Práctica 3. Divide y vencerás

Adrian Reyes Alba
adrian.reyesalba@alum.uca.es
Teléfono: 647243014
NIF: 49074565V

5 de enero de 2018

1. Describa las estructuras de datos utilizados en cada caso para la representación del terreno de batalla.

   Para el algoritmo sin-preordenación he usado una matriz y para los algoritmos de fusión, ordenación rápida, y montículo he usado un vector.

2. Implemente su propia versión del algoritmo de ordenación por fusión. Muestre a continuación el código fuente relevante.

```
//INICIO ALGORITMO DE FUSION
void fusion(std::vector<Cell>& v, int i, int k, int j) {
     int n=j-i+1, p=i , q=k+1;

    std::vector<Cell> vectorAux;

    for(int a=0; a<n; a++) {
        if(p<=k && (q>j || v[p].value > v[q].value)) {
            vectorAux.push_back(v[p]);
            p++;
        }
        else{
            vectorAux.push_back(v[q]);
            q++;
        }
    }

    for(int a=0; a<n; a++)
        v[i+a] = vectorAux[a];
}

void ordenacion_fusion(std::vector<Cell>& v, int i, int j) {
    int n = j-i+1;

    if(n <= 2) {
        if(v[i].value < v[j].value) {
            Cell vectorAux = v[j];
            v[j] = v[i];
            v[i] = vectorAux;
        }

    }
    else{

        int k = i-1 + n/2;
        ordenacion_fusion(v, i, k);
        ordenacion_fusion(v, k+1, j);
        fusion(v, i, k, j);
    }
}
//FIN ALGORITMO FUSION
```

3. Implemente su propia versión del algoritmo de ordenación rápida. Muestre a continuación el código fuente relevante.

```
//INICIO ALGORITMO ORDENACION RAPIDA

int pivote(std::vector<Cell>& v, int i, int j) {
    int p = i;
    int x = v[i].value;

    for(int k = i + 1; k <= j; ++k)
        if(v[k].value > x) {
            ++p;
            Cell aux = v[k];
            v[k] = v[p];
            v[p] = aux;
        }


    Cell v_aux = v[i];

    v[i] = v[p];
        v[p].position.x = v_aux.position.x;
        v[p].position.y = v_aux.position.y;

    v[p].value = x;

    return p;
}

void ordenacion_rapida(std::vector<Cell>& v, int i, int j) {
    int n = j-i + 1;

    if(n <= 2) {
        if(v[i].value < v[j].value) {
            Cell aux = v[j];
            v[j] = v[i];
            v[i] = aux;
        }
    }
    else {
        int p = pivote(v, i , j);
        ordenacion_rapida(v, i , p);
        ordenacion_rapida(v, p + 1, j);
    }
}//FIN ALGORITMO ORDENACION RAPIDA
```

4. Realice pruebas de caja negra para asegurar el correcto funcionamiento de los algoritmos de ordenación implementados en los ejercicios anteriores. Detalle a continuación el código relevante.

```
//INICIO FUNCION OK ORDENADO
bool ok(std::vector<Cell> v) {
    bool res = true;

    for(int i = 0; i < v.size(); ++i)
        if(v[i].value < v[i+1].value)
                res = false;

    return res;
}
//FIN FUNCION OK ORDENADO
//INICIO CAJA NEGRA
void blackBox(std::vector<Cell> v){
        ordenacion_fusion(v,0,v.size()-1);
        if(ok(v))
                std::cout << "Ordenacion por fusion ok!" << std::endl;

        ordenacion_rapida(v,0,v.size()-1);
        if(ok(v))
                std::cout << "Ordenacion rapida ok!" << std::endl;
}//FIN CAJA NEGRA
```

5. Analice de forma teórica la complejidad de las diferentes versiones del algoritmo de colocación de defensas en función de la estructura de representación del terreno de batalla elegida. Comente a continuación los resultados. Suponga un terreno de batalla cuadrado en todos los casos.

   Sin pre-ordenacion: n elevado a 2 (n cuadrado) Fusión,montículo y ordenación rápida: n log n

6. Incluya a continuación una gráfica con los resultados obtenidos. Utilice un esquema indirecto de medida (considere un error absoluto de valor 0.01 y un error relativo de valor 0.001). Es recomendable que diseñe y utilice su propio código para la medición de tiempos en lugar de usar la opción *-time-placeDefenses3* del simulador. Considere en su análisis los planetas con códigos 1500, 2500, 3500,..., 10500, al menos. Puede incluir en su análisis otros planetas que considere oportunos para justificar los resultados. Muestre a continuación el código relevante utilizado para la toma de tiempos y la realización de la gráfica.

