1. Analysis Tools

① $\begin{cases} d(n) = O(f(n)) \text{ iff } \exists c, n_0 \text{ s.t. } d(n) \le c \cdot f(n), \forall n > n_0 & -① \\ e(n) = O(g(n)) \text{ iff } \exists c', n_0' \text{ s.t. } e(n) \le c' g(n), \forall n > n_0' & -② \end{cases}$

$\therefore ① \times ② : d(n) \cdot e(n) \le (c \cdot c') \cdot f(n) \cdot g(n), \forall n > n_0 \cdot n_0'$

$\Rightarrow$ Thus for $c \cdot c'$ and $n > n_0 \cdot n_0' \rightarrow d(n) \cdot e(n) \le (c \cdot c') f(n) g(n)$

$\Rightarrow d(n) \cdot e(n) = O(f(n) \cdot g(n))$ QED.

② Set $\deg(p) = k \Rightarrow p(n) = a \cdot n^k + bn^{k-1} \cdots \Rightarrow \lim_{n\to\infty} \frac{n^k}{p(n)} \le \frac{1}{a}$ for $a > 0$.

$\therefore p(n) = O(n^k) \Leftrightarrow p(n) \le c \cdot n^k, \exists c, n_0, \forall n > n_0 \Rightarrow \log p(n) \le \log c \cdot n^k = \log c + k \cdot \log n$

$\rightarrow \exists c', n_0' \text{ s.t. } \log p(n) \le \log c + k \cdot \log n \le c' \cdot \log n, \forall n > n_0'$ ($\log c$ is constant).

Thus, $\log p(n) = O(\log n)$ QED.

③ $1 < f(n) \Rightarrow 0 < f(n) - 1 < \lceil f(n) \rceil \le f(n) \Rightarrow \frac{f(n)-1}{f(n)} < \frac{\lceil f(n) \rceil}{f(n)} \le \frac{f(n)}{f(n)}$

$\because \lim_{n\to\infty} \frac{f(n)-1}{f(n)} = \lim_{n\to\infty} \frac{f(n)}{f(n)} = 1 \quad \therefore$ By Squeeze Theorem $\therefore \lim_{n\to\infty} \frac{\lceil f(n) \rceil}{f(n)} = 1$

$\Rightarrow$ for $c = 1$ s.t. $\lceil f(n) \rceil \le c \cdot f(n), \forall n \in R$

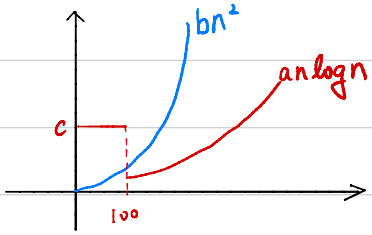
Thus $\lceil f(n) \rceil = O(f(n))$

④ There may be different kinds of reasons. For example, AI's algorithm might need more time for dynamic allocation or memory modification, etc.

So consider AI's algorithm to be $f(n) = \begin{cases} c, & n < 100 \\ a \cdot n \log n, & n \ge 100 \end{cases}$

Bob's algorithm to be $g(n) = bn^2$

So, chances that $f(n) > g(n)$ for $n < 100$ and $g(n) > f(n)$ for $n \geq 100$ holds.



$$\begin{cases} \text{for } n < 100 \Rightarrow c > bn^2 \rightarrow \text{Bob is faster} \\ \text{for } n \geq 100 \Rightarrow bn^2 \geq a \cdot n\log n. \rightarrow \text{Al is faster.} \end{cases}$$

However, time complexity only discribes how the size of data affect the algorithm; the constant still plays a significant role in execution time.
So algorithm with smaller Big-O might actually use more time.

⑤

Horner's method $p(x) = (\cdots((a_n \cdot x + a_{n-1}) \cdot x + a_{n-2} \cdot x) \cdots) x + a_0$

$= (\cdots(a_n \cdot x + a_{n-1}) \times$
$x + a_{n-2}) \times$
$x + a_{n-3}) \times$
$\vdots$
$x + a_1) \times$
$x + a_0$

} n multiplication, n addition

$\therefore$ for $\deg p(x) = n \rightarrow p(x)$ takes $2n$ arithmetic operation at most

$\Rightarrow p(x) \leq 2 \cdot n, \ \forall n > 0 \quad \therefore p(x) = O(n)$.

