Montículo Binomial

Xukai Chen; MARP -A

En esta memoria se explica todo lo relacionado con el coste y el tiempo de ejecución del algoritmo, la explicación del código está detallado en ***Heapbin.h***.

El TAD está compuesto por siguientes operaciones:

- Join (otro Montículo Binomial): unión con otro montículo.
- Insert (Elem): inserta un elemento al montículo.
- DecreaseKey (Nodo, Elem): decrece el valor contenido en Nodo al Elem.
- DeleteMin (): borra el nodo que contiene el mínimo.
- Min (): devuelve el mínimo del montículo.

Según la teoría dada durante las clases deberían tener los siguientes costes:

Operación	Coste
Join	O(log(n))
Insert	O(1)
DereaseKey	O(log(n))
DeleteMin	O(log(n))
Min	O(1)

Se ha ejecutado un ejemplo sencillo para ver que efectivamente da resultados esperados, ***prueba.cpp***:

```
Dos monticulos inicializados con siguientes valores, heap1 { 40,10,20,30,25,35,15 } y heap2 { 1,2,3,4,5,6,7 }
El minimo de heap1 esperado: 10
10
Borrado del minimo, el nuevo minimo esperado: 15
15
Union de ambos monticulos, minimo esperado: 0
0
Borrado del minimo, minimo esperado: 1
1
Un puntero al valor 35 que pertenece a un arbol de grado 3
35
Decrece su valor al 0, ahora el valor del puntero pasa a ser de su padre 25      el nuevo minimo sera 0
25      0
Borrado del minimo, minimo sera 1
1
Presione una tecla para continuar . . .
```

Luego, para comprobar que el TAD realmente ofrece estos costes, se ha diseñado ***graficaCoste.cpp*** y ***graficaTiempo.cpp*** para un tamaño suficientemente grande. Se considera que tanto para DecreaseKey como para Min no son necesarias las comprobaciones. ya que la primera operación cuyo coste depende totalmente del donde esté situado el nodo, en el peor caso resulta la hoja del árbol binomial de máximo grado está en O(log(n)), además esta

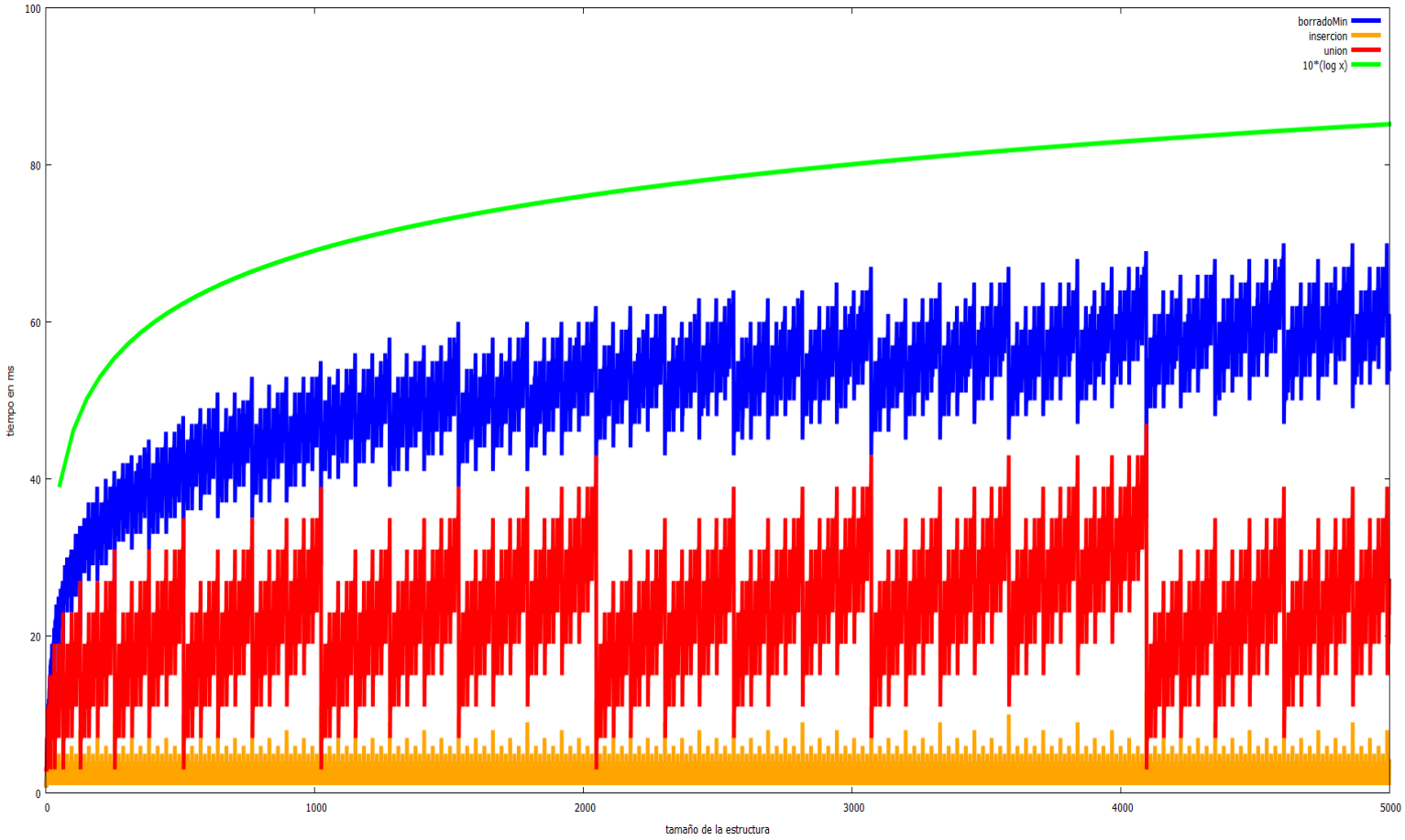
operación suele ser la auxiliar de DeleteMin porque para eliminar un nodo cualquiera (decrecer el valor al -infinito y borrar el mínimo). La operación Min su coste es de O(1) obviamente.

