Informatics II

Exercise 1

Sven Greuter

February 27, 2024

Recap - Last Week

- Installing `gcc`
- Compiler: Translates `C` code (human readable) to machine code readable (executable)
- `gcc <input.c> -o <output-name>`
  - (FYI: `https://tldr.inbrowser.app/pages/common/gcc`)

Recap - Algorithm

- Well-defined procedure with input and output
- Properties: Correctness and efficiency (runtime, space complexity, etc.)

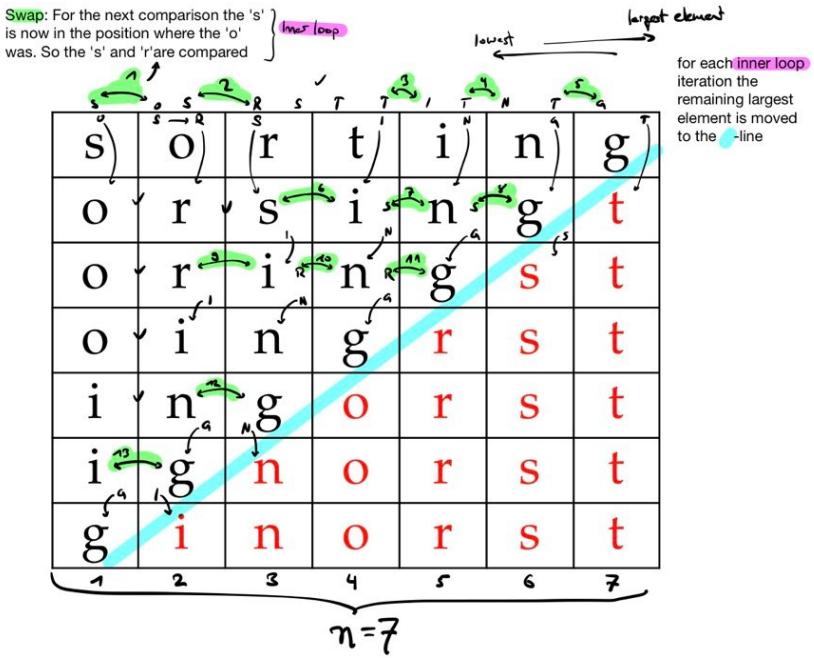
Recap - Sorting

- Fundamental Problem
- Random access: Each element can be accessed directly
- Examples:
  - n^2 comparisons: Bubble Sort, Selection Sort, Insertion Sort
  - $n \log n$ comparisons: Merge Sort, Heap Sort, Quick Sort

Recap - Sorting - Bubble Sort

The algorithm

```
Algo: BubbleSort(A)
for i = n to 2 do
    for j = 2 to i do
        if A[j]<A[j-1] then
            t = A[j];
            A[j] = A[j-1];
            A[j-1] = t;
```



$$C = \sum_{i=2}^{n}(i-1) = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = \frac{n^2-n}{2}$$

Best Case: $Mmin = 0$ Array sorted: no elements are moved

Worst Case: Inversed order $Mmax = \sum_{i=2}^{n} \underbrace{3(i-1)}_{\substack{3 \times \text{number of comparisons} \\ (\text{swaping})}} = \frac{3n(n-1)}{2} = \frac{3n^2-3n}{2}$

