Tarea 1. Análisis de algoritmos

Alumno	Ricardo Payan

Insertion-Sort.

1. Codificar el algoritmo deordenamiento por inserción(INSERTION-SORT), y calcular el tiempo de ejecución para diez valores distintos de n. Realizar lo anterior paraun arreglo ordenado en orden creciente (mejor caso), orde-nado en forma decreciente (peor caso) y un arreglo aleatorio (caso promedio). Para cada caso calcular los tiempos y graficarlos.

   Implementación:

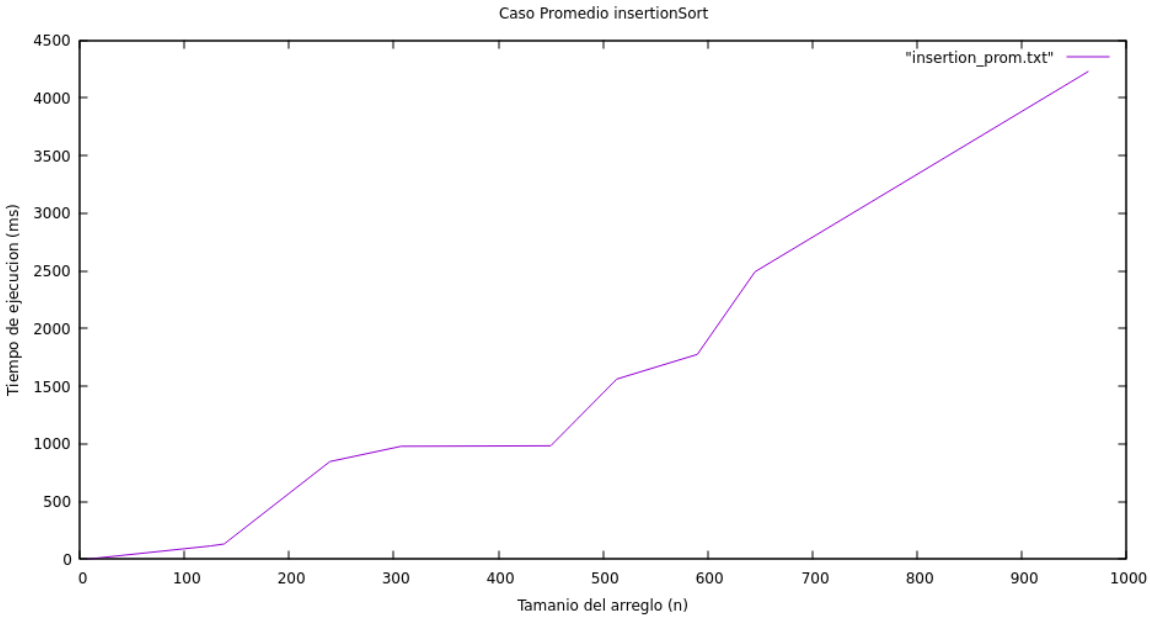
```
// Función de Insertion Sort
void insertionSort(vector<int> &arr) {
    int n = arr.size();
    int key, j;
    for (int i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

   En mi caso, para todos los algortimos, hice los vectores de tamaño aleatorio.

Caso Promedio.

