$O(n) = O(\frac{n}{2})$ or $n \neq 4$ ← dominant term

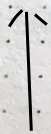
"to" = in terms of

BUT

$O(\frac{n}{2})$ is, in reality, twice as fast as $O(n)$ in clock time

Make spell checker

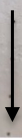
$\frac{7}{3}$ ~~$\frac{8}{3}$~~ 1



Brief notes on asymptotic time vs real world time

A breakdown, in my own words, of speller.c

Next Page



Note: if (a>5)
{
result = 3
{
else
{
result = 4
{

is result = a>5 ? 3 : 4

(1) ?: → Known as the ternary operator

(2) struct rusage contains ~~data types~~ variables pertaining to CPU usage

So struct rusage before, after;

simply ~~declares~~ refers to sets of the data, how much CPU is used before and after the program runs.

(3) getrusage is a function in sys/resource.h takes 2 parameters:

Sum of resources used by function → a pointer to rusage struct

1 int who → RUSAGE_SELF

2 struct rusage *usage

getrusage returns 0 when successful and -1 when failed

AND since rusage is a pointer

(3) continued all the variables in that structure get updated when getrusage is called

(4) we use getrusage before loading the dictionary and after loading the dictionary, presumably to [redacted] track efficacy during that process

(5) calculate(&before, &after)

* by passing pointers we can directly interface w/ the values stored at address of before.

a. if these pointers are NULL exit

b. otherwise, perform the following arithmetic and return result

* All converted to [redacted] Seconds

(After) UserCPU [redacted] (seconds) + UserCPU [redacted] (micro-)

(Before) UserCPU [redacted] (seconds) + UserCPU [redacted] (micro)

(After) KernelCPU [redacted] (seconds) + [redacted] (micro) -

(Before) [redacted]

(5) continued calculate(&a, &b) returns the total runtime for loading the dictionary

(6) The Fread loop scans the document entered as an argument and sends words one at a time into the spellchecker function, checking to make sure it's not reading numbers

Load load dictionary into memory, i.e.

Pseudocode

word \n
word \n
word \n
word
word

1. open dictionary
2. read from dictionary into buffer
3. pass buffer through hash function
4. malloc node at hash index
   e.g. if hash is 3 malloc node here
5. set char value at malloced node to word in buffer
6. Go to step 2