Universidad Andrés Bello

Informe Laboratorio 1

Análisis y Diseño de Algoritmos

Integrantes:

- André Ducheylard.
- Fabián Cárdenas.

Docente:

- Irene Zuccar Parrini.

Fecha: Viernes 14 de septiembre de 2018

Índice

Introducción

En el presente informe, se dará a conocer los datos obtenidos del análisis de dos algoritmos de ordenamiento de números programados en lenguaje c (QuickSort e InsertionSort). Para cada algoritmo se implementó una variable iteradora la cual refleja el número de repeticiones que realizo cada algoritmo en función del tamaño del conjunto de números que se ordenó, los tamaños son 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800 y 2000 datos, para cada tamaño de conjunto se realizó tres veces la medida variando entre mejor caso (los datos estaban ordenados), caso promedio (los datos se obtuvieron de forma aleatoria) y el peor caso (los datos estaban en orden invertido), mediante un análisis del código se obtuvo una función matemática en base a la cantidad de elementos del conjunto (t(n)), la cual también se utilizó para llegar a las conclusiones que se presentaran al final del informe.

Resultados Teóricos

Se llamara resultado teórico al obtenido mediante la expresión matemática (t(n)).

Para QuickSort.

En el mejor caso (los números están ordenados).

```
65  void quickSort2(int ini, int fin, int n, int b[n]){
66      int posPiv = 0; k
67
68      if(ini < fin){   k
69          posPiv = particion(ini, fin, n, b); k + TP(n) (particion 6kn +9k)
70          quickSort2(ini, posPiv - 1, n, b);
71          quickSort2(posPiv + 1, fin, n, b); } k + t(n-1)
72      }
73  }
```

```
75  int particion(int ini, int fin, int n, int b[n]){
76      int posUltMenor, k;
77      int temporal, pivote;
78      pivote = b[ini]; k
79      posUltMenor = ini; k
80      k = ini + 1; 2k
81      while(k <= fin){ k
82          if(b[k] < pivote){ k
83              posUltMenor++;
84              temporal = b[posUltMenor];
85              b[posUltMenor] = b[k];
86              b[k] = temporal;
87          }
88          iteraciones2++; 2k
89          k++; 2k
90      }
91          temporal = b[ini]; k
92          b[ini] = b[posUltMenor]; k
93          b[posUltMenor] = temporal; k
94
95      return posUltMenor; k
96  }
97
```

La expresión obtenida es:

$$t(n) = 2k \qquad si\ n = 0$$

$$t(n) = t(n-1) + 6kn + 13k \qquad si\ n > 0$$

$$t(n) = 3kn^2 + 16kn + 2k$$ *Ecuación desarrollada*

k: Tiempo que tarda el procesador en realizar una operación.

Para InsertionSort.

En el mejor caso (los números están ordenados).

```
4   int *b;
5   int iteraciones2 = 0;
6
7   int InsertionSort(int n, int a[n]){
8       int j, temporal;
9       int iteraciones = 0; k
10      int i = 1;  k
11      while (i < n){ k
12          iteraciones++; 2k
13          j = i; k     k          k       k
14          while (a[j] < a[j-1] && j-1 >= 0){
15              temporal = a[j-1];
16              a[j-1] = a[j];
17              a[j] = temporal;
18              j--;
19              iteraciones++;
20          }
21          i++; 2k
22      }
23      return iteraciones; k
24  }
```

La expresión obtenida es:

$$t(n) = 9kn + 4k$$

k: Tiempo que tarda el procesador en realizar una operación.

Reemplazando n (número de datos) en las ecuaciones se obtuvo.

n	InsertionSort	QuickSort
200	1805*k	123212*k
400	3605*k	486412*k
600	5405*k	1089612*k
800	7205*k	1932812*k
1000	9005*k	3016012*k
1200	10805*k	4339212*k
1400	12605*k	5902412*k
1600	14405*k	7705612*k
1800	16205*k	9748812*k
2000	18005*k	12032012*k

