University of Zurich UZH

Solution

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

Data Structures and Algorithms
Assignment 2 / **Solution**

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
```

```c
#include <stdio.h>
#include <stdlib.h>

void findKMax(int a[], int k, int n)
{
    int max;
    int i, j;

    printf("Results = ");
    for (i = 0; i <= n-k; i++)
    {
       max = a[i];

       for (j = 1; j < k; j++)
       {
          if (a[i+j] > max)
             max = a[i+j];
       }
       printf("%d ", max);
    }
    printf("\n");
}

void main() {
    int a[100];
    int k, i, n;

    printf("Type elements of A seperated by spaces (type 'end' to
         stop): ");
    i=0;
    while(scanf("%d", &a[i]) == 1) i++;
    n=i;

    // Read but do not store any terminating not integer values ('end')
    scanf("%*s");

    printf("Type k: ");
    scanf("%d", &k);

    findKMax(a, k, n);
}


// gcc task1.c -o task1; ./task1
```

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

Instruction	# of times executed	Cost
for = 1 to $n-k+1$ do	$n-k+2$	c_1
$max = A[i]$	$n-k+1$	c_2
for $j=1$ to $k-1$ do	$(n-k+1)k$	c_3
if $A[i+j] > max$ then	$(n-k+1)(k-1)$	c_4
$max = A[i+j]$	$\alpha(n-k+1)(k-1)$*	c_5
print(max)	$n-k+1$	c_6

* $0 \le \alpha \le 1$

$T(n) = c_1(n-k+2) + c_2(n-k+1) + c_3(n-k+1)k + c_4(n-k+1)(k-1) + c_5\alpha(n-k+1)(k-1) + c_6(n-k+1) \Rightarrow$
$T(n) = c_1 + (c_1 + c_2 - c_4 + c_6 - c_5 a + (c_3 + c_4 + c_5 a)k)(n-k+1)$

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

Best case

$\alpha = 0, k = n,$

$T_{\text{best}}(n) = 2c_1 + c_2 - c_4 + c_6 + (c_3 + c_4)n$

Worst case

$\alpha = 1, k = \frac{n}{2},$

$T_{\text{worst}}(n) = (2c_1 + c_2 - c_4 - c_5 + c_6) + (c_1 + c_2 + c_3 + c_6)\frac{n}{2} + (c_3 + c_4 + c_5)\frac{n^2}{4}$

Asymptotic complexity of best and worst case

$T_{\text{best}}(n) = \Theta(n), T_{\text{worst}}(n) = \Theta(n^2)$

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

Parameter k has an influence on asymptotic complexity and this is why it changes the order of the asymptotic complexity for the best and the worst case. However for each one of these two cases parameter k is fixed, it is eliminated from the asymptotic complexity expression and only n remains (in quadratic or linear form).

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]);
        $m = m - 1$;
```

Loop Invariant

- $A[1..k-1] = 0$
- $A[k..l-1] = 1$
- $A[m+1..n] = 2$

Initialization

- $k = 1$, $A[1..k-1] = 0$; True, because $A[1..0] = \emptyset$

- $k = 1$, $l = 1$, $A[k..l-1] = 1$; True, because $A[1..0] = \emptyset$
- $m = n$, $A[m+1..n] = 2$; True, because $A[n+1..n] = \emptyset$

Maintenance
We need to investigate three cases.
Case 1 $A[l] = 0$

- Before iteration it is true that $A[1..k-1] = 0$.
  After $swap(A[k], A[l])$ it is true that $A[1..k] = 0$, because of case assumption.
  After increment of k it is true that $A[1..k-1] = 0$.
- Before iteration it is true that $A[k..l-1] = 1$.
  After $swap(A[k], A[l])$ it is true that $A[k+1..l] = 1$.
  After increment of k and l it is true that $A[k..l-1] = 1$.
- Nothing is changed in $A[m+1..n]$

Case 2 $A[l] = 1$

- Nothing is changed in $A[1..k-1]$.
- Before iteration it is true that $A[k..l-1] = 1$.
  After increment of l it is true that $A[k..l-1] = 1$, because of case assumption.
- Nothing is changed in $A[m+1..n]$

Case 3 $A[l] = 2$

- Nothing is changed in $A[1..k-1]$.
- Nothing is changed in $A[k..l-1]$.
- Before iteration it is true that $A[m+1..n] = 2$.
  After $swap(A[l], A[m])$ it is true that $A[m..n] = 2$, because of case assumption.
  After decreasing m it is true that $A[m+1..n] = 2$

Termination
At the end of the last iteration of the while loop we have $l > m$. Because each loop iteration either increases l by 1 or decreases m by 1 (never both of them), we must have $l = m+1$. Substituting $m+1$ for l in the loop invariant we have that:

- $A[1..k-1] = 0$
- $A[k..m] = 1$
- $A[m+1..n] = 2$

Subarrays $A[1..k-1], A[k..m], A[m+1..n]$ compose the enire array which is sorted as required.

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.

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

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

Solution: $f_1, f_3, f_2, f_4, f_5, f_6$

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.

