Given an arithmetic expression with operators +, −, *, / and parantheis (,)

1) Build the equivalent expression in postfix form
2) Evaluate the expression given in postfix form.

3 + 2 − 1

3 2 + 1 −

3 * 2 + 1

3 2 * 1 +

3 + 2 * 1

3 2 1 * +

q

1 2 +

st: +

3 2 +

base priority: +

? −

1 −

3 2 *

greater priority: *

?+

+

1 +

3 2

lower priority: +

?*

+ *

3 2 1 * +

3 * (2 + 1)

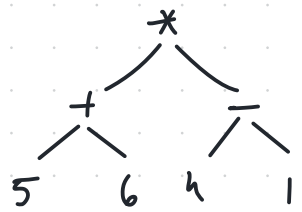
curr. symb		q	st
3	⇒	3	
*	⇒	3	*
(		3	*, (
2	⇒	3, 2	*, (
+	⇒	3, 2	*, (, +
1	⇒	3, 2, 1	*, (, +
)	⇒	3, 2, 1, +	*

3, 2, 1, +, *

(5 + 6) * (4 − 1)

current simbol	q	st
(		(
5	5	(
+	5	(, +
6	5, 6	(, +
)	5, 6, +	
*	5, 6, +	*
(	5, 6, +, 4	*, (
−	5, 6, +, 4	*, (, −
1	5, 6, +, 4, 1	*, (, −
)	5, 6, +, 4, 1, −	

5, 6, +, 4, 1, −, *



---

2 * (4 + 3) − 4 + 6 / 2 * 3

2 4 3 + * 4 − 6 2 / 3 * +

Function transform (expression)

```
    init (stack)
    init (q)    // build the postfix form

    for e in expression execute:
        if isOperand(e) then
            push(q, e)
        else if e == '(' then
            push(st, e)
        else if e == ')' then
            while (top(st) != '(') do
                push(q, top(st))
                pop(st)
            end_while
            pop(st)
        else // e is operator
            while (! isEmpty(st) and top(st) != '(' and priority(top(st)) > priority(e))
                push(q, pop(st))
            end_while
            push(st, e)
        end_if
    end_for
    while (! isEmpty(st))
        push(q, pop(st))
    end_while

    transform ← q
end_function
```

init
push
pop
isEmpty
top ← st

---

isOperand
isOperator
priority(e1) ≥ ≤ priority(e2)

1 2 + 3 * ⇒

= 9

4 3 * 6 +

5 6 + 4 1 − *

curr	st
5	5
6	5, 6
+	11
4	11, 4
1	11, 4, 1
−	11, 3
*	33

Function evaluation (postfix)

```
init (st)
while (! isEmpty (q)) execute
    e ← pop(q)
    if is Operand (e) then
        push (st, e)
    else
        e1 ← pop (st)
        e2 ← pop(st)
        result ← compute (e2, e, e1)
        push (st, result)
    end_if
end_while
evaluate ← pop (st)
```

Given

A vector with n distinct integers. compute the sum of n largest elements from there

V1) based on getting the maximum from an array

$\Theta(k \cdot n)$

V2) based on sorting algorithm

- sort in $\Theta(n \cdot \log n)$
- get Max n elems $\Theta(n)$

$\Theta(n \log n + n)$

- partial sort $\Theta(n+n)$

V3) Max heap

| add all elements into an initially empty heap
| $O(n \log n)$

| remove K-times from the heap
| $O(k \log n)$

$O(n + k \log n)$

init $(h, <^{\geq}_{\leq})$ MAX / MIN

$O(\log n)$ remove $(h) \Rightarrow e$

$O(\log n)$ add (h, e)

V4) Min-heap

95 81 40 90 60 100

$O(n * \log k)$