Hash-table performance

And how you can improve it

What affects performance?

Generally,

1) Choice of hash function
2) Search implementation

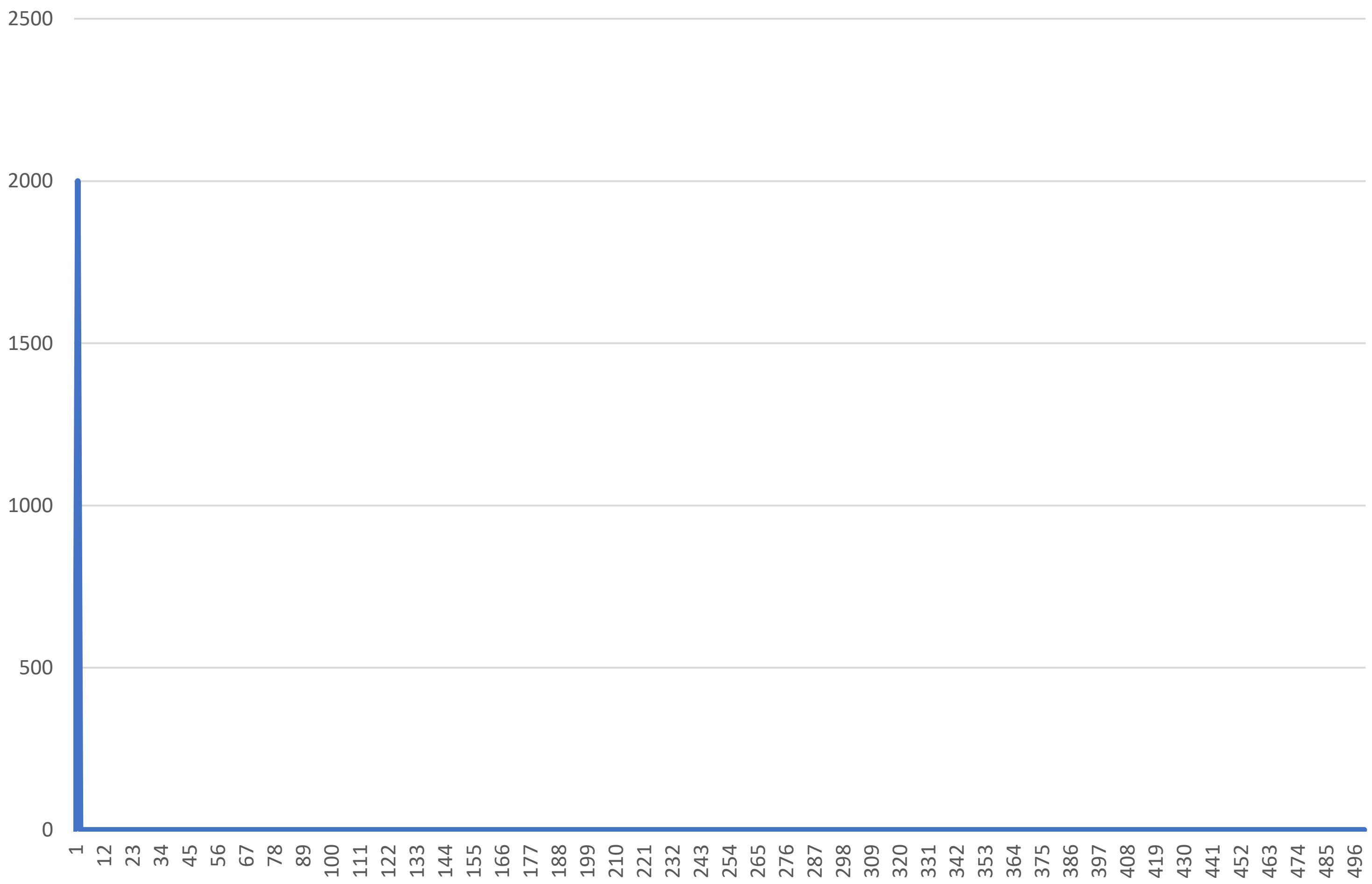
The easiest thing you can do is choose the good hash function