Tabla 1.1 Mejor Caso Teórico

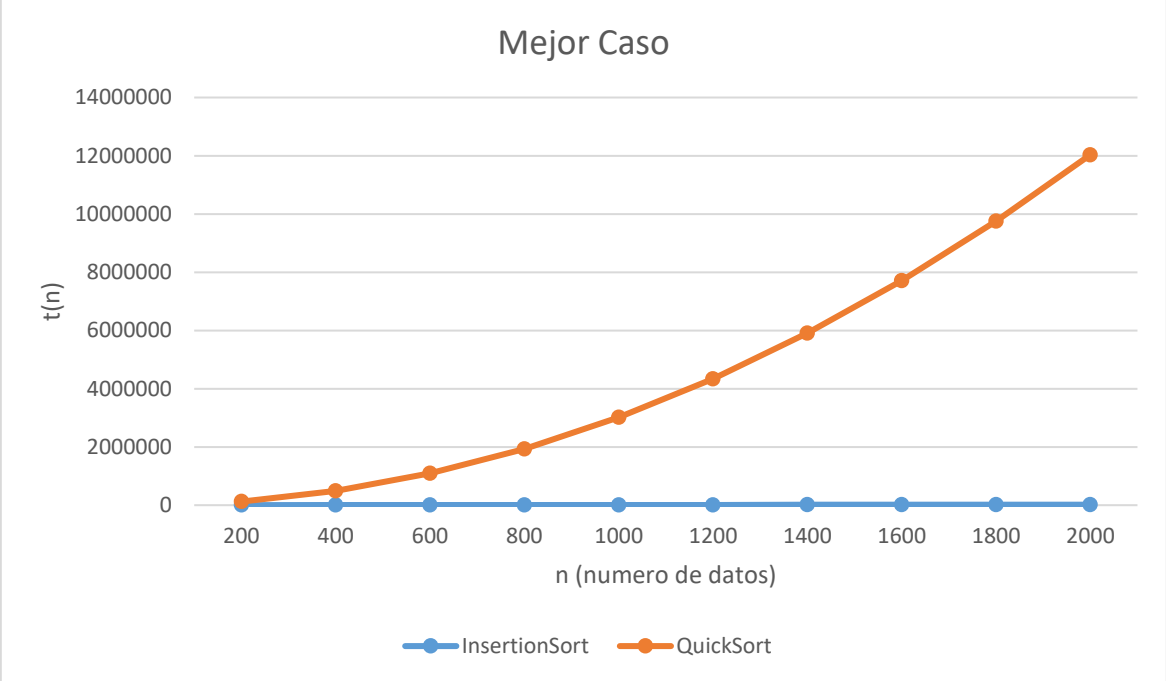

Grafico 1.1 InsertionSort vs Quicksort Mejor Caso Teórico

Para QuickSort.

En el peor caso (los números están ordenados al revés).

```
75  int particion(int ini, int fin, int n, int b[n]){
76      int posUltMenor, k;
77      int temporal, pivote;
78      pivote = b[ini];  k
79      posUltMenor = ini;  k
80      k = ini + 1;  2k
81      while(k <= fin){  k
82          if(b[k] < pivote){  k
83              posUltMenor++;  2k
84              temporal = b[posUltMenor];  k
85              b[posUltMenor] = b[k];  k
86              b[k] = temporal;  k
87          }
88          iteraciones2++;  2k
89          k++;  2k
90      }
91          temporal = b[ini];  k
92          b[ini] = b[posUltMenor];  k
93          b[posUltMenor] = temporal;  k
94
95      return posUltMenor;  k
96  }
97
65  void quickSort2(int ini, int fin, int n, int b[n]){
66      int posPiv = 0;  k
67
68      if(ini < fin){  k
69          posPiv = particion(ini, fin, n, b);  k + TP(n) (particion 11kn +9k)
70          quickSort2(ini, posPiv - 1, n, b);
71          quickSort2(posPiv + 1, fin, n, b);  }  k + t(n-1)
72      }
73  }
```

La expresión obtenida es:

$$t(n) = 2k \qquad si\ n = 0$$

$$t(n) = t(n-1) + 11kn + 13k \qquad si\ n > 0$$

$$t(n) = 5.5kn^2 + 18.5kn + 2k \quad \textit{Ecuación desarrollada}$$

k: Tiempo que tarda el procesador en realizar una operación.

Para InsertionSort.