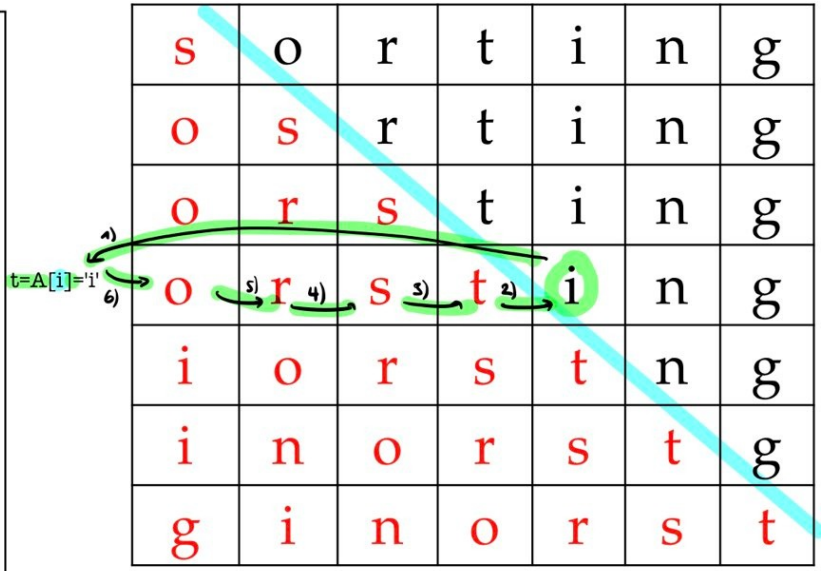
Recap - Sorting - Insertion Sort

The algorithm

Algo: InsertionSort(A)

```
for i = 2 to n do
    j = i-1;
    t = A[i];                          } 1)
    while j≥1 ∧ t<A[j] do              } 2) - 5)
        A[j+1] = A[j];
        j = j-1;
    A[j+1] = t;                        } 6)
```

t=A[i] represents the element we want to insert in the sorted subarray



s	o	r	t	i	n	g
o	s	r	t	i	n	g
o	r	s	t	i	n	g
o	r	s	t	i	n	g
i	o	r	s	t	n	g
i	n	o	r	s	t	g
g	i	n	o	r	s	t

Best Case: (sorted)

$$Cmin = \sum_{i=2}^{n} 1 = n - 1$$

if the element is the greatest and stays at that position (does not go into the while loop); so just one step (the false of the while)

$$Mmin = \sum_{i=2}^{n} 2 = 2(n-1) = 2n - 2$$

Worst Case: (reversed sorted)

$$Cmax = \sum_{i=2}^{n} \underbrace{(i-1)}_{\text{while loop swapping}} = \frac{n^2 - n}{2}$$

if the element is the smallest and needs to go to the beginning of the array; every element in the sorted subarray needs to be checked, and also swapped

$$Mmax = \sum_{i=2}^{n} (i+1) = \frac{n^2 + 3n - 4}{2}$$

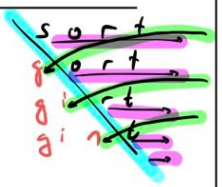
Recap - Sorting - Selection Sort

The algorithm

i is position of the smallest element, which is stored in key k, where the smallest element at index j is swapped with

Algo: SelectionSort(A)

```
for i = 1 to n-1 do
    k = i;
    for j = i+1 to n do
        if A[j]<A[k] then k = j;
    exchange A[i] and A[k];
```



s	o	r	t	i	n	g
g	o	r	t	i	n	s
g	i	r	t	o	n	s
g	i	n	t	o	r	s
g	i	n	o	t	r	s
g	i	n	o	r	t	s
g	i	n	o	r	s	t

In every iteration there is exactly one exchange (even if it's exchanged with itself)

$$C = \sum_{i=1}^{n-1} i = \frac{n^2 - n}{2}$$

$$M = \sum_{i=1}^{n-1} 3 = 3(n-1) = 3n - 3$$

Quick Poll

True or False?

The bubble sort algorithm can be implemented using two nested `while` loops.

True or False?

The insertion sort algorithm can be implemented using two nested `for` loops.

True or False?

Given the same input, all three sorting algorithms always need the same number of comparisons.

True or False?

All three sorting algorithms only compare two adjacent elements in an array.

Takeaways - Sorting

- There are slow and fast sorting algorithms
- They n^2 sorting algorithms have a nested `for` loop
  - That is why n^2 comparisons in worst case

Learning Goals

- Gain an initial understanding of the `C` programming language. This should include initial experience with the syntax, getting to know some peculiarities of `C` and being able to program simple algorithms yourself.
- Understand and implement simple sorting algorithms
- Gain some basic intuition for the runtime of an algorithm

Task 1.1 - Observation

- First number is as expected `2147483647`
- Then jumps to `-2147483648` and counts up from there

```c
int main () {
    int a = 2147483647;
    int b = 2147483648;
    int c = 2147483649;
    /* careful when copy pasting, migth screw up some characters */
    printf("%d, %d, %d", a, b, c);
    return 0;
}
```

```
2147483647, -2147483648, -2147483647
```

Task 1.1 - Explanation

- Integers have 4 bytes (32 bits)
- Most left bit is used for sign (+/-)
- Range from $-2'147'483'648$ to $2'147'483'647$
  - -2^{32-1} to $2^{32-1} - 1$
- Number stored in the integer exceeds this range, a so-called integer overflow occurs
- Counting starts at the minimum again
- `C` does not check for this danger and also does not warn you if such an overflow occurs
- If need longer numbers, user `long` (8 bytes)

