Laboratorio 2: Heaps

30 de Septiembre, 2025

Student	Mirko Peñailillo Vasquez
Profesor	José Fuentes Sepúlveda
Ayudante	Luciano Argomedo
N^{ro} Matricula	2021900322

1 Resumen

El objetivo de este laboratorio es entender el codigo de los distintos tipos de heap vistos en clases (heap binario, heap binomial, Fibonacci heap) e implementar el Fibonacci heap desde cero. Para el caso de la implementacion de el heap binario y el heap binomial se utilizaron implementaciones obtenidas de fuentes externas, las cuales seran referenciadas en la seccion correspondiente.

2 Análisis Teórico

2.1 Heap Binario

El heap binario es un arbol binario casi completo (todos los niveles del arbol estan completos menos ,posiblemente, las hojas) con propiedad de heap, es decir, para todo subarbol, la raiz tiene mayor, menor (max heap o min heap) o igual valor que sus hijos.
Complejidades:

- insert: $O(log\ n)$. Se inserta el nuevo elemento y se aplica sift-up, recorriendo a lo mas la altura del arbol (dada por $log\ n$).
- extractMin: $O(log\ n)$. Se reemplaza la raiz (en min heap) y se aplica sift down hasta restaurar la propiedad de heap.
- Espacio: $O(n)$

2.2 Heap Binomial

El heap binomial es un bosque de arboles binomiales en el que existe maximo un arbol por orden.
Complejidades:

- insert: $O(1)$ amortizado. Se inserta un nuevo arbol de grado 0. Si existe otro del mismo grado, se realiza un merge entre arboles hasta que no exista colision de grados.
- extractMin: $O(log\ n)$ amortizado. Se elimina la raiz minima y añade sus hijos a la lista de raices. Luego realiza merge de arboles hasta consolidar el heap.
- Espacio: $O(n)$

2.3 Fibonacci Heap

El Fibonacci heap es similar a el heap binomial, sin embargo, a diferencia de este tiene una mayor relajacion respecto a el "balance" de sus arboles. Ademas, el Fibonacci heap pospone la consolidacion de sus arboles para obtener mejores tiempos amortizados.
Complejidades:

- insert: $O(1)$. Se inserta un nuevo nodo a la lista de raices.
- extractMin: $O(log\ n)$ amortizado. Se elimina el nodo minimo, se añaden sus hijos a la lista de raices y se realiza la consolidacion de arboles.
- Espacio: $O(n)$

3 Implementación

Para el estudio de tanto el heap binario como el heap binomial, se utilizaron implementaciones encontradas en internet. Particularmente se utilizo [1] para el heap binario y [2] para el heap binomial. Por otro lado, la implementacion de el Fibonacci Heap se realizo desde cero siguiendo la explicacion encontrada en [3].

4 Resultados Experimentales

4.1 Decisiones de inicializacion

- Para el analisis experimental de los metodos insert() y extractMin() de cada tipo de heap, se realizaron los siguientes experimentos. Se utilizo un tamaño de heap n variable (desde 1000 hasta 100000 con steps de 1000).
  - El experimento 1, correspondiente a el metodo insert(), consiste en crear un heap de cada tipo de tamaño n, llenarlo de valores aleatorios y medir el tiempo que demora insertar un nuevo elemento.
  - El experimento 2, correspondiente a el metodo extractMin(), consiste en crear un heap de cada tipo de tamaño n, llenarlo de valores aleatorios y medir el tiempo que demora extraer el valor minimo.
- Para cada experimento realizado se utilizo $RUNS = 32$ en el programa "uhr.cpp" disponible en el repositorio del curso [4].

4.2 Uso de datasets externos

No se utilizaron datasets externos para la experimentacion.

4.3 Caracteristicas de la maquina donde se realizaron los experimentos

- Procesador: AMD Ryzen 5 3600 (3.6 GHz)
- Memoria RAM: 16GB DDR4 (3600 MHz)

4.4 Descripcion de los resultados mediante tablas o graficos

- En cada figura podemos observar los graficos correspondientes a los resultados obtenidos en cada experimento. Las figuras 1 a 3 corresponden a el experimento 1 mientras que las figuras 4 a 6 corresponden a el experimento 2.
- Notar que en cada figura aun se puede observar un ligero ruido por efecto de cache/OS, pero se pueden observar las tendencias de las curvas.
- Graficos
  Podemos observar en la figura 1 un leve crecimiento de la curva, coherente con la complejidad teorica de $O(log\ n)$ del algoritmo en el heap binario.