En el peor caso (los números están ordenados al revés).

```
 7  int InsertionSort(int n, int a[n]){
 8      int j, temporal;
 9      int iteraciones = 0; k
10      int i = 1; k
11      while (i < n){ k
12          iteraciones++;  2k
13          j = i; k     k          k          k
14          while (a[j] < a[j-1] && j-1 >= 0){
15              temporal = a[j-1]; k
16              a[j-1] = a[j];  k
17              a[j] = temporal; k
18              j--;   2k
19              iteraciones++;  2k
20          }
21          i++; 2k
22      }
23      return iteraciones;k
24  }
```

La expresión obtenida es:

$$t(n) = 10kn^2 + 7kn + 4k$$

k: Tiempo que tarda el procesador en realizar una operación.

Reemplazando n (número de datos) en las ecuaciones se obtuvo.

n	InsertionSort	QuickSort
200	401404*k	223702*k
400	1602804*k	887402*k
600	3604204*k	1991102*k
800	6405604*k	3534802*k
1000	10007004*k	5518502*k
1200	14408404*k	7942202*k
1400	19609804*k	10805902*k
1600	25611204*k	14109602*k
1800	32412604*k	17853302*k
2000	40014004*k	22037002*k

Tabla 1.2 Peor Caso Teórico

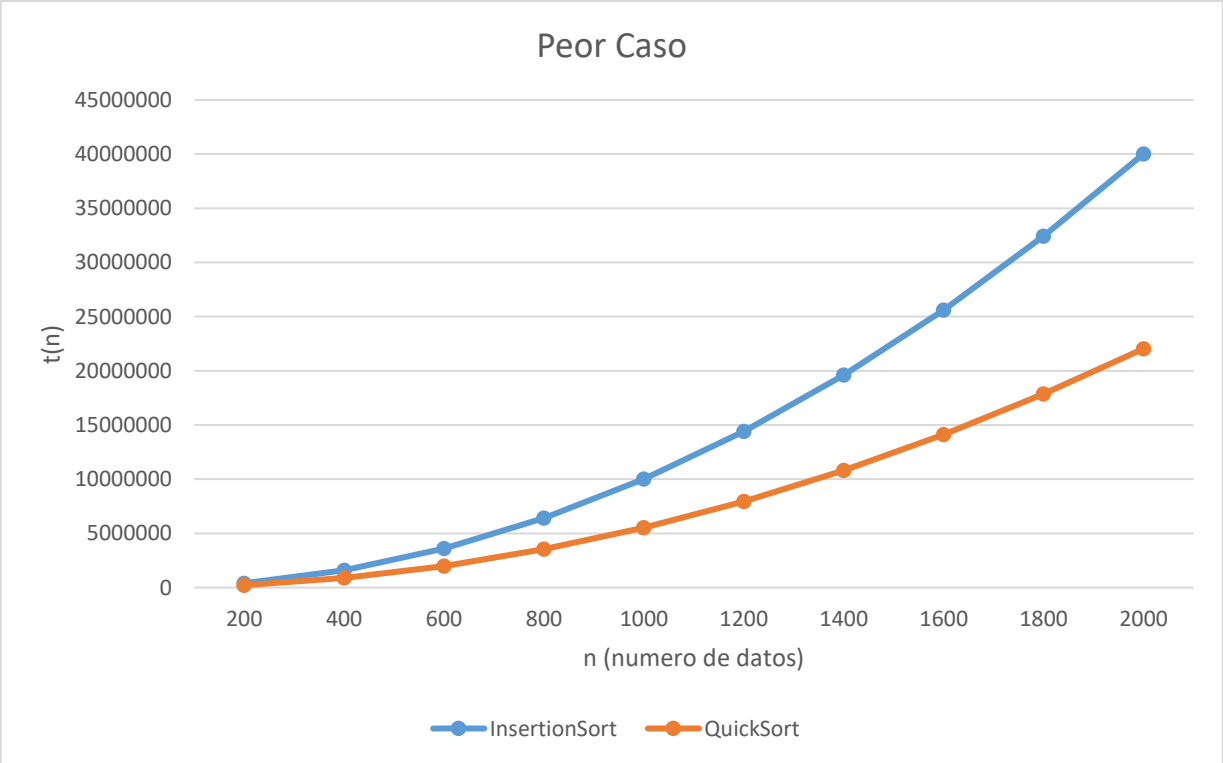

Grafico 1.2 InsertionSort vs Quicksort Peor Caso

Para QuickSort.

