## **Definición del problema**

Evitar que el procesador principal tenga que realizar tareas de computo intensivo usando un coprocesador matemático, dicho coprocesador tendrá 3 algoritmos de ordenamiento como operaciones nativas.

Se deben encontrar 3 algoritmos de ordenamiento que puedan ordenar, números enteros de tamaño arbitrariamente grande y números en formato de coma flotante de cualquier tamaño, muy eficientemente.

Para eso se deben pasar los algoritmos por ciertas pruebas y especificar el tiempo que se demoran en ejecutar estos casos de prueba.

|  |  |
| --- | --- |
| Nombre | R1. Generar floats |
| Resumen | Generar de manera aleatoria números de tipo coma flotante |
| Entrada | * Cantidad de números * Booleano para saber si se pueden repetir números con el mismo valor * Identificador que determinara el orden en el que estarán los números generados |
| Salida | Números de tipo coma flotante |

|  |  |
| --- | --- |
| Nombre | R2. Generar enteros |
| Resumen | * Cantidad de números * Booleano para saber si se pueden repetir números con el mismo valor * Identificador que determinara el orden en el que estarán los números generados |
| Entrada | Cantidad de números |
| Salida | Números enteros |

|  |  |
| --- | --- |
| Nombre | R3. Ordenar valores ascendentemente |
| Resumen | Se debe ordenar un conjunto de valores de manera ascendente |
| Entrada | Valores a ordenar |
| Salida | Valores ordenados ascendentemente |

|  |  |
| --- | --- |
| Nombre | R4. Ordenar valores descendentemente |
| Resumen | Se debe ordenar un conjunto de valores de manera descendente |
| Entrada | Valores a ordenar |
| Salida | Valores ordenados descendentemente |

|  |  |
| --- | --- |
| Nombre | R5. Desordenar valores |
| Resumen | Desordenar valores según un % |
| Entrada | Coeficiente de desordenamiento del conjunto |
| Salida | Valores desordenados |

|  |  |
| --- | --- |
| Nombre | R6. Ingresar valores |
| Resumen | Ingresar valores manualmente a través de una interfaz grafica |
| Entrada | Valores |
| Salida | Ninguna |

|  |  |
| --- | --- |
| Nombre | R7. Visualizar datos |
| Resumen | Interfaz gráfica que permita ver los valores ingresados o generados aleatoriamente, además de mostrar el tiempo de ejecución de los algoritmos de ordenamiento que se usan para organizar ese conjunto de valores |
| Entrada |  |
| Salida | Interfaz |

## **Recopilación de información**

Para llevar a la mejor solución del problema se debe conocer qué tipo de complejidad temporal conviene más al momento de buscar los algoritmos de ordenamiento más eficientes.

|  |  |
| --- | --- |
| Complejidad | Terminología |
| O(1) | Complejidad constante |
| O(log n) | Complejidad logarítmica |
| O(n) | Complejidad lineal |
| O(n log n) | Complejidad n log n |
| O(nk) | Complejidad polinómica |
| O(kn) | Complejidad exponencial |
| O(n!) | Complejidad factorial |

Fig 1. Tabla de tipos de complejidad mas comunes.

En la tabla, se inicia desde las complejidades computacionales más eficientes hasta las menos óptimas. En el caso de la solución del problema se desea encontrar cualquier algoritmo que esté entre complejidad constate hasta complejidad n log n.

En cuanto a algoritmos de ordenamiento se tienen los siguientes tipos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algoritmo | Mejor caso | Peor caso | Caso más común | Memoria |
| Bubble sort | O(n2) | O(n2) | O(n2) | O(1) |
| Selection sort | O(n2) | O(n2) | O(n2) | O(1) |
| Insertion sort | O(1) | O(n2) | O(n2) | O(1) |
| Quick sort | O(n log n) | O(n2) | O(n log n) | O(1) |
| Merge sort | O(n log n) | O(n log n) | O(n log n) | O(n) |
| Heap sort | O(n log n) | O(n log n) | O(n log n) | O(1) |
| Counting sort | O(n+k) | O(n+k) | O(n+k) | O(n+2k) |
| Radix sort | O(n.k/s) | O(2s.n/k) | O(n.k/s) | O(n) |
| Bucket sort | O(n.k) | O(n2.k) | O(n.k) | O(n.k) |
| Bogo sort | O(n!) | Doesn´t finish | O(n.n!) |  |