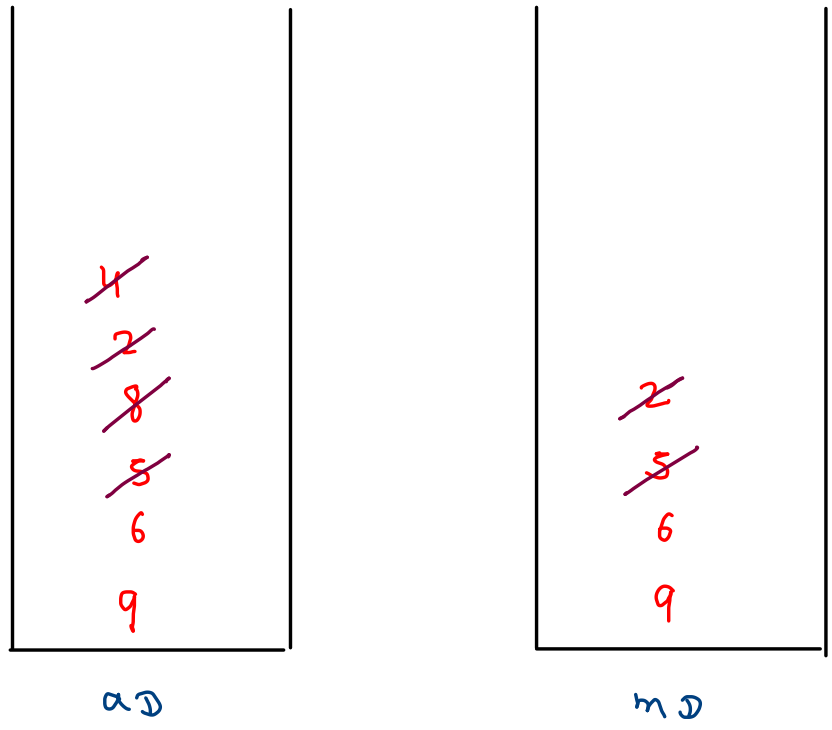
Minimum Stack - I

```
public MinStack() {
  allData = new Stack<>();
  minData = new Stack<>();
}


int size() {
  // write your code here
}

void push(int val) {
  // write your code here
}

int pop() {
  // write your code here
}

int top() {
  // write your code here
}

int min(){
  // write your code here
}
```

9 6 5 8 2 4 .



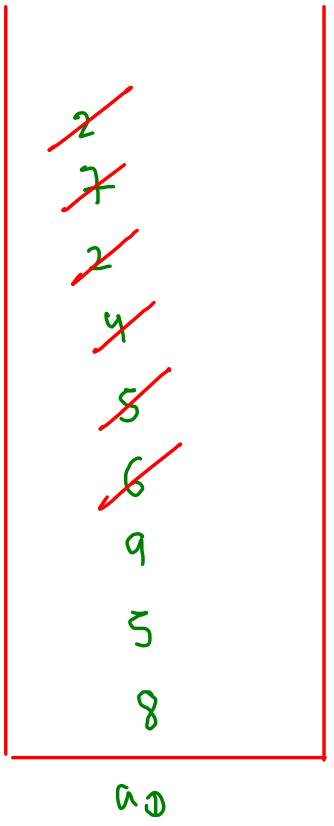
extra space

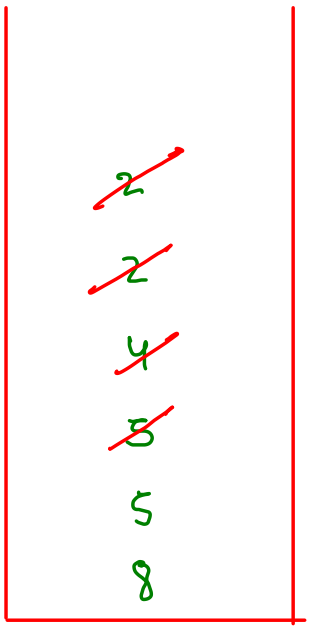
-> minData

T : O(1)