En el Caso Promedio (números en orden aleatorio).

```
75  int particion(int ini, int fin, int n, int b[n]){
76      int posUltMenor, k;
77      int temporal, pivote;
78      pivote = b[ini];  k
79      posUltMenor = ini;  k
80      k = ini + 1;  2k
81      while(k <= fin){  k
82          if(b[k] < pivote){  k
83              posUltMenor++;  2k
84              temporal = b[posUltMenor];  k
85              b[posUltMenor] = b[k];  k
86              b[k] = temporal;  k
87          }
88          iteraciones2++;  2k
89          k++;  2k
90      }
91          temporal = b[ini];  k
92          b[ini] = b[posUltMenor];  k
93          b[posUltMenor] = temporal;  k
94
95      return posUltMenor;  k
96  }
97
```

```
65  void quickSort2(int ini, int fin, int n, int b[n]){
66      int posPiv = 0;  k
67
68      if(ini < fin){  k
69          posPiv = particion(ini, fin, n, b);  k + TP( 9k + (n/2)*11k
70          quickSort2(ini, posPiv - 1, n, b);  } k + t(n-1)
71          quickSort2(posPiv + 1, fin, n, b);
72      }
73  }
```

La expresión obtenida es:

$$t(n) = 4.25kn^2 + 17.25kn + 2k$$

La ecuación promedio se obtuvo sumando la ecuación de mejor caso con la de peor caso y se dividió en dos

k: Tiempo que tarda el procesador en realizar una operación.

Para InsertionSort.

En el Caso Promedio (números en orden aleatorio).

```
7   int InsertionSort(int n, int a[n]){
8       int j, temporal;
9       int iteraciones = 0; k
10      int i = 1; k
11      while (i < n){k
12          iteraciones++; 2k
13          j = i;k      k        k        k
14          while (a[j] < a[j-1] && j-1 >= 0)
15              temporal = a[j-1];k
16              a[j-1] = a[j];k
17              a[j] = temporal; k
18              j--; 2k
19              iteraciones++; 2k
20          }
21          i++;2k
22      }
23      return iteraciones; k
24  }
```

La expresión obtenida es:

$$t(n) = 5kn^2 + kn + 4k$$

La ecuación promedio se obtuvo sumando la ecuación de mejor caso con la de peor caso y se dividió en dos

k: Tiempo que tarda el procesador en realizar una operación.

Reemplazando n (número de datos) en las ecuaciones se obtuvo.

n	InsertionSort	QuickSort
200	200204*k	173452*k
400	800404*k	686902*k
600	1800604*k	1540352*k
800	3200804*k	2733802*k
1000	5001004*k	4267252*k
1200	7201204*k	6140702*k
1400	9801404*k	8354152*k
1600	12801604*k	10907602*k
1800	16201804*k	13801052*k
2000	20002004*k	17034502*k

