Übung 9

Aufgabe 1

```
int max(int length, int array[length]) {
    int result = array[0];

    for (int i=1; i<length; i++) {
        if (array[i] > result) {
            result = array[i];
        }
    }
    return result;
}
```

Aufgabe 1

```
int max(int length, int array[length]) {
    int result = array[0];
    for (int i=1; i<length; i++) {
        if (array[i] > result) {
            result = array[i];
        }
    }
    return result;
}
```

Kosten: 2 → Zuweisung, Arrayzugriff

Aufgabe 1

```
int max(int length, int array[length]) {
    int result = array[0];                          Kosten: 2 → Zuweisung, Arrayzugriff

    for (int i=1; i<length; i++) {                  Kosten: 1 + length-1 * 3 (Vergleich, Zuweisung, Rechenoperation) + 1 (
        if (array[i] > result) {
            result = array[i];
        }
    }
    return result;
}
```

Aufgabe 1

```
int max(int length, int array[length]) {
    int result = array[0];                          Kosten: 2 → Zuweisung, Arrayzugriff

    for (int i=1; i<length; i++) {                  Kosten: 1 + length-1 * 3 (Vergleich, Zuweisung, Rechenoperation) + 1 (Vergleich)
        if (array[i] > result) {                    Kosten: 2 * length-1 → Arrayzugriff, Vergleich
            result = array[i];
        }
    }
    return result;
}
```

Aufgabe 1

```
int max(int length, int array[length]) {
    int result = array[0];                          Kosten: 2 → Zuweisung, Arrayzugriff

    for (int i=1; i<length; i++) {                  Kosten: 1 + length-1 * 3 (Vergleich, Zuweisung, Rechenoperation) + 1
        if (array[i] > result) {                    Kosten: 2 * length-1 → Arrayzugriff, Vergleich
            result = array[i];                      Kosten: 2* length-1 → Zuweisung, Arrayzugriff
        }
    }
    return result;                                  Gesamt: 2 + 1 + (n-1) * (3 + 2 + 2) + 1 → 4 + (n-1)*7 → O(n)
}
```

Aufgabe 2

```
int func1(char *string) {
    int i=0;
    while (string[i] != '\0') {
        i++;
    }
    return i;
}
```

Aufgabe 2

```
int func1(char *string) {
    int i=0;
    while (string[i] != '\0') {
        i++;
    }
    return i;
}
```

Kosten: 1 → Zuweisung

Aufgabe 2

```
int func1(char *string) {
    int i=0;                          Kosten: 1 → Zuweisung
    while (string[i] != '\0') {       Kosten: 2 * length(string) → Arrayzugriff und Vergleich
        i++;
    }
    return i;
}
```

Aufgabe 2

```
int func1(char *string) {
    int i=0;                          Kosten: 1 → Zuweisung
    while (string[i] != '\0') {       Kosten: 2 * length(string) → Arrayzugriff und Vergleich
        i++;                          Kosten: 2 * length(string-1) → Zuweisung und Rechenoperation
    }
    return i;
}
```

Aufgabe 2

```
int func1(char *string) {
    int i=0;                          Kosten: 1 → Zuweisung
    while (string[i] != '\0') {       Kosten: 2 * length(string-1) + 2 → Arrayzugriff und Vergleich
        i++;                          Kosten: 2 * length(string-1) → Zuweisung und Rechenoperation
    }
    return i;                         Kosten: 1 → return-Statement
                                      Gesamt: 1 + (length(string)-1) + 2 +1 → O(n)
}
```

Aufgabe 2

```
void func2(int arg, char *string) {
    for (int i=0; i<arg; i++) {
        string[i] = string[i]+12;
    }
}
```

Aufgabe 2

```
void func2(int arg, char *string) {
    for (int i=0; i<arg; i++) {          Kosten: 1 + (3* length(string)-1) + 1 → Zuweisung, (Vergleich, Zuweisung, Rechenop.), Vergleich
        string[i] = string[i]+12;
    }
}
```

Aufgabe 2

```
void func2(int arg, char *string) {
    for (int i=0; i<arg; i++) {
        string[i] = string[i]+12;
    }
}
```

Kosten: 1 + 3* (length(string)-1) + 1 → Zuweisung, (Vergleich, Zuweisung, Rechenoperation), Vergleich

Kosten: 4* (length(string)-1) → Arrayzugriff, Zuweisung, Arrayzugriff, Rechenoperation