⑥ Suppose $f(n) = \frac{1}{n} \Rightarrow f^2(n) = \frac{1}{n^2} \quad \therefore \forall n > 1, f(n) > f^2(n)$

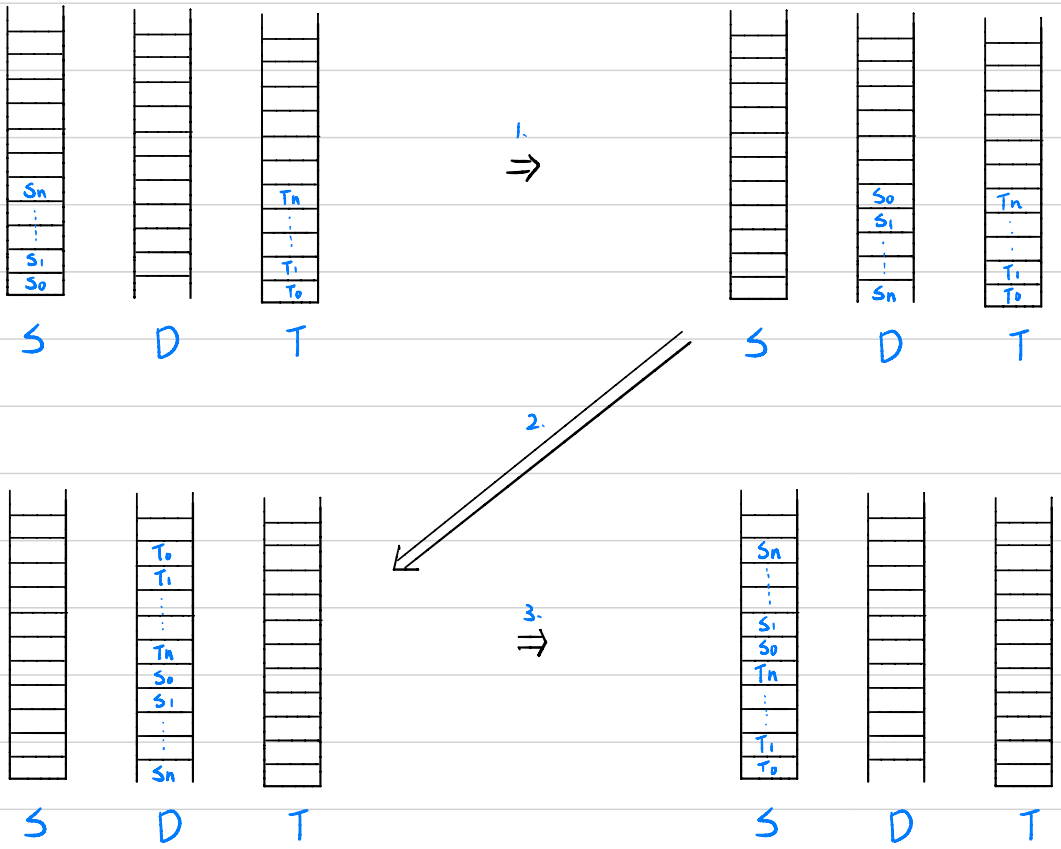
and there is no $c > 0$ s.t. $\frac{1}{n} < c \cdot \frac{1}{n^2}$ $\left(\lim_{n\to\infty} \frac{f(n)}{f^2(n)} = \lim_{n\to\infty} n = \infty\right)$.

Thus, for $f(n) = \frac{1}{n}$, $f(n) \neq O(f^2(n))$

$\Rightarrow$ So $f(n) = O(f^2(n))$ doesn't necessary holds. QED.