#	Case	A	B	result
1	A and B are NULL	NULL	NULL	TRUE
2	A is NULL	NULL	“hello”	FALSE
3	B is NULL	“hello”	NULL	FALSE
4	A and B are empty	“”	“”	TRUE
5	A is empty	“”	“hello”	FALSE
6	B is empty	“hello”	“”	FALSE
7	A is longer than B	“hello”	“hel”	FALSE
8	B is longer than A	“hel”	“hello”	FALSE
9	A = B	“hello”	“hello”	TRUE

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.

```
#include <stdio.h>

int permutation(char A[], char B[]) {
    int i,j;
    int temp;

    /* Case 1 */
    if(A == NULL && B == NULL)
        return 1;

    /* Cases 2 & 3 */
    if(A == NULL || B == NULL)
        return 0;

    /* Case 4 */
    if(A[0] == '\0' && B[0] == '\0')
        return 1;

    /* Cases 5 & 6*/
    if(A[0] == '\0' || B[0] == '\0')
        return 0;

    for (i=0; A[i] != '\0'; i++) {

        j = i;
        while (A[i] != B[j] && B[j] != '\0') j++;

        /* Case 7 */
        if (B[j] == '\0')
            return 0;
        else {
            temp = B[i];
            B[i] = B[j];
            B[j] = temp;
        }
    }
    if (B[i] == '\0')
        return 1;
    else
        /* Case 8 */
        return 0;
}


void main() {
    int i, maxstrlen = 100;
    char a[maxstrlen];
    char b[maxstrlen];
    char t[maxstrlen];
    a[0] = '\0';
    printf("Type a: ");
    scanf("%[^\n]s", a);
    getc(stdin);

    b[0] = '\0';
    printf("Type b: ");
    scanf("%[^\n]s", b);
    getc(stdin);

    if (permutation(a, b))
        printf("TRUE\n");
    else
        printf("FALSE\n");

    // You input different variables to a,b to check it for all the cases
}

// gcc task3.c -o task3; ./task3
```

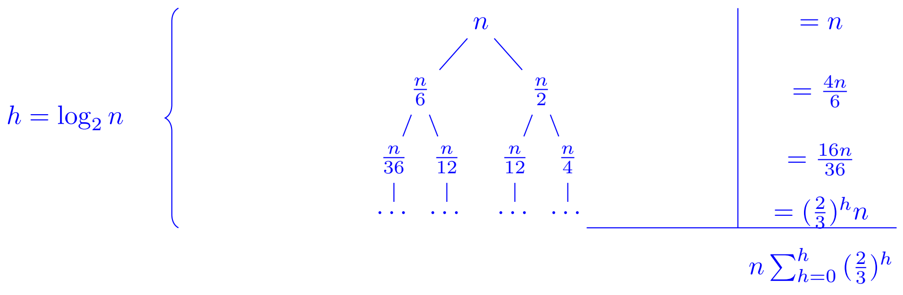
Attention! You are not allowed to use string-functions and/or `string.h`.

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]



Longest branch on the right. Tree grows until $\left(\frac{1}{2}\right)^h n = 1 \implies h = \lg$
Guess: $O(n)$

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

Inductive Step

- $T(n) \leq cn \implies T(n/6) + T(n/2) + n \leq c\frac{n}{6} + c\frac{n}{2} + n$

? $c\frac{n}{6} + c\frac{n}{2} + n \leq cn$

Proof

- $c\frac{n}{6} + c\frac{n}{2} + n \leq cn$, dividing both parts by n we get $c\frac{1}{6} + c\frac{1}{2} + 1 \leq c$
- $\frac{2}{6}c \geq 1$, $c \geq 3$

Asymptotic Complexity

$T(n) = O(n)$, for $c \geq 3$

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. $T(n) = 3T(\frac{n}{9}) + 32\sqrt{n}$
   $a = 3$, $b = 9$, $f(n) = 32\sqrt{n}$, Case 2:
   $T(n) = \Theta(\sqrt{n}\log_9 n)$

2. $T(n) = 16T(\frac{n}{4}) + n^3$
   $a = 16$, $b = 4$, $f(n) = n^3$, Case 3:
   Regularity Condition: if $c = \frac{1}{4}, a \cdot f(n/b) = 16 \cdot (\frac{n}{4})^3 \leq \frac{1}{4} \cdot n^3$
   $T(n) = \Theta(n^3)$

3. $T(n) = \sqrt{2}T(\frac{n}{2}) + \log n$
   $a = \sqrt{2}$, $b = 2$, $f(n) = \log n$, Case 1:
   $T(n) = \Theta(\sqrt{n})$

4. $T(n) = T(n-2) + n$

$$
\begin{aligned}
T(n) &= T(n-2) + n \\
&= T(n-4) + n + (n-2) \\
&= n + (n-2) + (n-4) + ... + 2 + c \\
&= (2 + 4 + 6 + ... + (n-2) + n) + c \\
&= 2(1 + 2 + 3 + ... + \frac{n}{2} - 1 + \frac{n}{2}) + c \\
&= 2(\frac{1}{2}\frac{n}{2}(1 + \frac{n}{2})) + c \\
&= \frac{n^2}{4} + \frac{n}{2} + c \\
&= \Theta(n^2)
\end{aligned}
$$

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**.