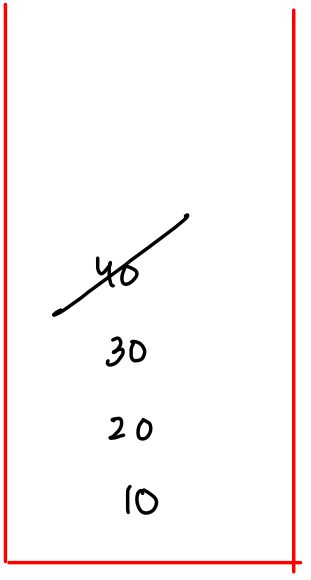
Stack

FILO: First in last out

[O(1) access] → top most element



st.push(10)

st.push(20)

st.push(30)

st.push(40)

st.peek() → 40

st.pop() → 40

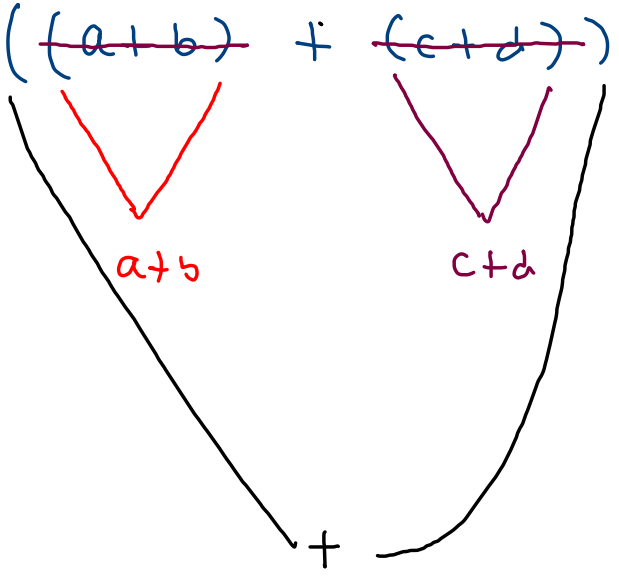
st.peek() → 30

push

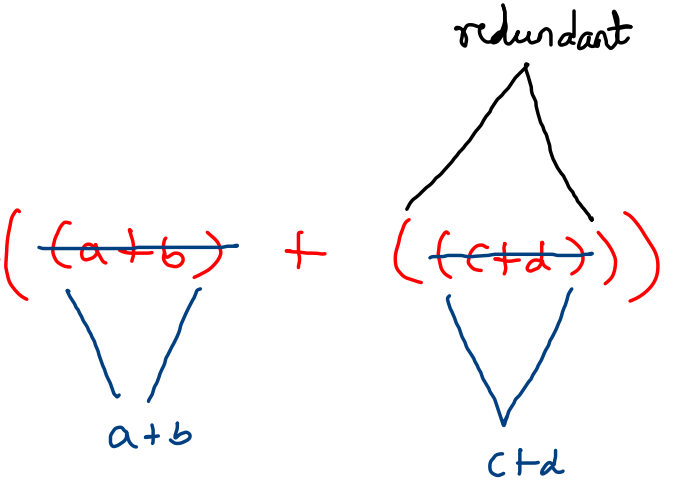
pop

peek

Duplicate Brackets



no duplicate brackets



$((a+b)+(c*(d+e)))$

i

~~e~~
~~+~~
~~d~~
~~(~~
~~*~~
~~c~~
~~(~~
~~+~~
~~b~~
~~+~~
~~a~~
~~(~~
~~(~~

(a + b * ((c - d)))

duplicate

c = 0

~~d~~
~~-~~
~~c~~
~~(~~
(
*
b
+
a
(

```
if (ch == ')') {
    // settle
}
else if(ch != ' ') {
    st.push(ch);
}
```

