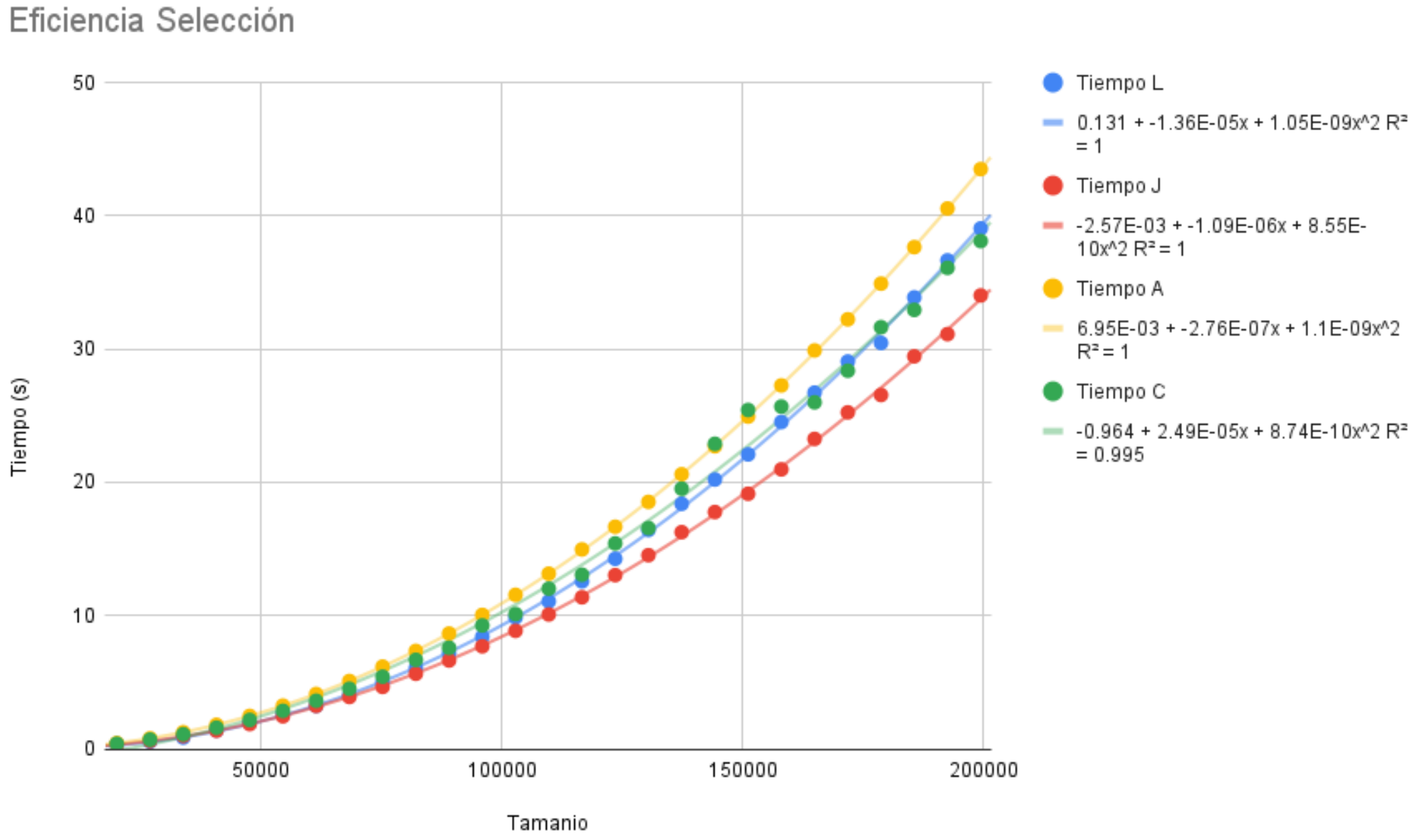
Análisis de Eficiencia de Algoritmos

JOAQUÍN ARCILA PÉREZ
LAURA LÁZARO SORALUCE
CRISTÓBAL MERINO SÁEZ
ÁLVARO MOLINA ÁLVAREZ

ÍNDICE

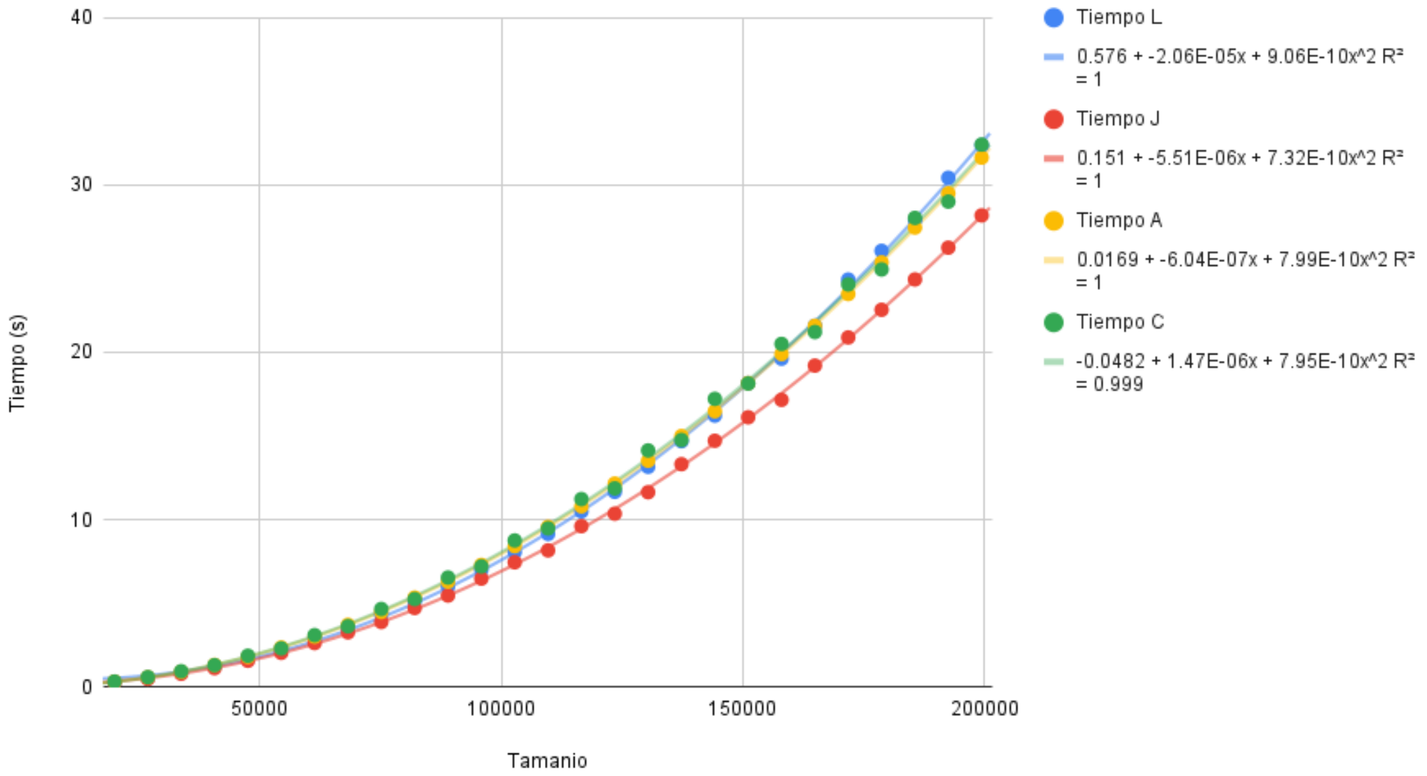
INTRODUCCIÓN

DESARROLLO
Algoritmo de selección



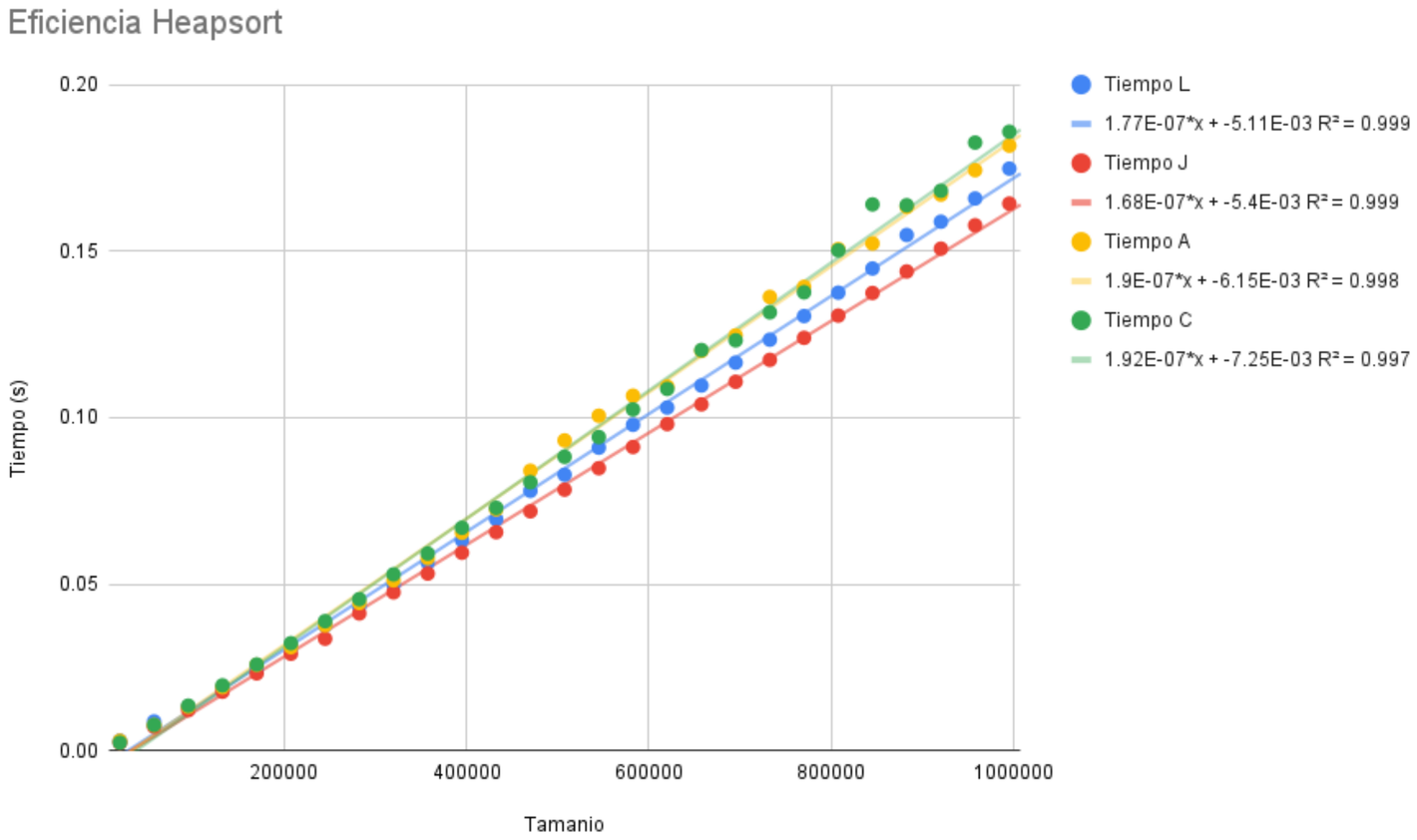

```
void seleccion(int T[], int num_elem)
{
 seleccion_lims(T, 0, num_elem);
}

static void seleccion_lims(int T[], int inicial, int final)
{
 int i, j, indice_menor;
 int menor, aux;
 for (i = inicial; i < final - 1; i++) {
 indice_menor = i;
 menor = T[i];
 for (j = i; j < final; j++)
 if (T[j] < menor) {
 indice_menor = j;
 menor = T[j];
 }
 aux = T[i];
 T[i] = T[indice_menor];
 T[indice_menor] = aux;
 };
}
```