Good hash provides an equal distribution

Let’s take a look

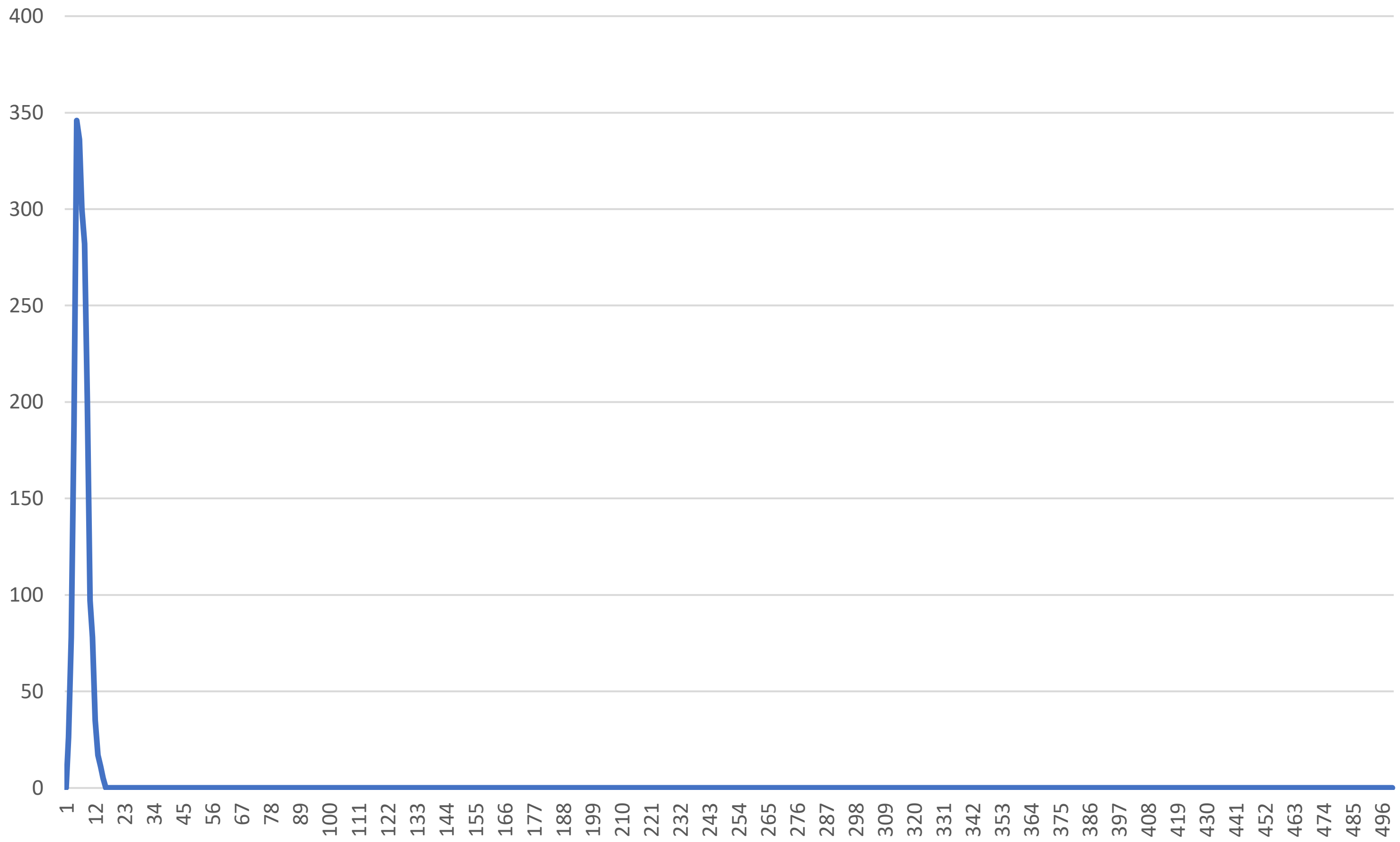
All to one hash