S : O(n)

data stream: 8 5 9 6 5 4 2 7 2

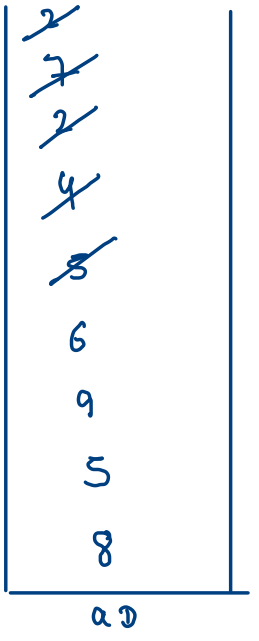


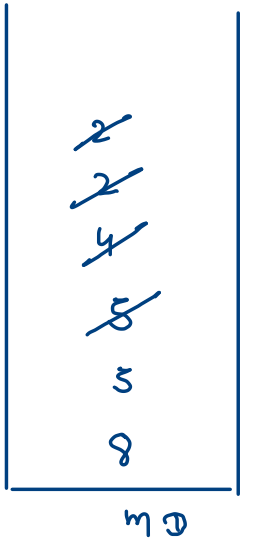


```java
int pop() {
    if(allData.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    }
    else {
        int val =  allData.pop();

        if(minData.peek() == val) {
            minData.pop();
        }

        return val;
    }
}

int top() {
    if(allData.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    }
    else {
        return allData.peek();
    }
}

int min(){
    if(allData.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    }
    else {
        return minData.peek();
    }
}
```

```java
int size() {
    return allData.size();
}

void push(int val) {

    allData.push(val);

    if(minData.size() == 0 || val <= minData.peek()) {
        minData.push(val);
    }
}
```

data stream: 8 5 9 6 5 4 2 7 2



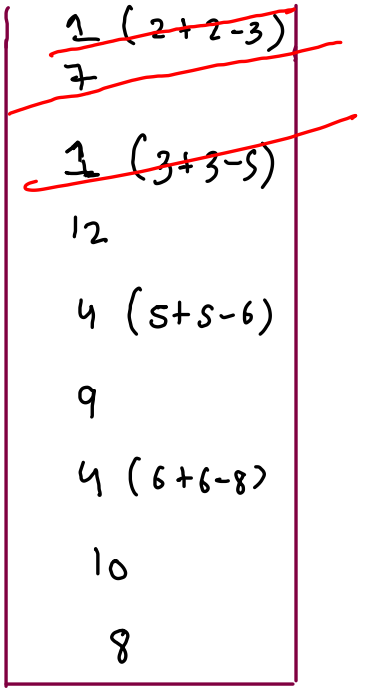


Minimum Stack - Constant Space

data stream: 8 10 6 9 5 12 3 7 2

T : O(1)

S : O(1)



(val<min)

encode => (val + val - min)

min = ~~8~~ ~~6~~ 5

fake vs real

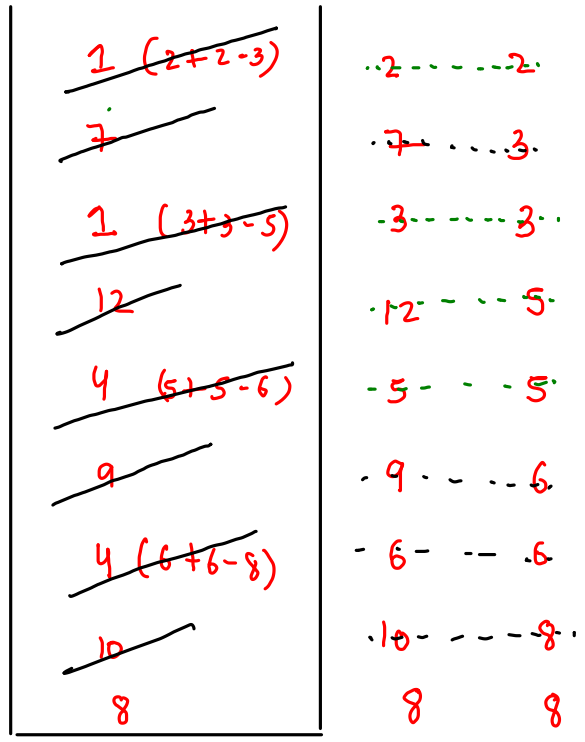
rv -> min

omin ->

val + val - omin = st.peek()