2. Stack, Queue, Deque.

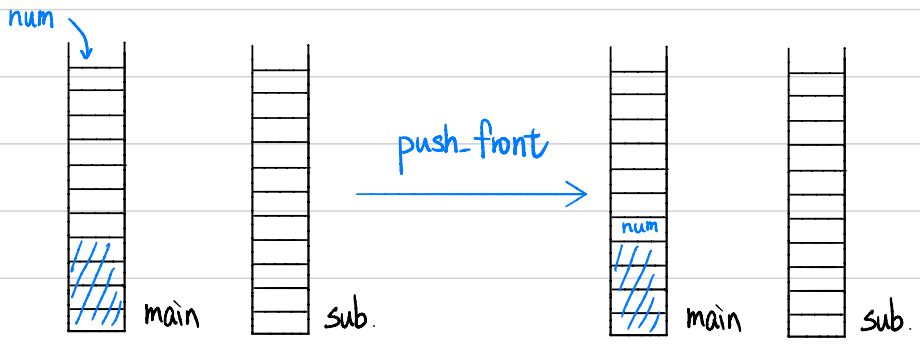
①



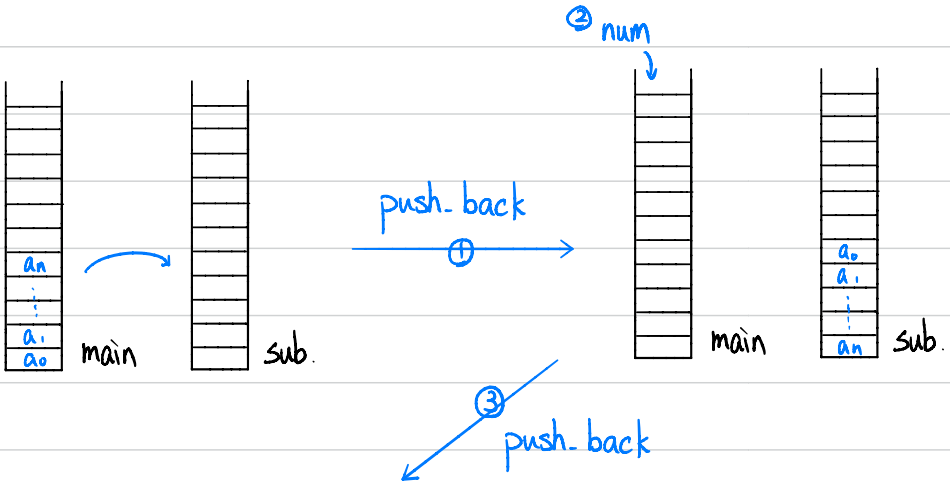
1. pop elements from stack S and push_front to Deque D.
2. pop elements from stack T and push_front to Deque D.
3. pop_front from Deque D to Stack S.

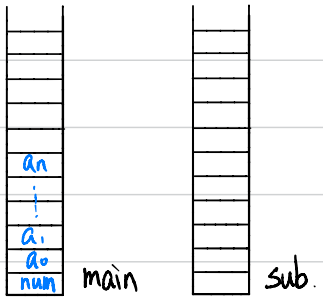
```
while(!S.empty()){
    D.push_front(S.top());
    S.pop();
}

while(!T.empty()){
    D.push_front(T.top());
    T.pop();
}

while(!D.empty()){
    S.push(D.front());
    D.pop_front();
}
```

② It will be faster if we switch the "main stack" but for readability, I will make the "main stack" be fixed. (Two stacks - main_stack, sub_stack).




```
function push_front(num)
    main_stack.push(num)
end
```



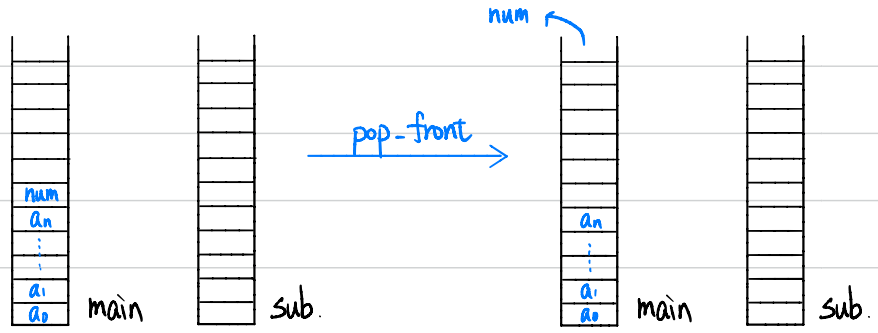



```
function push_back(num)
① while(main_stack.isempty() is false)
      tmp -> main_stack.pop()
      sub_stack.push(tmp)
  end while

② main_stack.push(num)

③ while(sub_stack.isempty() is false)
      tmp -> sub_stack.pop()
      main_stack.push(tmp)
  end while

end
```

