Práctica 2: Algoritmos Divide y Vencerás

Daniel Bolaños Martínez, José María Borrás Serrano,
Santiago De Diego De Diego, Fernando De la Hoz Moreno

ETSIIT

Introducción

Código Divide y Vencerás

Código secuencial

Listing 1: Función unimodal DyV

```
int unimodal(vector<int> v){
        bool fin=false;
        int maximo=v.size()-1;
        int indice=maximo/2;
        int minimo;

        while(!fin){
                if(v.at(indice-1)<v.at(indice))
                        if(v.at(indice+1)<v.at(
                            indice))
                                fin=true;
                        else{
                                minimo=indice;
                                indice=indice+((
                                    maximo-indice)
                                    /2);
                        }
```

Listing 2: Función main DyV

```
int main(int argc, char* argv[]){
        vector<int> array;
        int valor = -1;
        double suma=0;

        int v_size = atoi(argv[1]);
        array.resize(v_size);

        for(int i=0; i<100; ++i){
                int p = 1 + rand() %(v_size-2);
                array.at(p) = v_size-1;
                for (int i=0; i<p; i++)
                        array.at(i)=i;
                for (int i=p+1; i<v_size; i++)
                        array.at(i)=v_size-1-i+p;

        clock_t tantes;
        clock_t tdespues;
        tantes=clock();
        valor = unimodal(array);
        tdespues=clock();
        suma += (double)(tdespues - tantes) / CLOCKS_PER_SEC;
}
        cout << v_size <<" "<< suma/100 << endl;
}
```

Código secuencial

Listing 3: Función unimodal Secuencial

```
int unimodal_secuencial(vector<int> v){
        bool fin=false;
        int indice=1;

        while(!fin){
        if(v.at(indice+1)<v.at(indice))
          fin=true;
        else
          indice++;
        }

return indice;
}
```

Listing 4: Función main Secuencial

```
int main(int argc, char* argv[]){
        vector<int> array;
        int valor = -1;
        double suma=0;

        int v_size = atoi(argv[1]);
        array.resize(v_size);

        for(int i=0; i<100; ++i){
                int p = 1 + rand() %(v_size-2);
                array.at(p) = v_size-1;
                for (int i=0; i<p; i++)
                        array.at(i)=i;
                for (int i=p+1; i<v_size; i++)
                        array.at(i)=v_size-1-i+p;

        clock_t tantes;
        clock_t tdespues;
        tantes=clock();
        valor = unimodal_secuencial(array);
        tdespues=clock();
        suma += (double)(tdespues - tantes) / CLOCKS_PER_SEC;
}
        cout << v_size <<" "<< suma/100 << endl;
}
```

Comparación de la eficiencia

Eficiencia teórica

Figura: Pie de imagen

Eficiencia empírica

Eficiencia híbrida

Conclusión