Montículo Binomial

Pablo Daurell Marina

MARP (Práctica 3)

1. Introducción

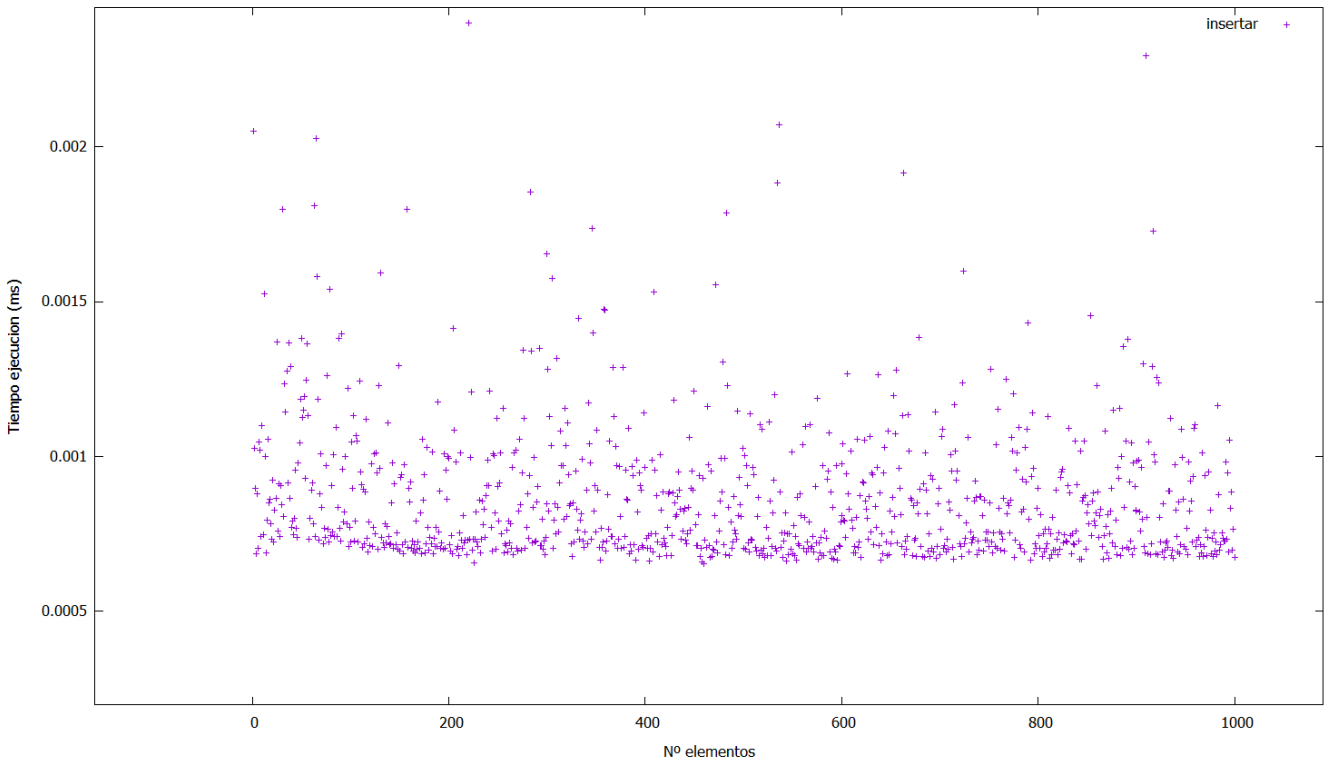
Para la realización de esta práctica se ha implementado un montículo binomial en C++ acompañado de las operaciones Insertar, Minimo, Borrar Minimo, Unir y Decrecer Clave.
Las implementaciones de estas operaciones están basadas en las diapositivas sobre montículos binomiales de la asignatura.

El monticulo está implementado con dos clases, la clase Nodo y la clase principal MonticuloBinomial en la cuál se implementan las operaciones principales del monticulo junto a varias operaciones complementarias.