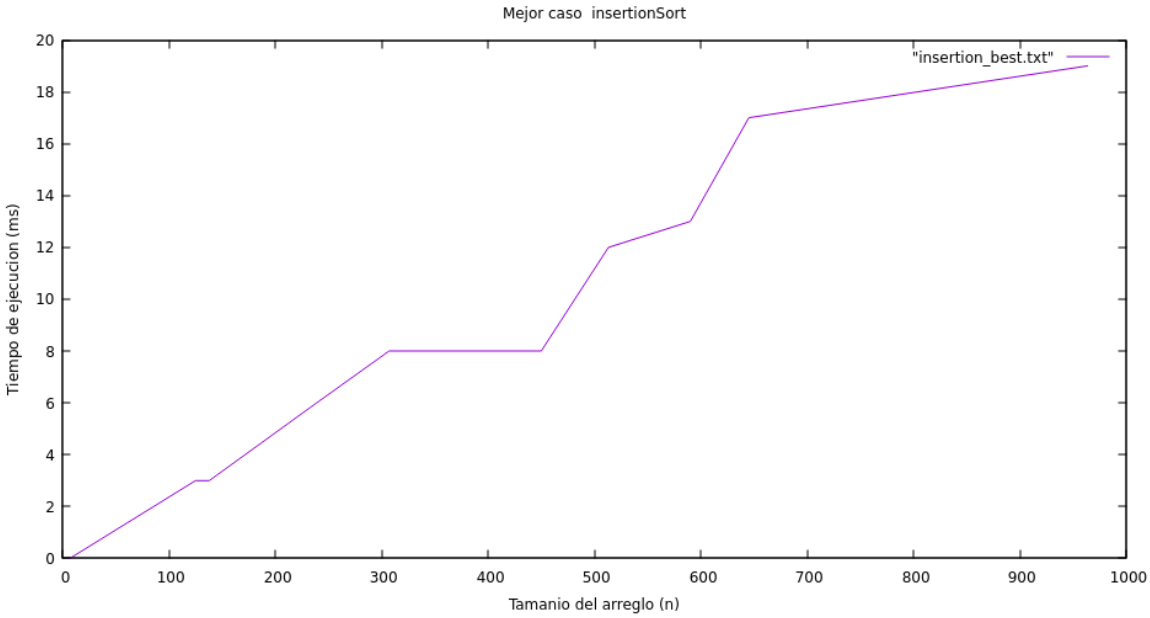
Tamaño del arreglo (n)	Tiempo (ms)
7	1
125	115
138	133
239	847
307	979

Tamaño del arreglo (n)	Tiempo (ms)
450	984
513	1562
590	1776
645	2492
963	4226



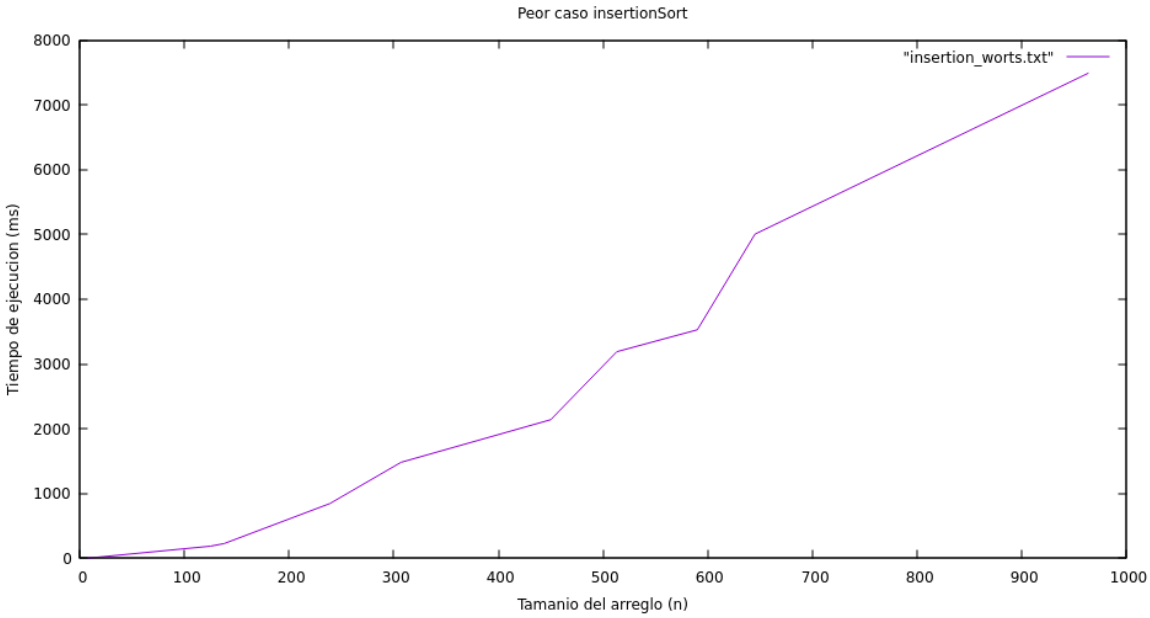
Mejor caso.

Tamaño del arreglo (n)	Tiempo (ms)
7	0
125	3
138	3
239	6
307	8
450	8
513	12
590	13
645	17
963	19



Peor Caso.

Tamaño del arreglo (n)	Tiempo (ms)
7	1
125	188
138	228
239	845
307	1483
450	2140
513	3190
590	3528
645	5005
963	7484



Selection-Sort.

