Boletín 2:
Heaps

Nombre del estudiante: Dazhi Enrique Feng Zong
Número de matrícula: 2019435710

Introducción

En este segundo boletín, se tiene como objetivo entender y comentar el código de un binomial heap, entender e implementar un binary heap con los algoritmos ya implementados en C++, y finalmente evaluar experimentalmente ambos algoritmos.

El binomial heap es una estructura de datos en forma de árbol que se utiliza para la implementación eficiente de colas de prioridad. Se caracteriza por ser un tipo de heap que sigue la estructura de un árbol binomial, que es un tipo particular de árbol con ciertas propiedades. El binomial heap se compone de varios árboles binomiales interconectados, y cada árbol binomial cumple con una propiedad llamada "propiedad binomial", la cual contribuye a mantener la eficiencia de las operaciones en la cola de prioridad, como inserción, extracción del mínimo y unión de heaps.

Por otro lado, el binary heap es una estructura de datos también utilizada para implementar colas de prioridad, pero se diferencia del binomial heap en su estructura y propiedades. Un binary heap es un árbol binario completo que cumple con la propiedad del heap, donde para cada nodo en el árbol, el valor almacenado en el nodo es menor o igual que los valores almacenados en sus hijos.

Ambos heaps pueden ser configurados como min heaps o max heaps. En el caso de un min heap, se tiene la operación para buscar y eliminar el elemento mínimo, mientras que en un max heap, se realiza la operación correspondiente para buscar y eliminar el elemento máximo.

Desarrollo

Análisis Teórico

En esta sección, se realiza un análisis teórico de dos tipos de heaps: Binary Heap y el Binomial Heap. Se evalúa sus tiempos de inserción, eliminación (del mínimo o máximo dependiendo del tipo de heap) y unión.

Binary Heap

- **Inserción:** La complejidad de la inserción en un Binary Heap es $O(\log n)$. Esto se debe a que, en el peor caso, se inserta un elemento menor que el mínimo (en un min heap) o mayor que el máximo (en un max heap), lo que requiere recorrer la altura del árbol desde la base hasta la raíz.
- **Eliminación:** La complejidad de la eliminación también es $O(\log n)$. Esto se debe a que se busca el mínimo/máximo (una operación constante), se elimina y luego se debe reorganizar el heap.
- **Unión:** La complejidad de la unión es $O(n)$. Esto se debe a que implica la concatenación (tiempo lineal) y luego la reorganización del heap, que es logarítmica.

Binomial Heap

- **Inserción:** En un Binomial Heap, la complejidad inicial de inserción es $O(\log n)$ debido a la fusión (merge) que se realiza al insertar un nuevo elemento. Sin embargo, a medida que el heap crece, el tiempo de inserción se vuelve constante, por lo que la complejidad amortizada es $O(1)$.
- **Eliminación:** La eliminación en un Binomial Heap tiene una complejidad logarítmica, es decir, $O(\log n)$. Esto se debe a que solo se recorren las raíces de los árboles, lo que toma tiempo logarítmico, y luego se reorganiza el árbol al quitar el elemento mínimo/máximo, lo cual también toma $O(\log n)$.
- **Unión:** La complejidad de unión en un Binomial Heap es $O(\log n)$, ya que esta operación implica fusionar dos heaps binomiales manteniendo la propiedad binomial y la estructura de heap.

Análisis Experimental

Se llevó a cabo un análisis experimental para investigar el impacto del tamaño de la secuencia en los tiempos de ejecución de ambos tipos de heaps. Se varió el tamaño de la secuencia, comenzando con 100 elementos y multiplicándolo por 10 en cada iteración hasta alcanzar 100.000.000 elementos. En cada iteración, se insertaban elementos al heap en rangos incrementales (por ejemplo, de 0 a 100, luego de 0 a 1000 en la siguiente iteración).

Para la prueba de inserción, se insertó el número -1 en el min-heap y n (tamaño del heap) en el max-heap. En la prueba de remover, se eliminó ese mismo número. En la prueba de unión, se unió el heap consigo mismo.

Cada experimento se repitió 40 veces. Las primeras 10 repeticiones se consideraron "cold runs" y no se incluyeron en el análisis final. Las 30 repeticiones restantes fueron consideradas en el análisis.

Las pruebas se realizaron utilizando el servidor *chome* de la Universidad de Concepción para garantizar un entorno de ejecución estable y consistente.