omin = 2 * val - st.peek()

data stream: 8 10 6 9 5 12 3 7 2



min = ~~∞~~ 8

```
if ( val < min) {
    encode -> (val + val - min) ->
    st.push (encode);
    min = val;
}
else {
    st.push (val);
}
```

why

val > val + val - min

2

-ve

rv -> min

rv + rv - omin = st.peek()

omin = rv * 2 - st.peek()

if(val < min)

fake_val = val + val − min

−ve

identify

decode

val > val + val − min

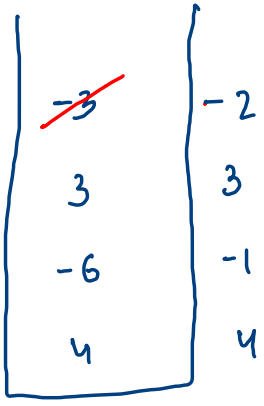
(−ve)

why not → val − min

data stream : 4 -1 3 -2

✔

fv: val + val - min

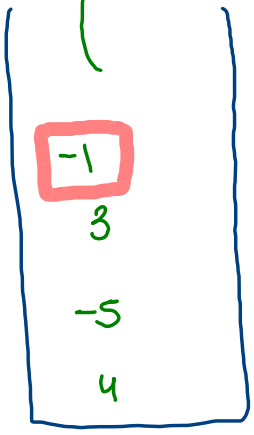


-1
min = ~~4~~

✘

fv: val - min

we can not identify blw fake vs real val



min = ~~∞~~ ~~4~~ ~~-1~~ -2

-2 - (-1)
= -1

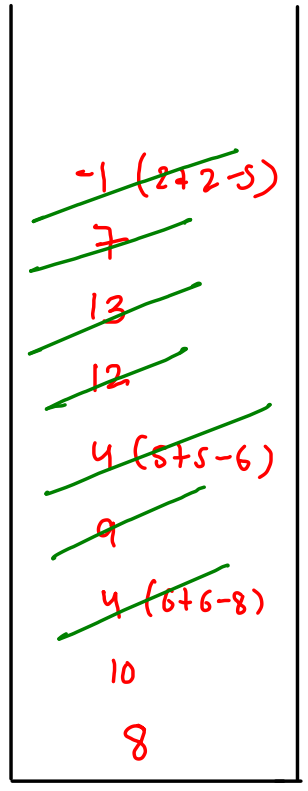
data stream: 8 10 6 9 5 12 13 7 2

```
int pop() {
    if (data.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    } else {
        if (data.peek() < min) {
            //data.peek() is a fake value
            int rv = min;
            int omin = 2 * rv - data.pop();
            min = omin;
            return rv;
        } else {
            //data.peek() is a real value
            return data.pop();
        }
    }
}
```

```
int top() {
    if (data.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    } else {
        if (data.peek() < min) {
            //data.peek() is a fake value
            return min;
        } else {
            //data.peek() is a real value
            return data.peek();
        }
    }
}
```

```
void push(int val) {
    if (data.size() == 0) {
        data.push(val);
        min = val;
        return;
    }

    if (val < min) {
        int enc = val + (val - min);
        data.push(enc);

        min = val;
    } else {
        data.push(val);
    }


}
```