The worst one



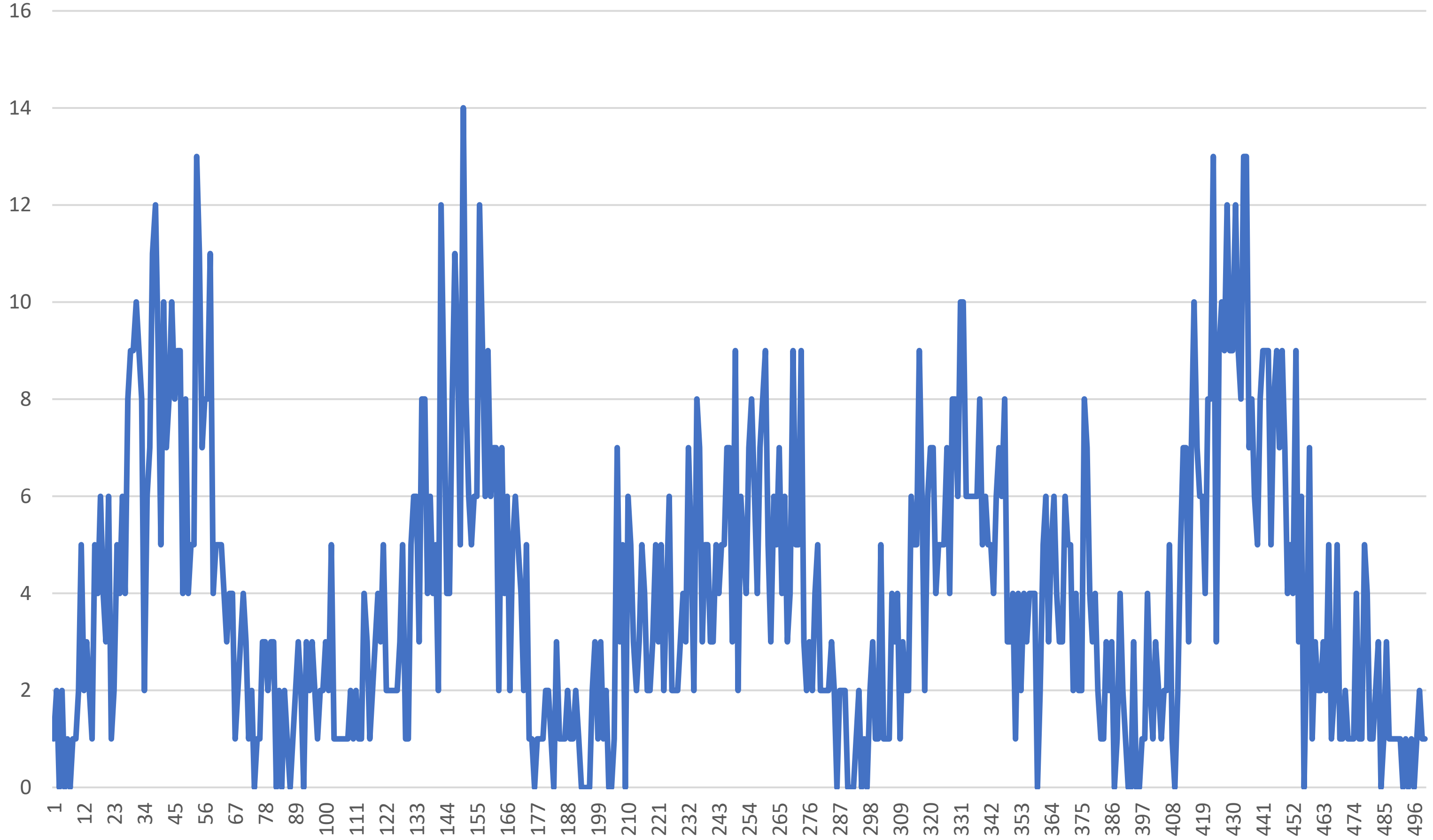
Word length hash

Still very bad



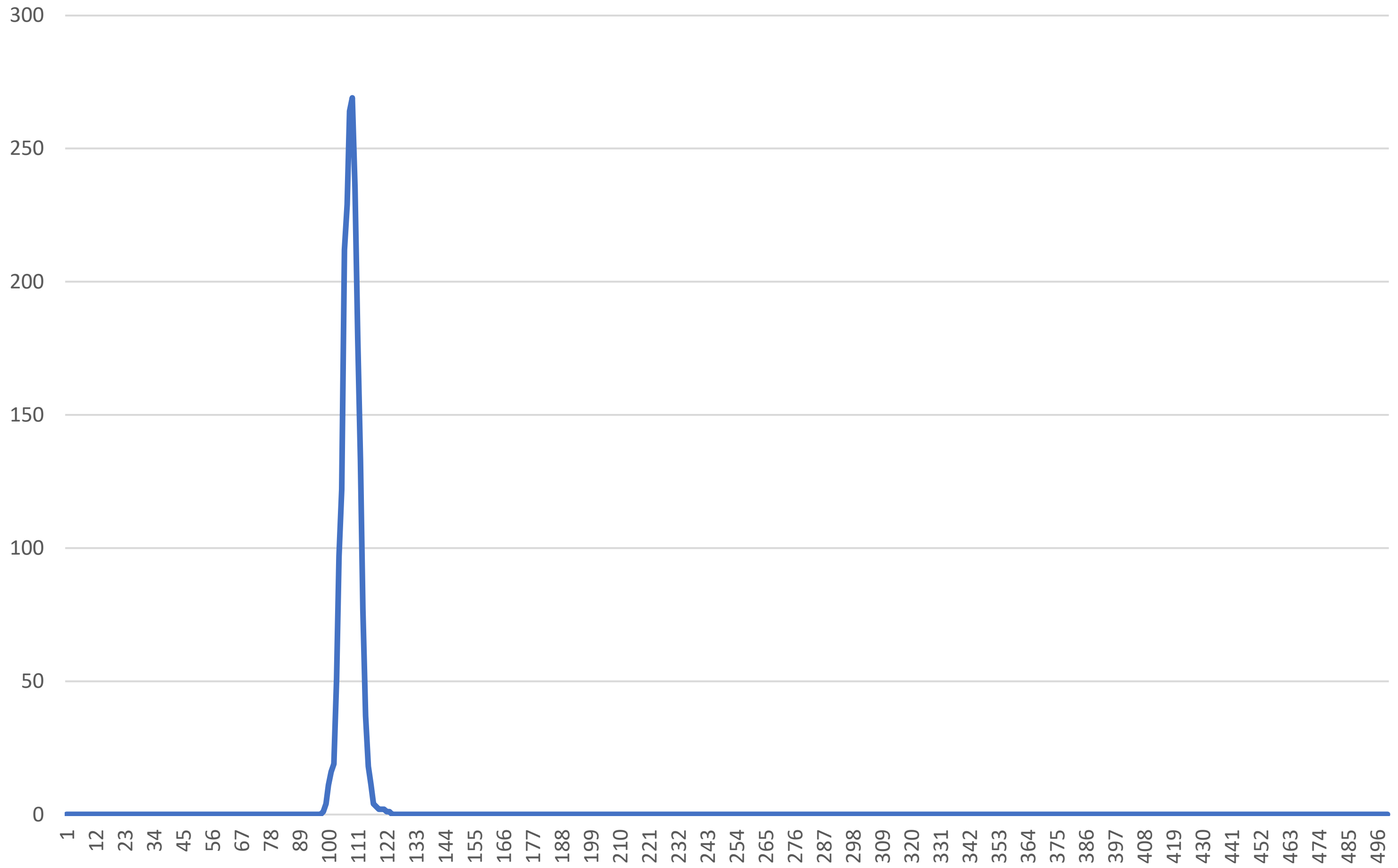