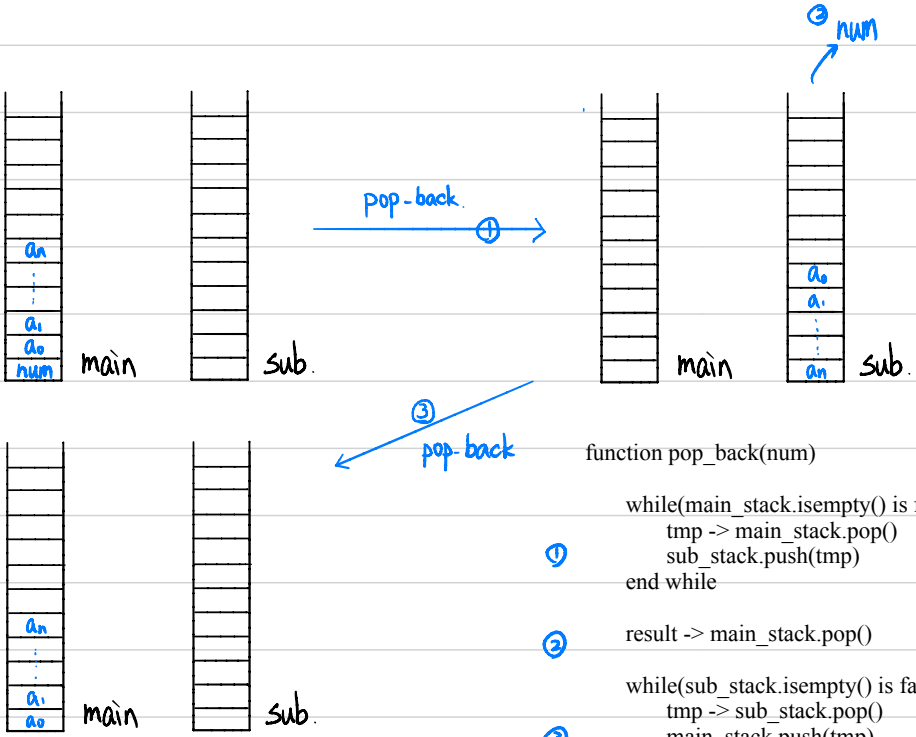
main → sub

sub → main

* push_back : O(1) , push_back : 2N → O(N)




```
function pop_front(num)
    return main_stack.pop()
end
```