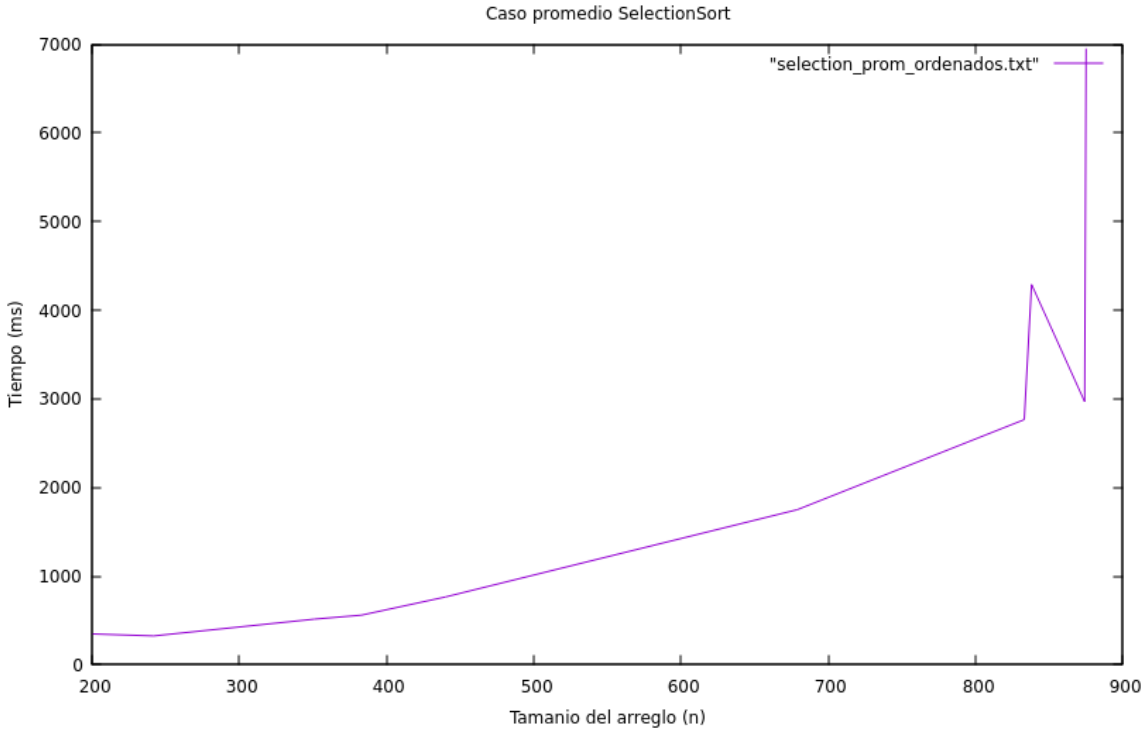
Implementación

```
void selectionSort(vector<int> &arr) {
    int n = arr.size();
    for (int i = 0; i < n-1; i++) {
        int minIndex = i;
        for (int j = i+1; j < n; j++) {
            if (arr[j] < arr[minIndex])
                minIndex = j;
        }
        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}
```

Caso Promedio.

