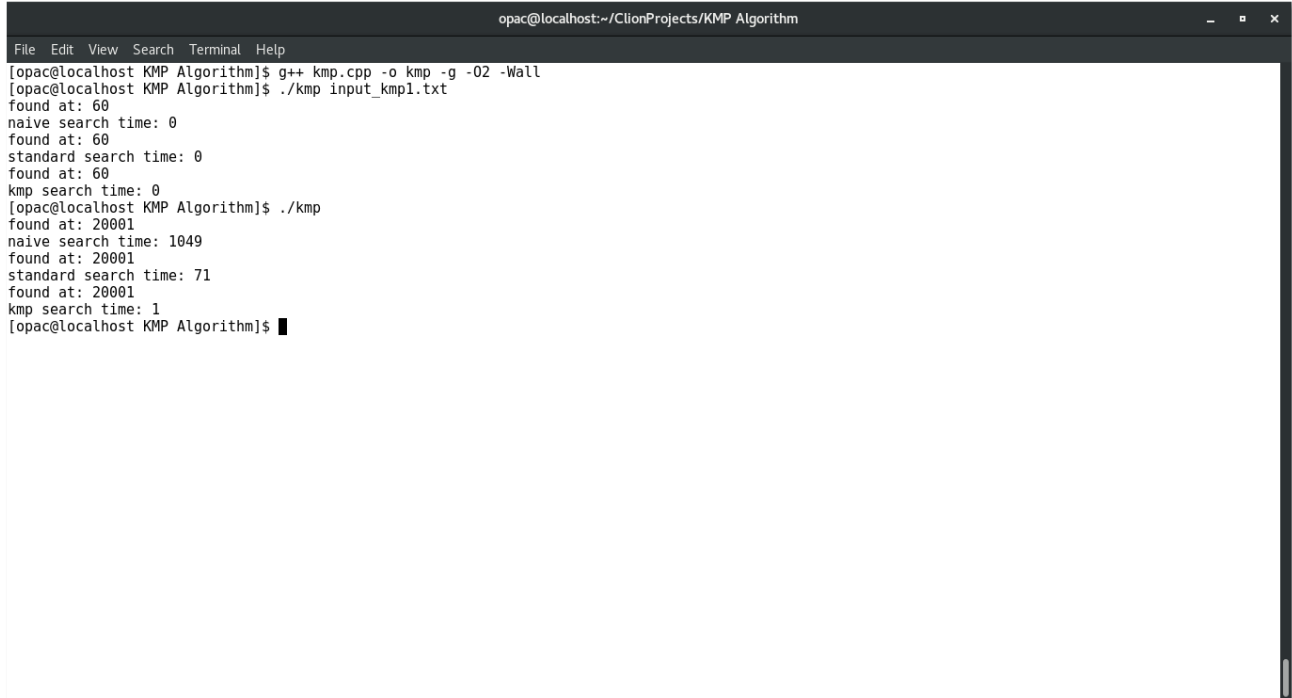


## KMP Performance

A terminal window titled 'opac@localhost:~/ClionProjects/KMP Algorithm' showing the execution of a C++ program. The user runs 'g++ kmp.cpp -o kmp -g -O2 -Wall' and then './kmp input\_kmp1.txt'. The output shows three test cases. In the first, both naive and standard search find a match at index 60 with 0 time. In the second, naive search takes 1049 seconds, standard search takes 71 seconds, and KMP search takes 1 second. In the third, KMP search takes 1 second.

```
opac@localhost:~/ClionProjects/KMP Algorithm
File Edit View Search Terminal Help
[opac@localhost KMP Algorithm]$ g++ kmp.cpp -o kmp -g -O2 -Wall
[opac@localhost KMP Algorithm]$ ./kmp input_kmp1.txt
found at: 60
naive search time: 0
found at: 60
standard search time: 0
found at: 60
kmp search time: 0
[opac@localhost KMP Algorithm]$ ./kmp
found at: 20001
naive search time: 1049
found at: 20001
standard search time: 71
found at: 20001
kmp search time: 1
[opac@localhost KMP Algorithm]$
```

The strings I am using to show that KMP algorithm works faster than in-built `std::find()` function in C++ and naïve  $O(nm)$  algorithm are:

**String:** aaaaaaaaaaaaaaaaaaaaaa...aaaaaaaaaaaaaaaaaab  
**Pattern:** aaaaaaaaaaaa...aaaaaaaaaaaaaab

String is of length 100002 where 'a' is repeating 100001 times and the last character is 'b'.  
Pattern is of length 80002 where 'a' is repeating 80001 times and the last character is 'b'.

KMP algorithm is better than the other two, because KMP algorithm figures out where the character match fails and reuses the work done so far and then continues. This is the case where KMP saves a lot of redundant work as compared to the other two. That's the reason why KMP takes 1 sec to get the output, while naïve algorithm takes too much time (1049 sec) and so the `std::find` (71 sec).

In the above example, KMP computes the prefix table and skips to the character where the match failed. So, in here, up to the position 80001, all the algorithms go on matching the pattern with the string sequentially. When they come across the last character 'b' in the pattern at position 80002 (1 indexed not 0), then naïve algorithm will start from the 2<sup>nd</sup> character the string. Again, it will fail and it will shift the pattern back to the 3<sup>rd</sup> character and so on. So, it takes  $O(mn)$  time. What KMP does here is it uses the value from the prefix table and that value here is almost equal to the current position as there are all 'a's till now. So, it doesn't need to go back right to the start. It saves a lot of computation.

In conclusion, for the strings where the pattern has very much similar sequence of characters at the start and at the end, KMP works better. As it will have high prefix value of the characters and it can use a lot of the work it has done previously.