Tabla 1.3 Caso Promedio Teórico

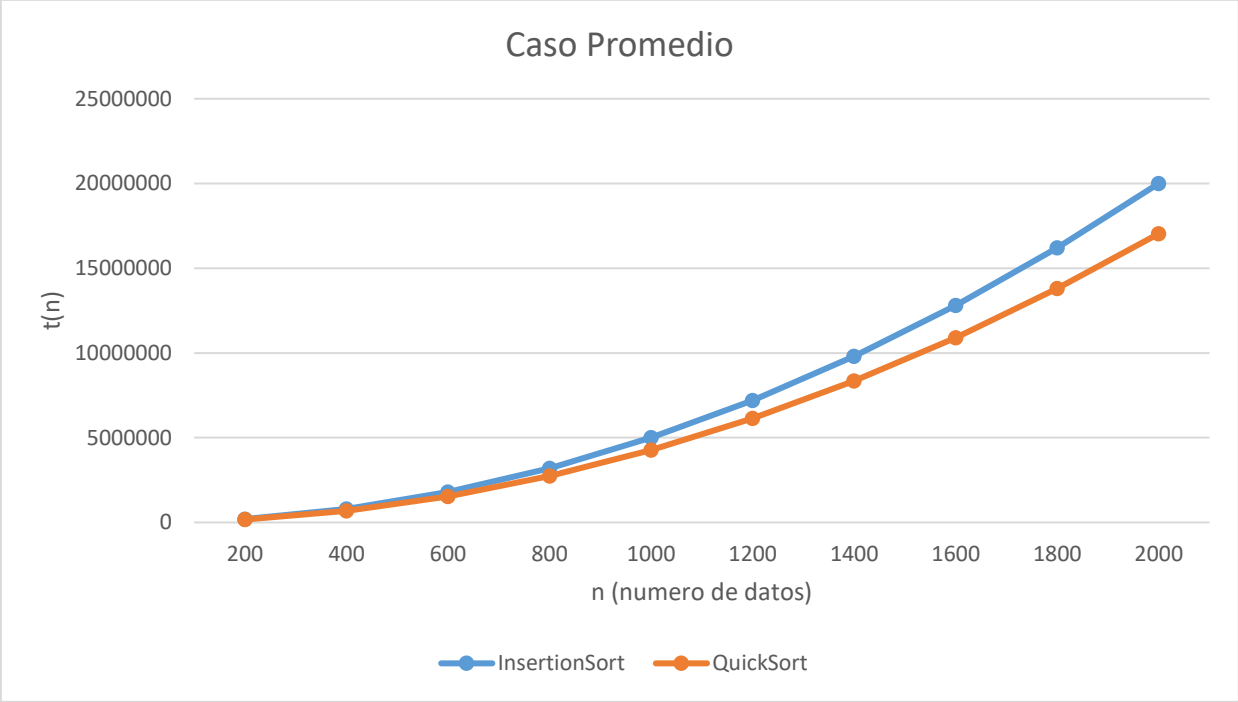

Grafico 1.3 InsertionSort vs Quicksort Caso Promedio Teórico

Resultados Empíricos

Se llamara resultado empírico al obtenido mediante un contador que se implementó en los algoritmos, el cual suma las iteraciones del procedo de ordenado de números, los datos en las tablas es la cantidad de iteraciones de cada algoritmo.

En el mejor caso (los números están ordenados).

n	QuickSort	InsertionSort
200	19900	199
400	79800	399
600	174700	599
800	314600	799
1000	494500	999
1200	714400	1199
1400	979300	1399
1600	1279200	1599
1800	1614100	1799
2000	1999000	1999

Tabla 2.1 Mejor Caso Empírico

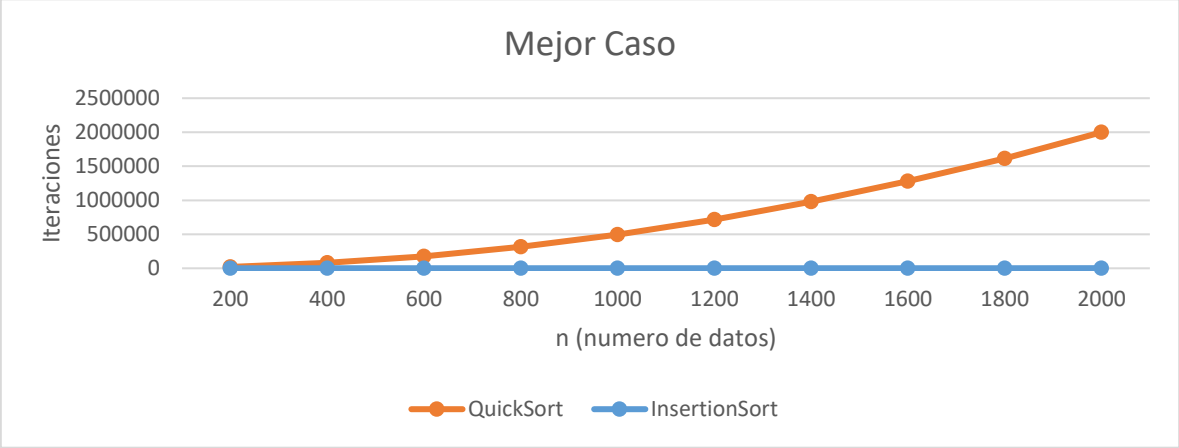

Grafico 2.1 InsertionSort vs Quicksort Mejor Caso Empírico

En el peor caso (los números están ordenados al revés).

n	QuickSort	InsertionSort
200	19900	2099
400	79800	80199
600	179700	180299
800	319600	320399
1000	499500	500499
1200	719400	720599
1400	979300	980699
1600	1279200	1280799
1800	1619100	1620899
2000	1999000	2000999