Ascii-sum hash

Slightly better



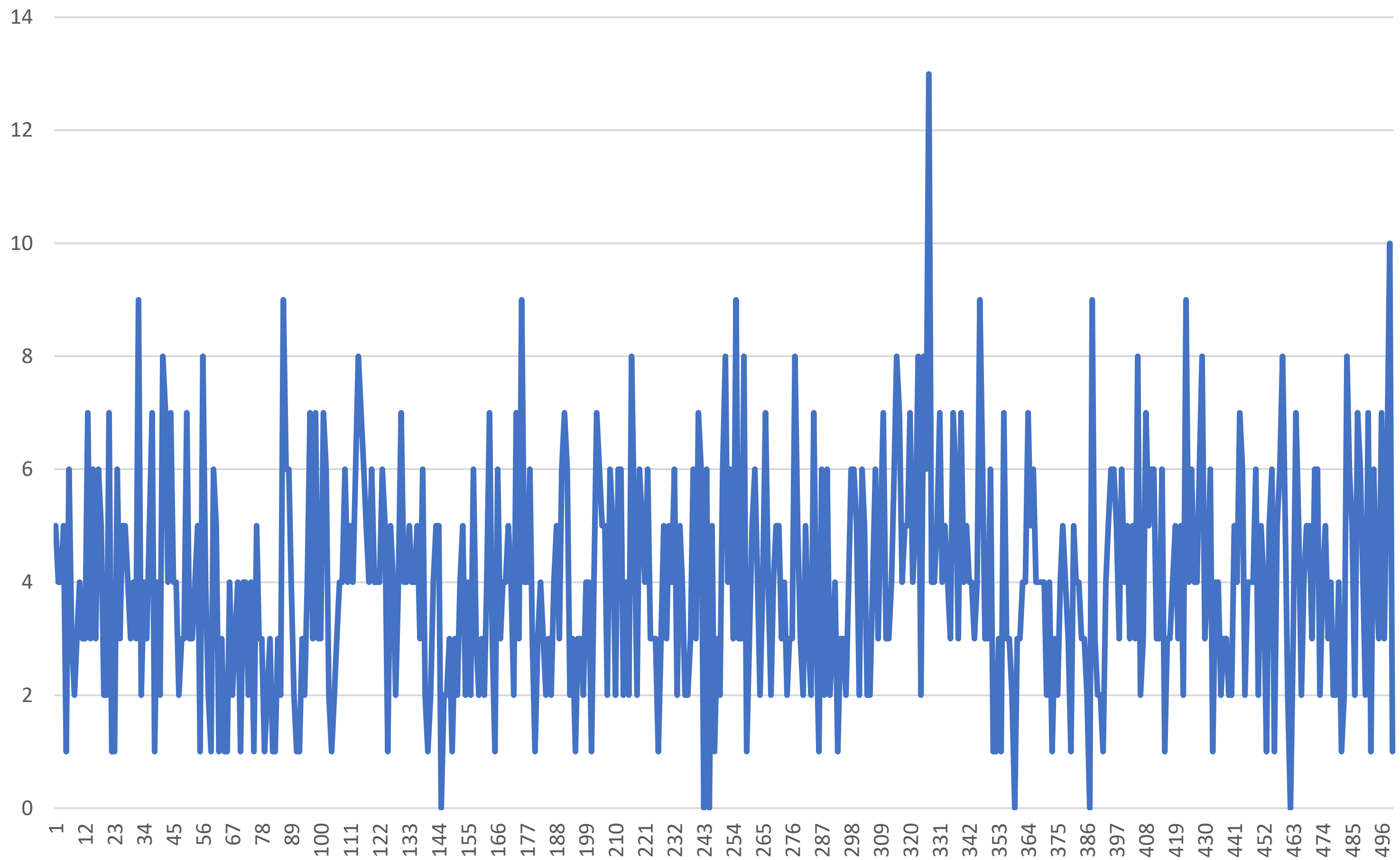
Ascii-sum divided by word length hash