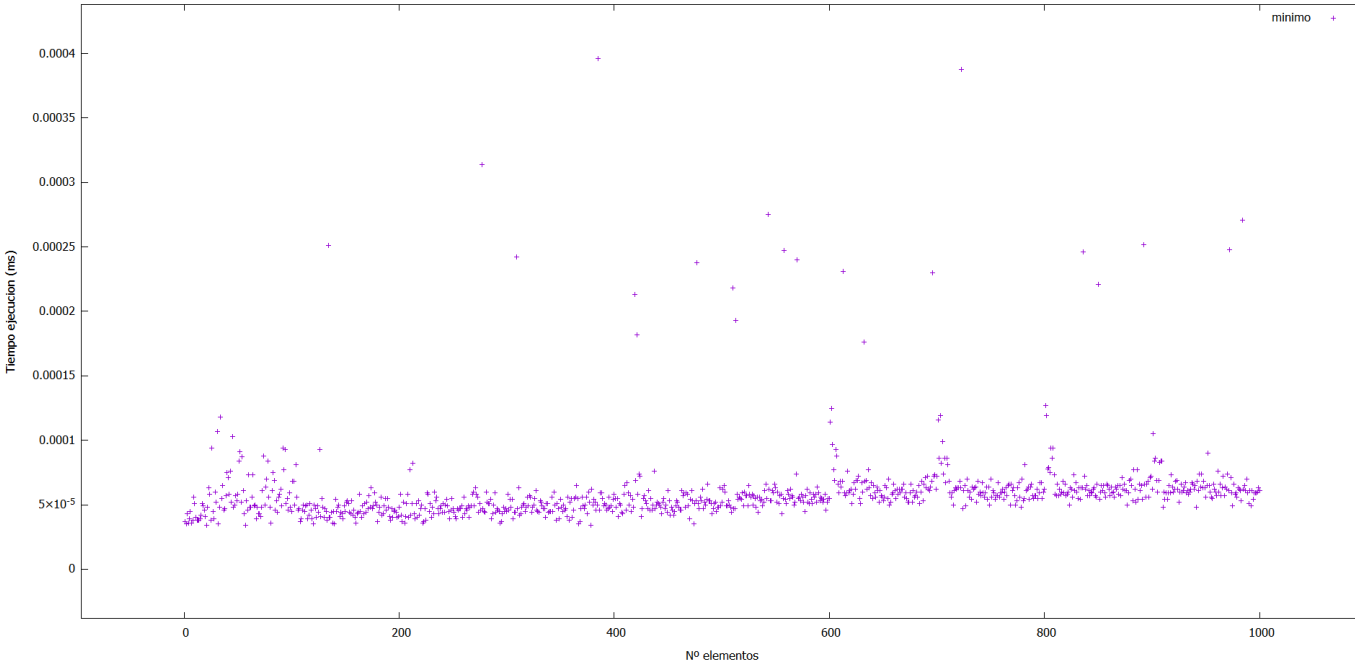
Todo el código (junto a pequeños comentarios sobre el funcionamiento de cada función) está incluido en el archivo "MonticuloBinomial.h"que acompaña a esta memoria.