Tabla 2.2 Peor Caso Empírico

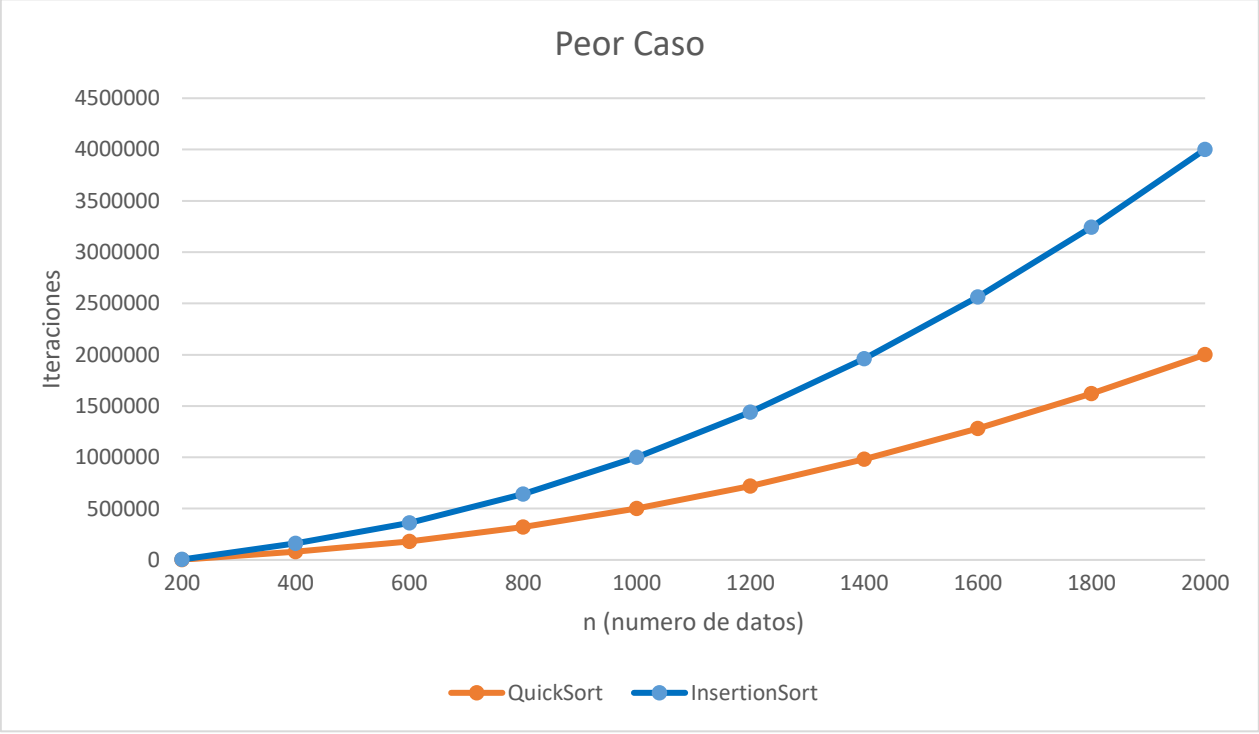

Grafico 2.2 InsertionSort vs Quicksort Peor Caso Empírico

En el Caso Promedio (números en orden aleatorio).

n	QuickSort	InsertionSort
200	1576	9715
400	3667	38661
600	5729	89882
800	8369	155998
1000	10863	256192
1200	13826	355828
1400	18257	493212
1600	18579	629389
1800	21273	813268
2000	24306	1029372

