Análisis de Algoritmos y Matemáticas Discretas
Tarea 2

Antonio Barragán Romero

Problema 1

En las sesiones de clases, se explicó que cuando en el vector de la `stl` *se hace un* `push_back`*, y no hay memoria suficiente para albergar otro elemento, se pide memoria adicional, y se procede a copiar todos los elementos que ya se tenían en el vector a las nuevas posiciones de memoria. Supongamos que inicialmente tenemos un vector vacío, con espacio reservado para un elemento, y que se van a realizar $n = 2^p$ inserciones.*

1. *Determine el número de escrituras en memoria que se realizarán si cada vez que el vector se queda sin memoria, se pide memoria para un espacio adicional. Especifique el número de escrituras ($e(n)$) de forma exacta, grafíquelo, y encuentre una función g(n) lo más simple posible, de forma que $e(n) = \theta(g(n))$.*
2. *Determine el número de escrituras en memoria que se realizarán si cada vez que el vector se queda sin memoria, se duplica la cantidad de espacio reservado. Especifique el número de escrituras ($e(n)$) de forma exacta, grafíquelo, y encuentre una función g(n) lo más simple posible, de forma que $e(n) = \theta(g(n))$*
3. *Determine el número de escrituras en memoria que se realizarán si cada vez que el vector se queda sin memoria, se multiplica por 4 la cantidad de espacio. Especifique el número de escrituras ($e(n)$) de forma exacta, grafíquelo, y encuentre una función g(n) lo más simple posible, de forma que $e(n) = \theta(g(n))$.*

Solución: Lo primero que hay que tener en cuenta es que el numero de escrituras en memoria es igual al numero de inserciones mas el numero de copias. Además cada vez que el vector se queda sin memoria se reserva memoria y se hacen las copias necesarias, por lo cual es necesario saber cuando se queda sin memoria el vector.

1. En este caso como solo se reserva memoria para un elemento extra entonces cada vez que se inserta un elemento el vector se queda sin memoria. Lo anterior implica que cada vez que se inserta un elemento se hace la copia de los elementos anteriores, por lo cual tenemos que para insertar $n+1$ elementos el numero de escrituras en memoria es igual al numero escrituras en memoria de los n anteriores ($e(n)$), más la copia de los primero n elementos del vector, mas la escritura del nuevo valor, lo anterior lo podemos ver como

$$e(n+1) = e(n) + n + 1,$$

además $e(1) = 1$. De lo anterior podemos notar que

$$e(n) = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \frac{n^2+n}{2} = \frac{n^2}{2} + \frac{n}{2}.$$

Primero veamos que $e(n) = \Theta(n^2)$, para ello notemos que

$$\frac{1}{2} \cdot n^2 \leq \frac{n^2}{2} + \frac{n}{2} \leq n^2,$$

para todo $n \in \mathbb{N}$, por lo cual tomamos $c_1 = \frac{1}{2}$, $c_2 = 1$ y $n_0 = 1$. Por ultimo veamos la gráfica de $e(n)$ y n^2:

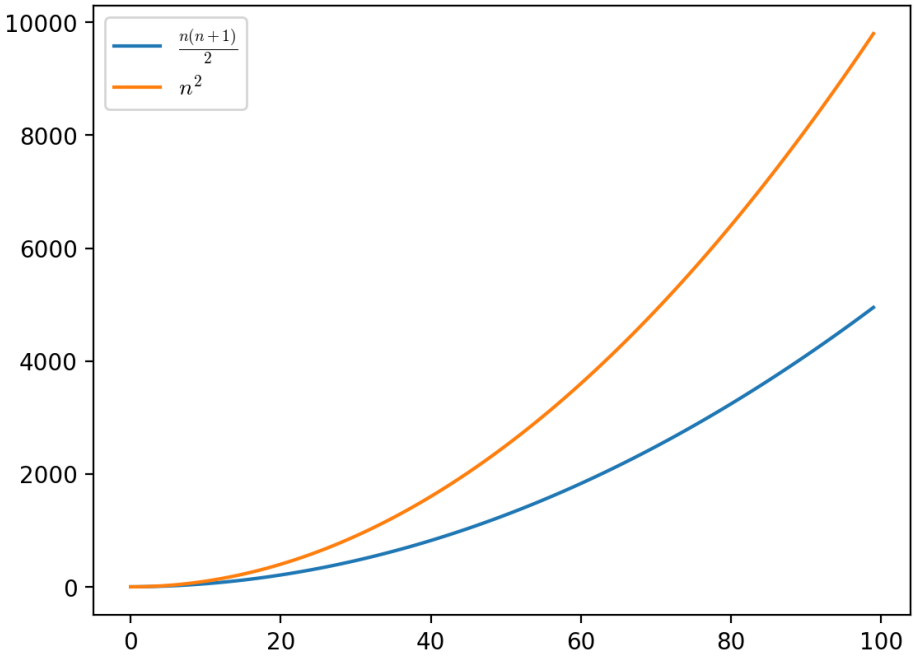


Figure 1: $e(n) = \Theta(n^2)$

2. En este caso, como inicialmente tenemos memoria para un elemento (potencia de dos) cuando el vector se llene, al duplicar su capacidad también seguirá siendo una potencia de dos por lo cual el vector se queda sin memoria justo en las potencias de 2, más aún esto nos dice que cuando se insertan 2^k elementos, la memoria se reservo al insertar 2^{k-1} elementos, después se hizo la copia de los primeros números 2^{k-1} y al final se insertaron los 2^{k-1} números siguientes. De lo anterior tenemos que

$$e(2^k) = e(2^{k-1}) + 2^{k-1} + 2^{k-1} = e(2^{k-1}) + 2^k,$$

con $e(1) = 1$. Luego

$$e(n) = e(2^p) = \sum_{k=0}^{p} 2^k = 2^{p+1} - 1 = 2 \cdot 2^p - 1 = 2n - 1,$$

se sigue entonces que $e(n) = \Theta(n)$, para ello notemos que

$$n \leq 2n - 1 \leq 2n.,$$

para todo $n \geq 2$, por lo cual tomamos $c_1 = 1$, $c_2 = 2$ y $n_0 = 2$. Por ultimo veamos la gráfica de $e(n)$ y n:

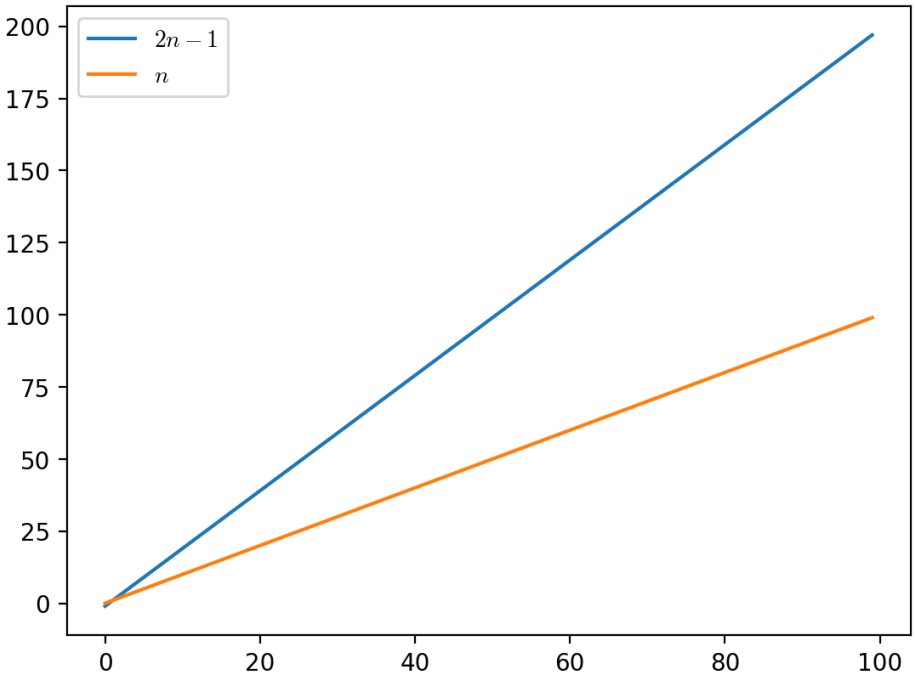


Figure 2: $e(n) = \Theta(n)$

3. Este caso es análogo al anterior solo que como se reserva cuatro veces la capacidad anterior tenemos que se pide memoria (y por tanto se hacen las copias) solo en las potencias pares de dos, por lo mismo en el caso de las potencias impares no se hacen copias solo inserciones. Entonces tenemos que:

$$e(2^k) = \begin{cases} e(2^{k-1}) + 2^k \text{ si } k \text{ es impar} \\ e(2^{k-2}) + 2^k \text{ si } k \text{ es par} \end{cases},$$

se sigue que

$$e(2^p) = \sum_{k \text{ es impar}}^{p} e(2^{k-1}) + 2^k + \sum_{k \text{ es par}}^{p} e(2^{k-2}) + 2^k$$

Problema 2

Implemente un árbol binario de búsqueda, sin balancear, que permita insertar números enteros. Realice N operaciones de inserción, insertando números en el rango $[1, 10^{18}]$ *generados de forma aleatoria. Si el número ya estaba insertado, no lo reinserte.*

1. *Determine, para el caso peor, cuál es el número de comparaciones entre números llevadas a cabo y expréselo usando notación* θ*. El número de comparaciones se refiere al total después de la N inserciones. No hace falta hacer una demostración formal de que ese es el caso peor, pero indique cuándo se genera el peor caso e indique por qué se obtiene esa complejidad. No hace falta la demostración de la complejidad, por ejemplo, si el caso peor es 5n, puede decir que es* $O(n)$ *sin demostrar que* $5n = O(n)$*.*
2. *De forma experimental, cuente cuántas comparaciones se hace y grafíquelo para valores de N hasta* 10^7 *(no hay que probar todas los valores solo un subconjunto). Nótese que dependerá de los valores aleatorios generados, así que ejecútelo varias veces y analice los datos en la forma que considere más oportuna (box-plots, desviación estándar, medias, mínimos, máximos). Encuentre una función que se ajuste de forma adecuada a los datos generados, y trate de explicar qué es lo que está ocurriendo, apoyándose de medidas experimentales sobre el árbol*

que se está generando, así como describiendo su intuición sobre lo que está ocurriendo e incluyendo fórmulas que apoyen a su intuición.

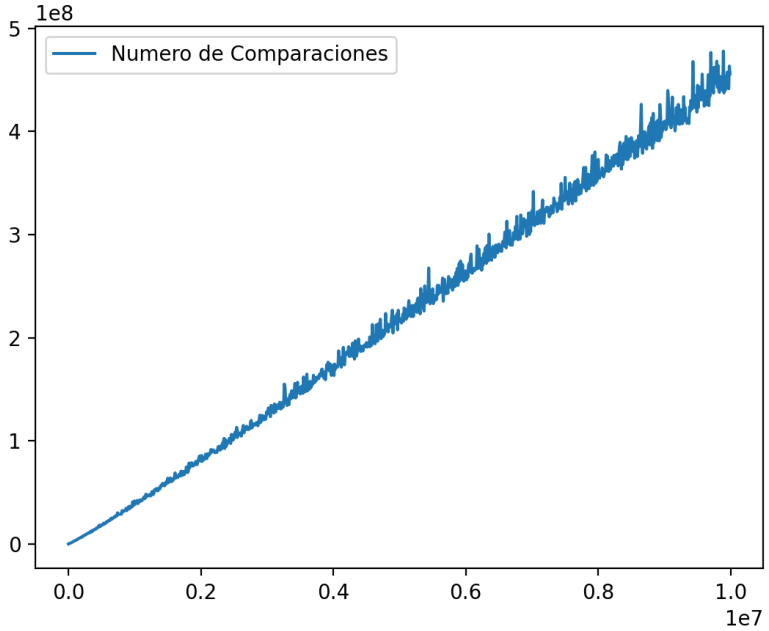
Solución:

1. Primero notemos que cuando se inserta un valor al `BST` tenemos que conocer en que parte del árbol debe ir, para ello buscamos su padre, esto se hace recorriendo el árbol empezando desde la raíz y en cada nodo que visitemos decidimos si irnos a la izquierda o la derecha, entonces en el peor caso se hacen las dos comparaciones[1] y se visitan todos los nodos[2]. Después para insertar el elemento, nuevamente debemos decidir en que parte ira, entonces son dos comparaciones más. También podemos notar que en el peor caso todos los valores son distintos[3]. Por lo cual cuando se inserta el n-esimo se visitan los $(n-1)$ nodos anteriores generando $2n$ comparaciones para encontrar al padre mas dos para saber en que lado del padre debe ir, entonces se hacen $2(n-1)+2$ comparaciones.

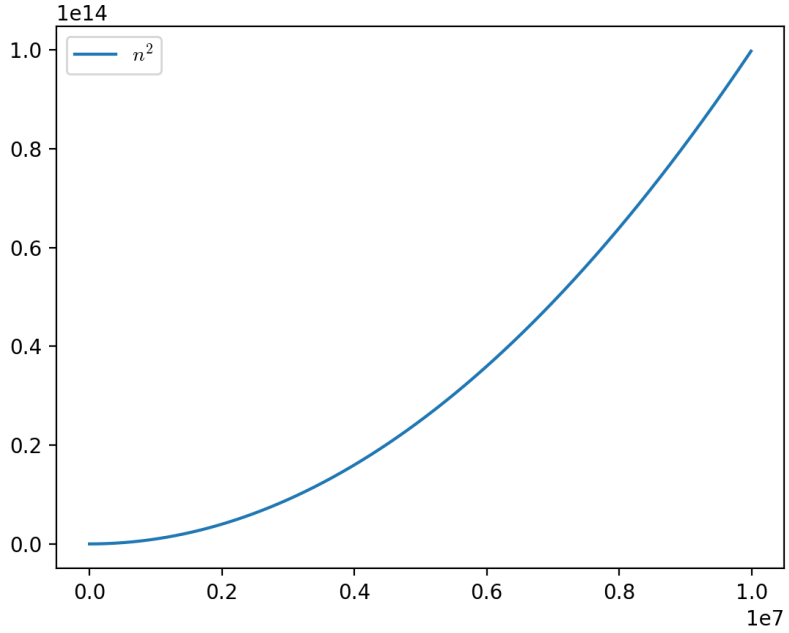
   Esto por cada nodo y el primero no necesita compararse, por lo que en total se hacen

   $$T(n) = \sum_{k=2}^{n} 2(k-1)+2 = \sum_{k=1}^{n-1} 2k+2 = 2\sum_{k=1}^{n-1} k + 2(n-1) = (n-1)n + 2n - 2 = n^2 + n - 2,$$

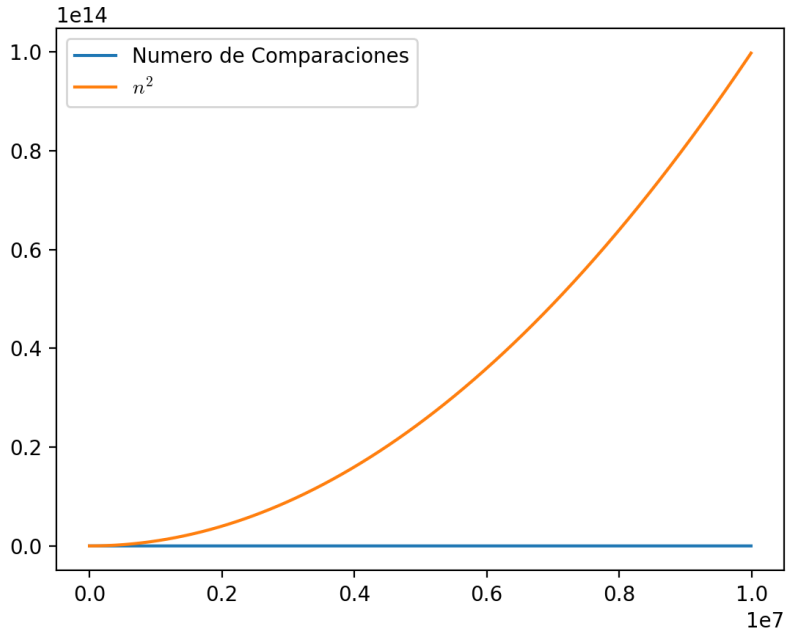
   comparaciones y por tanto se tiene que en el peor caso el numero de comparaciones es $\Theta(n^2)$.
2. En este caso parti el intervalo $[1, 10^7]$ en intervalos de 10^4 elementos[4]. Primero veamos su gráfica:



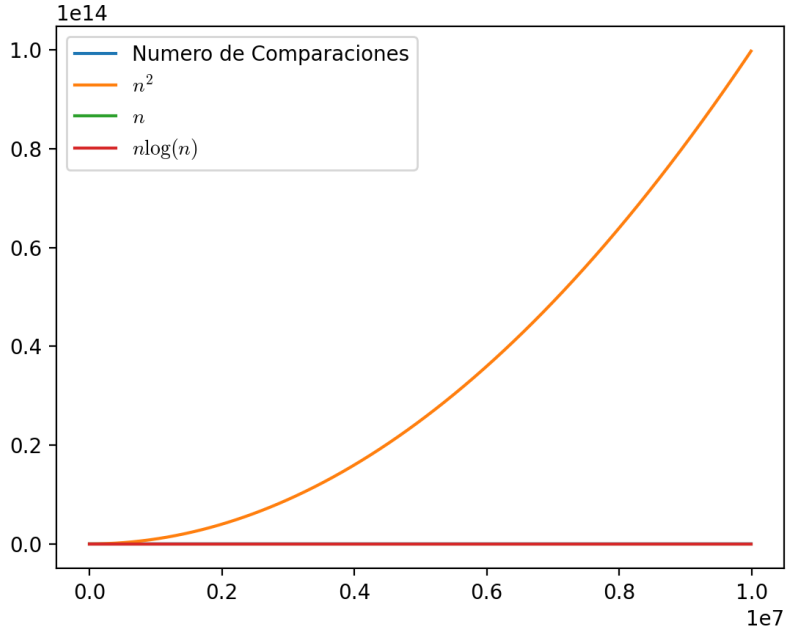
   Ahora veamos la gráfica de n^2



Podemos notar que los valores en el eje y varían mucho, entonces grafiquemos los juntos:



De lo anterior parece ser que el numero de comparaciones es lineal, o $O(n\log(n))$, para comprobar grafiquemos todo:



Sin embargo veamos que el numero de comparaciones sea $\Theta(n\log(n))$ tiene sentido. Suponiendo que la uniformidad de los datos aleatorios generados es muy buena entonces como que dado un nodo la cantidad de hijos a la izquierda y a la derecha es la misma, eso de cierta manera nos dice que nuestro árbol esta balanceado. Si el árbol esta balanceado tenemos que la cantidad de nodos que se tienen que recorrer para hallar un nodo padre es igual a la altura del árbol, que al estar balanceado es $\log(n)$, entonces el numero de comparaciones de un nodo esta acotado por $2\log(n)$ y como son n nodos tenemos que el numero de comparaciones totales esta acotado por $2n\log(n)$.

Problema 3

Demuestre por reducción al absurdo que para todo entero k mayor o igual a 1, $n^{k-1} = o(n^k)$.

Demostración: Supongamos que existe $k \in \mathbb{N}$ tal que $n^{k-1} \neq o(n^k)$, entonces existe $c > 0$ tal que para todo $n \in \mathbb{N}$ se cumple que $n^{k-1} > c \cdot n^k$, lo cual implica que

$$1 > cn \Longrightarrow \frac{1}{c} > n,$$

para todo $n \in \mathbb{N}$, lo cual es absurdo, pues los números naturales no están acotados. De lo anterior obtenemos lo deseado.

[1]Para evitar repetidos.
[2]Esto se logra si insertamos los numero en orden.
[3]Cuando son iguales se evitan comparaciones.
[4]Lo cual tardo como 6 hrs en ejecutarse.