2. Gráficas de tiempos de ejecución

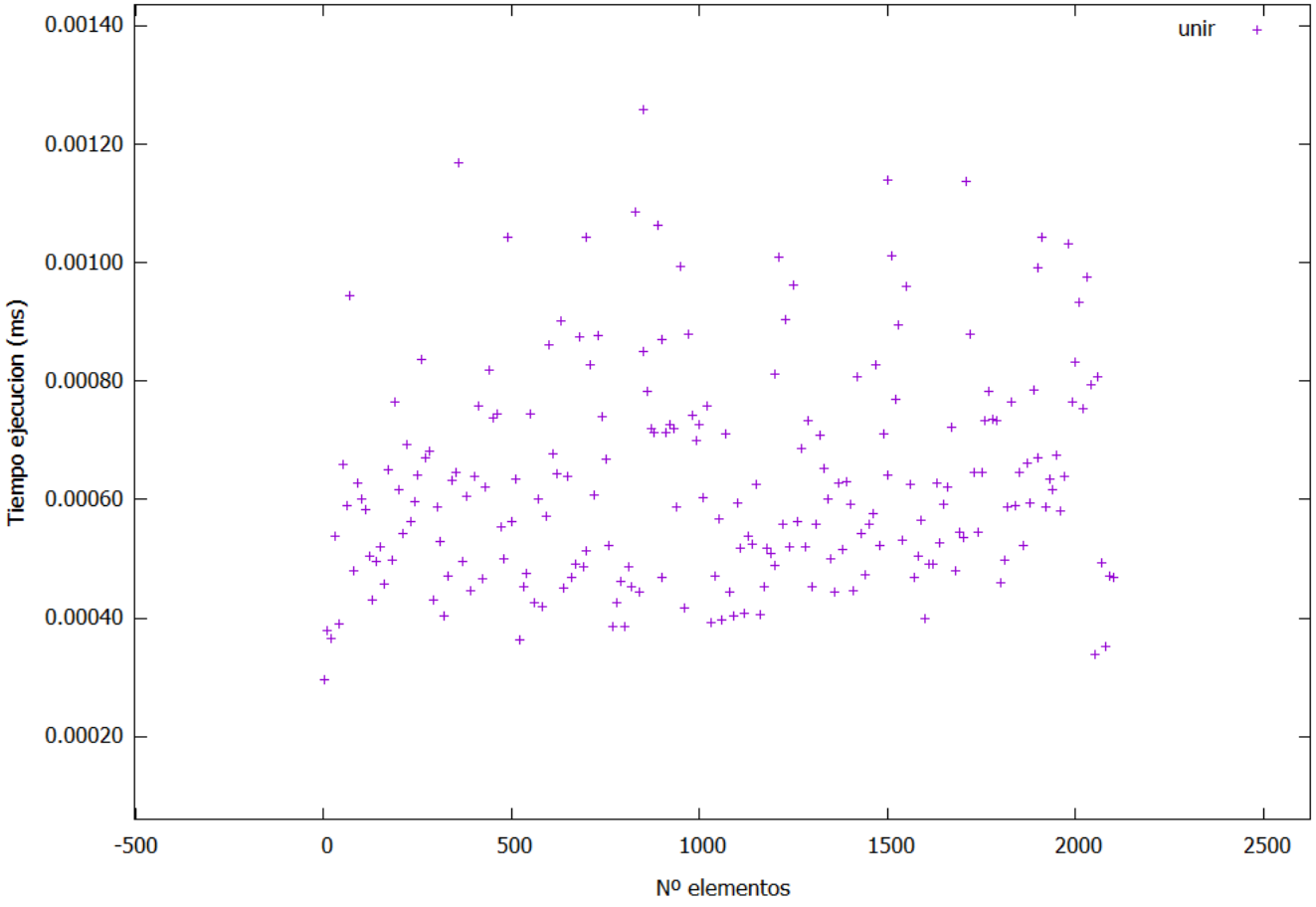
2.1. Insertar



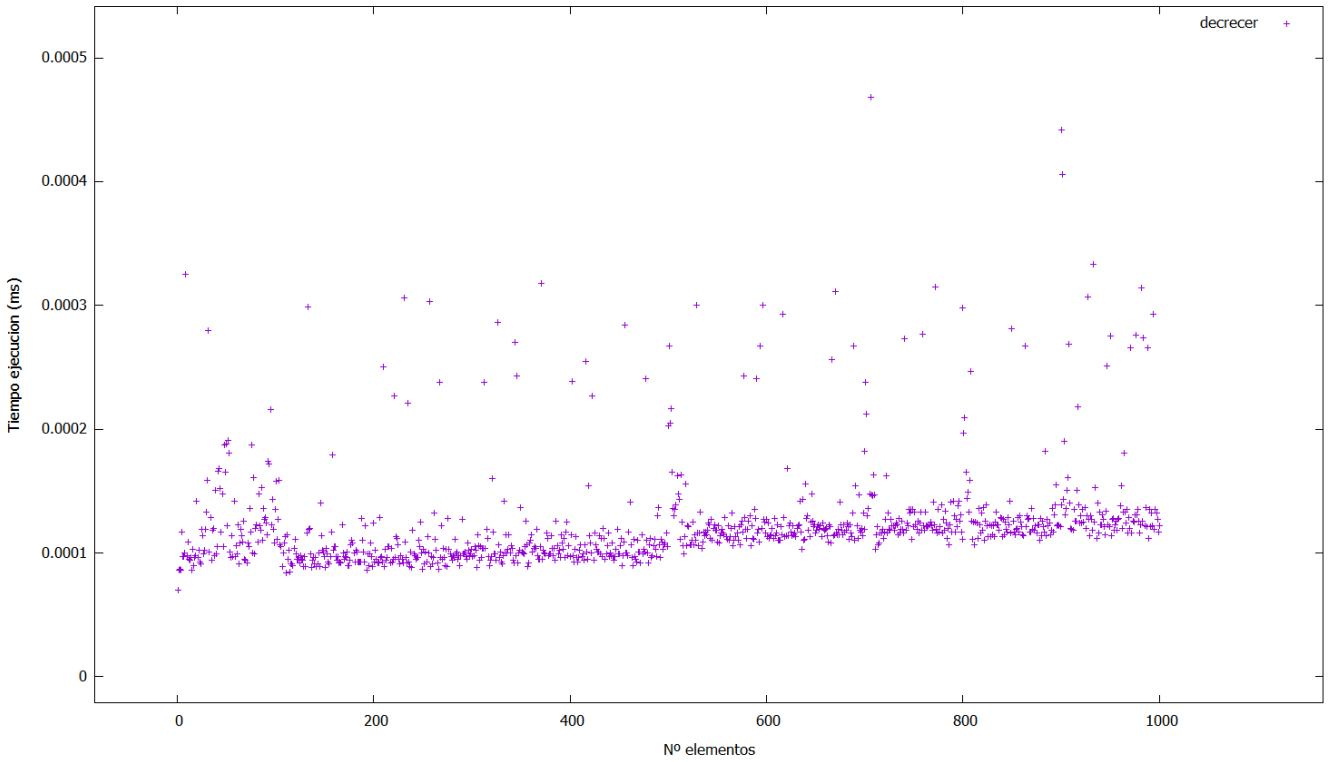
2.2. Mínimo



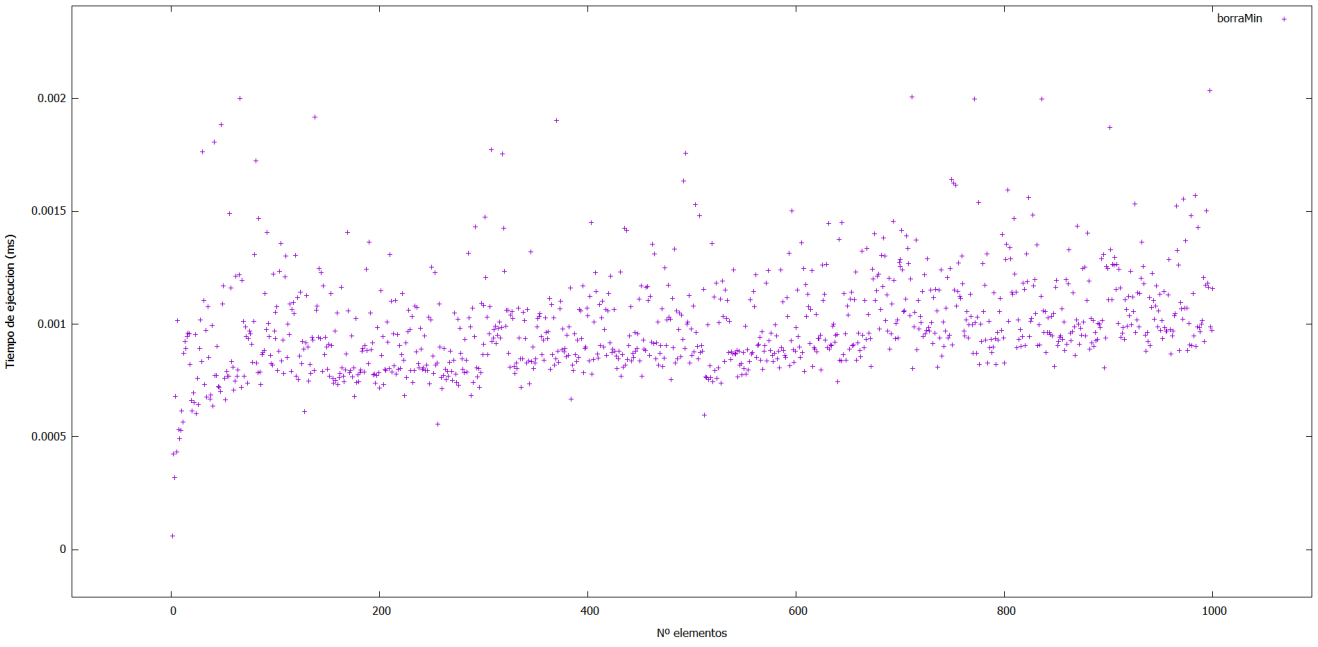
2.3. Unir



2.4. Decrecer Clave



2.5. Borrar Mínimo



3. Análisis de los costes

Los costes de las operaciones de las graficas anteriores son similares a los previstos por la teoría de los montículos binomiales.
Como podemos ver en las gráficas *insertar*, *unir*, *decrecer* y *borrarMínimo*, estas describen un crecimiento más o menos logarítmico (sobretodo borrar mínimo) que coincide con el coste $\mathcal{O}(logn)$ esperado. Estos costes se deben a que todas estas operaciones llaman en algún momento a *unir*, que como ya hemos

visto tiene coste logarítimco. Además en el borrado del mínimo vemos un crecimiento logarítmico algo más definido ya que, a parte de hacer uso de la unión, tiene que buscar el nuevo mínimo del monticulo recorriendo todas las raíces de los árboles binomiales que forman el montículo, lo cuál supone con coste $\mathcal{O}(logn)$ añadido.
