float minimo(int i, int j, float A[N][N])

{

float min = FLT\_MAX;

for(int m = i; m < j+1; m++)

{

if(A[i][m-1] + A[m+1][j] < min)

{

min = A[i][m-1] + A[m+1][j];

}

}

return min;

}

float sumatoria(int i, int j, float p[])

{

float sum = 0;

for(int m = i; m < j+1; m++)

{

sum += p[m];

}

return sum;

}

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()

{

cout << "Gilbert Moore: " << gilbertymoore(arr4, 4) <<endl;

}

1. **¿Cuál es el promedio de búsqueda de estas llaves en el ABB óptimo?** 3.6
2. **¿Cuál es el orden de complejidad del algoritmo que implementaste para la función mínimo?** O(n)
3. **¿Cuál es el orden de complejidad del algoritmo que implementaste para la función sumatoria?** O(n)

**Floyd – Problema 3 Examen EM15**

//

// main.cpp

// floyd

//

// Created by Layla Tame on 3/13/19.

// Copyright © 2019 Layla Tame. All rights reserved.

//

#include <cstdlib>

#include <iostream>

using namespace std;

int costos[7][7] = { {0, 9999, 9999, 500, 9999, 9999, 9999}, { 9999, 0, 9999, 9999, 9999, 9999, 480}, { 340, 9999, 0, 9999, 9999, 9999, 9999}, { 9999, 9999, 9999, 0, 9999, 1075, 9999}, { 9999, 900, 9999, 9999, 0, 9999, 9999}, { 9999, 9999, 760, 9999, 9999, 0, 120}, { 9999, 9999, 9999, 9999, 230, 9999, 0} };

int iSize[7][7] = {0};

//Problema 3

// Algoritmo de Floyd

int min(int a,int b)

{

if(a<b)

return(a);

return(b);

}

void floyd(int p[7][7],int n)

{

for(int k=0;k<n;k++)

{

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

if (p[i][k]+p[k][j] < p[i][j])

{

iSize[i][j] = k;

p[i][j] = p[i][k]+p[k][j];

}

}

}

}

}

void sinCamino(int p[7][7], int m[7][7])

{

for(int i=0;i<7;i++)

{

for(int j=0;j<7;j++)

{

if (p[i][j] == 9999)

{

iSize[i][j] = -1;

}

}

}

}

void imprimirMatriz(int iSize[7][7])

{

for(int i=0;i<7;i++)

{

for(int j=0;j<7;j++)

{

cout << iSize[i][j] << " ";

}

cout << endl;

}

}

int main()

{

floyd(costos, 7);

sinCamino(costos, iSize);

imprimirMatriz(iSize);

}

1. **¿Cuántas trayectorias de ciudad a ciudad no implican hacer escalas?** 9
2. **¿Cuál es la cantidad máxima de escalas que se realizan las trayectorias óptimas entre las ciudades?** 6
3. **¿Cuántas escalas se realizarían en la trayectoria óptima de la ciudad 3 a la ciudad 5?** 0

**Programación Dinámica –**