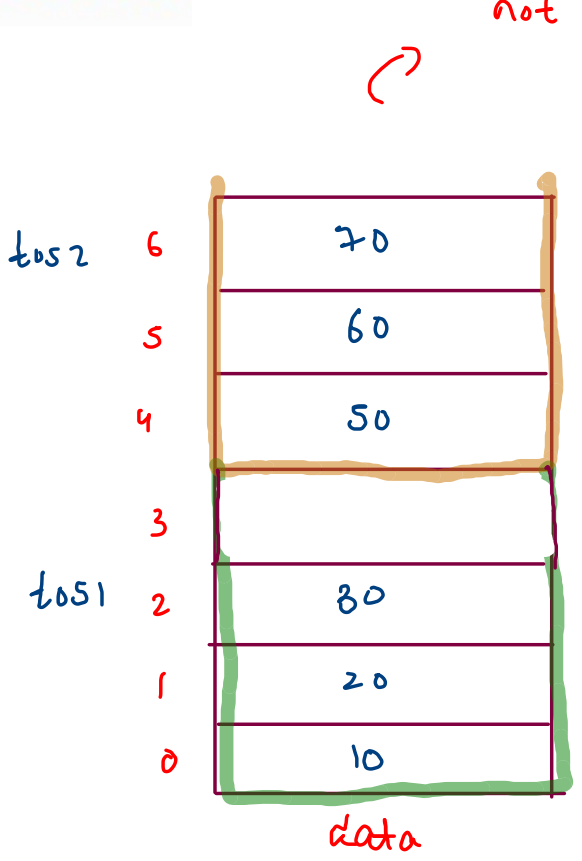
```
int min() {
    if (data.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    } else {
        return min;
    }
}
```



min = 8

```
TwoStack st = new TwoStack(n);
```

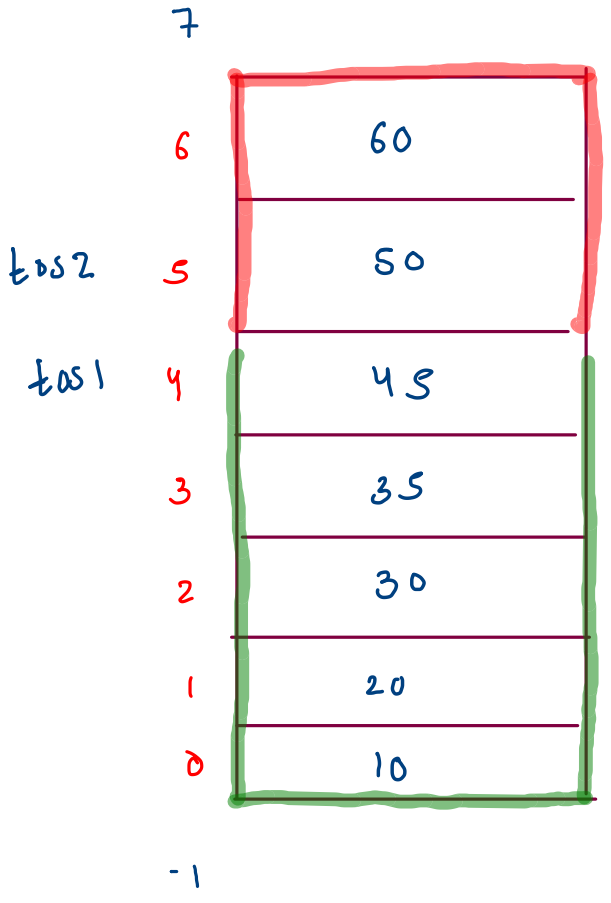
```
public static class TwoStack {
  int[] data;
  int tos1;
  int tos2;

  public TwoStack(int cap) {
    // write your code here
  }

  int size1() {
    // write your code here
  }

  int size2() {
    // write your code here
  }

  void push1(int val) {
    // write your code here
  }

  void push2(int val) {
    // write your code here
  }

  int pop1() {
    // write your code here
  }

  int pop2() {
    // write your code here
  }

  int top1() {
    // write your code here
  }

  int top2() {
    // write your code here
  }
}
```



push1(10)
push1(20)
push1(30)
push1(40)
push2(50)
push2(60)
push2(70)
pop1() -> 40
push2() X

n = 7

```
public static class TwoStack {
  int[] data;
  int tos1;
  int tos2;

  public TwoStack(int cap) {
    // write your code here
  }

  int size1() {
    // write your code here
  }

  int size2() {
    // write your code here
  }

  void push1(int val) {
    // write your code here
  }

  void push2(int val) {
    // write your code here
  }

  int pop1() {
    // write your code here
  }

  int pop2() {
    // write your code here
  }

  int top1() {
    // write your code here
  }

  int top2() {
    // write your code here
  }
}
```



push1(10)
push1(20)
push2(60)
push2(50)
push2(40)
push1(30)
push1(35)
push2(38) -> of
pop2 ->
push1(45)