Instituto Politécnico Nacional

Escuela Superior de Cómputo

REPORTE DE LA SESIÓN 3: Divide y vencerás 1

Materia:

Análisis de algoritmos

Grupo:

3CM3

Integrantes:

Castro Cruces Jorge Eduardo

Fecha:

Miércoles, abril 1, 2020

a) Una vez realizado el análisis lógico-sintáctico de la expresión, se puede concluir y afirmar que es posible calcular correctamente el Máximo Común Divisor.

b)

```c
#include <stdio.h>

/*int MCD(int a, int b){
    return b ? MCD(b, a % b) : a;
}*/

int MCD(int a, int b){
    if(b==1) return a;
    else if((a%2 == 0) && (b%2 == 0))
        return MCD(a/2, b/2)*2;
    else if((a%2 != 0) && (b%2 == 0))
        return MCD(a, b/2);
    else if((a%2 != 0) && (b%2 != 0))
        return MCD((a-b)/2, b);
}

int main(){
    int a, b, mcd;
    printf("Ingrese el valor de a: ");
    scanf("%d", &a);
    printf("Ingrese el valor de b: ");
    scanf("%d", &b);
    if (b > a) mcd=MCD(b, a);
    else mcd=MCD(a, b);
    printf("El MCD de %d y %d es: %d", a, b, mcd);
    return 0;
}
```

c) **Análisis de complejidad:**

Complejidad de tiempo: Este algoritmo cuenta con una complejidad de O(logn), ya que con cada llamada recursiva se va dividiendo el problema en subproblemas.

```
Ingrese el valor de a: 20
Ingrese el valor de b: 18
El MCD de 20 y 18 es: 2
--------------------------------
Process exited after 3.175 seconds with return value 0
Presione una tecla para continuar . . .
```

```cpp
#include <iostream>
using namespace std;

int Horner(int P[], int n, int x){
    if (n<0) return 0;
    int Res = P[n-1];
    n--;
    Res += Horner(P, n, x) * x;
    return Res;
}

int main() {
    int P[] = {1, 3, 5, 7, 9}; //P(x) = x^4 + 3x^3 + 5x^2 + 7x + 9
    int x = 2; //P(2)=83
    int n = sizeof(P)/sizeof(P[0]);
    cout << "El valor del polinomio es: " << Horner(P, n, x);
    return 0;
}
```

Análisis de complejidad:

Complejidad de tiempo: El algoritmo cuenta con una complejidad de O(n*logn),

a)

```cpp
#include <iostream>
using namespace std;

int NumInv(int A[], int n){
    int Cont = 0;
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            if ((A[i] > A[j]) || (A[i] > (2*A[j])))
                Cont++;

    return Cont;
}

int main(){
    int A[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    int n = sizeof(A) / sizeof(A[0]);
    cout << "El numero de inversiones es: " << NumInv(A, n);
    return 0;
}
```

Análisis de complejidad:

Complejidad de tiempo: O (n^2), se necesitan dos bucles anidados para atravesar la matriz de principio a fin, por lo que la complejidad de tiempo es O (n ^ 2)

Complej idad espacial: O (1), no se requiere espacio adicional.

b)

```cpp
#include <bits/stdc++.h>
using namespace std;

int NumInv(int A[], int Aux[], int Izq, int Der);
int Merge(int A[], int Aux[], int Izq, int Mid, int Der);

int MergeSort(int A[], int n){
    int Aux[n];
    return NumInv(A, Aux, 0, n-1);
}

int NumInv(int A[], int Aux[], int Izq, int Der){
    int Mid, Cont=0;
    if(Der > Izq){
        Mid = (Der + Izq) / 2;
        Cont += NumInv(A, Aux, Izq, Mid);
        Cont += NumInv(A, Aux, Mid+1, Der);
        Cont += Merge(A, Aux, Izq, Mid+1, Der);
    }
    return Cont;
}

int Merge(int A[], int Aux[], int Izq, int Mid, int Der){
    int i=Izq, j=Mid, k=Izq, Cont=0;
    while((i <= Mid-1) && (j <= Der)){
        if(A[i] <= A[j])
            Aux[k++] = A[i++];
        else{
            Aux[k++] = A[j++];
            Cont = Cont + (Mid-i);
        }
    }
    while (i <= Mid-1)
        Aux[k++] = A[i++];
    while (j <= Der)
        Aux[k++] = A[j++];
    for (i=Izq; i <= Der; i++)
        A[i] = Aux[i];
    return Cont;
}

int main() {
    int A[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    int n = sizeof(A) / sizeof(A[0]);
    cout<<"El numero de inversiones es: "<<MergeSort(A, n);
```

Análisis de complejidad:

Complejidad del tiempo: O (n log n), el algoritmo utilizado es dividir y conquistar, por lo que en cada nivel se necesita un recorrido completo de la matriz y hay niveles de log n, por lo que la complejidad del tiempo es O (n log n).

Complejidad espacial: O (1), no se requiere espacio adicional.

Tenga en cuenta que el código anterior modifica (o clasifica) la matriz de entrada. Si queremos contar solo las inversiones, entonces necesitamos crear una copia de la matriz original y llamar a mergeSort () en la copia.