# Práctica 1. "Análisis experimental de la eficiencia de algoritmos de ordenamiento"

Christian Miguel Hernández Mejía
Departamento de Ciencias e Ingeniería de la Computación,
Análisis de Algoritmos, ESCOM-IPN
(email: christian.mhm@outlook.com)

## I. Objetivo

El alumno realizará un análisis a posteriori de diversos algoritmos de ordenamiento. Implementará y comparará la eficiencia de estos algoritmos en los casos mejor, peor y promedio.

## II. Intrucciones

Para mostrar la eficiencia de diferentes algoritmos que solucionan un mismo problema, se consideró el problema de ordenamiento de una lista de números enteros. Los algoritmos que se implementarán son:

1) Ordenamiento por inseción
2) El método de la burbuja
3) Ordenamiento por mezcla
4) Ordenamiento rápido (Quick-sort)

Considere como entrada, conjuntos de números enteros con $\{100, 200, 300, \ldots, 10,000\}$ elementos. Deberá realizar la ejecución de cada algoritmo para el mejor y peor caso. Además debe usar los datos de entrada proporcionados por el profesor para el caso promedio. Deberá reportar el tiempo de ejecución (en segundos) de cada instancia como se pide en la Tabla 1. Una vez que se tengan los datos del tiempo de ejecución grafique (de preferencia en python o gnuplot) estos resultados (ver ejemplo en Figura 1).

Deberá enviar al correo miriam.pescador@gmail.com, una carpeta comprimida con la implementación de los códigos python y el reporte en formato latex y pdf. El asunto del correo deberá decir "Practica 1 Analisis de Algoritmos [nombre completo del alumno comenzando con el apellido paterno]". La carpeta debe tener el nombre del alumno (comenzando por apellido paterno). La fecha límite de entrega es el próximo Martes 5 de febrero de 2019 a las 10:00 pm. Por cada día de retraso se penalizará al alumno con 15% de la calificación obtenida.

El reporte en latex debe considerar las siguientes secciones:

- Introducción: descripción sobre la implementación de la práctica
- Marco teórico. En esta sección deberá poner los conceptos de:
  - Algoritmo
  - Análisis a priori y posteriori
  - Análisis del mejor y peor caso
  - Caso promedio

  Incluya la bibliografía que fue consultada (use el formato de ejemplo para agregarlo a su reporte).
- Implementación. Coloque en esta sección los algoritmos proporcionados en la plantilla de latex. Proporcione información sobre las características del equipo de cómputo donde realizó las pruebas (sistema operativo, tipo de procesador, memoria, etc.). Además deberá documentar las bibliotecas que empleo para la implementación de los algoritmos.
- Resultados. Incluya la tabla de resultados y graficas solicitadas.
- Conclusiones. Describa cuáles fueron los mejores algoritmos y para qué casos y/o número de datos de entrada. Proporcione una justificación del por qué se obtuvieron estos resultados.

### A. Algoritmos

**Algorithm 1**: Insertion Sort Algorithm

```
Data: A : list of sortable items
begin
    InsertionSort(A)
    for i ← 2 to n do
        j ← i − 1;
        while j ≥ 1 and A[j] > A[j + 1] do
            swap(A[j], A[j+1]);
            j ← j − 1;
    return A;
end
```

**Algorithm 2**: Merge Sort Algorithm

```
MergeSort(A, p, r)
if p < r then
    q ← ⌊(p + r)/2⌋;
    MergeSort(A, p, q);
    MergeSort(A, q + 1, r);
    Merge(A, p, q, r);
return A;
```

La figura 1 muestra el comportamiento de las funciones.

**Algorithm 3**: Bubble Sort Algorithm

```
Data: A : list of sortable items
begin
    n ← length(A);
    repeat
        swapped ← false;
        for i ← 1 to n do
            if A[i − 1] > A[i] then
                swap(A[i-1], A[i]);
                swapped ← true;
        n ← n − 1;
    until not swapped ;
    return A;
end
```

**Algorithm 4**: Quick Sort Algorithm

```
QuickSort(A, p, r)
if p < r then
    q ← Partition(A, p, r);
    QuickSort(A, p, q);
    QuickSort(A, q + 1, r);
return A;
```

**Algorithm 5**: Partition Algorithm

```
Partition(A, p, r)
x ← A[p];
i ← p − 1 j ← r + 1  while true do
    repeat
        j ← j − 1;
    until A[j] ≤ x ;
    repeat
        i ← i + 1;
    until A[j] ≥ x ;
    if i ¡ j then
        exchange A[i] ↔ A[j];
    else
        return j;
```

## REFERENCES


[1] P. Moscato, *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms,* Technical Report C3P Report 826, 1989.