DESARROLLO
Algoritmo de inserción



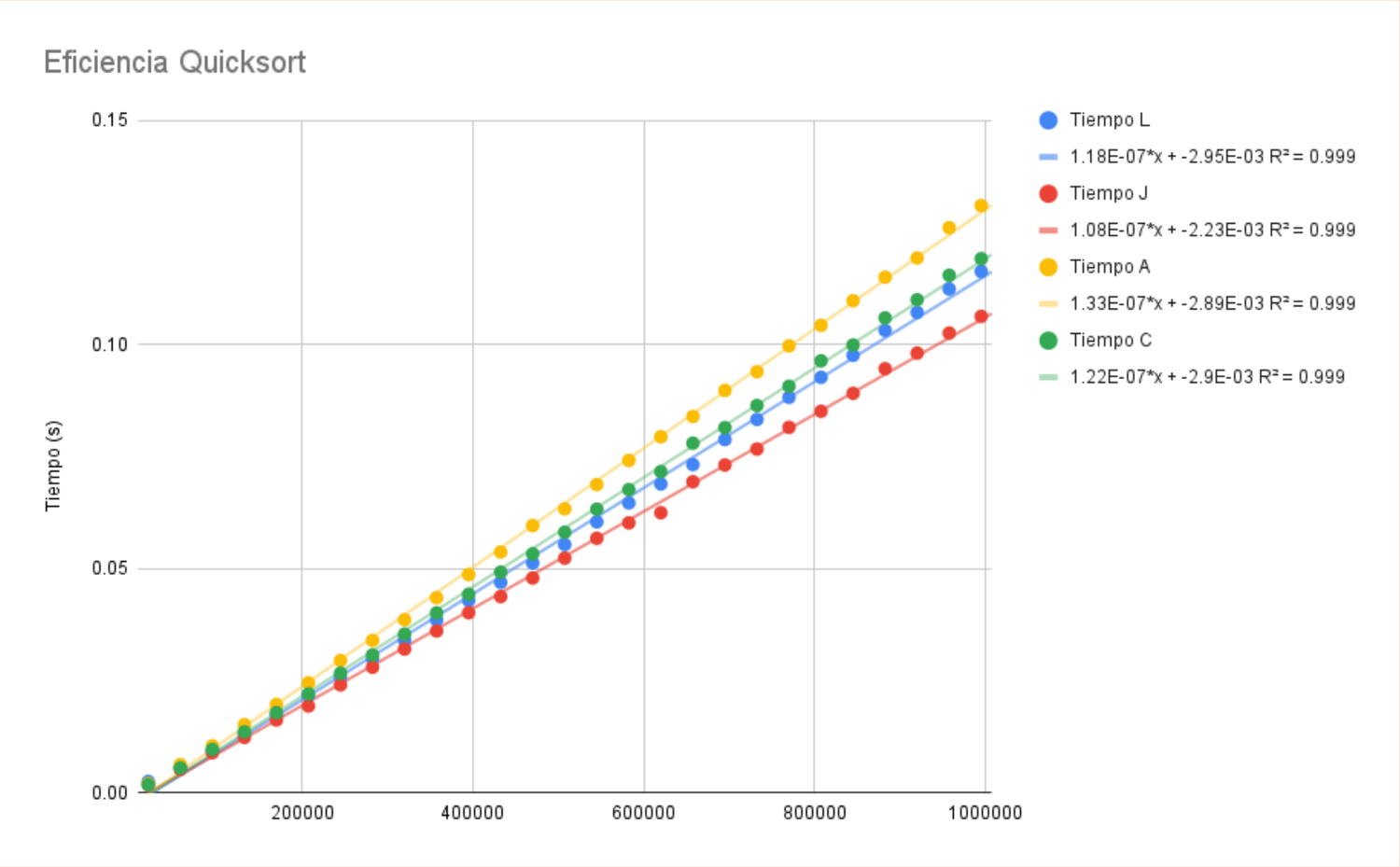

```
inline static void insercion(int T[], int
num_elem)
{
 insercion_lims(T, 0, num_elem);
}

static void insercion_lims(int T[], int inicial, int
final)
{
 int i, j;
 int aux;
 for (i = inicial + 1; i < final; i++) {
 j = i;
 while ((T[j] < T[j-1]) && (j > 0)) {
 aux = T[j];
 T[j] = T[j-1];
 T[j-1] = aux;
 j--;
 };
 };
}
```