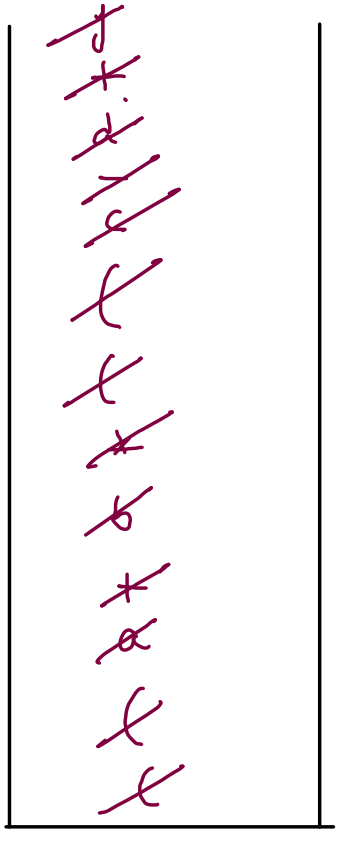
(, operators, operands

((a+b) * ((c - d) + f))

```java
for(int i=0; i < exp.length();i++) {
    char ch = exp.charAt(i);

    if(ch == ')') {
        //settle
        int count = 0;

        while(st.peek() != '(') {
            count++;
            st.pop();
        }

        //this pair of bracket is redundant
        if(count == 0) {
            return true;
        }

        st.pop(); //for '(' bracket
    }
    else if(ch != ' ') {
        //'(',operand,operators
        st.push(ch);
    }
}
```

c = ~~0~~ 1

i



O(n)

```java
for(int i=0; i < exp.length();i++) {
    char ch = exp.charAt(i);

    if(ch == ')') {
        //settle
        int count = 0;

        while(st.peek() != '(') {
            count++;
            st.pop();
        }

        //this pair of bracket is redundant
        if(count == 0) {
            return true;
        }

        st.pop(); //for '(' bracket
    }
    else if(ch != ' ') {
        //'(',operand,operators
        st.push(ch);
    }
}
```

(a+b)*((c-d))

i

duplicate c

c = 0

~~d~~
~~-~~
~~c~~
~~(~~
(
*
~~b~~
~~+~~
~~a~~
~~(~~

Balanced Brackets

$\{ (a+b) + [c+d] \}$

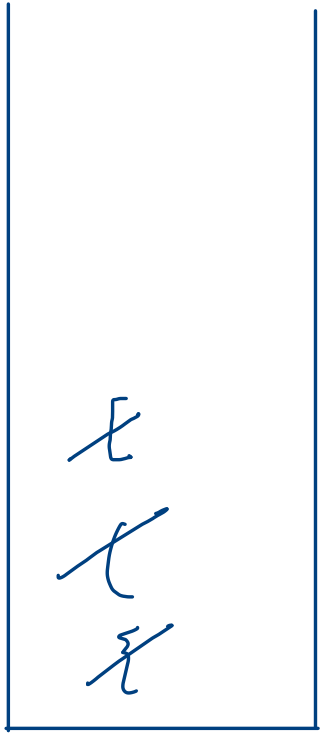
(i) mismatch $(a+b\}$

(ii) no. of op == no. of closing bracket $((a+b)$ or $(a+b))$

(iii) pairing (closing bracket should have a same type opening and vice-versa should be also true).

{ (a+b) + [c+d] }

```java
for(int i=0; i < exp.length();i++ ) {
    char ch = exp.charAt(i);

    if(ch == '(' || ch == '[' || ch == '{') {
        st.push(ch);
    }
    else if(ch == ')' || ch == ']' || ch == '}') {
        //validate
        if(ch == ')') {
            if(st.size() == 0 || st.peek() != '(') {
                return false;
            }
        }
        else if(ch == ']') {
            if(st.size() == 0 || st.peek() != '[') {
                return false;
            }
        }
        else if(ch == '}') {
            if(st.size() == 0 || st.peek() != '{') {
                return false;
            }
        }

        st.pop(); //to pop opening bracket
    }
}
```



