University of Zurich UZH

Department of Informatics
Database Technology Group
Prof. Dr. M. Böhlen

Data Structures and Algorithms
Assignment 2

Mar 7, 2016

Algorithmic Complexity and Correctness [25 points]

Task 1. [20 points] Given an unsorted array $A[1..n]$ of integers and an integer k, the following algorithm calculates the maximum value of every contiguous subarray of size k. For instance, if $A = [8, 5, 10, 7, 9, 4, 15, 12, 90, 13]$, and $k = 4$, then findKMax(A, 4, 10) returns *10 10 10 15 15 90 90*.

```
Algo: findKMax(A, k, n)
Input: array $A[1..n]$ of length $n$, $1 \leq k \leq n$
Output: print the maximum value of every contiguous
        subarray of size $k$
for $i = 1$ to $n - k + 1$ do
    $max = A[i]$;
    for $j = 1$ to $k - 1$ do
        if $A[i + j] > max$ then
            $max = A[i + j]$;
    print($max$)
```

a) Implement the algorithm as a C program that reads the elements of A, reads k and then prints the result of findKMax. An input/output example is illustrated bellow (input is typeset in bold):

```
Elements of A: 8 5 10 7 9 4 15 12 90 13 end
Type k: 4
Results: 10 10 10 15 15 90 90
```

b) Do an exact analysis of the running time of the algorithm.

c) Determine the best and the worst case of the algorithm. What is the running time and asymptotic complexity in each case?

d) What influence has the parameter k on the asymptotic complexity?

Task 2. [5 points] Given an unsorted array $A[1..n]$ that contains only numbers 0, 1, and 2, the following algorithm rearranges the elements of A, such that all occurrences of 0 come before all occurrences of 1 and all occurrences of 1 come before all occurrences of 2. State a loop invariant for the algorithm partitionValues above and show that it is correct.

```
Algo: partitionValues(A, n)
Input: array $A[1..n]$ of length $n$
Output: array $A[1..n]$ rearranged
$k = 1$;
$l = 1$;
$m = n$;
while $l \leq m$ do
    if $A[l] = 0$ then
        swap(A[k], A[l]);
        $k = k + 1$;
        $l = l + 1$;
    else if $A[l] = 1$ then
        $l = l + 1$;
    else
        swap(A[l], A[m]);
        $m = m - 1$;
```

Asymptotic Complexity [3 points]

Task 3. [3 points] Calculate the asymptotic tight bound for the following functions and rank them by their order of growth (lowest first). Clearly work out the calculation steps in your solution.

$$f_1(n) = \log(\pi n) + \log(100^{\log n})$$
$$f_2(n) = 10^{\lg 20} n^4 + 8^{229} n^3 + 20^{231} n^2 + 128 n \log n$$
$$f_3(n) = \log n^{2n+1}$$
$$f_4(n) = 101^{\sqrt{n}}$$
$$f_5(n) = 2^n + \sqrt{n}$$
$$f_6(n) = (n+1)!$$

Special Case Analysis [15 points]

Task 4. [15 points] In mathematics, the act of rearranging the elements of an array A is called permuting and a resulting array is called a permutation of A. Strings in C are represented as arrays of characters, terminated by a special character $'\backslash 0'$. Given two strings A and B, develop an algorithm that checks if B is a permutation of A. For example, if `B = "aabb"` and `A = "baba"`, the return value will be **TRUE**. If `B = "ab"` and `A = "baba"`, the return value will be **FALSE**.

a) Specify all the special cases that need to be considered and provide examples of the input data for each of them.

b) Write a C program implementing your algorithm and make sure it runs for all the special cases you provided. Include a function `int permutation(char A[], char B[])` which returns 1 if $B[]$ is a permutation of $A[]$, and 0 otherwise.

Attention! You are not allowed to use string-functions and/or `string.h`.

Recurrences [12 points]

Task 5. [6 points] Consider the recurrence:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/6) + T(n/2) + n & \text{if } n > 1 \end{cases}$$

a) Draw a recursion tree and use it to estimate the asymptotic upper bound of $T(n)$. Include the tree-based calculations that led to your estimate. [3 points]

b) Prove the correctness of your estimate using the substitution method. [3 points]

Task 6. [6 points] Calculate the asymptotic tight bound of the following recurrences. If the Master Theorem can be used, write down a, b, $f(n)$ and the case (1-3).

1. $T(n) = 3T(\frac{n}{9}) + 32\sqrt{n}$
2. $T(n) = 16T(\frac{n}{4}) + n^3$
3. $T(n) = \sqrt{2}T(\frac{n}{2}) + \log n$
4. $T(n) = T(n-2) + n$

Submission

Submit a zipped folder *a<exercise number>_<family name>_<matriculation number>.zip* where `family name` and `matriculation number` correspond to your personal data. This folder should include:

a) the C-files you created for the tasks where an implementation was needed. Each C-file should be named as *task<task number>.c*

b) a pdf named *a<exercise number>.pdf* with the solutions for the rest of the tasks.

Make sure that both in the C-files as well as in the pdf file you submit, your personal data is included (in the form of comments or a note).

Deadline: **Sunday, March 20th at 23:59**.