Tiempo de Ejecución (I)

Tiempo real de ejecución

- Sean 2 algoritmos de ordenación:
  - Selección
  - Mergesort

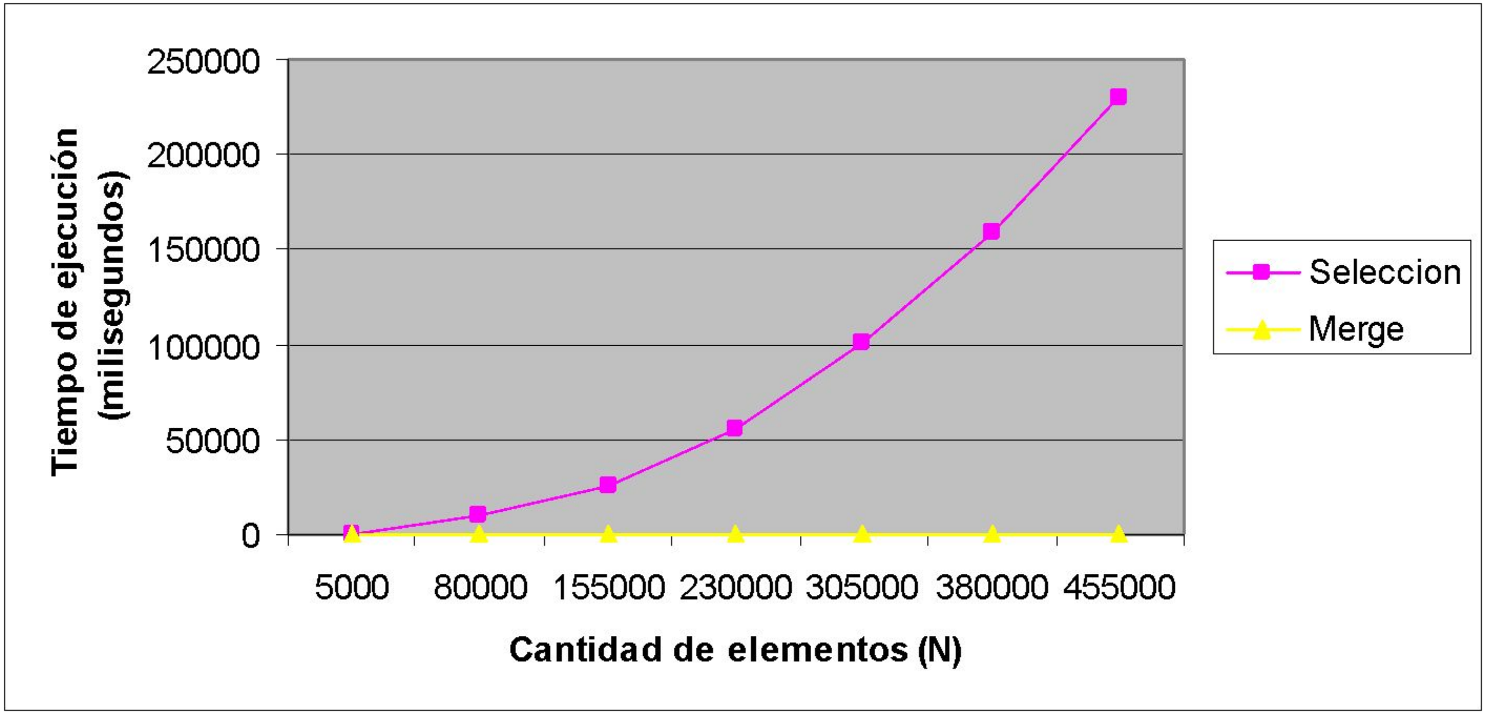
- A continuación se muestran datos reales de tiempo que ejecución

Tiempo real de ejecución

Cantidad (N)	Selección (milisegundos)	Merge (milisegundos)
5000	47	0
80000	9391	15
155000	25188	31
230000	55782	47
305000	100312	62
380000	158875	78
455000	229985	109

Tiempo real de ejecución

- Los algoritmos poseen los siguientes órdenes:
  - Selección $O(n^2)$
  - Mergesort $O(n*Log(n))$



Punto clave

- Calculamos el T(N) y el O(N) para comparar el desempeño de los algoritmos, sin necesidad de cronometrar su tiempo.