Terrible again



Xor hash

Much better!



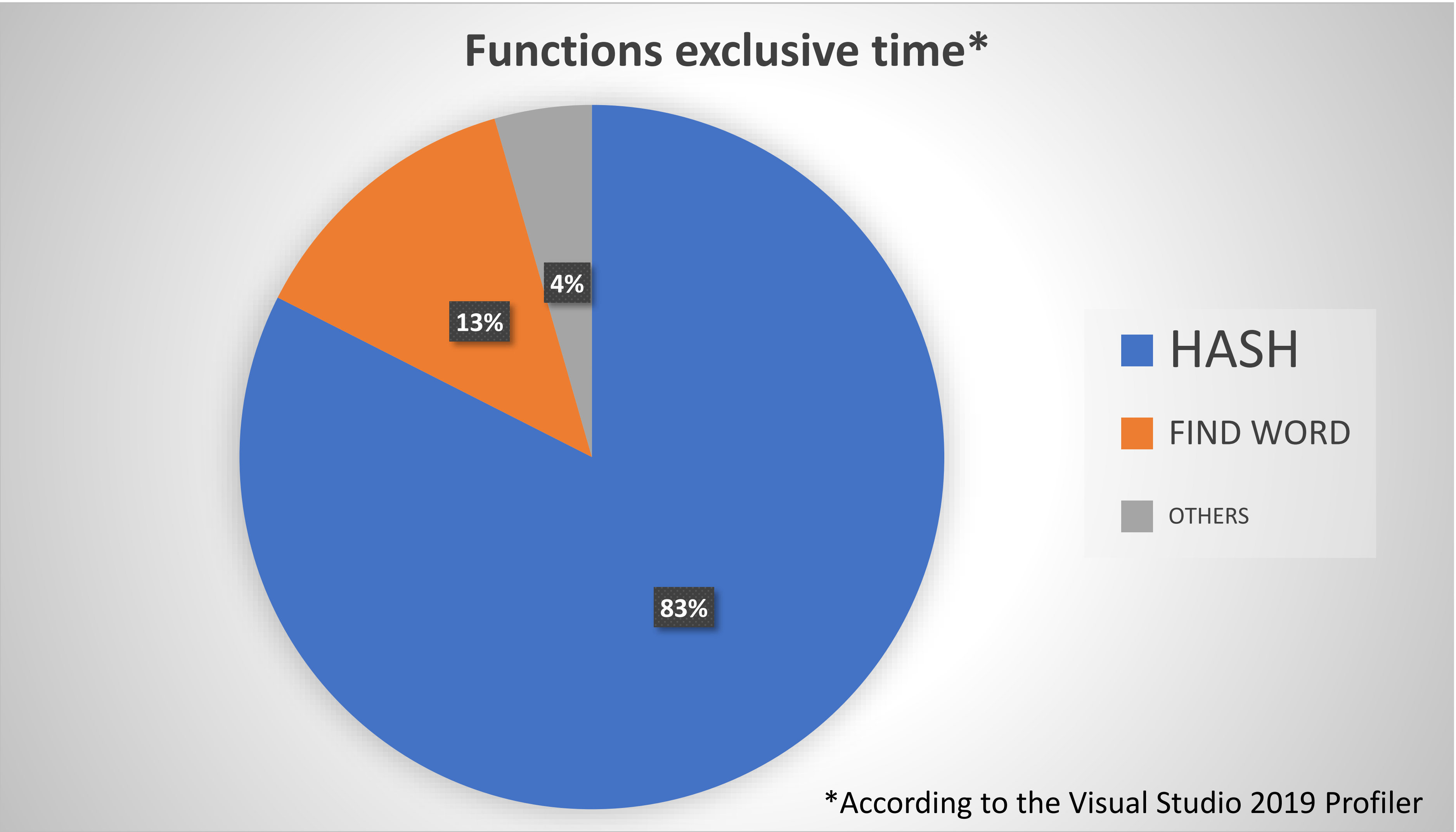
And what’s next?

