



# Chapter 4 Amortized Analysis

## Algorithm Theory WS 2018/19

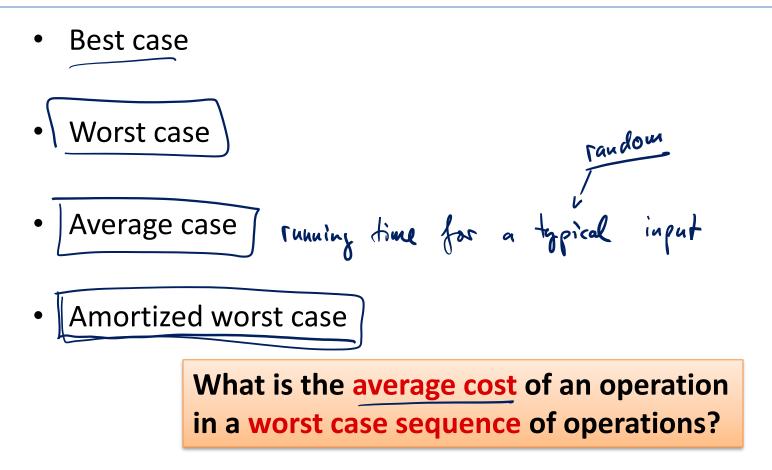
Fabian Kuhn

### Amortization



- Consider sequence o<sub>1</sub>, o<sub>2</sub>, ..., o<sub>n</sub> of n operations (typically performed on some data structure D)
- *t<sub>i</sub>*: execution time of operation <u>o</u><sub>i</sub>
- $T \coloneqq \underline{t_1} + \underline{t_2} + \cdots + \underline{t_n}$ : total execution time
- The execution time of a single operation might vary within a large range (e.g.,  $t_i \in [1, O(i)]$ )
- The worst case overall execution time might still be small
  - → average execution time per operation might be small in the worst case, even if single operations can be expensive

## Analysis of Algorithms



UNI FREIBURG



#### Stack Data Type: Operations

- S.push(x) : inserts x on top of stack
- <u>S.pop()</u> : removes and returns top element

#### **Complexity of Stack Operations**

• In all standard implementations: O(1)

#### **Additional Operation**

- **S.multipop(k)** : remove and return top k elements
- Complexity: <u>O(k)</u>
- What is the amortized complexity of these operations?



#### **Amortized Cost**

- Sequence of operations i = 1, 2, 3, ..., n
- Actual cost of op. i: t<sub>i</sub>
- Amortized cost of op. i is  $a_i$  if for every possible seq. of op.,

$$T = \sum_{i=1}^{n} t_i \leq \sum_{i=1}^{n} a_i$$

#### **Actual Cost of Augmented Stack Operations**

- S.push(x), S.pop(): actual cost  $t_i = O(1)$
- S.multipop(k) : actual cost  $\underline{t_i = O(k)}$
- Amortized cost of all three operations is constant
  - The total number of "popped" elements cannot be more than the total number of "pushed" elements: cost for pop/multipop < cost for push</li>





**Amortized Cost** 

$$T = \sum_{i} t_i \le \sum_{i} a_i$$

#### **Actual Cost of Augmented Stack Operations**

- S.push(x), S.pop(): actual cost  $t_i \leq c$
- S. multipop(k) : actual cost  $t_i \leq c \cdot k$ <u>n operations</u>  $p \leq n$  push ops. total push cost  $\leq c \cdot p$ total #del. elem.  $\leq p$  total pap/multipop cost  $\leq c \cdot p$ total cost  $\leq 2 \cdot c \cdot p$  $avg. cost por op. \leq \frac{2 \cdot c \cdot p}{n} \leq \frac{2 \cdot c \cdot p}{p} = 2c$

## Example 2: Binary Counter



Incrementing a binary counter: determine the bit flip cost:

Operation	Counter Value	Cost	
	00000		
1	00001	1	
2	000 <b>10</b>	2	
3	0001 <mark>1</mark>	1	
4	00 <b>100</b>	3	
5	0010 <mark>1</mark>	1	
6	001 <b>10</b>	2	
7	0011 <mark>1</mark>	1	
8	01000	4 3	
9	0100 <mark>1</mark>	1	
10	010 <b>10</b>	2	
11	0101 <mark>1</mark>	1	
12	01 <b>100</b>	3	
13	0110 <mark>1</mark>	1	

111(11(111))

Algorithm Theory, WS 2018/19



#### **Observation:**

• Each increment flips exactly one 0 into a 1

 $00100011111 \Longrightarrow 0010010000$ 

#### Idea:

- Have a bank account (with initial amount 0)
- Paying x to the bank account costs x
- Take "money" from account to pay for expensive operations

#### Applied to binary counter:

- Flip from 0 to 1: pay 1 to bank account (cost: 2)
- Flip from 1 to 0: take 1 from bank account (cost: 0)
- Amount on bank account = number of ones
   → We always have enough "money" to pay!

Algorithm Theory, WS 2018/19

Fabian Kuhn

## Accounting Method



Op.	Counter	Cost	To Bank	From Bank	Net Cost	Credit	
	00000					0	
1	00001	1	I	0	2	l	
2	000 <b>10</b>	2	1	1	2	l	
3	00011	1	1	0	2	2	
4	00 <b>100</b>	3	I	2	2	l	
5	00101	1	l	0	2	2	
6	001 <b>10</b>	2	(	l	2	2	
7	0011 <b>1</b>	1	1	0	2	3	
8	01000	4	ſ	3	2	1	
9	01001	1	l	0	2	2	
10	010 <b>10</b>	2			2	2	
$C + B = A = A$ Algorithm Theory, WS 2018/19 Fabian Kuhn $C \leq A$ 9							