¿2^{n+1} es $O(2^n)$?

(es decir, 2^{n+1} crece a una velocidad <= que 2^n?)

Para que 2^{n+1} sea $O(2^n)$, usando definición de BigOh, tiene que verificarse que $\mathbf{2^{n+1} <= c2^n}$ para todo $n>=n_0$.

Ahora bien, $2^{n+1} = 2*2^n$

En particular, podemos decir que $2^{n+1} <=$ **2***2^n.

Considerando **c=2** y dado que vale para todo

$\mathbf{n_0 >=0}$ logramos acotar 2^{n+1} con $c2^n$ por lo cual, 2^{n+1} es $O(2^n)$.

Puntos claves

- Definición de BigOh: $T(n)$ es $O(n)$ si existen constantes $c>0$ y $n_{0,}$ tal que
  $$T(n) <= cO(n) \text{ para todo } n>=n_0$$
- Deben definirse arbitrariamente c y n_0 para verificar la desigualdad
- Las propiedades de potenciación siguen valiendo

¿2^{2n} es $O(2^n)$?

Usando definición de BigOh, tiene que verificarse que **$2^{2n} <= c2^n$ para todo $n>=n_0$**

Ahora bien, $2^{2n} = 2^n*2^n$.

Por lo cual, podemos escribir $2^n*2^n <= c2^n$

Despejando c, $2^n*2^n/2^n<= c$.

Simplificando, $2^n<= c$.

Sin embargo, esto es absurdo, puesto que **nunca se puede acotar con una constante a una función creciente**. Por lo cual 2^{2n} no es $O(2^n)$.

Encontrar T(n)

```
public static void uno (int n)  {
      int  i, j, k ;
      int [] [] a, b, c;
      a = new int [n] [n];
      b = new int [n] [n];
      c = new int [n] [n];
      for ( i=1; i<=n-1; i++) {
            for ( j=i+1; j<=n; j++) {
                  for ( k=1; k<=j; k++)  {
                     c[i][j] = c[i][j]+
  a[i][j]*b[i][j];
                  }
            }
      }
   }
```

Encontrar T(n)

$$T(n) = c1 + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{j} c2 = c1 + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(j * c2) = c1 + c2 * \left(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} j\right) =$$

$$T(n) = c1 + c2 * \sum_{i=1}^{n-1}\left(\sum_{j=1}^{n} j - \sum_{j=1}^{i} j\right) = c1 + c2 * \sum_{i=1}^{n-1}\left(\frac{n*(n+1)}{2} - \frac{i*(i+1)}{2}\right) =$$

$$T(n) = c1 + \frac{c2}{2}\sum_{i=1}^{n-1} n*(n+1) - \frac{c2}{2}\sum_{i=1}^{n-1} i*(i+1) = c1 + \frac{c2}{2}(n-1)*n*(n+1) - \frac{c2}{2}\sum_{i=1}^{n-1}(i^2 + i) =$$

$$c1 + \frac{c2}{2}(n-1)*n*(n+1) - \frac{c2}{2}\sum_{i=1}^{n-1}(i^2) - \frac{c2}{2}\sum_{i=1}^{n-1}(i) =$$

$$c1 + \frac{c2}{2}(n-1)*n*(n+1) - \frac{c2}{2}\left(\frac{(n-1)*n(2(n-1)+1)}{6}\right) - \frac{c2}{2}\left(\frac{(n-1)n}{2}\right)$$

Puntos claves

- Traducir constantes o iteraciones correctamente.
- Respetar los límites de las iteraciones al traducirlas a sumatorias (respetando las variables).
- Prestar atención a si dentro de una sumatoria **se suma o no la variable índice**.
- Tener presente que las equivalencias para la suma de los n primeros números naturales **comienza en 1** y no en un número arbitrario.