Tamaño del arreglo (n)	Tiempo (ms)
200	349
242	327

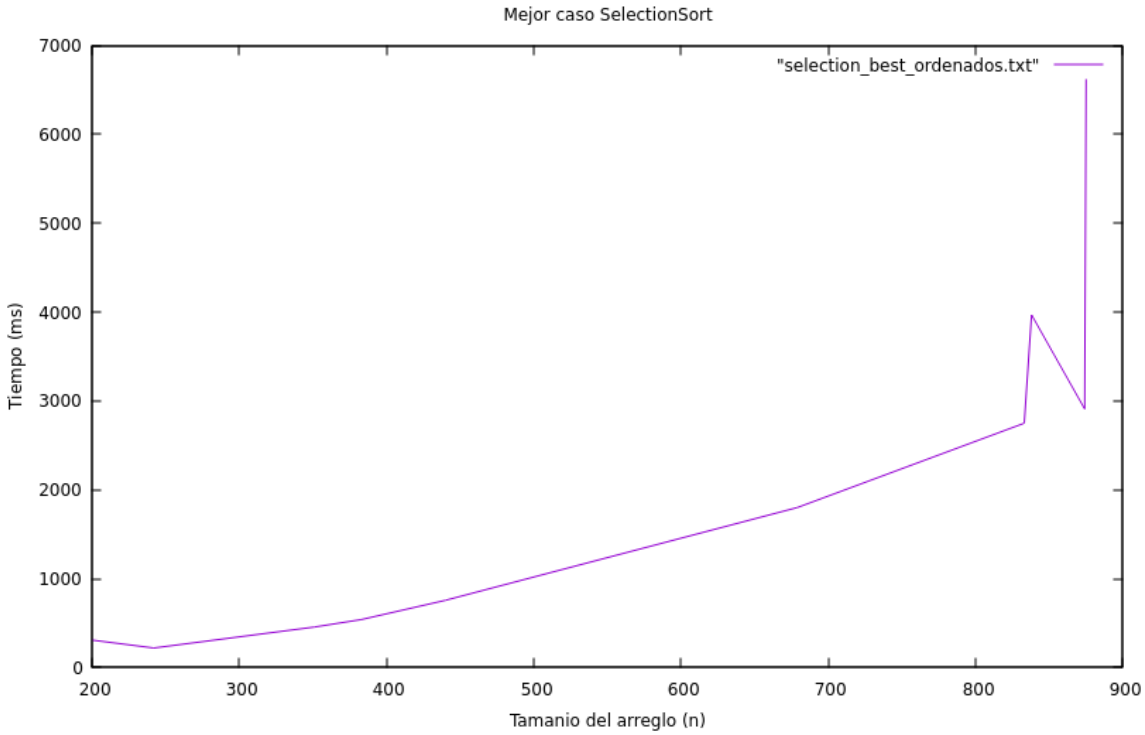
Tamaño del arreglo (n)	Tiempo (ms)
351	515
383	560
440	766
679	1751
833	2765
838	4290
874	2968
875	6943



Mejor Caso.

Tamaño del arreglo (n)	Tiempo (ms)
200	311
242	221
351	457
383	541

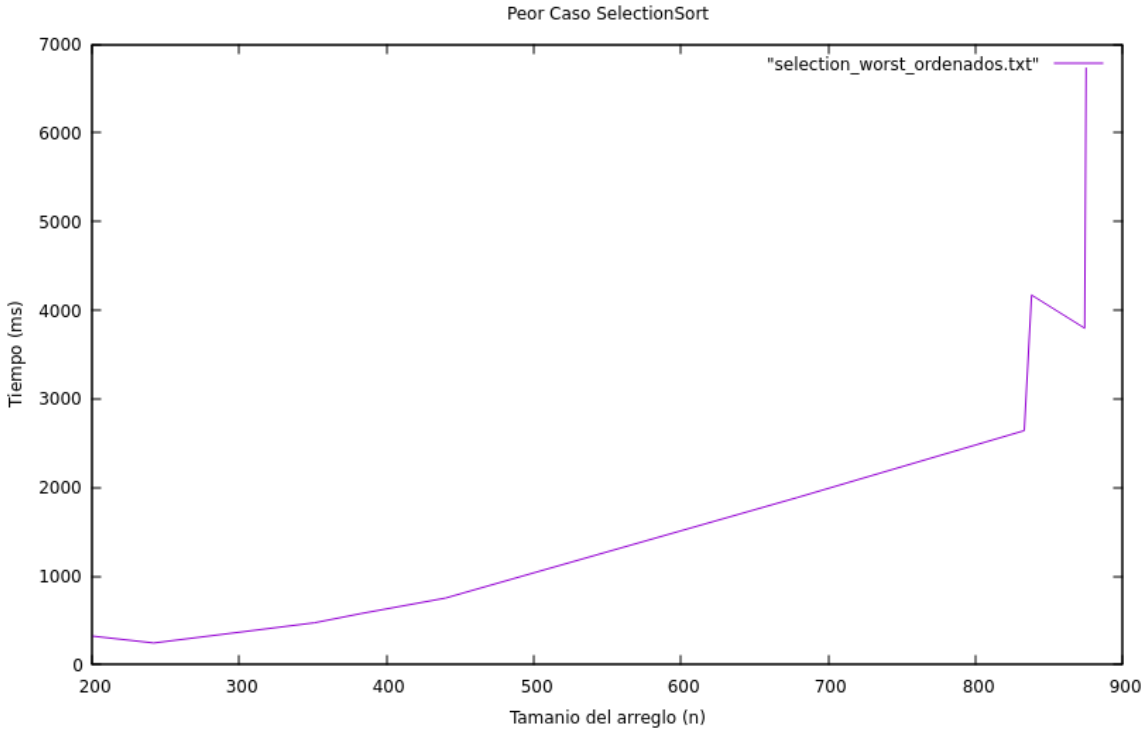
Tamaño del arreglo (n)	Tiempo (ms)
440	757
679	1801
833	2750
838	3969
874	2910
875	6612



Peor caso.

