5.2. Algoritmos de Ordenación por mezcla

UNIVERSIDAD
DE GRANADA

Julio José Reyes Hurtado
Antonio García Castillo

Información hardware

Architecture: x86_64

CPU op-mode(s): 32-bit, 64-bit

Byte Order: Little Endian

Model name: Intel(R) Core(TM) i5-4258U CPU @ 2.40GHz

Mem RAM: 8 gb ram 1600 MHz DDR3

Stepping: 1 CPU MHz: 2400.000

L1d cache: 32K

L1i cache: 32K

L2 cache: 256K

L3 cache: 3072K NUMA node0

Pruebas realizadas en una maquina virtual de ubuntu soportado por Mac OS.

Opciones de compilación: g++ -std=c++11 ejecutable.cpp -o ejecutable

Comandos gnuplot.

Se ha utilizado GNUplot para poder realizar las pruebas empíricas en cada uno de los algoritmos. En una primera aproximación se ha realizado un visionado de la gráfica, con plot 'archivo.dat', y posteriormente hemos utilizados las funciones generales para sacar las constantes y contrastarlo. Así podremos comparar los datos de distintos umbrales del mismo algoritmo

```
inline static void insercion(int T[], int num_elem)
{
  insercion_lims(T, 0, num_elem);
}

static void insercion_lims(int T[], int inicial, int final)
{
  int i, j;
  int aux;
  for (i = inicial + 1; i < final; i++) {
    j = i;
    while ((T[j] < T[j-1]) && (j > 0)) {
      aux = T[j];
      T[j] = T[j-1];
      T[j-1] = aux;
      j--;
    };
  };
}
const int UMBRAL_MS = 5;

void mergesort(int T[], int num_elem)
{
  mergesort_lims(T, 0, num_elem);
}

static void mergesort_lims(int T[], int inicial, int final)
{
  if (final - inicial < UMBRAL_MS)
    {
      insercion_lims(T, inicial, final);
    } else {
      int k = (final - inicial)/2;

      int * U = new int [k - inicial + 1];
      assert(U);
      int l, l2;
```

```
    for (l = 0, l2 = inicial; l < k; l++, l2++)
        U[l] = T[l2];
    U[l] = INT_MAX;

    int * V = new int [final - k + 1];
    assert(V);
    for (l = 0, l2 = k; l < final - k; l++, l2++)
        V[l] = T[l2];
    V[l] = INT_MAX;

    mergesort_lims(U, 0, k);
    mergesort_lims(V, 0, final - k);
    fusion(T, inicial, final, U, V);
    delete [] U;
    delete [] V;
   };
}

static void fusion(int T[], int inicial, int final, int U[], int V[])
{
  int j = 0;
  int k = 0;
  for (int i = inicial; i < final; i++)
    {
     if (U[j] < V[k]) {
        T[i] = U[j];
        j++;
     } else{
        T[i] = V[k];
        k++;
     };
   };
}
int main(int argc, char * argv[])
{
  if (argc != 2)
    {
```

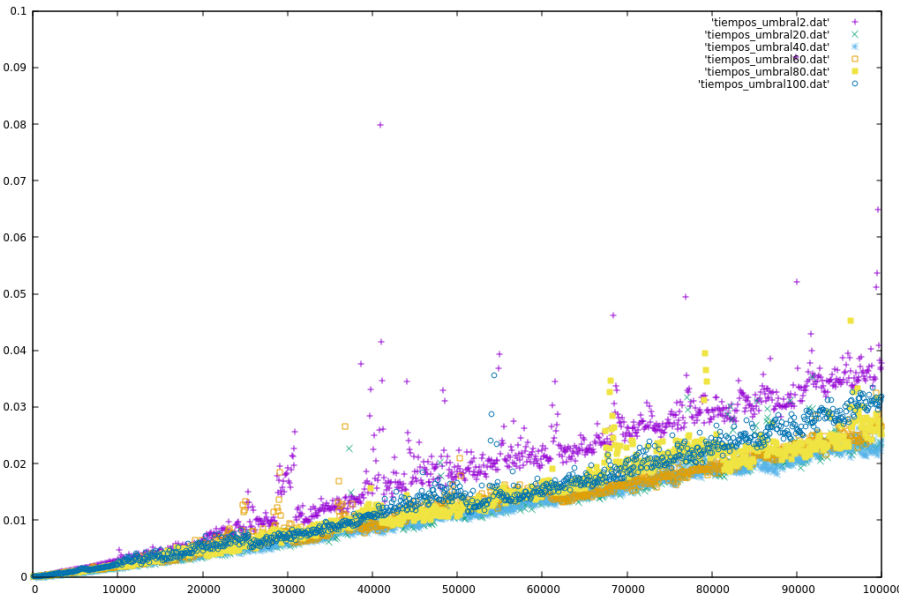
```
    cerr << "Formato " << argv[0] << " <num_elem>" << endl;
    return -1;
  }
 int n = atoi(argv[1]);
 int * T = new int[n];
 assert(T);
srandom(time(0));

 for (int i = 0; i < n; i++)
  {
    T[i] = random();
  };

 const int TAM_GRANDE = 10000;
 const int NUM_VECES = 1000;

 if (n > TAM_GRANDE)
  {
    clock_t t_antes = clock();
    mergesort(T, n);
    clock_t t_despues = clock();

    cout << n << "  " << ((double)(t_despues - t_antes)) / CLOCKS_PER_SEC
            << endl;
  } else {
    int * U = new int[n];
    assert(U);

    for (int i = 0; i < n; i++)
        U[i] = T[i];

    clock_t t_antes_vacio = clock();
    for (int veces = 0; veces < NUM_VECES; veces++)
        {
          for (int i = 0; i < n; i++)
            U[i] = T[i];
        }
```

```
    clock_t t_despues_vacio = clock();

    clock_t t_antes = clock();
    for (int veces = 0; veces < NUM_VECES; veces++)
        {
         for (int i = 0; i < n; i++)
           U[i] = T[i];
         mergesort(U, n);
        }
    clock_t t_despues = clock();
    cout << n << " \t  "
          << ((double) ((t_despues - t_antes) -
                        (t_despues_vacio - t_antes_vacio))) /
        (CLOCKS_PER_SEC * NUM_VECES)
          << endl;

    delete [] U;
   }
  delete [] T;
  return 0;
};
```

Merge sort

Se va a estudiar el algoritmo de mezcla comparando los tiempos de distintios umbrales y además con el algoritmo de inserción.

Esto se realiza a que el algoritmo por mezcla es mas eficiente cuando se introducen una gran cantidad de datos.



En esta gráfica podemos ver que para distintos tipos de umbrales los que tienen un crecimiento mas lento son cuando metemos los umbrales de 20 y de 40. Existen diferencias ya que para un umbral dado cuando el tamaño del vector dividido sea menor que el umbral se aplicará la insercción.

En la siguiente gráfica se va a comparar estos umbrales con el algoritmo de inserción en su caso peor y se puede ver como a pesar de tener distintos umbrales se puede seguir apreciando que el peor de merge sort, Tm(n)= n*log(n) no llega a acercarse al crecimiento tan elevado como es el de inserción, Tm(n) = n^2.