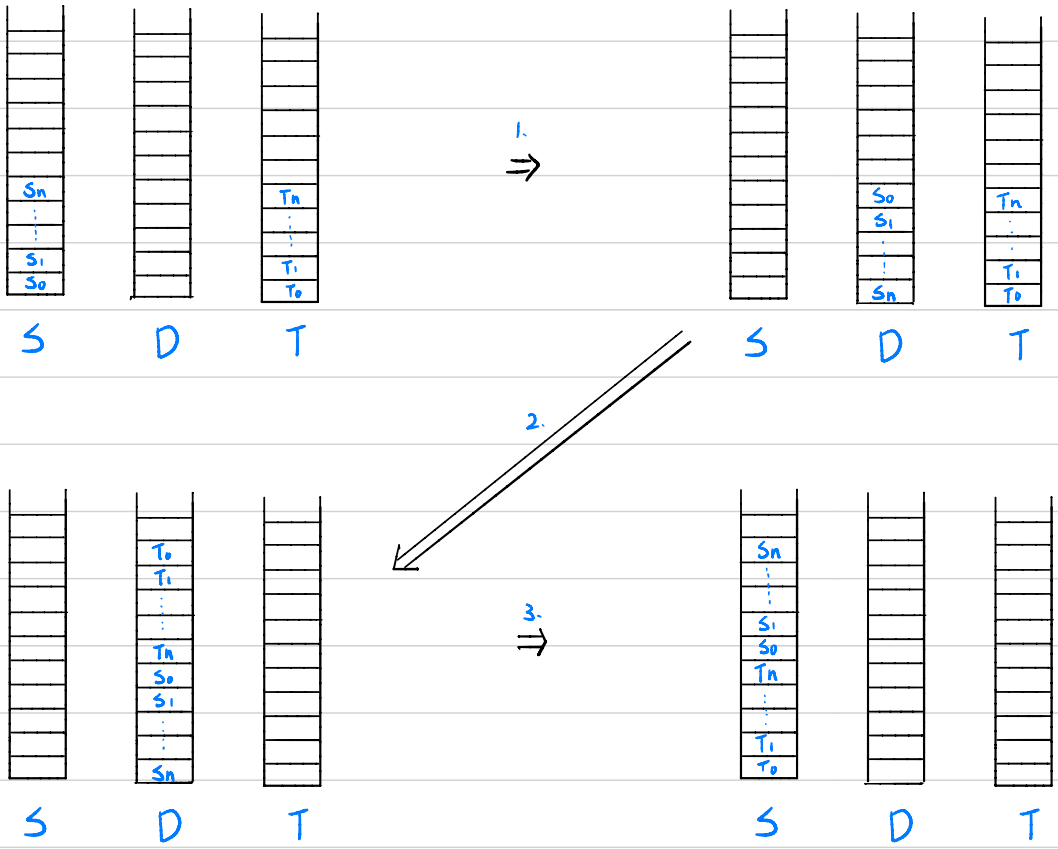

```
function pop_back(num)

    while(main_stack.isempty() is false)
        tmp -> main_stack.pop()
        sub_stack.push(tmp)                    ① main → sub
    end while

    result -> main_stack.pop()                 ②

    while(sub_stack.isempty() is false)
        tmp -> sub_stack.pop()
        main_stack.push(tmp)                   ③ sub → main
    end while

    return result

end
```

* pop-front : $O(1)$, pop-back : $2N \rightarrow O(N)$

∴ After N operation , the worst time : $O(N^2)$

the best time : $O(N)$.

③



1. pop elements from stack S and push into Stack D
2. pop elements from stack T and push into Stack D.
3. pop elements from stack D to Stack S.

(Quite the same as Problem 1).

```
while(!S.empty()){
    D.push(S.top());
    S.pop();
}

while(!T.empty()){
    D.push(T.top());
    T.pop();
}

while(!D.empty()){
    S.push(D.front());
    D.pop();
}
```

3. List, Iterator

① Set after N push operations, we perform k extension. (initial size $a_0 = 1$).

$\Rightarrow a_k = a_{k-1} + \lceil \frac{a_{k-1}}{4} \rceil \Rightarrow a_{k-1} + \frac{a_{k-1}}{4} \le a_{k-1} + \frac{a_{k-1}}{3}$

$\therefore$ When perform N push operation, we perform $\sum_{i=1}^{k} a_i$ times copy and N times push

$\Rightarrow$ total $T(N) = \sum_{i=1}^{k} a_i + N = \sum_{i=0}^{k-1} (a_i + \lceil \frac{a_i}{4} \rceil) + N \Rightarrow \sum_{i=0}^{k-1} (a_i \times \frac{5}{4}) \le T(N) \le \sum_{i=0}^{k-1} (a_i \times \frac{4}{3})$ — ①

$\because a_0 = 1$, $a_k = a_{k-1} \cdot r \Rightarrow a_k = r^k \Rightarrow \sum_{i=0}^{k-1} a_i = \frac{a_0 \cdot (r^n - 1)}{r - 1}$

Thus ①: $\frac{5}{4} \times \frac{1 \cdot (\frac{5}{4}^k - 1)}{\frac{5}{4} - 1} \le T(N) \le \frac{4}{3} \times \frac{1 \cdot (\frac{4}{3}^k - 1)}{\frac{4}{3} - 1} \Rightarrow 5 \cdot (\frac{5}{4}^k - 1) \le T(N) \le 4 (\frac{4}{3}^k - 1)$

$\because \frac{4}{3}^k \le a_k = N \le \frac{5}{4}^k \Rightarrow \log_{\frac{5}{4}} N \le k \le \log_{\frac{4}{3}} N \quad \therefore 4(N-1) \le T(N) \le 5(N-1)$

Thus $T(N)$ is a polynomial of $N^1 \Rightarrow T(N) = O(N)$ QED

② Proof of Knuth Shuffle, want to show all ordering has the same probability.