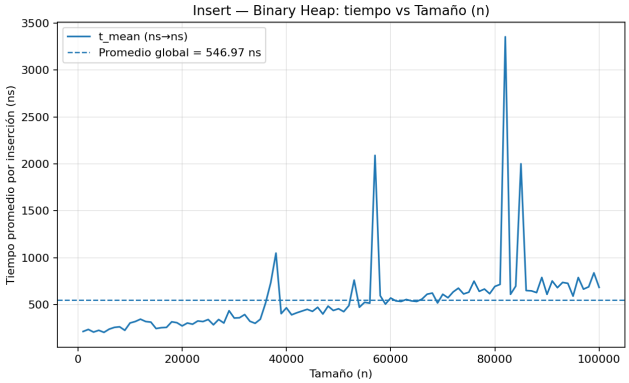

Figure 1: Insert heap binario

En la figura 2 podemos observar un sobre costo evidente respecto a la figura 1. Esto se puede explicar porque, aunque la complejidad amortizada de insert en un binomial heap es $O(1)$, el merge de arboles pueden elevar el costo temporal.

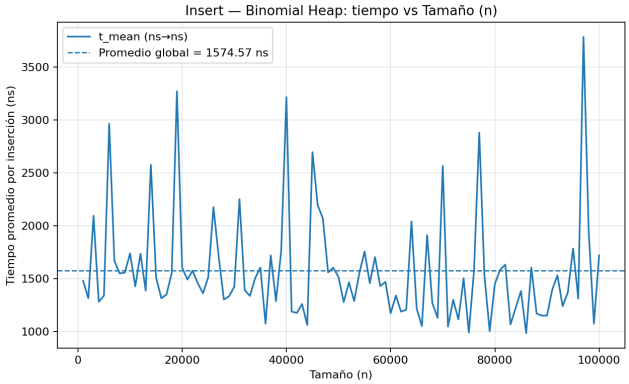

Figure 2: Insert heap binomial

En la figura 3 podemos observar que la curva corresponde a una complejidad temporal de $O(1)$.

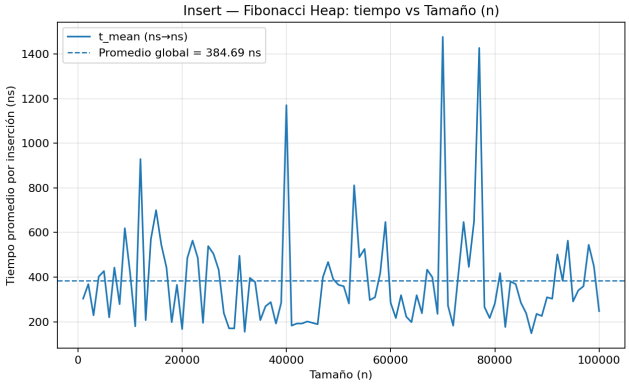

Figure 3: insert Fibonacci heap

En la figura 4 se aprecia un leve crecimiento de la curva, consistente con una complejidad temporal de $O(log\ n)$.

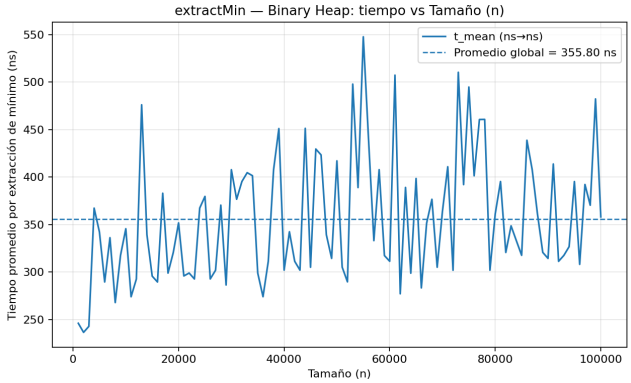

Figure 4: extractMin heap binario

La curva de la figura 5 tambien presenta una tendencia a un comportamiento logaritmico. Tambien presenta un claro sobre costo en relacion a la figura 4, lo cual es consistente con la necesidad de el heap binomial de "balancear" los arboles que contiene.