[2] N. Krasnogor and J. Smith, *A Memetic Algorithm With Self-Adaptive Local Search: TSP as a case study,* Genetic Evolutionary Computation Conference pp. 987-994, 2000.


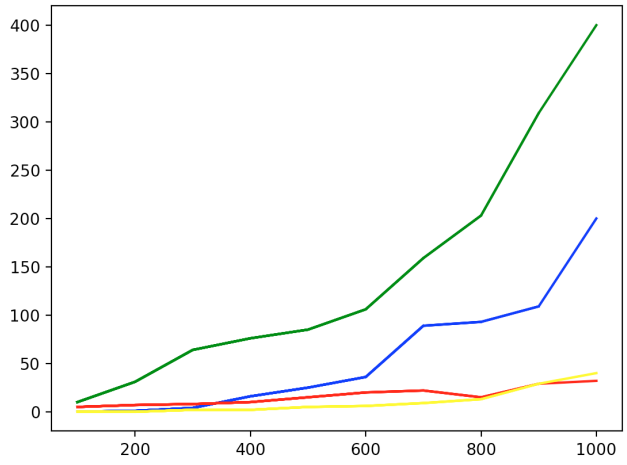


Fig. 1. Comparación del comportamiento de los algoritmos

| n | Mejor Caso | | | | Peor Caso | | | | Caso promedio | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | insertion | merge | bubble | quick-sort | insertion | merge | bubble | quick-sort | insertion | merge | bubble | quick-sort |
| 100 | 0.00004 | 0.00051 | 0.00002 | 0.00041 | 0.0051 | 0.00072 | 0.00451 | 0.00138 | 0.00269 | 0.00073 | 0.00302 | 0.0005 |
| 200 | 0.00007 | 0.00119 | 0.00004 | 0.0008 | 0.01916 | 0.00142 | 0.01697 | 0.00493 | 0.00992 | 0.00144 | 0.01065 | 0.00109 |
| 300 | 0.00012 | 0.00177 | 0.00006 | 0.00123 | 0.0445 | 0.00214 | 0.03682 | 0.00961 | 0.02035 | 0.0022 | 0.02188 | 0.0015 |
| 400 | 0.00015 | 0.00236 | 0.00008 | 0.00189 | 0.07573 | 0.00304 | 0.06223 | 0.01633 | 0.0357 | 0.00273 | 0.03949 | 0.00208 |
| 500 | 0.00021 | 0.00309 | 0.00011 | 0.00223 | 0.11081 | 0.00303 | 0.09037 | 0.0257 | 0.05057 | 0.00324 | 0.05765 | 0.00252 |
| 600 | 0.00024 | 0.00387 | 0.00014 | 0.00285 | 0.15409 | 0.00399 | 0.13395 | 0.03691 | 0.07657 | 0.00415 | 0.08583 | 0.00337 |
| 700 | 0.0003 | 0.00457 | 0.00016 | 0.00346 | 0.22882 | 0.00506 | 0.18777 | 0.05388 | 0.11413 | 0.00496 | 0.12575 | 0.00476 |
| 800 | 0.00034 | 0.00528 | 0.00019 | 0.00405 | 0.32366 | 0.00631 | 0.26037 | 0.06509 | 0.14958 | 0.00547 | 0.16437 | 0.00491 |
| 900 | 0.00038 | 0.0059 | 0.00021 | 0.00456 | 0.37904 | 0.00647 | 0.30892 | 0.07697 | 0.20877 | 0.00615 | 0.20181 | 0.00495 |
| 1000 | 0.00043 | 0.00665 | 0.00024 | 0.00493 | 0.47107 | 0.00787 | 0.45145 | 0.11483 | 0.227 | 0.00683 | 0.25352 | 0.00568 |
| 1100 | 0.00049 | 0.0077 | 0.00028 | 0.00574 | 0.60479 | 0.00821 | 0.53072 | 0.12596 | 0.29485 | 0.00762 | 0.31975 | 0.00652 |
| 1200 | 0.00052 | 0.00833 | 0.00029 | 0.00623 | 0.73021 | 0.012 | 0.64184 | 0.17054 | 0.35064 | 0.00859 | 0.37459 | 0.00726 |
| 1300 | 0.00059 | 0.00919 | 0.00034 | 0.00718 | 0.80978 | 0.00976 | 0.72414 | 0.17043 | 0.41011 | 0.00942 | 0.42794 | 0.00766 |