   Adjunto 2 gráficas una es con la medida indirecta que viene en el pdf del campus, lo de errores relativos y absolutos y la segunda con el while menor q 1 tal como viene en el example. Teorícamente la que está bien es la segunda. Que sale que la ordenación rápida es mejor. En este caso siguiendo el pdf de utilizar la medida del tiempo de forma indirecta, sale que la sin-preordenación es más rápida.

Todo el material incluido en esta memoria y en los ficheros asociados es de mi autoría o ha sido facilitado por los profesores de la asignatura. Haciendo entrega de este documento confirmo que he leído la normativa de la asignatura, incluido el punto que respecta al uso de material no original.

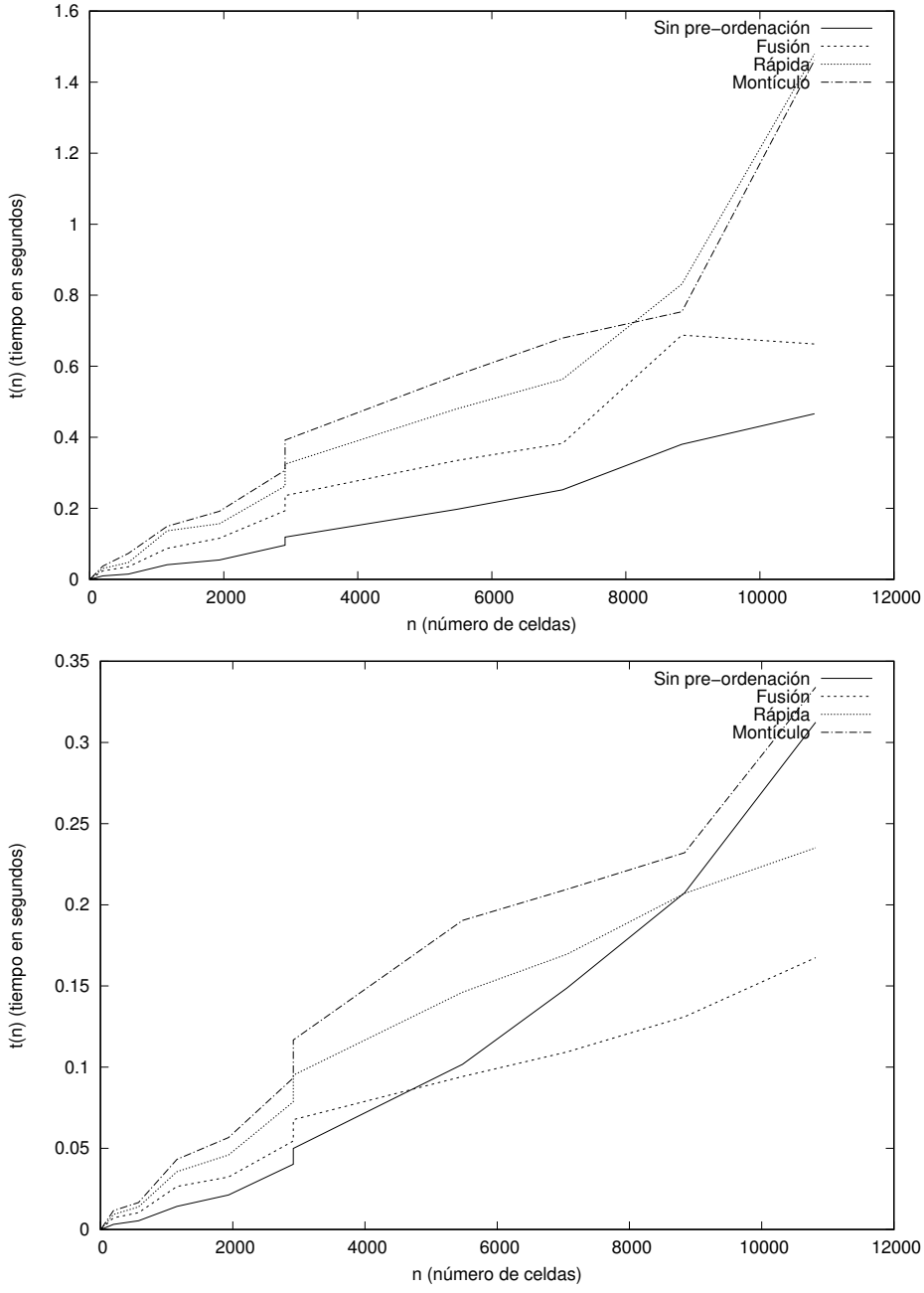

Figura 1: Grafica