DESARROLLO
Algoritmo de heapsort



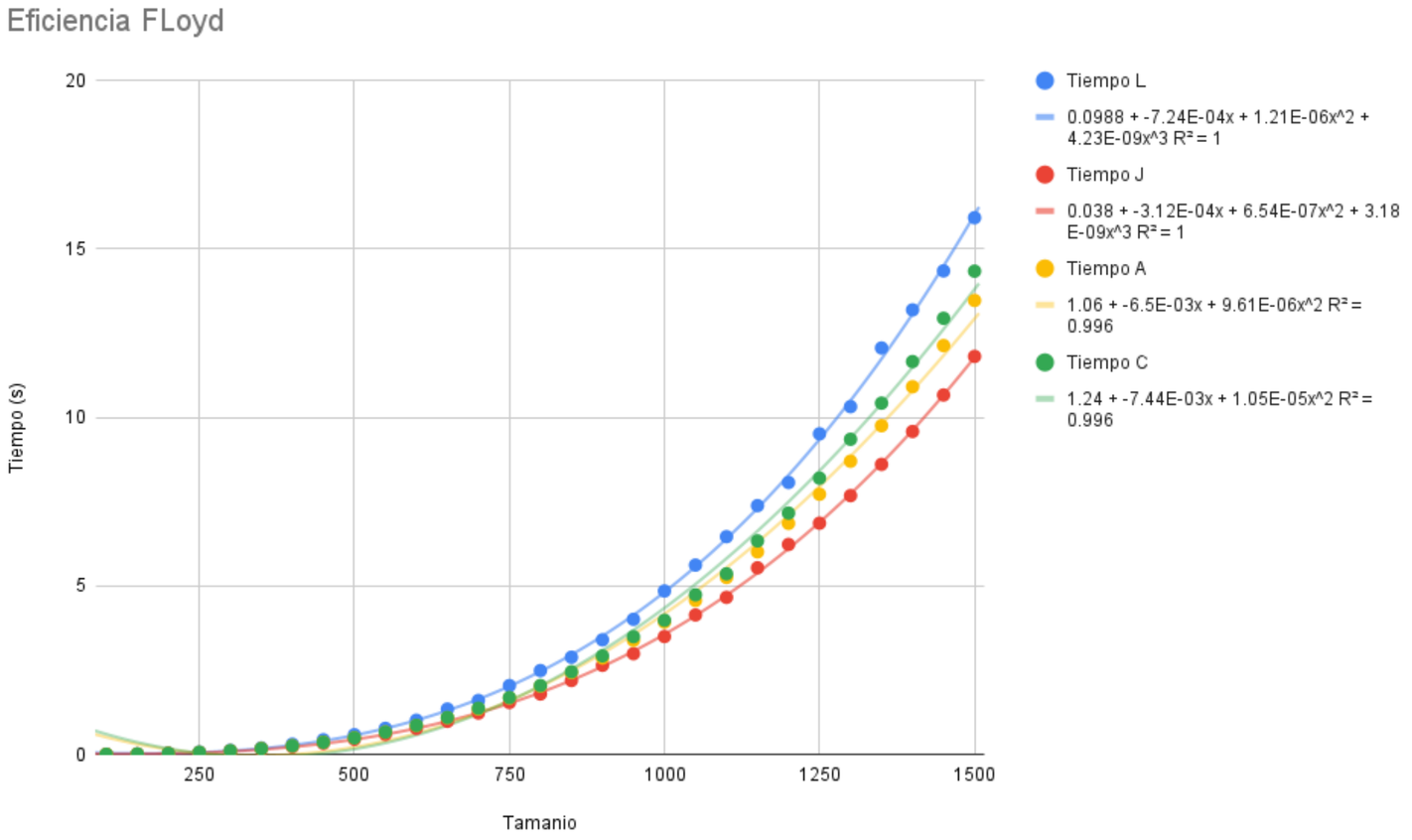

```
static void heapsort(int T[], int num_elem)
{
 int i;
 for (i = num_elem/2; i >= 0; i--)
 reajustar(T, num_elem, i);
 for (i = num_elem - 1; i >= 1; i--)
 {
 int aux = T[0];
 T[0] = T[i];
 T[i] = aux;
 reajustar(T, i, 0);
 }
}

static void reajustar(int T[], int num_elem, int
k)
{
 int j;
 int v;
 v = T[k];
 bool esAPO = false;
 while ((k < num_elem/2) && !esAPO)
 {
 j = k + k + 1;
 if ((j < (num_elem - 1)) && (T[j] < T[j+1]))
 j++;
 if (v >= T[j])
 esAPO = true;
 T[k] = T[j];
 k = j;
 }
 T[k] = v;
}
```

DESARROLLO
Algoritmo de quicksort



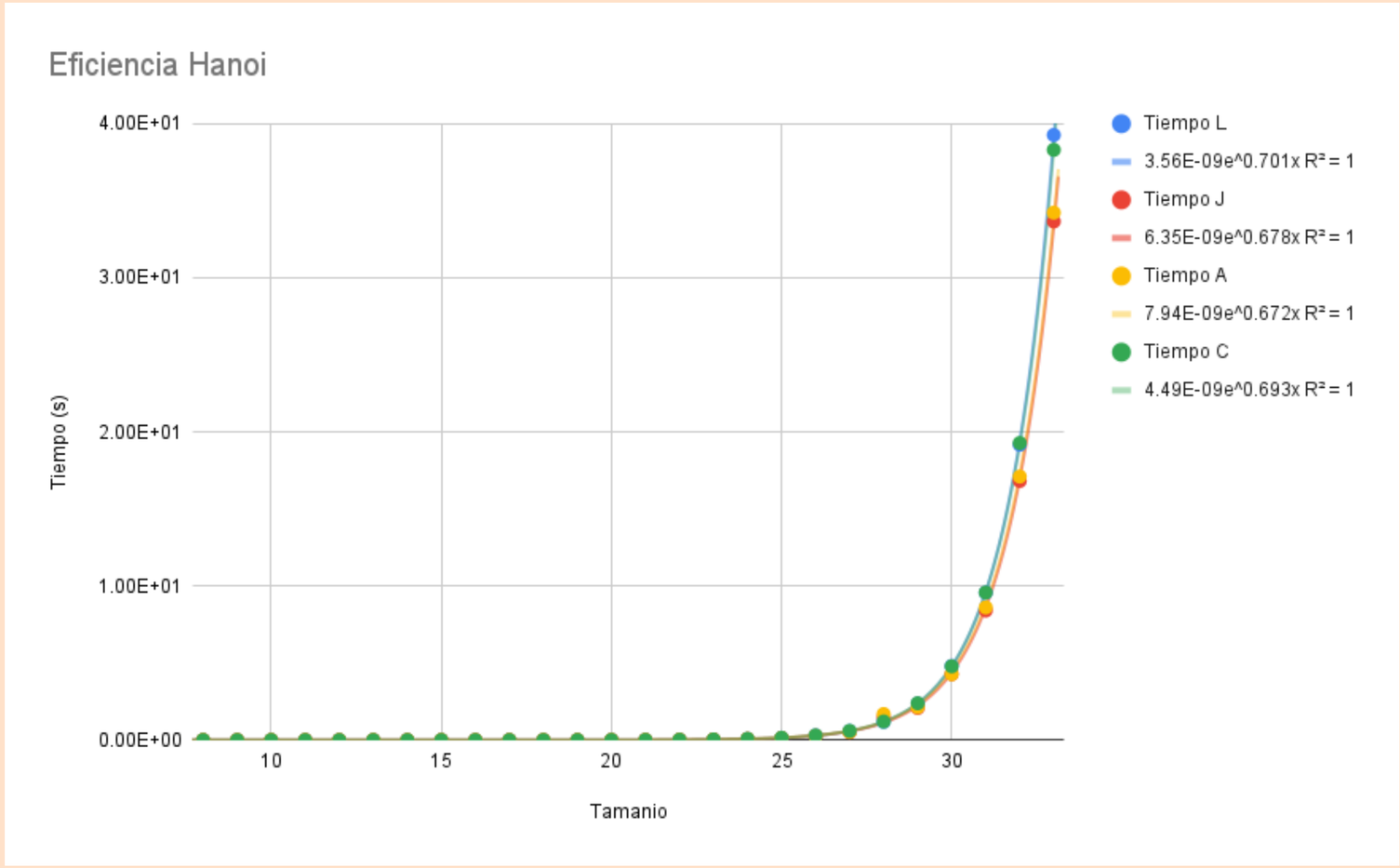

```
inline void quicksort(int T[], int num_elem)
{ quicksort_lims(T, 0, num_elem); }

static void quicksort_lims(int T[], int inicial, int final)
{
int k;
 if (final - inicial < UMBRAL_QS) {
 insercion_lims(T, inicial, final);
 } else {
 dividir_qs(T, inicial, final, k);
 quicksort_lims(T, inicial, k);
 quicksort_lims(T, k + 1, final); };
}

static void dividir_qs(int T[], int inicial, int final, int & pp)
{
int pivote, aux;
 int k=inicial, l=final;
 pivote = T[inicial];

 do k++; while ((T[k] <= pivote) && (k < final-1));
 do l--; while (T[l] > pivote);
 while (k < l) {
 aux = T[k];
 T[k] = T[l];
 T[l] = aux;
 do k++; while (T[k] <= pivote);
 do l--; while (T[l] > pivote); };
 aux = T[inicial];
 T[inicial] = T[l];
 T[l] = aux;
 pp = l;
};
```