Now when we choose the right hash-func, lets go further and try to optimize it.

I’m going to show the assembly optimization.

As we’ll see even not very complicated one can improve the performance pretty well.

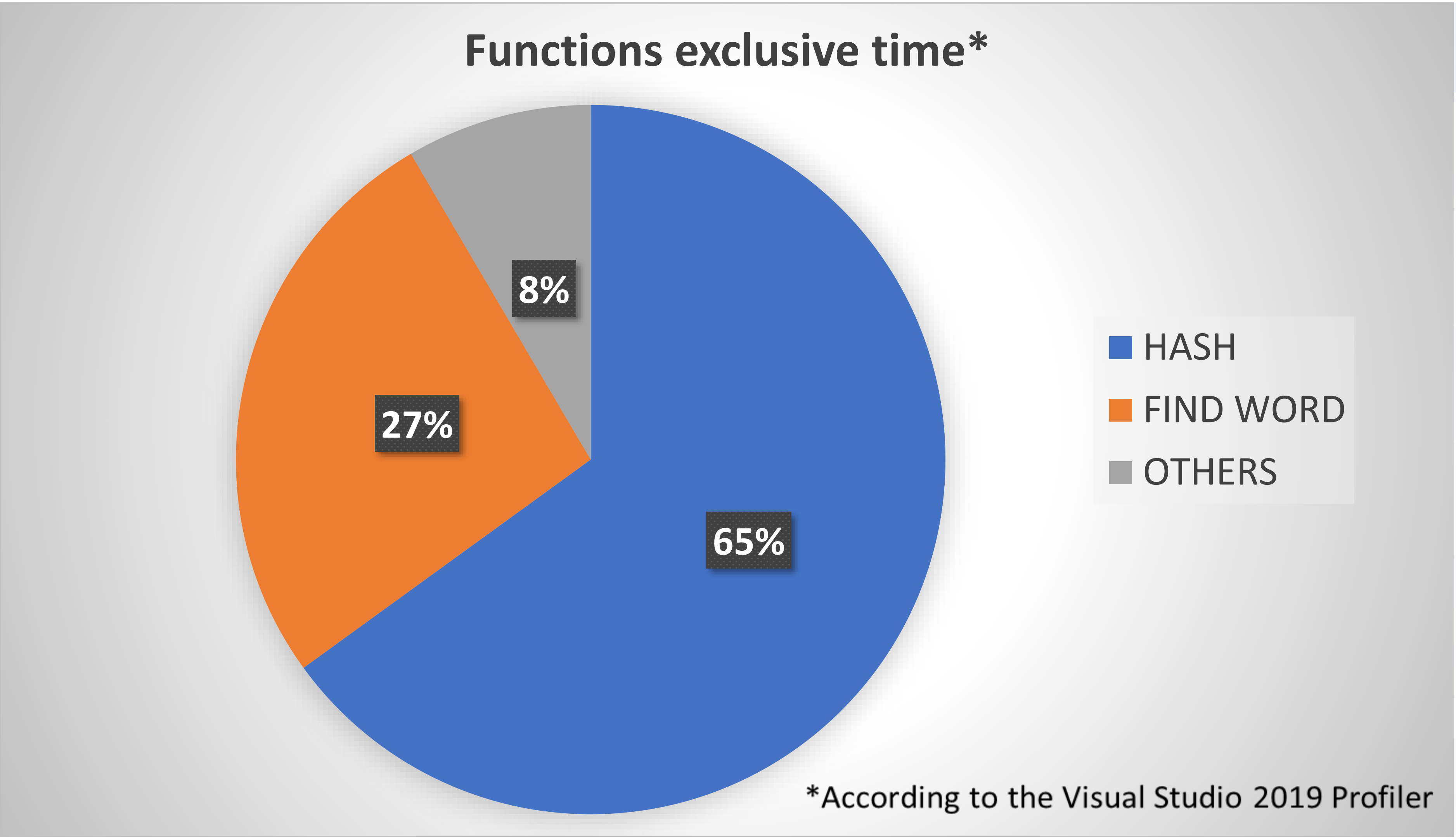
Before optimization



