Algorithms & Data Structures

(Algorithmen & Datenstrukturen)

S. Scheele, T. Dose

Exercise Sheet 1

Warm-Up

Exercise 1.1. Implement the **Selection Sort** algorithm.

Description: Selection Sort is an in-place comparison-based sorting algorithm:

Steps:

1. Find the smallest element in the unsorted portion of the array.
2. Swap it with the leftmost unsorted element.
3. Move the boundary between the sorted and unsorted sections one element to the right.
4. Repeat until the array is fully sorted.

You can find a graphical illustration of Selection Sort here: Wikipedia - Selection Sort.

a) **Implementation:**
- Implement the `selectionSort(int arr[], int n)` function in your preferred programming language.
- The function should take an array as input, its length and sort the array.
- Use an in-place sorting approach (i.e., do not create a new array).

b) **Runtime Simulation**

Measure the runtime of your implementation using arrays of size n, including n random integers. Start with $n = 1000$, then $n = 2000$ up to $n = 50000$. Measure the runtime (using a timer) to record the execution runtime.

- How does the runtime of selection sort grow as the input size doubles each time?
- Is the growth linear, logarithmic, or quadratic?
- How does the runtime of selection sort compare to linear search for large values of n?

c) **Unit Testing:**

For this exercise, write unit tests to verify the correctness of your implementation. Each test should check at least one non-trivial case, including:

- Sorting an even number of elements (e.g., `[4, 2, 6, 1, 5]` → `[1, 2, 4, 5, 6]`).
- Sorting an odd number of elements (e.g., `[7, 3, 5, 9, 2]` → `[2, 3, 5, 7, 9]`).
- Sorting an already sorted array.
- Sorting an array in reverse order.
- Handling special cases like an empty array `[]` or an array with only one element.

Exercise 1.2. Write a function `contains` in C that checks whether an element exists in a sorted array. Implement it using **binary search** strategy both iteratively and recursively.

Description: Binary search is an efficient algorithm for searching a sorted array by repeatedly dividing the search interval in half. The key steps are:

1. Compare the target value with the middle element of the array.
2. If they match, return `true` (element found).
3. If the target value is smaller, repeat the process in the left subarray.
4. If the target value is larger, repeat the process in the right subarray.
5. If the search interval becomes empty, return `false` (element not found).

A graphical illustration of Binary Search can be found at: Wikipedia - Binary Search.

a) **Implementation:**
- Implement the function `int containsIterative(int arr[], int size, int target)` using an iterative approach.
- Implement the function `int containsRecursive(int arr[], int left, int right, int target)` using recursion.
- Assume the array is sorted in ascending order.
- The function should return 1 if the element exists, otherwise 0.

b) **Runtime Simulation:**
Measure the runtime of your implementation using arrays of size n, including n random integers. Start with $n = 1000$, then $n = 2000$ up to $n = 50000$. For each input size, run your binary search for a random target T that is guaranteed to be inside the array. Measure the runtime (using a timer) to record the execution runtime.
- How does the runtime of binary search grow as the input size doubles each time?
- Is the growth linear, logarithmic, or quadratic?
- How does the runtime of binary search compare to linear search for large values of n?

c) **Unit Testing:**
Write unit tests to verify the correctness of your implementation. Each test should check at least:
- Searching for an element that exists in the array.
- Searching for an element that does not exist.
- Searching in an empty array.
- Searching for the first and last elements.
- Searching in an array with a single element.

Induction

Exercise 1.3. Prove each of the following statements using induction.

a) For every $n \in \mathbb{N}$ with $n \geq 4$:
$$2^n \geq n^2.$$

b) For every $n \in \mathbb{N}$ with $n \geq 1$:
$$\sum_{i=1}^{n} i^2 = \frac{1}{6}n(n+1)(2n+1).$$

c) For every $n \in \mathbb{N}$:
$$\sum_{i=0}^{n} (2i+1)^2 = \frac{1}{3}(n+1)(2n+1)(2n+3).$$

Exercise 1.4. Let Σ be a finite set of symbols and Σ^* be a set of strings over Σ.
Let $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$ be the **concatenation** operator, defined recursively as:
$$\varepsilon \cdot w = w \quad \text{for } w \in \Sigma^*$$
$$(av) \cdot w = a(v \cdot w) \quad \text{for } a \in \Sigma,\ v, w \in \Sigma^*$$

Furthermore, the function $|\_| : \Sigma^* \to \mathbb{N}_0$ defined recursively by
$$|\varepsilon| = 0 \quad \text{and} \quad |aw| = 1 + |w| \quad \text{for } a \in \Sigma,\ w \in \Sigma^*$$

is called the **(word) length** of a word $w \in \Sigma^*$.
Task: Use structural induction, to prove that $|x \cdot y| = |x| + |y|$, where $x, y \in \Sigma^*$.