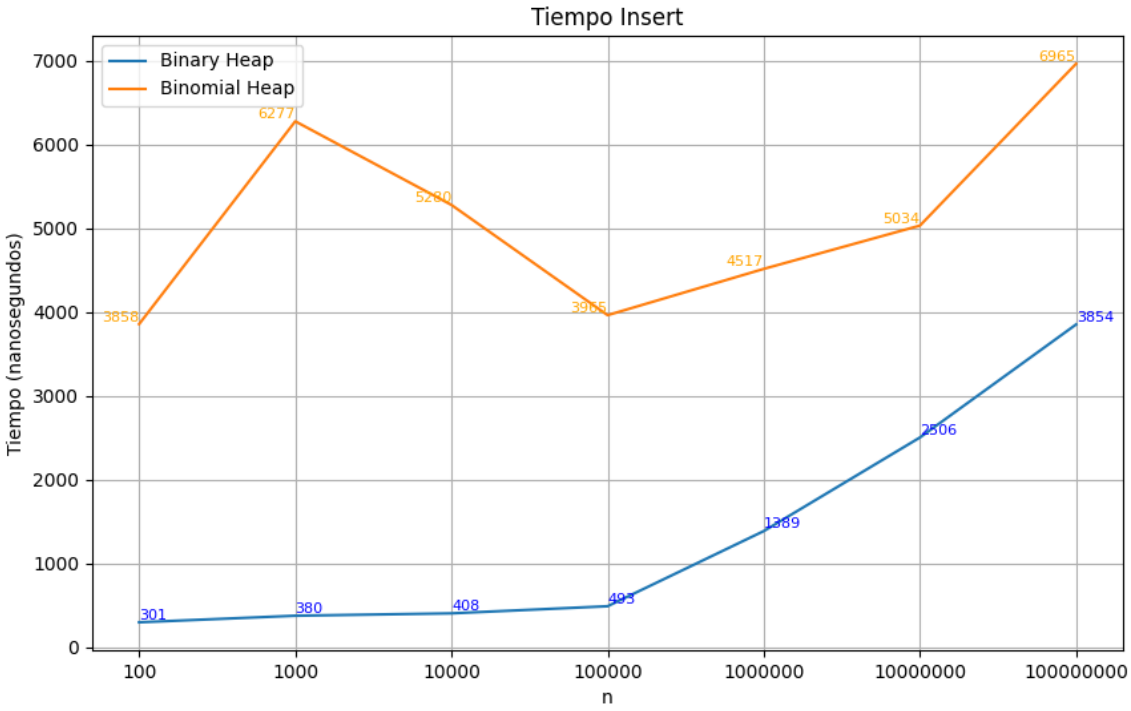
Hipótesis

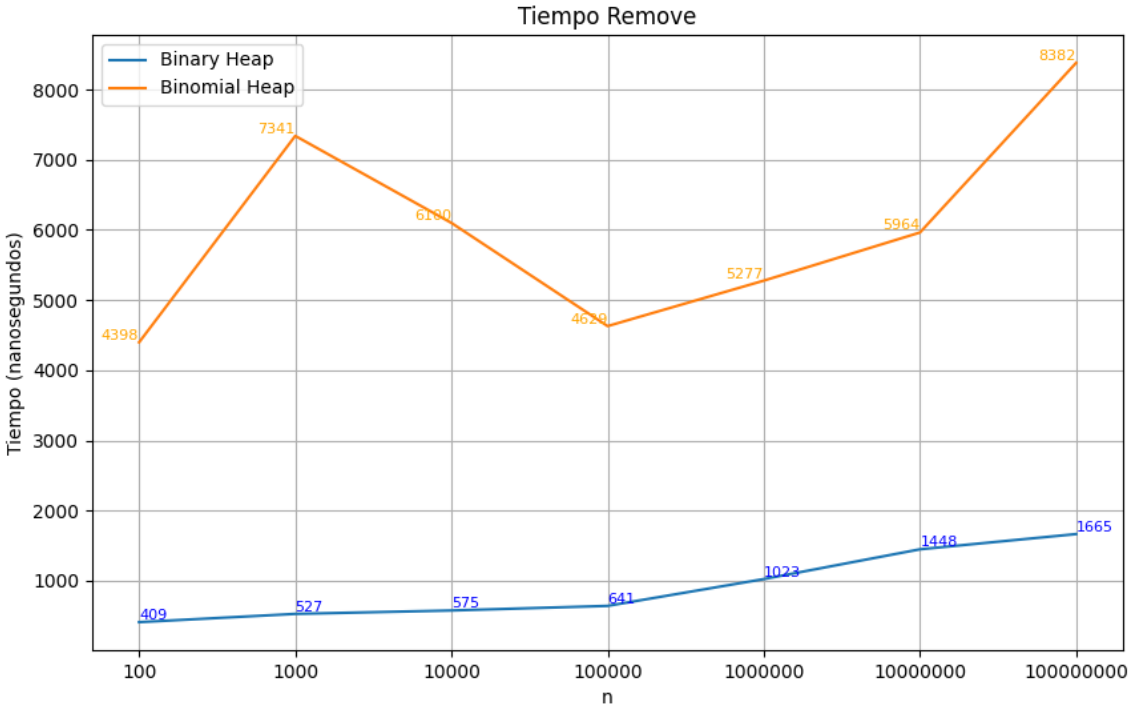
1. Se plantea que la inserción en el binomial heap debería ser más rápida que en el binary heap, dado que el binomial heap tiene una complejidad amortizada constante, mientras que el binary heap siempre es logarítmico.