*According to the Visual Studio 2019 Profiler

Total time, ms
21640
21710
21430
21530
21650
Average: 21590

After optimization

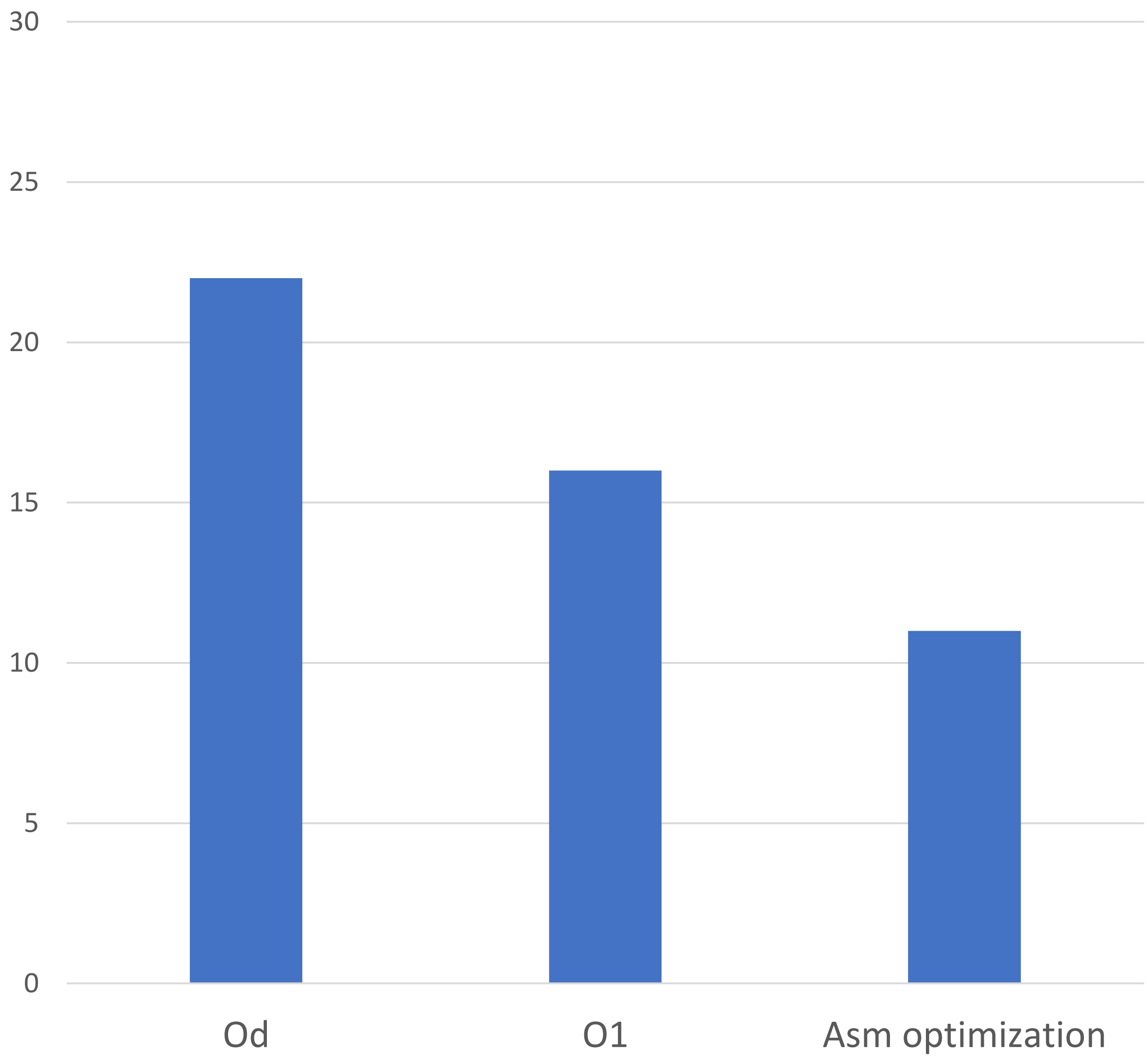


*According to the Visual Studio 2019 Profiler

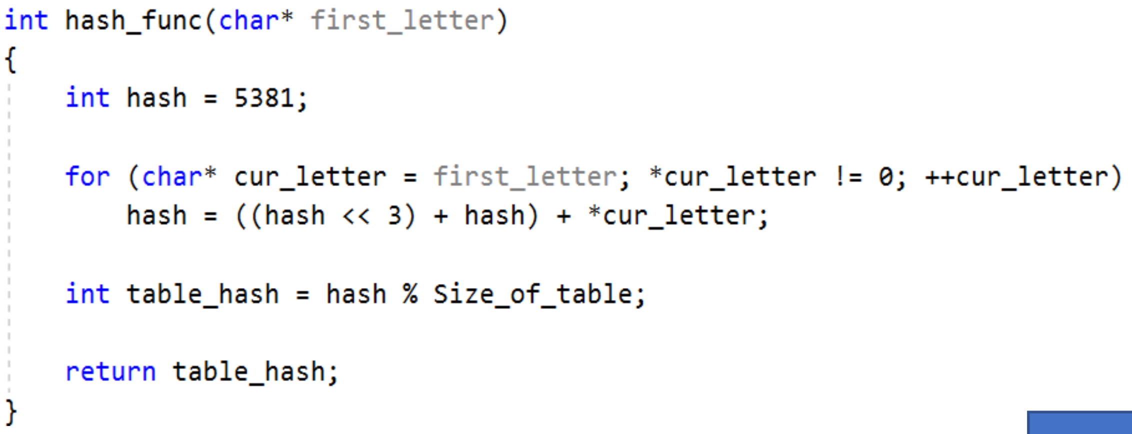
Total time, ms
10360
10320
10340
10320
10440
Average: 10356