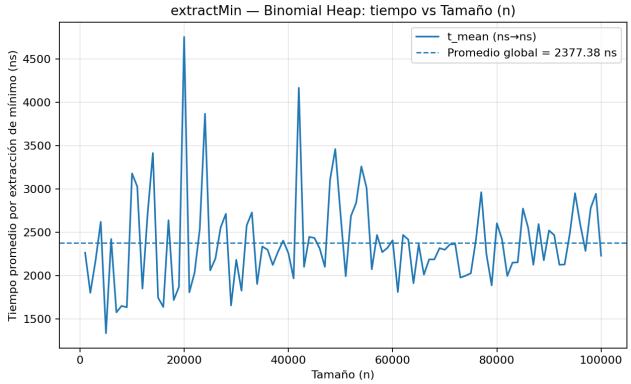

Figure 5: extractMin heap binomial

Finalmente en la figura 6 podemos observar que la curva parace describir una complejidad temporal $O(n)$, lo que contradice el analisis teorico de extractMin en un Fibonacci heap. La explicacion a este comportamiento se puede encontrar en la implementacion del fibonacci heap y el como fue realizado el experimento 2. Notemos que el Fibonacci tree acumula muchas raices sin consolidar, lo que causa que al utilizar extractMin() una sola vez este deba recorrer la lista de raices que es un conjunto proporcional a n.

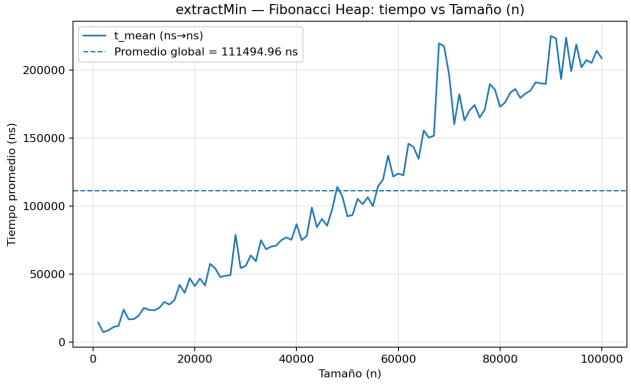

Figure 6: extractMin Fibonacci heap

5 Conclusiones

5.1 Insert

- En las inserciones, el heap binario resulto mas rapido que el binomial (por aproximadamente 3 veces en promedio global), incluso cuando ambos tienen complejidad teorica de $O(log\ n)$. Mientras que el fibonacci heap obtuvo las inserciones mas rapidas.

5.2 ExtractMin

- Tanto para el heap binario como el heap binomial, los resultados experimentales mostraron un comportamiento logaritmico, sin embargo, el heap binario obtuvo menores tiempos de ejecucion en promedio (por aproximadamente 6 veces). Mientras tanto, el fibonacci heap experimentalmente se comporto como $O(n)$, lo que contradice la teoria. Este comportamiento ya fue analizado previamente en este informe, en la seccion Resultados Experimentales.

5.3 Decisiones de uso

- En base a los resultados experimentales, el heap binario es la mejor opcion practica, debido a su simplicidad y resultados. Los resultados de el heap binomial no mostro ventajas practicas en ninguno de los metodos estudiados en este laboratorio.
- El Fibonacci heap obtiene mejores resultados en la operacion de insert respecto a el heap binario, sin embargo, podemos asegurar que para problemas con pocas operaciones de extractMin, el heap binario sigue siendo mejor opcion. Para continuar el estudio de el Fibonacci heap, se recomienda realizar un experimento 3 donde se realizen multiples llamados a extractMin, para corroborar su complejidad temporal de $O(log\ n)$ amortizado.

6 Referencias

References

[1] GeeksforGeeks. "C++ program to implement binary heap." Accessed: 2025-09-29. [Online]. Available: `https://www.geeksforgeeks.org/cpp/cpp-program-to-implement-binary-heap/`.

[2] GeeksforGeeks, *C++ Program to Implement Binomial Heap*, `https://www.geeksforgeeks.org/cpp/cpp-program-to-implement-binomial-heap/`, Accedido: 28 septiembre 2025.

[3] GeeksforGeeks, *Fibonacci Heap | Set 1 (Introduction)*, `https://www.geeksforgeeks.org/dsa/fibonacci-heap-set-1-introduction/`, Accedido: 29 septiembre 2025.

[4] jfuentess. "Edaa." Repositorio en GitHub, Accessed: Sep. 28, 2025. [Online]. Available: `https://github.com/jfuentess/edaa`.