Prof. Dr. Goran Glavaš,
M.Sc. Fabian David Schmidt
M.Sc. Benedikt Ebing
Lecture Chair XII for Natural Language Processing, Universität Würzburg

2. Exercise for "Algorithmen, KI & Data Science 1"

1 Complexity

1. Complete the following table with the symbols O, Ω, Θ. Use O and Ω only if Θ cannot be used.

   Hint: Try plotting the functions if you are unsure (you can find a code snippet in the provided .ipynb). You do not need to prove your solution mathematically.

   Additional information:
   Ω-notation characterizes a lower bound on the asymptotic behavior of a function (similar as O-notation characterizes an upper bound). It says that a function grows at least as fast as a certain rate. The formal definition is given by:

   For a given function $g(n)$, $\Omega(g(n))$ denotes a set of functions: $\Omega(g(n)) = \{f(n)$: there exists positive constants c, and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$

	$log(n)$	$2^{n/2}$	$\sqrt{n}$	5	2^n	$1/n$	n	e^n	n^2
$log(n)$							O		
$2^{n/2}$									
$\sqrt{n}$									
5									
2^n									
$1/n$									
n									
e^n									
n^2									

	$log(n)$	$2^{n/2}$	$\sqrt{n}$	5	2^n	$1/n$	n	e^n	n^2
$log(n)$	Θ	O	O	Ω	O	Ω	O	O	O
$2^{n/2}$	Ω	Θ	Ω	Ω	O	Ω	Ω	O	Ω
$\sqrt{n}$	Ω	O	Θ	Ω	O	Ω	O	O	O
5	O	O	O	Θ	O	Ω	O	O	O
2^n	Ω	Ω	Ω	Ω	Θ	Ω	Ω	O	Ω
$1/n$	O	O	O	O	O	Θ	O	O	O
n	Ω	O	Ω	Ω	O	Ω	Θ	O	O
e^n	Ω	Ω	Ω	Ω	Ω	Ω	Ω	Θ	Ω
n^2	Ω	O	Ω	Ω	O	Ω	Ω	O	Θ

2. For each of the following functions f_i, provide a function g_i having as few terms as possible and satisfying $f_i \in \Theta(g_i)$.

   Hint: Try plotting your solution if unsure.

   - Example: $f_0(n) = 3n^2 + 3 \in \Theta(n^2)$

- $f_1(n) = n^2 2^n + 4^n + 3^n$
- $f_2(n) = n(n-1)/2$
- $f_3(n) = log(n^{70})$
- $f_4(n) = 9nlog(n) + 30n(log(n))^2 + n$

> - $f_1(n) \in \Theta(4^n)$
> - $f_2(n) \in \Theta(n^2)$
> - $f_3(n) \in \Theta(logn)$
> - $f_4(n) \in \Theta(n(log(n))^2)$

3. Given Algorithm 1, explain in your own words, what the algorithm does. Determine its time complexity in Big-O notation.
   Assuming that $nums$ is sorted. Implement the algorithm $algo2(nums, v)$ in the provided .ipynb that solves the problem in $O(log(n))$, where n is the length of $nums$.

Algorithm 1 algo1(nums, v)

```
for i = 0 to nums.length − 1 do
  if nums[i] == v then
    return i
  end if
end for
return NIL
```

> The algorithm searches for the element v in the array $nums$. If v is in $nums$, the index of v in $nums$ is returned. Otherwise, the algorithm returns NIL. The algorithm is known as linear search.
>
> Its time complexity is $O(n)$ (if we only look at worst-case running time $\Theta(n)$ would be more precise).
>
> If $nums$ is sorted, binary search is an algorithm that performs the task in $O(log(n))$.

2 Sorting

1. Illustrate each step of merge sort as shown in lecture 5 slides 16-17 on the following sequence: <3, 9, 1, 2, 7, 3, 9, 6>.

3	9	1	2	7	3	9	6
3	9	1	2	7	3	9	6
3	9	1	2	7	3	9	6
3	9	1	2	7	5	9	6
3	9	1	2	3	7	6	9
1	2	3	9	3	6	7	9
1	2	3	3	6	7	9	9

2. What value does $partition$ (as presented in the lecture slides) return when all elements in the subarray $A[p : r]$ have the same value?

   The value r

3. Give a brief argument that running time of $partition$ on a subarray of size n is $O(n)$ (Big-O notation).

   In the loop, there are $p - (r - 1)$ iterations that take constant time plus the additional $exchange$ outside the loop. This yields, $r - p$ iterations that take at most constant time. Since $r - p$ is the size of the subarray $partition$ is called on, this yields $O(n)$, if $partition$ is called on a subarray of size n.

4. Implement the method $insertsort(nums)$ in the corresponding .ipynb file. Given a list of integers $nums$, the method should sort the list in **descending order** using insertion sort. Sorting should be done in-place.
5. Implement the method $quicksort(nums, p, r)$ and $partition(nums, p, r)$ in the corresponding .ipynb file. Given a list of integers $nums$, a starting index p, and an end index r, the method $quicksort$ should sort the list in **descending order** using quicksort. Sorting should be done in-place.