Task 1.2 - Observation

```
1  int main() {
2      int myArray[20];
3      for(int i = 0; i < 20; i++) {
4          printf("%d \n", myArray[i]);
5      }
6      return 0;
7  }
```

```
-569409472
22002
2020231228
32698
3376
0
-73007495
32765
-72269824
32765
16777216
257
2
0
```

Task 1.2 - Explanation

- Variables are not initialized with a uniform default value
- Value after declaration is the binary data previously stored in that memory location
- Source of potential bugs

Task 1.3 - Observation

- Note you might have to add `-fno-stack-protector` (Error: stack smashing detected)

```c
int main() {
    int myArray[1];
    myArray[0] = 0;
    myArray[1] = 1;
    myArray[2] = 2;
    printf("%d, %d, %d", myArray[0], myArray[1], myArray[2]);
    return 0;
}
```

```
0, 1, 2
```

Task 1.3 - Explanation

- You can manipulate an array beyond the set length
- `C` does not warn you
  - Because of the design philosopy of `C`
- Can overwrite other data stored that our program needs (but should be prevented by OS)
- Careful to have boundaries under control
- Source of potential bugs
  - Similar to `SEGFAULT` error

Task 1.4 - Observation

```
1  int main() {
2    int myArray[] = {72, 101, 108, 108, 111, 32,
3                     87, 111, 114, 108, 100, 33};
4
5    for(int i = 0; i < 12; i++) {
6        printf("%d", myArray[i]);
7    }
8    printf("\n");
9
10   for(int i = 0; i < 12; i++) {
11       printf("%c", myArray[i]);
12   }
13   return 0;
14 }
```

```
721011081081113287111114108100 33
Hello World!
```

Task 1.4 - Explanation

- Read binary content of variable as other types
- Numbers are interpreted as ASCII characters
- Avoid: Source of bugs and poorly understandable code

Task 1.5 - Observation

```c
int main() {
    int myArray[5];
    int size1 = sizeof(myArray);
    int size2 = sizeof(myArray[0]);
    int size3 = size1 / size2;
    printf("%d, %d, %d", size1, size2, size3);
    return 0;
}
```

```
20, 4, 5
```

Task 1.5 - Explanation

- `sizeof(myArray[0])`: Size of a single element in bytes, thus 4
- `sizeof(myArray)`: Size of whole array, because 5 elements with 4 bytes each results in total size of $4\text{bytes} \cdot 5\text{elements} = 20\text{bytes}$
- `size1/size2`: Is thus the length, i.e., the number of elements of the array

Task 1.6 - Observation

```c
int main() {
    char myString[] = "hello";

    int stringSize = sizeof(myString)/sizeof(myString[0]);
    printf("%d, ", stringSize);

    for(int i = 0; i < stringSize; i++) {
        printf("%c", myString[i]);
    }
    return 0;
}
```

```
6, hello
```

Task 1.6 - Explanation

- Strings are arrays of type `char`
- Mark end of string with `\0` (null character)
- Null character is also an element in the array
- Be careful when initializing an String with fixed length

Task 2

```
1  void rleCompression(char string[], int length) {
2      if (length == 0) {                              /* safeguard */
3          return;
4      }
5      int charCount = 1;
6      char mostRecentChar = string[0];
7      for (int i = 1; i < length; i++) {              /* iterate string */
8          if (mostRecentChar == string[i]) {          /* count adj chars */
9              charCount++;
10         } else {                                    /* print & reset */
11             printf("%d%c", charCount, mostRecentChar);
12             charCount = 1;
13             mostRecentChar = string[i];             /* update last seen */
14         }
15     }
16     printf("%d%c", charCount, mostRecentChar);
17 }
18
19 int main() {
20     char string[] = "AAABBAAAA";
21     rleCompression(string, (sizeof(string)/sizeof(string[0]))-1);
22     return 0;
23 }
```