Tabla 2.3 Caso Promedio Empírico

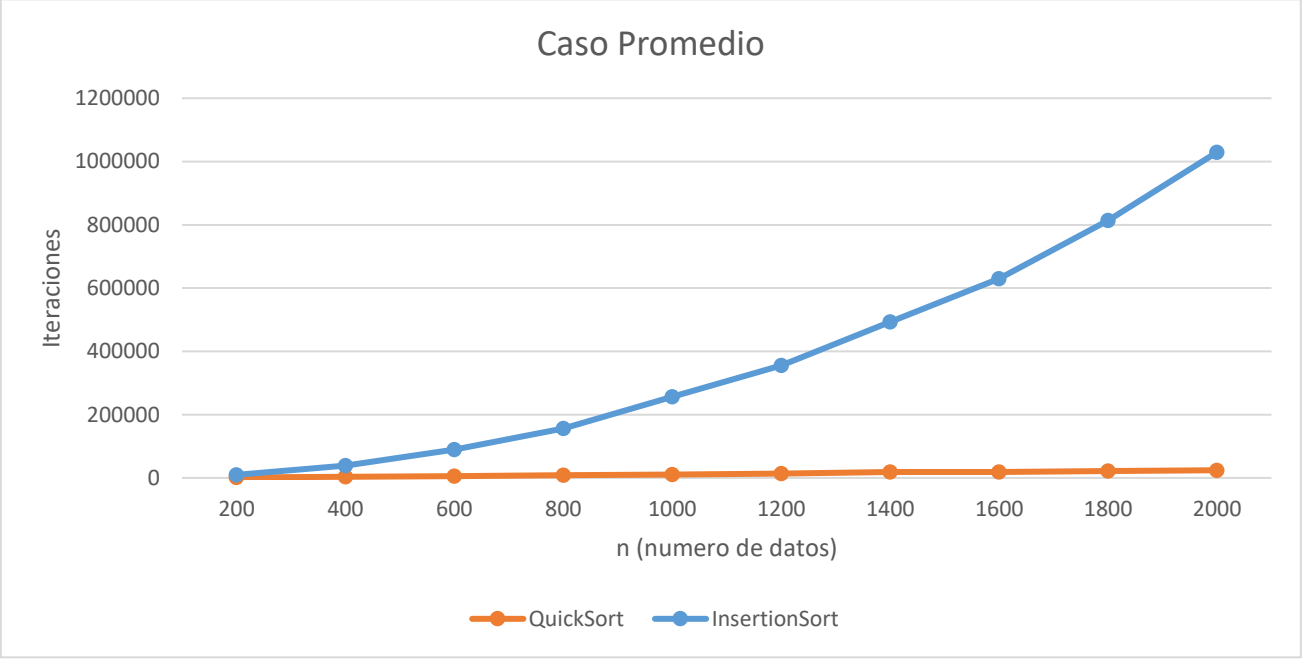

Grafico 2.3 InsertionSort vs Quicksort Caso Promedio Empírico

Análisis de Resultados

Pregunta 1: En términos teóricos ¿Cuál algoritmo es más eficiente en el mejor caso?

Observando lo datos de la tabla 1.1 y el grafico 1.1, se ve claramente un tiempo excesivamente mayor en el algoritmo QuickSort, por lo cual para el mejor caso, el algoritmo más eficiente es InsertionSort.

Pregunta 2: En términos teóricos ¿Cuál algoritmo es más eficiente en el peor caso?

Observando los datos de la tabla 1.2 y el grafico 1.2, podemos deducir que el algoritmo QuickSort es más eficiente que InsertionSort en el peor caso.

Pregunta 3: En términos teóricos ¿Cuál algoritmo es más eficiente en el caso promedio?

Observando los datos de la tabla 1.3 y el grafico 1.3, podemos ver que ambos algoritmos tienen una tendencia similar no obstante, QuickSort es más eficiente que InsertionSort, aunque la diferencia es muy poca.

Pregunta 4: En términos empíricos ¿Cuál algoritmo es más eficiente en el mejor caso?

Observando los datos de la tabla 2.1 y el grafico 2.1, podemos concluir que el algoritmo InsertionSort es más eficiente que QuickSort.

Pregunta 5: En términos empíricos ¿Cuál algoritmo es más eficiente en el peor caso?

Observando los datos de la tabla 2.2 y el grafico 2.2, podemos ver que el Algoritmo QuickSort es más eficiente que InsertionSort.

Pregunta 6: En términos empíricos ¿Cuál algoritmo es más eficiente en el caso promedio?

Observando los datos de la tabla 2.3 y el grafico 2.3, podemos concluir que el algoritmo QuickSort es mucho más eficiente que InsertionSort.

Pregunta 7: En definitiva ¿Cuál algoritmo de ordenamiento recomendaría?

Analizando los datos de las diferentes tablas, el algoritmo recomendado es QuickSort, ya que tiene un mejor rendimiento en datos aleatorios, y en su mayoría por probabilidades los datos que se ingresaran al programa no estarán ordenados por completo, ni completamente invertidos, por lo cual para la mayoría de los casos Quicksort sería más eficiente que InsertionSort