Suppose the original sequence: $S = [a_1, a_2 \cdots a_n]$ and $R = [b_1, b_2 \cdots b_n]$ is one of the random state of a_i with all $b_i \neq b_j$.

$\therefore P(R[i] = b_i \text{ for } i \in 1 \sim N)$ is the possibility of $S \to R$ by Knuth-Shuffle.

$P(R[1] = b_1) \times P(R[2] = b_2 \mid R[1] = b_1) \times \cdots \times P(R[N] = b_n \mid R[i] = b_i \text{ for } i \in 1 \sim N-1)$. — ②

[for $P(R[k] = b_k \mid R[i] = b_i \text{ for } i \in 1 \sim k-1) = \frac{1}{n-k+1}$ since there's $n-k+1$ items left and the possibility of getting each of them is the same.]

$\Rightarrow$ Thus ② $= \frac{1}{n} \times \frac{1}{(n-1)} \times \cdots \times \frac{1}{1} = \frac{1}{n!}$. And since there are $n!$ kinds of S permutation

$\Rightarrow$ each permutation has the possibility of $\frac{1}{n!} = P(S \to R)$.

$\therefore$ So the possibility for each permutation is the same. QED.

```
for(int i = n; i > 0; i—){
    int j = randomInterger(i);
    swap(vector[I], vector[j]);
}
```

$\therefore$ The time complexity = $O(n)$.

(Assuming randomInteger, Swap is $O(1)$).

4. Calculator

①

Infix: -+-(2+3*4-(7*2)/!4)+~2<<3%13!=7&&3||0
---
push - to stack
push + to stack
push - to stack
push (to stack
2 to postfix
push + to stack
3 to postfix
push * to stack
4 to postfix
pop * from stack and add to postfix
pop + from stack and add to postfix
push - to stack
push (to stack
7 to postfix
push * to stack
2 to postfix
pop * from stack and add to postfix
pop (from stack
push / to stack
push ! to stack
4 to postfix
pop ! from stack and add to postfix
pop / from stack and add to postfix
pop - from stack and add to postfix
pop (from stack
pop - from stack and add to postfix
pop + from stack and add to postfix
pop - from stack and add to postfix
push + to stack
push ~ to stack
2 to postfix
pop ~ from stack and add to postfix
pop + from stack and add to postfix
push << to stack
3 to postfix
push % to stack
13 to postfix
pop % from stack and add to postfix
pop << from stack and add to postfix
push != to stack
7 to postfix
pop != from stack and add to postfix
push && to stack
3 to postfix
pop && from stack and add to postfix
push || to stack
0 to postfix

Postfix: 2 3 4 * + 7 2 * 4 ! / - - + - 2 ~ + 3 13 % << 7 != 3 && 0 ||

RESULT: 1
Answer: 1

Infix: 3*-+—2%5*12/5
---
3 to postfix
push * to stack
push - to stack
push + to stack
push - to stack
push - to stack
2 to postfix
pop - from stack and add to postfix
pop - from stack and add to postfix
pop + from stack and add to postfix
pop - from stack and add to postfix
pop * from stack and add to postfix
push % to stack
5 to postfix
pop % from stack and add to postfix
push * to stack
12 to postfix
pop * from stack and add to postfix
push / to stack
5 to postfix
---
3 2 - - + - * 5 % 12 * 5 /

