# Chapter 4
# Amortized Analysis

## Algorithm Theory
## WS 2019/20


**Fabian Kuhn**

# Amortized Cost

**Amortized Cost of sequence of operations $\boldsymbol{i = 1, 2, \dots, n}$**

- Actual cost of op. $i$: $\boldsymbol{t_i}$
- Amortized cost of op. $i$ is $\boldsymbol{a_i}$ if for every possible seq. of ops.,

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

**Amortized Analysis: Techniques**

1. Directly analyze the total cost of all operations
2. Accounting method
   - Bank account with initial balance $0$
   - Paying $x$ to bank costs $x$
   - Use money from the bank to pay for expensive operations
3. Potential function method

# Potential Function Method

- Most generic and elegant way to do amortized analysis!
  - But, also more abstract than the others…

- State of data structure / system: $S \in \mathcal{S}$ (state space)

  **Potential function $\boldsymbol{\Phi}: \boldsymbol{\mathcal{S}} \to \mathbb{R}_{\geq 0}$**

- **Operation $\boldsymbol{i}$:**
  - $\boldsymbol{t_i}$: actual cost of operation $i$
  - $\boldsymbol{S_i}$: state after execution of operation $i$ ($S_0$: initial state)
  - $\boldsymbol{\Phi_i} \coloneqq \Phi(S_i)$: potential after exec. of operation $i$
  - $\boldsymbol{a_i}$: amortized cost of operation $i$:

$$\boldsymbol{a_i \coloneqq t_i + \Phi_i - \Phi_{i-1}}$$

# Potential Function Method

**Operation $i$:**

actual cost: $t_i$   amortized cost: $a_i = t_i + \Phi_i - \Phi_{i-1}$

**Overall cost:**

$$T := \sum_{i=1}^{n} t_i = \left(\sum_{i}^{n} a_i\right) + \Phi_0 - \Phi_n$$

# Example 3: Dynamic Array

- How to create an array where the size dynamically adapts to the number of elements stored?
  – e.g., Java "ArrayList" or Python "list"

**Implementation:**

- Initialize with initial size $N_0$
- Assumptions: Array can only grow by appending new elements at the end
- If array is full, the size of the array is increased by a factor $\beta > 1$

**Operations (array of size $N$):**

- read / write: actual cost $O(1)$
- append: actual cost is $O(1)$ if array is not full, otherwise the append cost is $O(\beta \cdot N)$ (new array size)

# Example 3: Dynamic Array

**Notation:**

- $n$: number of elements stored
- $N$: current size of array

**Cost $t_i$ of $i^{th}$ append operation:** $t_i = \begin{cases} 1 & \text{if } n < N \\ \beta \cdot N & \text{if } n = N \end{cases}$

**Claim:** Amortized append cost is $O(1)$

**Potential function $\Phi$?**

- should allow to pay expensive append operations by cheap ones
- when array is full, $\Phi$ has to be large
- immediately after increasing the size of the array, $\Phi$ should be small again

# Dynamic Array: Potential Function

**Cost $t_i$ of $i^{th}$ append operation:** $t_i = \begin{cases} 1 & \text{if } n < N \\ \beta \cdot N & \text{if } n = N \end{cases}$

# Dynamic Array: Amortized Cost

**Cost $t_i$ of $i^{th}$ append operation:** $t_i = \begin{cases} 1 & \text{if } n < N \\ \beta \cdot N & \text{if } n = N \end{cases}$