Assignment 3 - Profilling with GPROF

João Pedro Almeida Santos Secundino jp.secundino@usp.br
10692054

The objective of this assignment was to analyze algorithms with different complexities using the *GPROF* tool. The algorithms were analyzed and the results generated by this tool, being shown graphically with the help of *gprof2dot*, a program that transforms the .dot generated by *GPROF* into a visual graph to facilitate the analysis.

1. Analysed Algorithms

The analysed algorithms were the famous:

- Radix Sort (with Bucket Sort) - $O(n)$
- Heap Sort - $O(nlogn)$
- Merge Sort - $O(nlogn)$
- Bubble Sort - $O(n^2)$

Each of these algorithms was executed after a function call that clears the cache for them. They were also analyzed from the perspective of three input sizes: 1,000, 10,000, 100,000. A code snippet that shows the routine used in the analysis is shown below:

```
...
    fill(v, VEC_SIZE); // fills an array with values between 0 and array.size
    cache_clean(); // cleans the cache before running sorting algorithm
    radixsort(v, VEC_SIZE);//O(n)
    printf("radix");

    fill(v, VEC_SIZE); // fills an array with values between 0 and array.size
    cache_clean(); // cleans the cache before running sorting algorithm
    heapsort(v, VEC_SIZE);//O(nlogn)
    printf("heap");

    fill(v, VEC_SIZE); // fills an array with values between 0 and array.size
    cache_clean(); // cleans the cache before running sorting algorithm
    mergesort(v, 0,VEC_SIZE);//O(nlogn)
    printf("merge");

    fill(v, VEC_SIZE); // fills an array with values between 0 and array.size
    cache_clean(); // cleans the cache before running sorting algorithm
    bubblesort(v, VEC_SIZE);//O( n  )
    printf("bubble");
...
```

Listing 1: main.c Snippet

To make sure that only the important routines were shown in the final gragh, even that ones that got insignificant processing time, the code was compiled with the following commands:

```
gcc main.c sorting.c -o prog.exe -pg -g
./prog.exe
gprof prog.exe gmon.out -Pcache_clean -Pfill -Qcache_clean -Qfill > chart.prev
gprof2dot --node-thres=0.0 --edge-thres=0.0 chart.prev > chart.dot
dot -Tpng -o chart.png chart.dot
```

Listing 2: Compilation Directives

2. Results

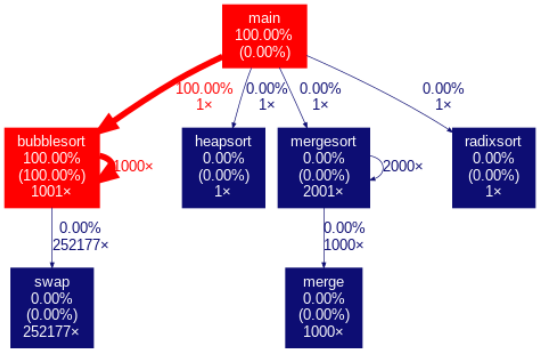

Figure 1: N = 1.000

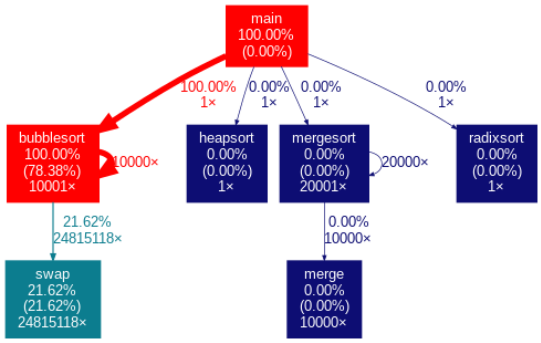

Figure 2: N = 10.000

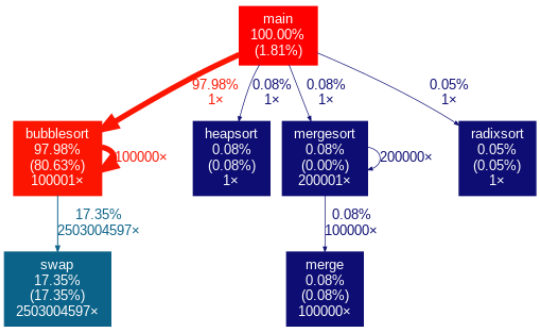

Figure 3: N = 100.000

As expected, the analysis shows that the algorithm that took the longest to process the input was Bubble Sort, whose complexity is $O(n^2)$. It was followed by Heap Sort and Merge Sort ($O(nlogn)$, with almost the same performance) and, finally, Radix Sort ($O(n)$, whose processing time only became significant with the largest input size.+

Is important to note that in the first two figures, Bubble Sort was protagonist in terms of processing time compared to the other algorithms, whose percentage in the graph appears as 0%. This protagonism remained true in Figure 3, but the processing time of the other algorithms in it showed some sign of life. This is due to the precision of the GPROF tool: it only has two decimal places of precision to count the execution time.

Important: Note that some functions were omitted to make analysis more clear such as "cache_clean" and "fill" routines.