DESARROLLO
Algoritmo de Floyd




```
void Floyd(int **M, int dim)
{
 for (int k = 0; k < dim; k++)
 for (int i = 0; i < dim;i++)
 for (int j = 0; j < dim;j++)
 {
 int sum = M[i][k] + M[k][j];
 M[i][j] = (M[i][j] > sum) ? sum : M[i][j];
 }
}
```

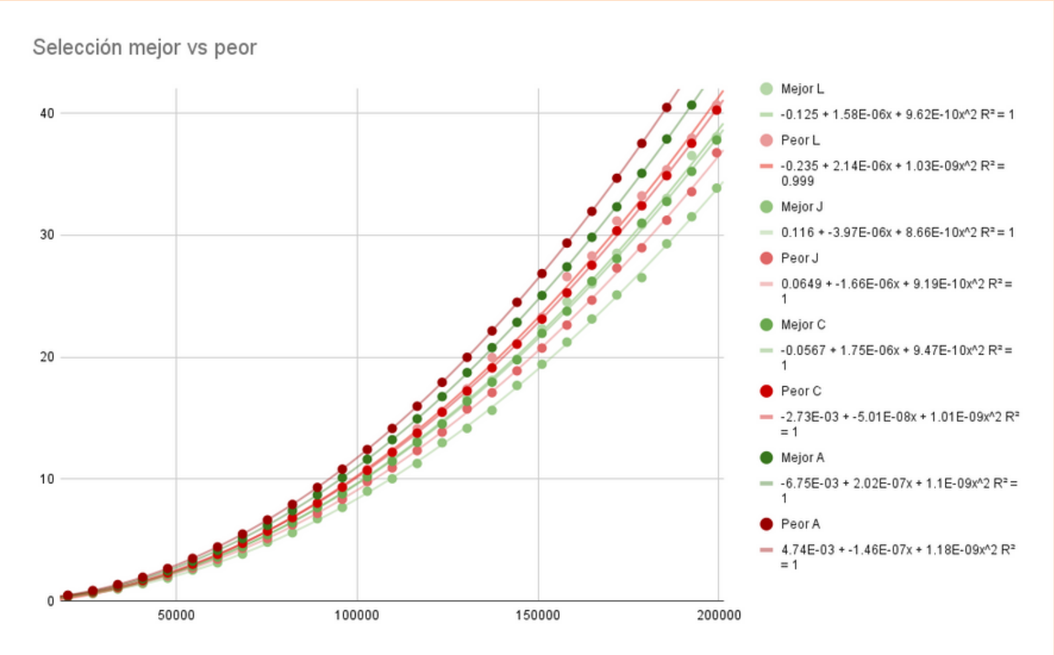
DESARROLLO
Algoritmo de Hanoi