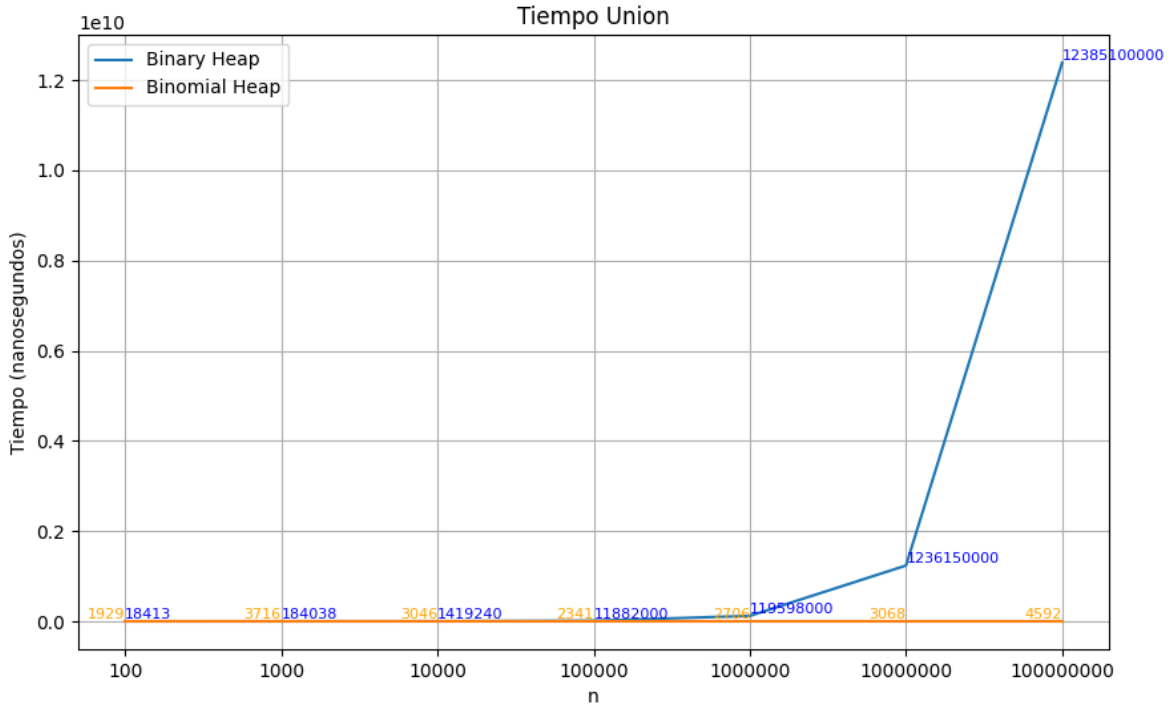
2. Para la operación de remover el mínimo/máximo, se espera que ambos heaps tengan tiempos de ejecución más o menos iguales, en un orden logarítmico.

3. Basándose en el análisis teórico que indica que la complejidad del binomial heap es menor ($O(\log n)$) en comparación con el binary heap ($O(n)$), se espera que el binomial heap debería demostrar mayor eficiencia en las pruebas realizadas.