Gesamt: 1 + (length(string-1)) * 7 + 1 → O(n)

Aufgabe 2

```
void allIncluded(char *string1, char *string2) {
    int a = func1(string1);
    for (int i=0; i<a; i++) {
        string2[i] = string1[i];
    }
    string2[a] = '\0';
    func2(a, string2);
}
```

Aufgabe 2

```
void allIncluded(char *string1, char *string2) {
    int a = func1(string1);                 Kosten: 2 + Kosten von func1 → Zuweisung, Funktionsaufruf
    for (int i=0; i<a; i++) {               Kosten: 1 + (length(string)-1)* (1+2)
        string2[i] = string1[i];            Kosten: 3 * (length(string)-1) → Arrayzugriff, Zuweisung, Arrayzugriff
    }
    string2[a] = '\0';                      Kosten: 2 → Arrayzugriff, Zuweisung
    func2(a, string2);                      Kosten: 1 + Kosten von func1 → Funktionsaufruf
}
```

Gesamt: O(n)

Aufgabe 3

```
void algorithmus(int data[], int length) {
    int i, k, m, t;
    for (i = 0; i < length - 1; i++) {
        m = i;
        for (k = i + 1; k < length; k++) {
            if (data[k] < data[m]) {
                m = k;
            }
        }
        temp = data[m];
        data[m] = data[i];
        data[i] = temp;
    }
}
```

Aufgabe 3

```
void algorithmus(int array[], int length) {
    int current_index, min_index, temp_value;
    for (current_index = 0; current_index < length - 1; current_index++) {
        min_index = current_index;
        for (int j = current_index + 1; j < length; j++) {
            if (array[j] < array[min_index]) {
                min_index = j;
            }
        }
        temp_value = array[min_index];
        array[min_index] = array[current_index];
        array[current_index] = temp_value;
    }
}
```

Aufgabe 3

```
void selection_sort(int array[], int length) {
    int current_index, min_index, temp_value;
    for (current_index = 0; current_index < length - 1; current_index++) {
        min_index = current_index;
        for (int j = current_index + 1; j < length; j++) {
            if (array[j] < array[min_index]) {
                min_index = j;
            }
        }
        temp_value = array[min_index];
        array[min_index] = array[current_index];
        array[current_index] = temp_value;
    }
}
```

Aufgabe 3

```
void algorithmus(int data[], int length) {
    int i, k, m, t;
    for (i = 0; i < length - 1; i++) {
        m = i;
        for (k = i + 1; k < length; k++) {
            if (data[k] < data[m]) {
                m = k;
            }
        }
        temp = data[m];
        data[m] = data[i];
        data[i] = temp;
    }
```

Kosten: 1 + 2 * (length-1) + 2

Kosten: 1 * (length-1)

Kosten: 2 * (length-1) + 3 * ((length-1)² + (length-1)) / 2 + 1

Kosten: 3 * ((length-1)² + (length-1)) / 2

Kosten: 1 * ((length-1)² + (length-1)) / 2

Kosten: 2* (length-1)

Kosten: 3* (length-1)

Kosten 2* (length-1)

n = length-1

Gesamt: $1 + 2n + 2 + n + 2n + 7 * (n^2+n)/2 + n + 7n \rightarrow O(n^2)$

Aufgabe 4

```
int algo(int data[], int len, int x) {

    int left = 0;
    int right = len-1;
    while(left <= right) {
        int middle = left + ((right-left)/2);
        if(data[middle] == x) return middle;
        else if (x<data[middle]) right = middle - 1;
        else left = middle + 1;
    }
    return -1;
}
```

1. Was macht der Algorithmus?
2. Was ist der Best-Case?
3. Was ist der Worst-Case?

Aufgabe 4

```
int algo(int data[], int len, int x) {

    int left = 0;

    int right = len-1;

    while(left <= right) {

        int middle = left + ((right-left)/2);

        if(data[middle] == x) return middle;

        else if (x<data[middle]) right = middle - 1;

        else left = middle + 1;

    }

    return -1;

}
```

Kosten: 1 → Zuweisung
Kosten: 2 → Zuweisung, Rechenoperation
Kosten: 1 → Vergleich
Kosten: 4 → Zuweisung, 3x Rechenoperation
Kosten: 2 → Arrayzugriff, Vergleich

Kosten: 4 → Vergleich, Arrayzugriff, Zuweisung, Rechenop.

Kosten: 2 → Zuweisung, Rechenoperation

Beispiel $O(2^n)$