Tamaño del arreglo (n)	Tiempo (ms)
200	325
242	247
351	474
383	580
440	753
679	1888
833	2641

Tamaño del arreglo (n)	Tiempo (ms)
838	4170
874	3796
875	6728



Merge-Sort

Implementación:

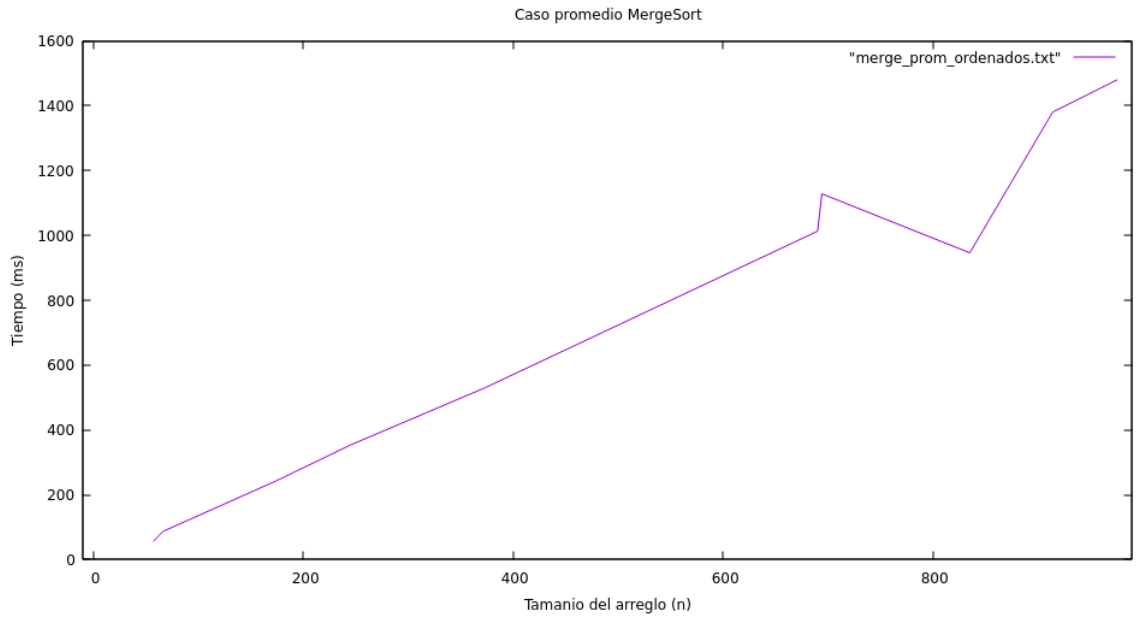
```
// Función para combinar dos subarrays de arr[].
// El primer subarray es arr[l..m]
// El segundo subarray es arr[m+1..r]
void merge(vector<int> &arr, int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    vector<int> L(n1), R(n2);
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    i = 0;
    j = 0;
```

```
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Función principal de Merge Sort
void mergeSort(vector<int> &arr, int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
```

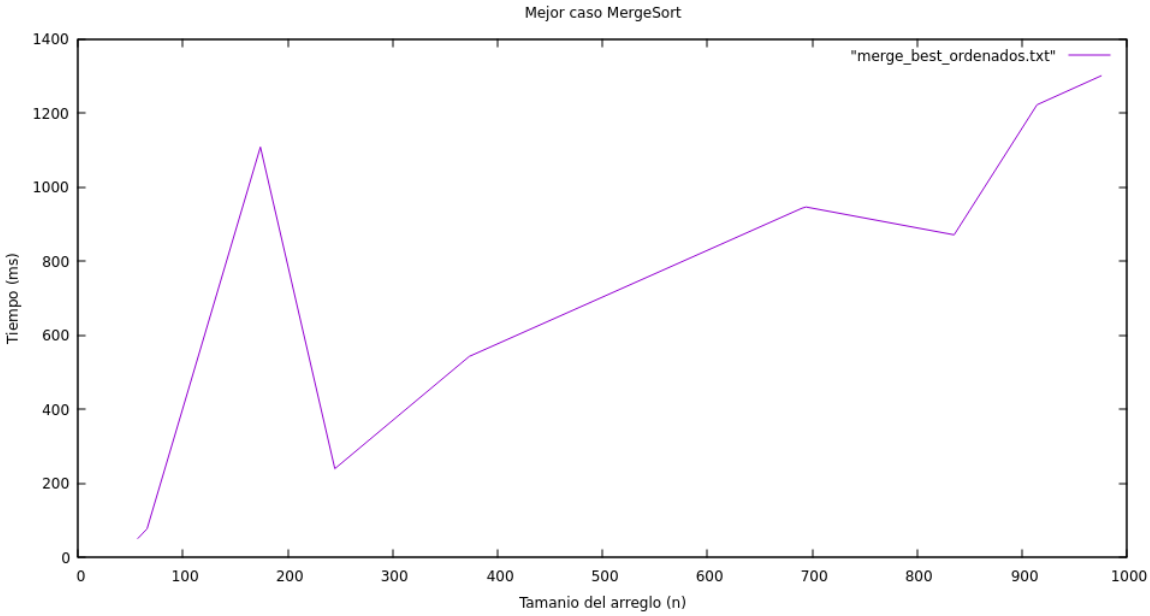
Caso Promedio.