opening -> st.push

closing -> validate

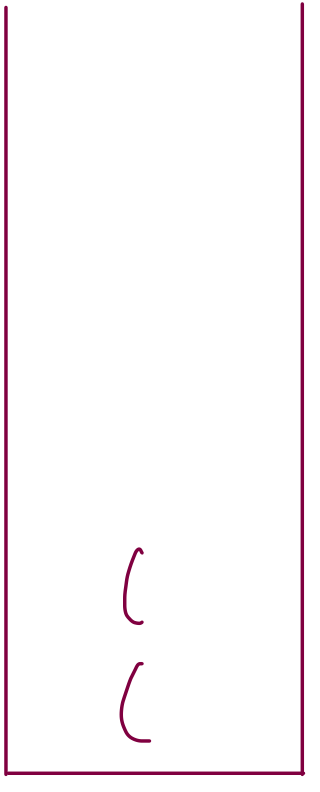
((a+b] + [c+d])

mismatch

ch =]

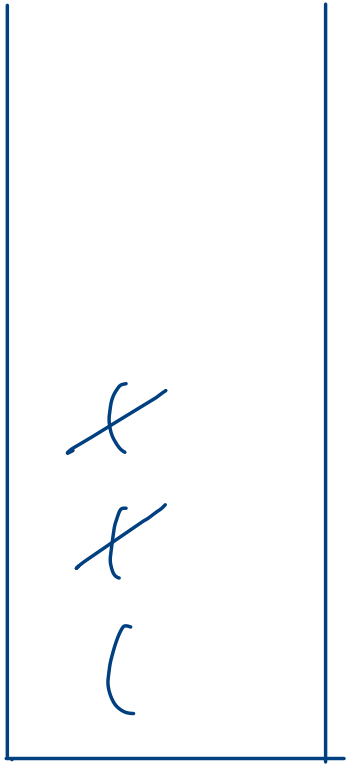
```
for(int i=0; i < exp.length();i++ ) {
    char ch = exp.charAt(i);

    if(ch == '(' || ch == '[' || ch == '{') {
        st.push(ch);
    }
    else if(ch == ')' || ch == ']' || ch == '}') {
        //validate
        if(ch == ')') {
            if(st.size() == 0 || st.peek() != '(') {
                return false;
            }
        }
        else if(ch == ']') {
            if(st.size() == 0 || st.peek() != '[') {
                return false;
            }
        }
        else if(ch == '}') {
            if(st.size() == 0 || st.peek() != '{') {
                return false;
            }
        }

        st.pop(); //to pop opening bracket
    }
}
```

F T



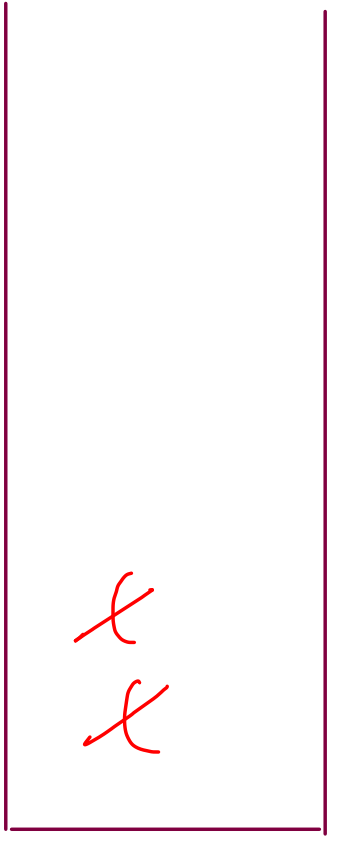
(((a+b)*d);

```java
for(int i=0; i < exp.length();i++ ) {
    char ch = exp.charAt(i);

    if(ch == '(' || ch == '[' || ch == '{') {
        st.push(ch);
    }
    else if(ch == ')' || ch == ']' || ch == '}') {
        //validate
        if(ch == ')') {
            if(st.size() == 0 || st.peek() != '(') {
                return false;
            }
        }
        else if(ch == ']') {
            if(st.size() == 0 || st.peek() != '[') {
                return false;
            }
        }
        else if(ch == '}') {
            if(st.size() == 0 || st.peek() != '{') {
                return false;
            }
        }

        st.pop(); //to pop opening bracket
    }
}
```