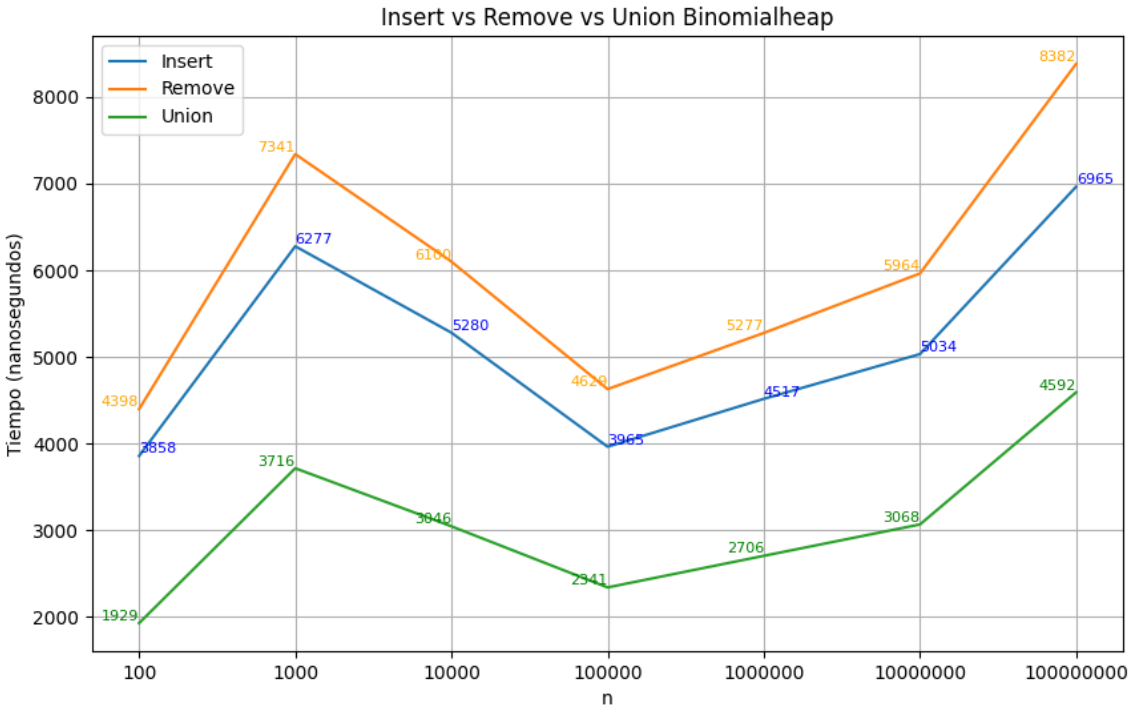
Tiempo de cada operación: Binary heap vs Binomial heap






Tiempo Union
1e10
Binary Heap
Binomial Heap
Tiempo (nanosegundos)
n
1.2
1.0
0.8
0.6
0.4
0.2
0.0
100
1000
10000
100000
1000000
10000000
100000000
12385100000
1236150000
118598000
11882000
1419240
184038
18413
4592
3068
2706
2341
3046
3716
1929

Tiempo de las operaciones de un heap



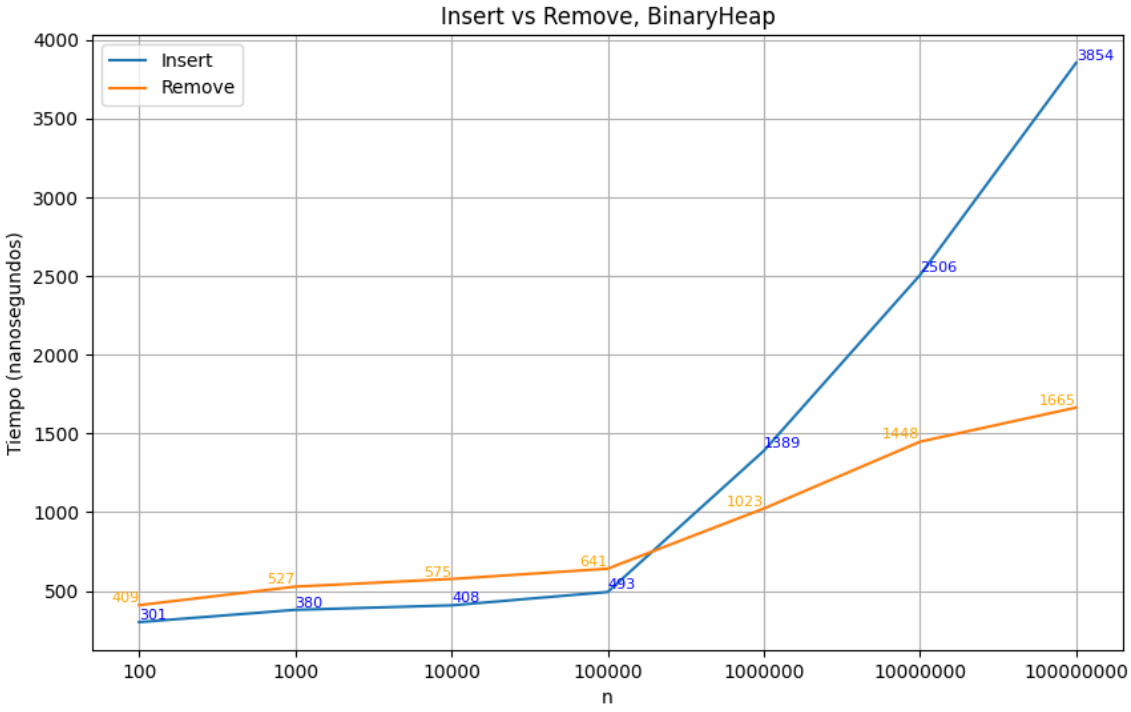


Tabla de los resultados, donde n representa el número de elementos, t representa el tiempo, y v representa la varianza