| 1400 | 0.0006 | 0.00986 | 0.00038 | 0.00805 | 1.00332 | 0.00999 | 0.86745 | 0.19028 | 0.46963 | 0.0097 | 0.51216 | 0.00835 |
| 1500 | 0.00058 | 0.01049 | 0.00039 | 0.00882 | 1.15588 | 0.01163 | 0.928 | 0.21852 | 0.52731 | 0.01053 | 0.58435 | 0.00964 |
| 1600 | 0.00069 | 0.01135 | 0.00042 | 0.00923 | 1.24741 | 0.01139 | 1.01678 | 0.24528 | 0.63197 | 0.01191 | 0.65451 | 0.00968 |
| 1700 | 0.00076 | 0.01272 | 0.00048 | 0.01011 | 1.6366 | 0.01329 | 1.12716 | 0.27465 | 0.7209 | 0.01262 | 0.75385 | 0.01022 |
| 1800 | 0.00082 | 0.01285 | 0.00047 | 0.01014 | 1.50069 | 0.01236 | 1.22014 | 0.30347 | 0.80239 | 0.01377 | 0.84367 | 0.01075 |
| 1900 | 0.00084 | 0.01461 | 0.00048 | 0.0103 | 1.7747 | 0.01406 | 1.43062 | 0.34033 | 0.87809 | 0.0142 | 0.99564 | 0.01197 |
| 2000 | 0.00077 | 0.01474 | 0.00055 | 0.01159 | 1.91349 | 0.01463 | 1.71074 | 0.40753 | 1.02944 | 0.01859 | 1.22941 | 0.01654 |
| 2100 | 0.00094 | 0.01559 | 0.00059 | 0.01173 | 2.13514 | 0.01541 | 1.81542 | 0.43846 | 1.02876 | 0.01565 | 1.14635 | 0.01275 |
| 2200 | 0.00098 | 0.01625 | 0.00061 | 0.01214 | 2.30065 | 0.01619 | 1.95848 | 0.46913 | 1.20191 | 0.01673 | 1.30875 | 0.01391 |
| 2300 | 0.00103 | 0.01701 | 0.00064 | 0.01301 | 2.66175 | 0.02212 | 2.30951 | 0.54539 | 1.31769 | 0.01752 | 1.42972 | 0.01465 |
| 2400 | 0.00106 | 0.01696 | 0.00059 | 0.01387 | 2.94397 | 0.02053 | 2.46941 | 0.59705 | 1.6125 | 0.02102 | 1.61563 | 0.01539 |
| 2500 | 0.00112 | 0.01854 | 0.0007 | 0.0145 | 3.09781 | 0.01917 | 2.63058 | 0.6382 | 1.51991 | 0.02061 | 1.70267 | 0.01712 |
| 2600 | 0.00118 | 0.0191 | 0.00062 | 0.01502 | 3.39612 | 0.01877 | 2.7743 | 0.74055 | 1.64607 | 0.02099 | 1.84184 | 0.01689 |
| 2700 | 0.00116 | 0.02014 | 0.00071 | 0.01621 | 3.73651 | 0.02184 | 3.05087 | 0.72472 | 1.72787 | 0.02079 | 1.87635 | 0.01702 |
| 2800 | 0.00126 | 0.02049 | 0.00067 | 0.01682 | 3.98676 | 0.02571 | 3.29633 | 0.83525 | 1.97331 | 0.02183 | 2.35468 | 0.02058 |
| 2900 | 0.00131 | 0.02073 | 0.00073 | 0.01774 | 4.1186 | 0.02124 | 3.23006 | 0.77287 | 2.30956 | 0.02337 | 2.29632 | 0.01913 |
| 3000 | 0.00121 | 0.02196 | 0.00075 | 0.01807 | 4.36995 | 0.0224 | 3.51441 | 0.88998 | 2.15485 | 0.02522 | 2.33197 | 0.02223 |

TABLE I

RESULTADOS DE LOS TIEMPOS DE EJECUCIÓN PARA CADA ALGORITMO.