Por otro lado, en la gráfica de la operación *mínimo* vemos una gráfica más o menos constante que coincide con el coste $\mathcal{O}(1)$ esperado. Este coste se consigue al incluir en la implementación del montículo un puntero al elemento mínimo de este.

4. Código utilizado para generar los datos de las gráficas:

C++

```
#include <Windows.h>
#include <stdlib.h>
#include <time.h>
#include <fstream>
#include "MonticuloBinomial.h"
using namespace std;

// Crea un monticulo binomial de n elementos coin valores aleatorios
MonticuloBinomial<int>* crearMonticulo(int n) {
        MonticuloBinomial<int>* m = new MonticuloBinomial<int>();

        for (int i = 0; i < n; i++) {
                int numero = rand() % 100;
                m->insertar(numero);
        }

        return m;
}

int main() {
        srand(time(NULL));

        ofstream archivo;
        string fichero = "tiempos.dat";
        archivo.open(fichero);  archivo << "#   X   Y" << endl;

        int elementos;
        int n = 100;

        for (elementos = 1; elementos <= 1000; elementos += 1) {
                double t_total = 0;

                for (int i = 0; i < n; i++) {
                        MonticuloBinomial<int>* m1 = crearMonticulo(elementos);

                        LARGE_INTEGER frecuencia;
                        LARGE_INTEGER ini;
                        LARGE_INTEGER fin;

```

```
40                          QueryPerformanceFrequency(&frecuencia);
41                          QueryPerformanceCounter(&ini);
42
43                          // Operacion a medir
44                          m1->borraMin();
45
46                          QueryPerformanceCounter(&fin);
47
48                          double tiempo = (double)(fin.QuadPart - ini.QuadPart)
49                                          / frecuencia.QuadPart * 1000;
50
51                          t_total += tiempo;
52
53                          delete m1;
54                  }
55
56                  cout << endl << "ELEMENTOS: " << elementos << ", TIEMPO MEDIO: "
57                      << t_total / n << endl;
58
59                  archivo << "  " << elementos << "   " << t_total / n << endl;
60          }
61          archivo.close();
62
63          return 0;
64  }
```