3A2B4A

Task 3.1 - Bubble Sort Implementation

- Pseudocode from slide 5

```
1  void bubbleSort(int array[], int length) {
2      int counter = 0;
3      for (int i = length - 1; i > 0; i--) {
4          for (int j = 1; j <= i; j++) {
5              counter++;                          /* Counter for task 3.3 */
6              if (array[j] < array[j - 1]) { swap(array, j, j - 1 ); }
7          }
8      }
9      printf("Counter: %d \n", counter);
10 }
11 int main() {
12     int array[] = {0, 1, 3, 4, 2, 8, 9, 5, 6, 7};
13     int length = sizeof(array) / sizeof(array[0]);
14     bubbleSort(array, length);
15     for (int i = 0; i < length; ++i) { printf("%d, ", array[i]); }
16     return 0;
17 }
```

```
Counter: 45
0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
```

Task 3.2 - Insertion Sort Implementation

- Pseudocode from slide 6

```c
void insertionSort(int array[], int length) {
    int counter = 0;
    for (int i = 1; i < length; i++) {
        int j = i - 1;
        int current = array[i];
        while (j >= 0 && array[j] > current) {
            counter++;                          /* Counter for task 3.3 */
            array[j + 1] = array[j];  j--;
        }
        array[j + 1] = current;
    }
    printf("Counter: %d \n", counter);
}
int main() {
  int array[] = {0, 1, 3, 4, 2, 8, 9, 5, 6, 7};
  int length = sizeof(array) / sizeof(array[0]);
  insertionSort(array, length);
  for (int i = 0; i < length; i++) { printf("%d, ", array[i]); }
  return 0;
}
```

```
Counter: 8
0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
```

Task 3.3 - Reverse Sorting Modification

Observations

- Insertion Sort runs through its innermost loop significantly less than the Bubble Sort
  - Bubble Sort: Inner loop always completes
  - Insertion Sort: Breaks of the innermost loop as soon as the subarrray is sorted

Task 3.4 - Worst Case Array

Find an array that causes the highest possible counter

$$C = \sum_{i=2}^{n} (i-1) = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = \frac{n^2 - n}{2}$$

Figure: Bubble Sort: comparisons are always the same

Best Case: (sorted)

$$Cmin = \sum_{i=2}^{n} 1 = n - 1$$

if the element is the greatest and stays at that position (does not go into the while loop); so just one step (the false of the while)

$$Mmin = \sum_{i=2}^{n} 2 = 2(n-1) = 2n - 2$$

Worst Case: (reversed sorted)

$$Cmax = \sum_{i=2}^{n} \underbrace{(i-1)}_{\text{while loop swapping}} = \frac{n^2 - n}{2}$$

if the element is the smallest and needs to go to the beginning of the array; every element in the sorted subarray needs to be checked, and also swapped

$$Mmax = \sum_{i=2}^{n} (i+1) = \frac{n^2 + 3n - 4}{2}$$

Figure: Insertion Sort: Maximum counter if reversed sorted (if element is the smallest and needs to go to the beginning, every element in sorted subarray needs to be checked)

Task 3.5 - Try with Lots of Elements

Counter: 704982704

Task 4 - Implementation

```
1  int zeroSubarray(int const array[], int length) {
2      /* loop through possible subarrays */
3      for (int i = 0; i < length; i++) {
4          int sum = 0;
5          for (int j = i; j < length; j++) { /* widen subarray */
6              sum += array[j];        /* add to sum the current element */
7              if (sum == 0) {         /* found possible subarray? */
8                  return 1;
9              }
10         }
11     }
12     return 0;
13 }
14 int main() {
15     int array[] = {3, -2, 4, 2, 1, -5};
16     int arrayLength = sizeof(array) / sizeof(array[0]);
17     printf("Solution: %d", zeroSubarray(array, arrayLength));
18     return 0;
19 }
```

```
Solution: 1
```

Task 4 - Analysis

- What is complexity? (How many times is `if` statement executed?)
- Is there a more efficient way?
- Yes, we will look at this kind of problem in Exercise 11 (Dynamic Programming)

Main Takeaways

- Be careful with arrays in `C`
- Sorting algorithms can have n^2 runtime
  - They implement nested `for` or `while` loops then

Next Week

- Submit Questions:
  - `https://cryptpad.fr/form/#/2/form/view/43W06FSiZRjEDGf5k9nUpfbFFQ7+zzDiUNg13tUr5+s/`
- Hints for next week:
  - Concrete mathematics on recursion (PDF on OLAT; `Materials/Labs/Lab 2/exercise02_concrete_mathematics.pdf`)
  - Try to find key takeaways and share them next week

More on C

- Understand this by the end of the course:
  `https://www.youtube.com/watch?v=G7LJC9vJluU`