

# Verificación y Validación del Software

## Práctica 4: Reporte Sonarlint de análisis estático

### Alumno:

❖ Coronel Vilca, Brisa Valeria.

## Práctica 4

### 1. Módulo

El presente módulo, desarrollado en el marco de un proyecto sobre programación paralela y concurrente, presenta un estudio comparativo del rendimiento del algoritmo Heapsort, tanto en su versión secuencial como paralela, al ordenar un conjunto de un millón de elementos. Se procede a la evaluación del código una vez implementadas todas las correcciones detalladas en el archivo “README”

#### Antes de la Refactorización

```

HeapSortParallel.py 4 X
HeapSortParallel.py > main
34 def parallel_heap_sort(arr, left, right, depth=0):
35
60     # while i < len(left_arr) and j < len(right_arr):
61     #     if left_arr[i] < right_arr[j]:
62     #         arr[k] = left_arr[i]
63     #         i += 1
64
65     while i < len(left_arr):
66         arr[k] = left_arr[i]
67         i += 1
68         k += 1
69
70     while j < len(right_arr):
71         arr[k] = right_arr[j]
72         j += 1
73         k += 1
74
75 # Función principal
76 def main():
77     num_iterations = 1
78     parallel_times = [0.0] * num_iterations
79     serial_times = [0.0] * num_iterations
80
81     # Secuencial
82     for i in range(num_iterations):
83         with open("numeros_binarios.txt", "rb") as file:
84             arr = list(struct.unpack("i" * (os.path.getsize("numeros_binarios.txt") // 4), file.read()))
85
86             start_serial = time.time()
87             heap_sort(arr)
88             end_serial = time.time()
89             serial_times[i] = end_serial - start_serial
90
91     # Paralelo
92     for i in range(num_iterations):
93         with open("numeros_binarios.txt", "rb") as file:
94             arr = list(struct.unpack("i" * (os.path.getsize("numeros_binarios.txt") // 4), file.read()))
95
96             start_parallel = time.time()
97             parallel_heap_sort(arr, 0, len(arr) - 1)
98             end_parallel = time.time()
99
100
PROBLEMS 16 OUTPUT DEBUG CONSOLE TERMINAL PORTS
HeapSortParallel.py 16
Unused import math Pylint(W0611:unused-import) [Ln 3, Col 1]
Rename function "Heapify" to match the regular expression ^[a-z_][a-z0-9_]*$. sonarlint(python:S1542) [Ln 8, Col 5]
Remove this commented out code. sonarlint(python:S125) [Ln 60, Col 9]
Define a constant instead of duplicating this literal "numeros_binarios.txt" 4 times. [+3 locations] sonarlint(python:S1192) [Ln 83, Col 19]

```

## Después de la Refactorización

```

heap_sort_parallel.py X
heap_sort_parallel.py > parallel_heap_sort
25
26 # Función para realizar heap sort
27 def heap_sort(arr):
28     n = len(arr)
29     for i in range(n // 2 - 1, -1, -1):
30         heapify(arr, n, i)
31
32     for i in range(n - 1, 0, -1):
33         arr[i], arr[0] = arr[0], arr[i]
34         heapify(arr, i, 0)
35
36 # Función para hacer el heap sort en paralelo
37 def parallel_heap_sort(arr, left, right, depth=0):
38     max_depth = 9
39     if depth >= max_depth:
40         heap_sort(arr[left:right + 1])
41     else:
42         mid = left + (right - left) // 2
43         left_arr = arr[left:mid + 1]
44         right_arr = arr[mid + 1:right + 1]
45
46         left_thread = threading.Thread(
47             target=parallel_heap_sort,
48             args=(left_arr, 0, len(left_arr) - 1, depth + 1)
49         )
50         left_thread.start()
51         parallel_heap_sort(right_arr, 0, len(right_arr) - 1, depth + 1)
52
53         left_thread.join()
54
55         i = j = 0
56         k = left
57         while i < len(left_arr) and j < len(right_arr):
58             if left_arr[i] < right_arr[j]:
59                 arr[k] = left_arr[i]
60                 i += 1
61             else:
62                 arr[k] = right_arr[j]
63                 j += 1

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

No problems have been detected in the workspace.