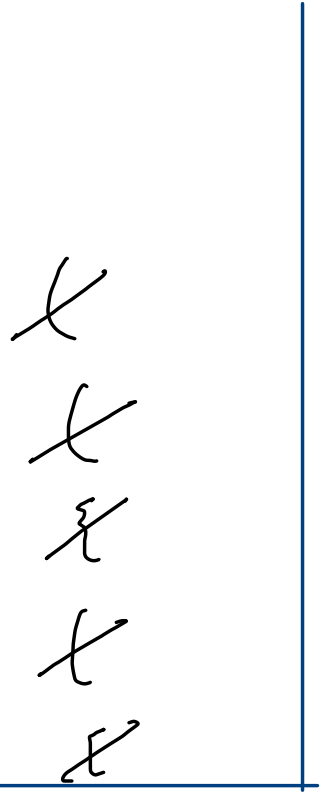
(a+b) + (c * d)]

```java
for(int i=0; i < exp.length();i++ ) {
    char ch = exp.charAt(i);

    if(ch == '(' || ch == '[' || ch == '{') {
        st.push(ch);
    }
    else if(ch == ')' || ch == ']' || ch == '}') {
        //validate
        if(ch == ')') {
            if(st.size() == 0 || st.peek() != '(') {
                return false;
            }
        }
        else if(ch == ']') {
            if(st.size() == 0 || st.peek() != '[') {
                return false;
            }
        }
        else if(ch == '}') {
            if(st.size() == 0 || st.peek() != '{') {
                return false;
            }
        }

        st.pop(); //to pop opening bracket
    }
}
```



[(a+b) + {(c+d) * (e+f)}])

```java
for(int i=0; i < exp.length();i++ ) {
    char ch = exp.charAt(i);

    if(ch == '(' || ch == '[' || ch == '{') {
        st.push(ch);
    }
    else if(ch == ')' || ch == ']' || ch == '}') {
        //validate
        if(ch == ')') {
            if(st.size() == 0 || st.peek() != '(') {
                return false;
            }
        }
        else if(ch == ']') {
            if(st.size() == 0 || st.peek() != '[') {
                return false;
            }
        }
        else if(ch == '}') {
            if(st.size() == 0 || st.peek() != '{') {
                return false;
            }
        }

        st.pop(); //to pop opening bracket
    }
}
```



extra closing brackets.

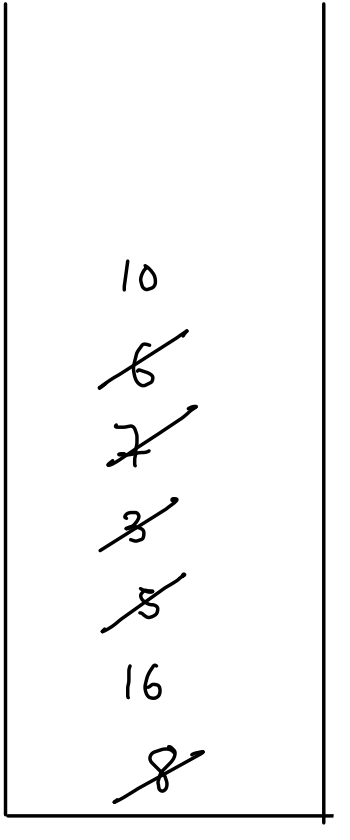
Next Greater Element To The Right

O(n)