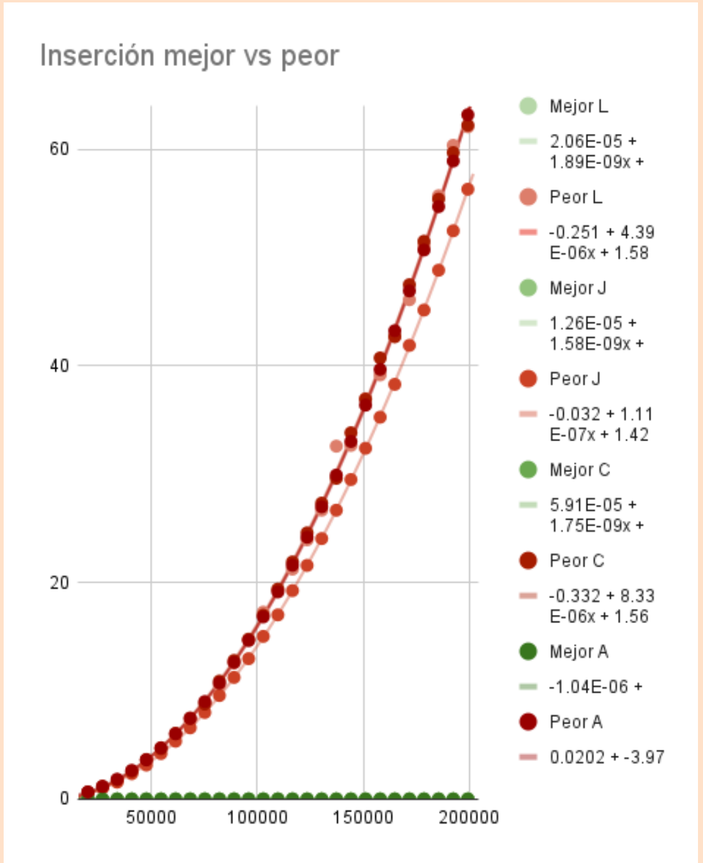

```
void hanoi (int M, int i, int j)
{
 if (M > 0)
 {
 hanoi(M-1, i, 6-i-j);
 //cout << i << " -> " << j << endl;
 hanoi (M-1, 6-i-j, j);
 }
}
```

COMPARACIONES
Mejor vs Peor Caso

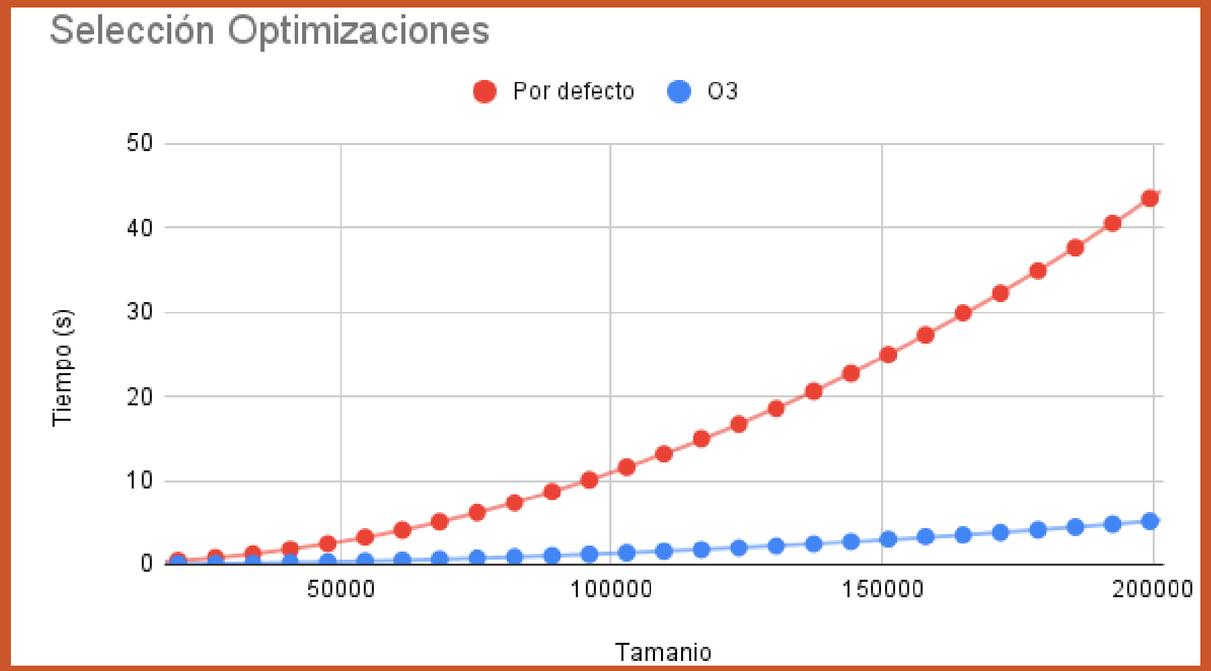
Selección:

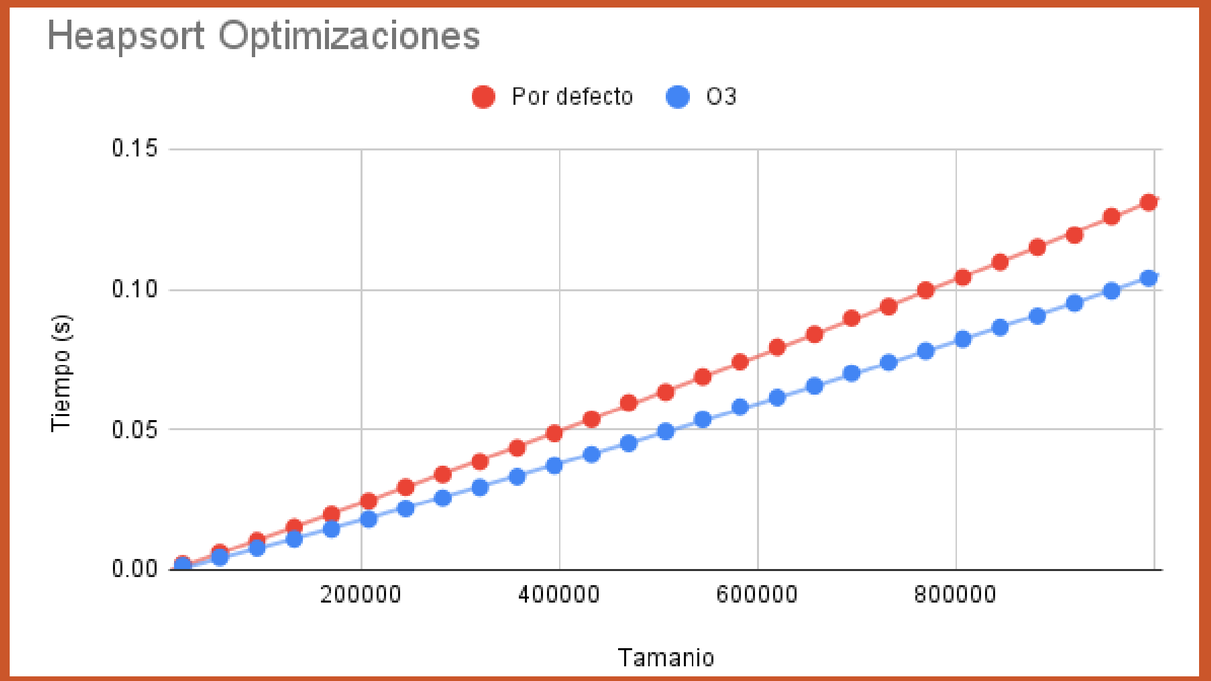


Inserción:

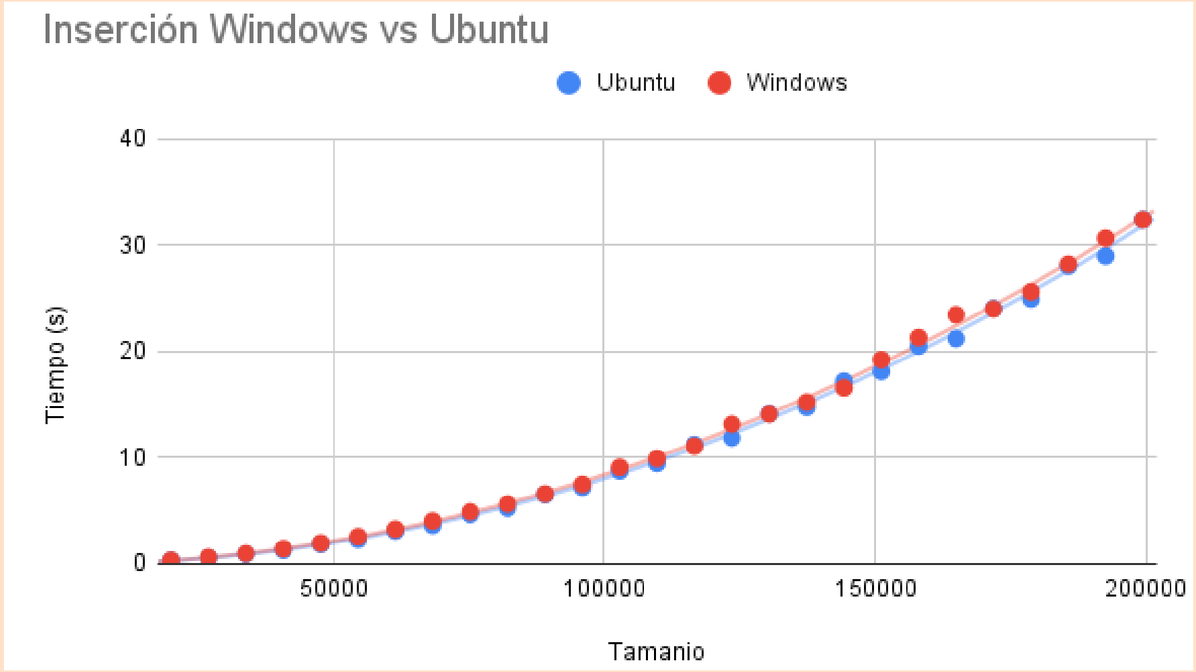


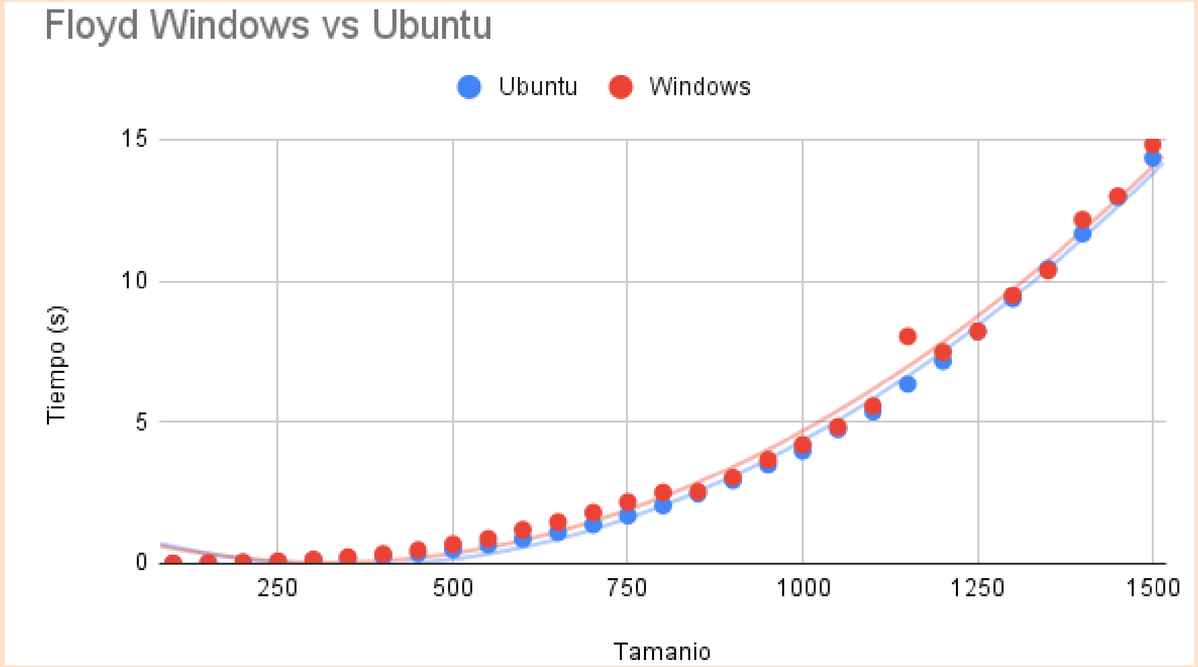
COMPARACIONES
Optimizaciones



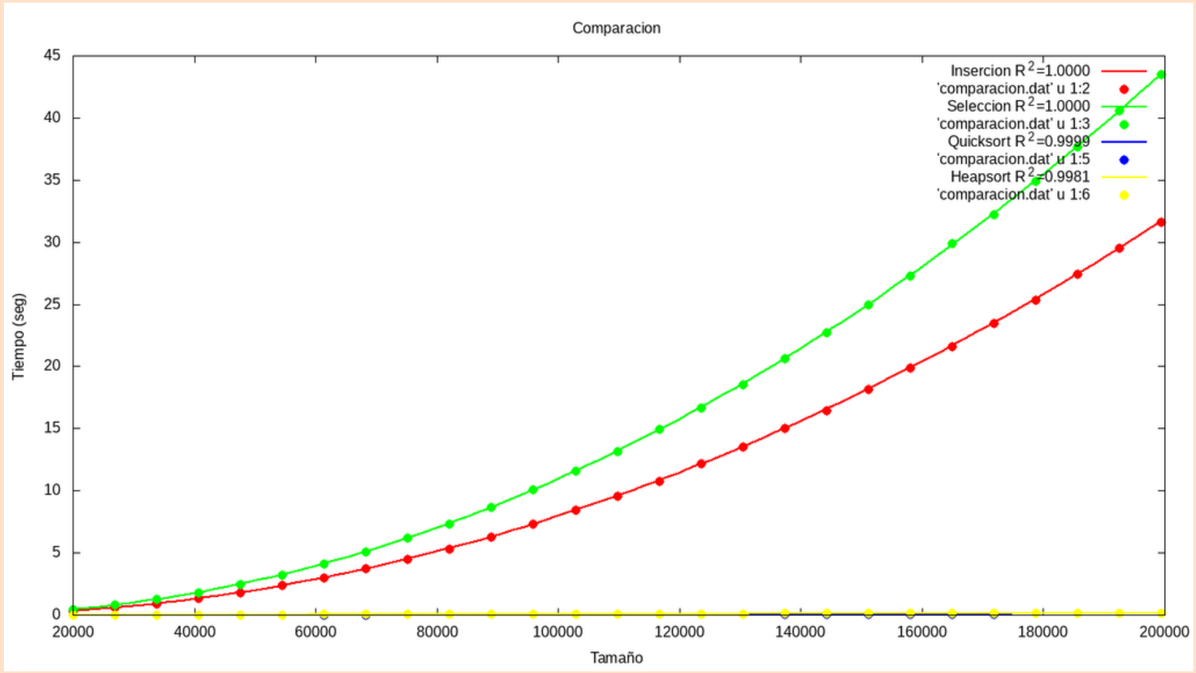


COMPARACIONES
Windows vs Ubuntu

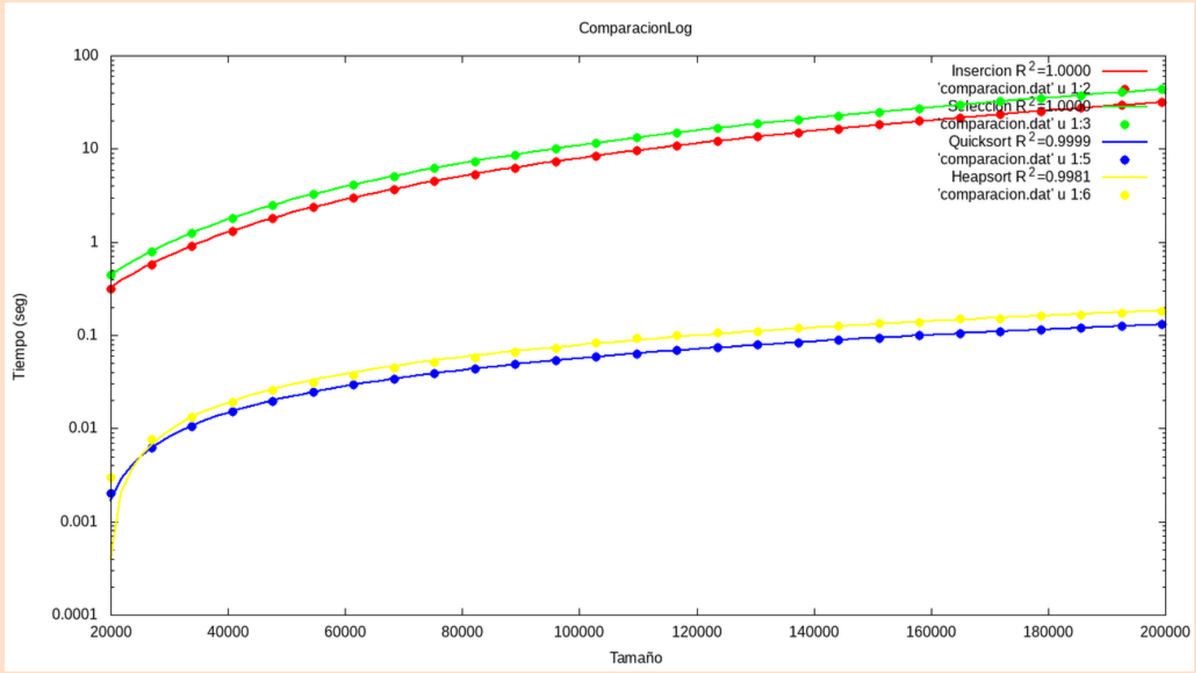




COMPARACIONES
Algoritmos



Escala Normal



Escala Logarítmica

CONCLUSIÓN

-Mayor importancia del orden de eficiencia que de otros factores de ejecución
-Poca diferencia entre la ejecución en sistemas operativos distintos
-Pequeña diferencia entre el peor y el mejor caso de selección
-Gran diferencia entre el peor y el mejor caso de inserción

MUCHAS GRACIAS