n	tInsert	vInsert	tRemove	vRemove	tUnion	vUnion
100	3858.03	5923.7	4398.03	2302.5	1929.43	1763.91
1000	6277.43	5181.51	7340.9	5276.89	3716.17	1733.21
10000	5280.07	5539.2	6100.13	8908.85	3046.43	3894.38
100000	3964.83	3933.81	4629.27	3436.93	2341.43	1523.05
1000000	4517.07	32568.3	5277.33	3189.82	2705.87	3385.78
10000000	5033.67	7768.82	5963.63	2973.17	3068.07	1584.93
100000000	6964.8	4884.16	8382.17	6196.61	4592.33	2866.76

Cuadro 1: Resultados de los tiempos de ejecución en nanosegundos para cada operación en un **Binomial heap**. Var indica la varianza.

n	tInsert	vInsert	tRemove	vRemove	tUnion	vUnion
100	301.267	1785.26	409.267	956.996	18413.2	23002.7
1000	379.5	775.183	526.867	1810.05	184038	2.24975e+06
10000	407.9	10701.6	575.467	13147.8	1.41924e+06	8.80553e+10
100000	492.8	2092.49	640.833	4577.54	1.1882e+07	8.93039e+09
1000000	1389.47	393055	1023.1	33061.1	1.19598e+08	7.08908e+11
10000000	2506.03	2.53236e+06	1447.63	36704.6	1.23615e+09	1.95634e+13
100000000	3854.5	4.45476e+06	1664.67	43634.6	1.23851e+10	2.25525e+14

Cuadro 2: Resultados de los tiempos de ejecución en nanosegundos para cada operación en un **Binary heap**. Var indica la varianza.

Discusión y Conclusión

De los resultados obtenidos, se observa que se cumplen 2 de los 3 puntos de la hipótesis. En cuanto a la inserción, el análisis experimental muestra que el binomial heap logra un tiempo logarítmico menor que el tiempo de inserción del binary heap. El binary heap tiene un tiempo irregular debido a la precisión en nanosegundos utilizada en el análisis, pero en una escala diferente, podría mostrar un comportamiento constante o logarítmico. Sin embargo, la diferencia es mínima debido al uso de nanosegundos. Otra consideración importante es que, en teoría, el tiempo puede ser constante, pero esto puede ocultar varios factores que influyen en este tiempo constante. Además, es relevante considerar que este tiempo es amortizado, por lo que hay ocasiones en que puede tener un tiempo mayor.

Respecto al tiempo de eliminación, ambos logran un tiempo más o menos logarítmico, con el binary heap teniendo un tiempo ligeramente menor, aunque la diferencia es pequeña debido a la escala de nanosegundos.

Finalmente, se confirma el análisis teórico sobre la operación de unión en ambos heaps, donde efectivamente el binary heap muestra un tiempo lineal en comparación con el tiempo logarítmico del binomial heap.

Es importante destacar que las diferencias en los tiempos de inserción y eliminación no son muy grandes debido a la escala de nanosegundos utilizada, lo que indica que tienen un rendimiento muy similar. Además, es relevante mencionar que el binary heap utiliza funciones de C++ que ya están optimizadas, a diferencia del binomial heap que fue implementado desde cero.

En resumen, ambos heaps tienen un desempeño muy similar en cuanto a tiempo de inserción y eliminación. Sin embargo, si se requiere la operación de unión, es mucho mejor utilizar un binomial heap debido al tiempo logarítmico que este presenta. También es importante destacar que si se realizan pocas inserciones, entonces el binomial heap puede no lograr un tiempo amortizado, por lo que en ese caso, la mejor opción sería el binary heap.

La elección entre estos dos heaps depende de varios factores y no hay uno que sea mejor que el otro en todos los casos. Además, es crucial considerar la complejidad de la implementación de cada heap y las funciones ya implementadas en el lenguaje de programación que pueden ayudar en la implementación del heap y proporcionar más optimizaciones.