10	6	7	3	5	16	8
0	1	2	3	4	5	6

nge

16	7	16	5	16	-1	-1



10	6	9	17	5	4	11
0	1	2	3	4	5	6
17	9	17	-1	11	11	-1

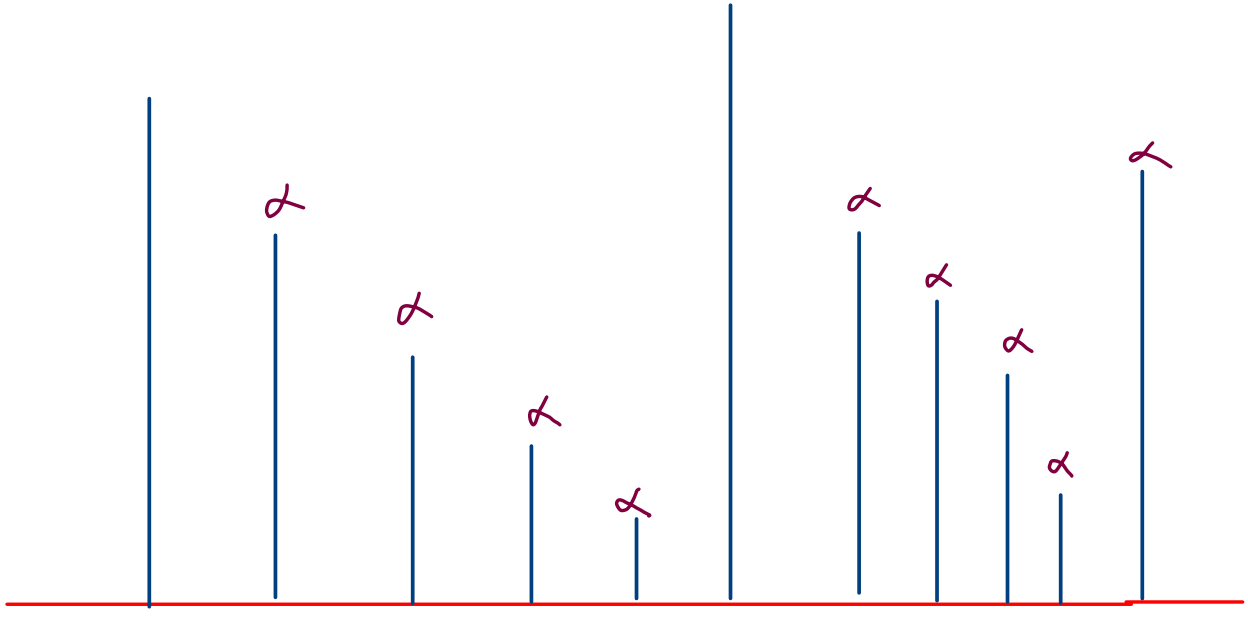
nge

```
for(int i = n-2; i >= 0;i--) {
    while(st.size() > 0 && st.peek() <= arr[i]) {
        st.pop();
    }

    if(st.size() == 0) {
        nge[i] = -1;
    }
    else {
        nge[i] = st.peek();
    }

    st.push(arr[i]);
}
```

