Práctica 3. Divide y vencerás

Raúl Arcos Herrera
raul.arcosherrera@alum.uca.es
Teléfono: xxxxxxxx
NIF: 77175935X

19 de diciembre de 2021

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

   Para la representación del terreno de batallas, este se dividirá en celdas.
   Para las celdas he utilizado una estructura *Cell* que contiene el valor de la celda junto a su posición en el mapa.

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

```
void fusionFinal(Cell cell[], int i, int k, int j, int N){
    int n = j-i+1, p = i, q = k+1;
    Cell aux[N];
    for (int a = 0; a <= n; a++){
        if (p <= k && (q > j || cell[p].valor >= cell[q].valor)){
            aux[a] = cell[p];
            p++;
        } else {
            aux[a] = cell[q];
            q++;
        }
        cell[i+a]=aux[a];
    }
}

void ordenacionFusion(Cell cell[], int i, int j, int N){
    if (i<j){
        int k=(i+j)/2;
        ordenacionFusion(cell, i, k, N);
        ordenacionFusion(cell, k + 1, j, N);
        fusionFinal(cell, i, k, j, N);
    }
}
```

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

```
void CellRapida (Cell *cell, int i, int j){
    if (i<j){

        //Pivote
        int p = i;
        Cell aux = cell[i];
            for (int k = i+1; k<=j; k++)
                    if (cell[k].valor > aux.valor)
            {
                            p++;
                            std::swap(cell[p], cell[k]);
                    }

        cell[i] = cell[p];
```

```
            cell[p] = aux;
        //End-Pivote
        CellRapida(cell, i, p-1);
        CellRapida(cell, p + 1, j);
    }
}
```

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.

   Para las pruebas de caja negra se ha utilizado el algoritmo voraz de la P1, básicamente rellenamos la estructura *Cell* con los valores de cada celda por defecto y llamamos al procedimienton de ordenación correspondiente, después de eso utilizamos la función voraz de la P1 para colocar la mejor celda en el mapa y se mide el tiempo del proceso completo.

   Se utiliza a continuación como ejemplo la prueba de caja negra de la ordenación rápida.

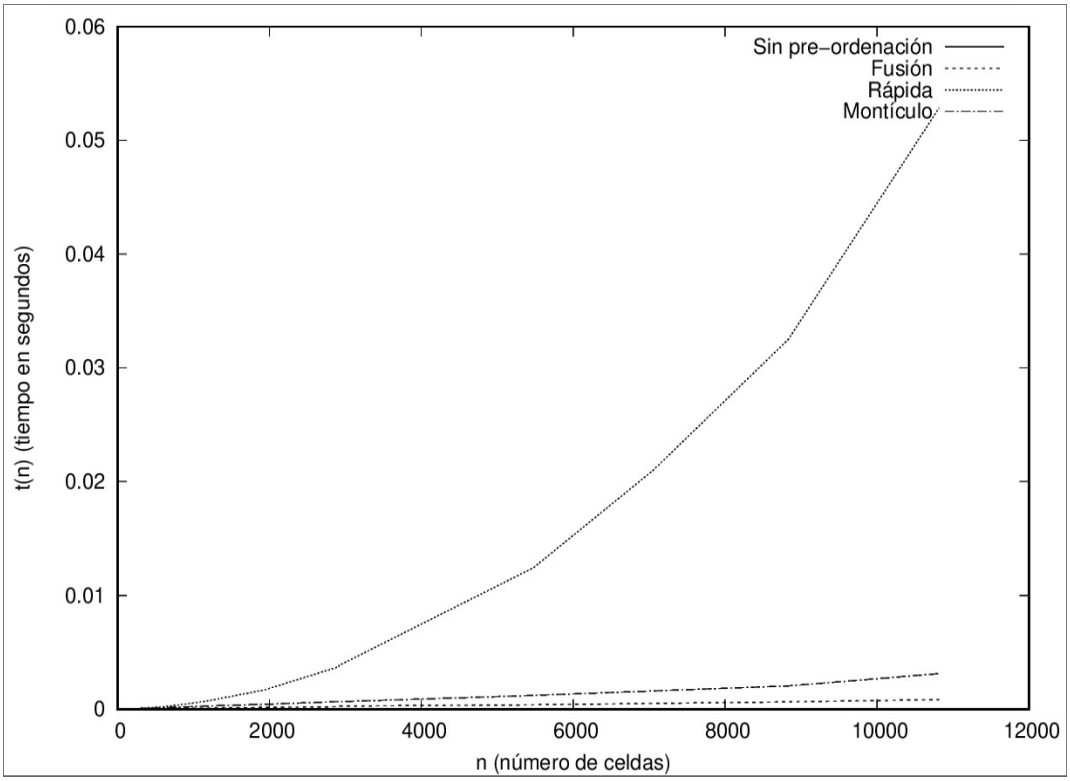
```
//TERCER METODO: ORDENACION RAPIDA.
    cronometro c3;
    long int r3 = 0;
    c3.activar();
    do{
        CellRapida(cell, 0, N-1);

       while(currentDefense != defenses.end()) {
            pos = voraz(freeCells,cell,nCellsWidth, nCellsHeight, mapWidth, mapHeight,
                obstacles, defenses, (*currentDefense)->radio,(*currentDefense)->id,N);
            (*currentDefense)->position.x = pos.x;
            (*currentDefense)->position.y = pos.y;
            (*currentDefense)->position.z = 0;
            ++currentDefense;
        }
        ++r3;
    }while(c3.tiempo() < e_abs / e_rel + e_abs);
    c3.parar();
```

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 Ordenación**, consta de dos "for" que recorren la función, lo que nos lleva a $\mathbf{O(n\hat{2})}$.
   - **Ordenación por fusión**, Dividimos el problema en dos iguales, esto está representado por "log n", y el número de pasos es "log(n+1)" en el peor de los casos, por otro lado para encontrar el punto medio de la matriz necesitamos un paso O(1), para las submatrices será de O(n), por lo que la complejidad final será de **O(n*log n)**.
   - **Ordenación rápida**, en el mejor caso, el pivote termina en el centro de la lista, en ese caso será de O(n*log n), en cambio, en el peor de los casos el pivote termina en un extremo de la lista, $O(n\hat{2})$, por lo que el promedio es: **O(n·log n)**.
   - **Ordenación por montículos**, El algoritmo es de orden n en ambas funciones, lo que resulta en **O(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). Considere en su análisis los planetas con códigos 1500, 2500, 3500,..., 10500. Incluya en el análisis los planetas que considere oportunos para mostrar información relevante.



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.