Time comparison

- 2 times faster than unoptimized code
- 1.4 times faster than -O1 optimization



Details

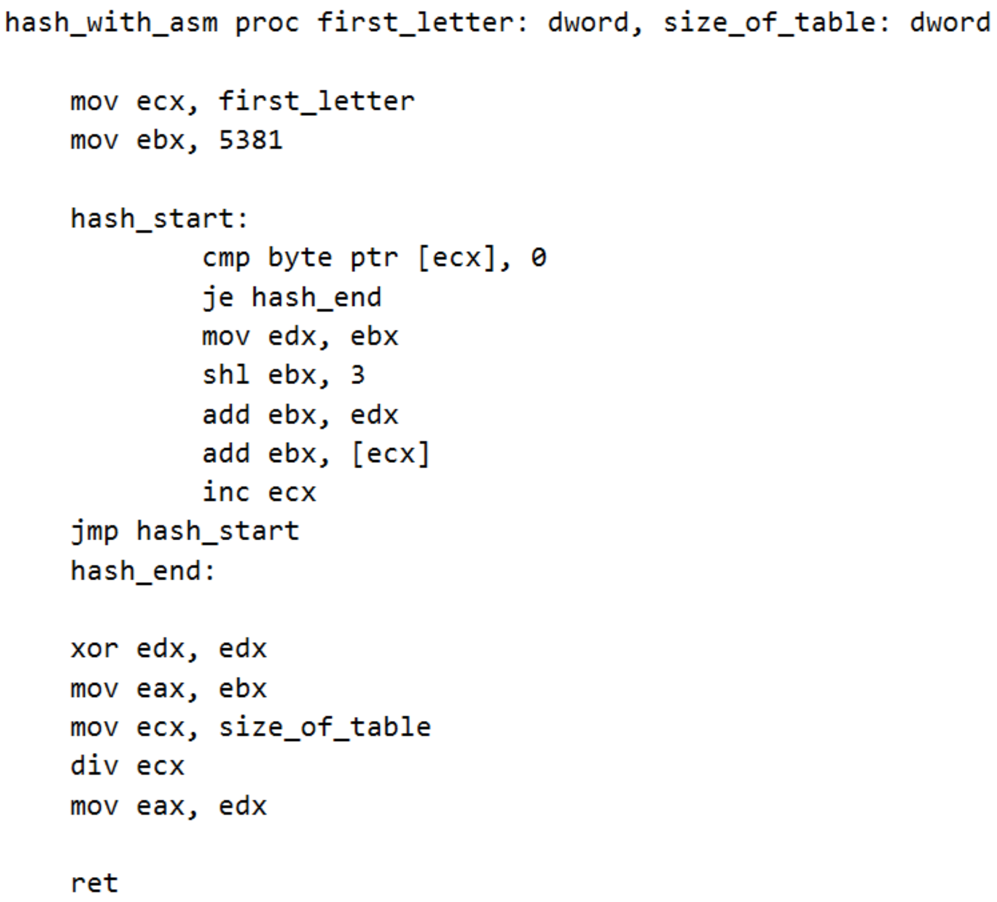


```c
int hash_func(char* first_letter)
{
    int hash = 5381;

    for (char* cur_letter = first_letter; *cur_letter != 0; ++cur_letter)
        hash = ((hash << 3) + hash) + *cur_letter;

    int table_hash = hash % Size_of_table;

    return table_hash;
}
```



```
hash_with_asm proc first_letter: dword, size_of_table: dword

    mov ecx, first_letter
    mov ebx, 5381

    hash_start:
            cmp byte ptr [ecx], 0
            je hash_end
            mov edx, ebx
            shl ebx, 3
            add ebx, edx
            add ebx, [ecx]
            inc ecx
    jmp hash_start
    hash_end:

    xor edx, edx
    mov eax, ebx
    mov ecx, size_of_table
    div ecx
    mov eax, edx

    ret
```

Thank you