RESULT: -2
Answer: -2

Infix: (2&3)<<4|~512+!2!=3
---
push (to stack
2 to postfix
push & to stack
3 to postfix
pop & from stack and add to postfix
pop (from stack
push << to stack
4 to postfix
pop << from stack and add to postfix
push | to stack
push ~ to stack
512 to postfix
pop ~ from stack and add to postfix
push + to stack
push ! to stack
2 to postfix
pop ! from stack and add to postfix
pop + from stack and add to postfix
push != to stack
3 to postfix
---
2 3 & 4 << 512 ~ 2 ! + 3 != |

RESULT: 33
Answer: 33

②

Infix: 2.41*-+--3.22/4.113*(1.2+2/1.5)

2.41 to postfix
push * to stack
push - to stack
push + to stack
push - to stack
push - to stack
3.22 to postfix
pop - from stack and add to postfix
pop - from stack and add to postfix
pop + from stack and add to postfix
pop - from stack and add to postfix
pop * from stack and add to postfix
push / to stack
4.113 to postfix
pop / from stack and add to postfix
push * to stack
push (to stack
1.2 to postfix
push + to stack
2 to postfix
push / to stack
1.5 to postfix
pop / from stack and add to postfix
pop + from stack and add to postfix
pop * from stack and add to postfix

2.41 3.22 - - + - * 4.113 / 1.2 2 1.5 / + *

RESULT: -4.779765
Answer: -4.77976497285

Infix: exp(log(2*3.14)*fabs(-134.2))+sin(1.3)/tan(1.3)*-+--+1.402
push exp to stack
push (to stack
push log to stack
push (to stack
2 to postfix
push * to stack
3.14 to postfix
pop * from stack and add to postfix
pop log from stack and add to postfix
push * to stack
push fabs to stack
push (to stack
push - to stack
134.2 to postfix
pop - from stack and add to postfix
pop fabs from stack and add to postfix
pop * from stack and add to postfix
pop exp from stack and add to postfix
push + to stack
push sin to stack
push (to stack
1.3 to postfix
pop sin from stack and add to postfix
push / to stack
push (to stack
1.3 to postfix
pop / from stack and add to postfix
push * to stack
push - to stack
push + to stack
push - to stack
push - to stack
push + to stack
1.402 to postfix
pop + from stack and add to postfix
pop - from stack and add to postfix
pop - from stack and add to postfix
pop + from stack and add to postfix
pop - from stack and add to postfix
pop * from stack and add to postfix
pop + from stack and add to postfix
---
2 3.14 * log 134.2 - fabs * exp 1.3 sin 1.3 / 1.402 + - - + - * +

RESULT: 12195066967729491527806099908162082703430792187669781770782006825566771576384256966099752864653635423620300 8.000000
Answer: 1.2195066967729492e+107 (12195066967729491527806099908162082703430792187669781770782006825566771576384256966099752864653635423620300 8)

Infix: -sin(pow(2+3,3+5*1.4)*cos(exp(3.1)))+log(4.23*sqrt(5)+fabs(-100.02))
---
push - to stack
push sin to stack
push (to stack
push (to stack
2 to postfix
push + to stack
3 to postfix
pop + from stack and add to postfix
push pow to stack
3 to postfix
push + to stack
5 to postfix
push * to stack
1.4 to postfix
pop * from stack and add to postfix
pop + from stack and add to postfix
pop pow from stack and add to postfix
push * to stack
push cos to stack
push (to stack
push exp to stack
push (to stack
3.1 to postfix
pop exp from stack and add to postfix
pop cos from stack and add to postfix
pop * from stack and add to postfix
pop sin from stack and add to postfix
pop - from stack and add to postfix
push + to stack
push log to stack
push (to stack
4.23 to postfix
push * to stack
push sqrt to stack
push (to stack
5 to postfix
pop sqrt from stack and add to postfix
pop * from stack and add to postfix
push + to stack
push fabs to stack
push (to stack
push - to stack
100.02 to postfix
pop - from stack and add to postfix
pop fabs from stack and add to postfix
pop + from stack and add to postfix
pop log from stack and add to postfix
pop + from stack and add to postfix
---
2 3 + 3 5 1.4 * + pow 3.1 exp cos * sin - 4.23 5 sqrt * 100.02 - fabs + log +

RESULT: 5.694330
Answer: 5.694329880507852