10
~~6~~
~~9~~
17
~~5~~
~~4~~
~~11~~

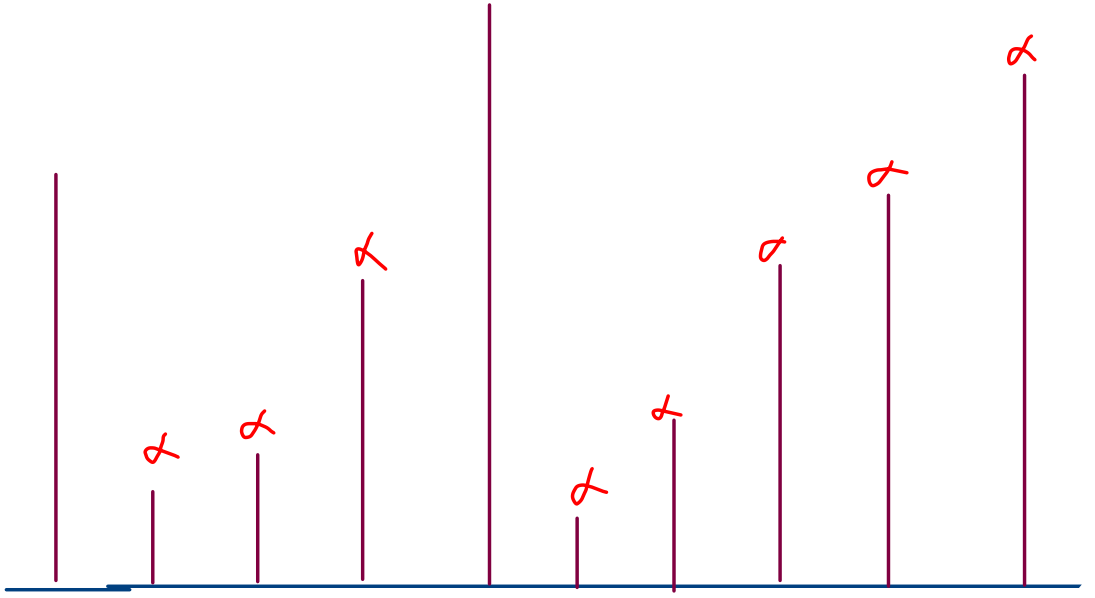
```java
for(int i = n-2; i >= 0;i--) {
    while(st.size() > 0 && st.peek() <= arr[i]) {
        st.pop();
    }

    if(st.size() == 0) {
        nge[i] = -1;
    }
    else {
        nge[i] = st.peek();
    }

    st.push(arr[i]);
}
```

$O(n)$



push	1	1	1	1	1	1	1	1	1	1	1
pop	1	1	1	1	0	2	1	1	1	0	0

n push + n pop



push	1	1	1	1	1	1	1	1	1	1
pop	3	0	0	0	5	0	0	0	0	0

n pop

n push

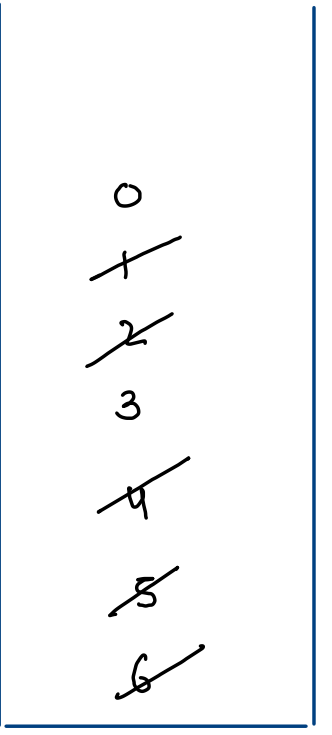
nge or	nge ol
right to left	left to right

nge or	nse or
while (st.size() > 0 && st.peek() <= arr[i]) { st.pop(); }	while (st.size() > 0 && st.peek() ≥ arr[i]) { st.pop(); }

nge or (index - based)

	10	6	9	17	5	4	11
	0	1	2	3	4	5	6
ngeori	3	2	3	-1	6	6	-1
dist	3	1	1	-1	2	1	-1

arr [st.peek()]



stack <-> index

	10	6	9	17	5	4	11
	0	1	2	3	4	5	6
nge	3	1	1	-1	2	1	-1

gap between me (ith ele) and nge on right for ith ele.

0
~~1~~
~~2~~
3
~~4~~
~~5~~
~~6~~

```
for(int i=n-2; i >= 0;i--) {
    while(st.size() > 0 && arr[st.peek()] <= arr[i]) {
        st.pop();
    }

    if(st.size() == 0) {
        nge[i] = -1;
    }
    else {
        nge[i] = st.peek() - i;
    }

    st.push(i);
}
```