Tamaño del arreglo (n)	Tiempo (ms)
57	56
66	87
174	243
245	354
373	530
690	1013
694	1128
835	946
914	1380
975	1479



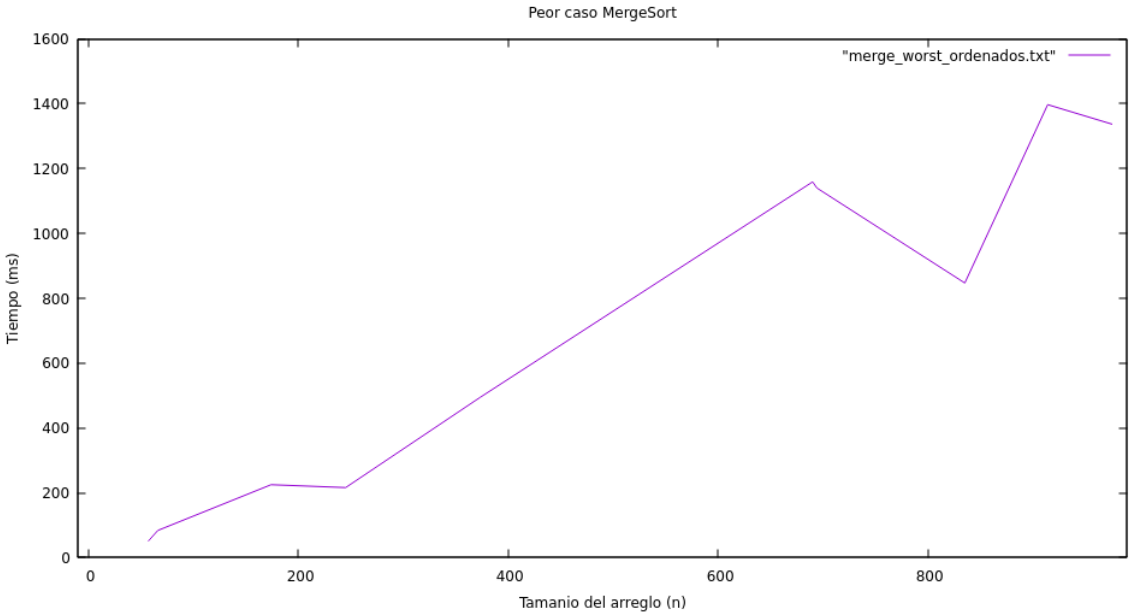
Mejor Caso.

Tamaño del arreglo (n)	Tiempo (ms)
57	51
66	78
174	1108
245	240
373	543
690	943
694	946
835	871
914	1222
975	1300



Peor Caso.

Tamaño del arreglo (n)	Tiempo (ms)
57	51
66	84
174	225
245	216
373	494
690	1159
694	1140
835	847
914	1397
975	1337



5. Realizar la graficación de las siguientes funciones: $f(n) = log(n)$, $f(n) = n$, $f(n) = n \cdot log(n)$, $f(n) = n^2$, $f(n) = n^3$.Todas las curvas deben ser colocadas en la misma gráfica.