Las tres operaciones restantes resultan que todas miden por debajo de 10 ms, entonces se ha tenido que ejecutarlas varias veces de las siguientes maneras:

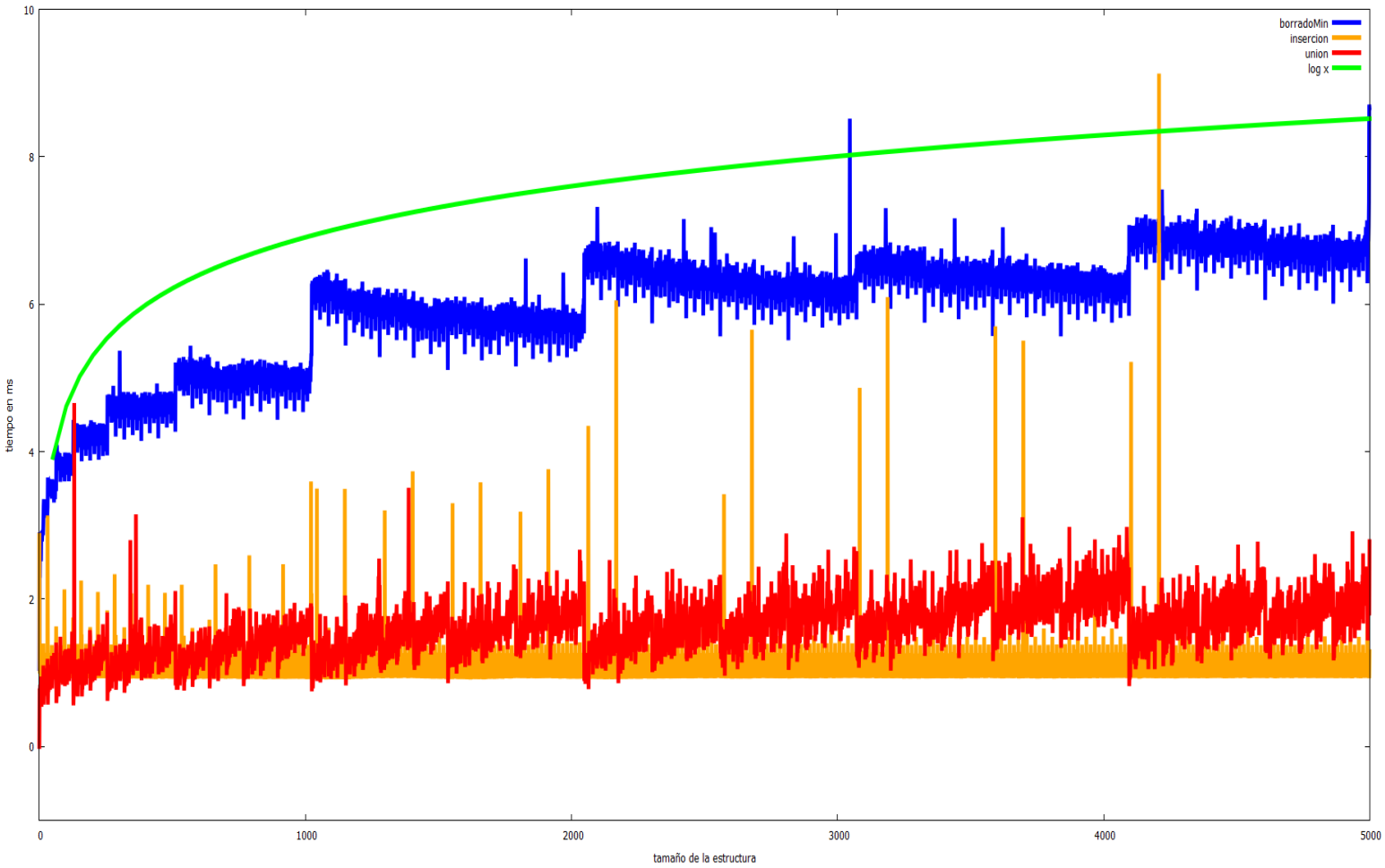
- Insert: repetir 1000 veces insertar 5000 números aleatorios entre 0-5000, y calcular la media. Supone un coste al recorrer un árbol.
- Join: repetir 10 veces, insertar y unir i números para i entre 0 y 5000, y calcular la media. Se calcula el coste como dos veces el número de árboles en montículo después de mergesort.
- DeleteMin: repetir 100 veces, insertar 5000 números aleatorios entre 0-5000 y borrar mínimo 5000 veces, calcular la media. El coste sería la suma de: la búsqueda del árbol anterior al árbol que contiene el mínimo, 2 veces el número de arboles colgados del nodo mínimo por la utilización de la pila y el coste de la unión con el montículo formado con esos arboles descolgados.

Los resultados obtenidos:

Grafica de coste



Grafica de tiempo



Se ve que borrado del minimo y union están en O(log(n)), mientras que inserción O(1).