University of Zurich

Department of Informatics
Database Technology Group
Prof. Dr. Anton Dignös

Informatics II
Exercise 3

Mar 08, 2020

Goals:

- practice calculation of asymptotic tight bound.
- practice running time analysis
- practice best case and worst case analysis
- identify the influence of a parameter

Algorithmic Complexity and Correctness

Task 1. Consider algorithm `whatDoIDo(A, n, k)` below. Input array `A[]` contains n integers, and k is an integer.

```
Algo: WHATIDO(A, n, k)
sum = 0;
for i = 1 to k do
    maxi = i;
    for j = i + 1 to n do
        if A[j] > A[maxi] then
            maxi = j;
    sum = sum + A[maxi];
    swp = A[i];
    A[i] = A[maxi];
    A[maxi] = swp;
return sum
```

a) What does algorithm `whatDoIDo(A,k)` do?

- The algorithm returns the sum of its k-biggest elements in array $A[]$. For example, given the input array $A = [12, 4, 10, 2, 8]$, if $k = 3$, 3-biggest elements are 12, 10, and 8, hence, sum= $12 + 10 + 8 = 30$; if $k = 4$, 4-biggest elements are 12, 10, 8, and 4, and sum $= 12 + 10 + 8 + 4 = 34$.

b) Implement algorithm `whatDoIDo(A,k)` in C, and call your implementation in a complete C program.

Given an array a and an integer k, run the nested loop where outer loop runs from $i = 0$ to k and inner loop starts from $i + 1$ to array size. The inner loop finds the index of i^{th} maximum element. After exiting from inner loop, i^{th} maximum element is added to sum variable and then this i^{th} maximum value is swaped with the element present in i^{th} index so that the same maximum element is not found again during the next iteration of outer loop.

```
int sumKBiggest(int a[], int n, int k) {
  int i, j, swp;
  int maxi;
  int sum = 0;

  for (i = 0; i < k; i++) {
    maxi = i;
    for (j = i + 1; j < n; j++) {
      if (a[j] > a[maxi]) { maxi = j; } //replace element with higher
    }
    sum += a[maxi]; //add kth biggest element to sum
    swp = a[i]; //swap to the left so it's not checked again
    a[i] = a[maxi];
    a[maxi] = swp;
  }
  return sum;
}
```

c) Conduct an exact analysis of the running time of algorithm `whatDoIDo(A,k)`.

Instruction	# of times executed	Cost
$sum := 0$	1	c_1
for $i := 1$ **to** k **do**	$k+1$	c_2
$maxi := i$	k	c_3
for $j := i+1$ **to** n **do**	$\left(kn - \frac{k(k+1)}{2}\right)^{*} + 2k^{**}$	c_4
if $A[j] > A[maxi]$ **then**	$kn - \frac{k(k+1)}{2}$	c_5
$maxi := j$	$\alpha\left(kn - \frac{k(k+1)}{2}\right)^{***}$	c_6
$sum := sum + A[maxi]$	k	c_7
$swp := A[i]$	k	c_8
$A[i] := A[maxi]$	k	c_9
$A[maxi] := swp$	k	c_{10}
return sum	1	c_{11}

$$^{*}\ (n-2+1)+(n-3+1)+\ldots+(n-k) = \sum_{q=1}^{k}(n-q) = kn - \frac{k(k+1)}{2}$$

** k times for $i+1$ and k times for termination condition

*** $0 \leq \alpha \leq 1$

$$T(n) = c_1 + c_2(k+2) + c_3k + c_4(kn - \tfrac{k(k+1)}{2} + 2k) + c_5(kn - \tfrac{k(k+1)}{2}) +$$
$$+ c_6(\alpha(kn - \tfrac{k(k+1)}{2})) + (c_7 + c_8 + c_9 + c_{10})k + c_{11}$$

In conclusion, $T(n) = k * n$.

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

Best case
$\alpha = 0, k = 1,$

$T_{\text{best}}(n) = c_1 + 2c_2 + c_3 + c_4(n+1) + c_5(n-1) + 0 + c_7 + c_8 + c_9 + c_{10} + c_{11}$

$T_{\text{best}}(n) = O(n)$

Worst case
$\alpha = 1, k = n,$

$T_{\text{worst}}(n) = c_1 + c_2(n+1) + c_3n + c_4(\frac{n^2}{2} + \frac{3}{2}n) + c_5(\frac{n^2}{2} - \frac{n}{2}) + c_6(\frac{n^2}{2} - \frac{n}{2}) +$

$(c_7 + c_8 + c_9 + c_{10})n + c_{11}$

$T_{\text{worst}}(n) = O(n^2)$

Asymptotic complexity of best and worst case $T_{\text{best}}(n) = O(n)$

$T_{\text{worst}}(n) = O(n^2)$

e) What influence does the parameter k have in the asymptotic complexity?

The complexity of the algorithm is $O(k * n)$, so k determines the time complexity of the algorithm. If k is close to n, the complexity is $O(n^2)$; if k is a small integer, the complexity is $O(n)$.

f) List special cases, and provide an example for each special case if possible.

- k is negative or 0. For example, $k = -1$ or $k = 0$.
- k is bigger than n.
- Array $A[...]$ is empty. For example, $n = 0$.

Task 2. Calculate the asymptotic tight bound for the following functions and rank them by their order of growth (lowest first). Clearly work out the calculation step by step in your solution.

$$
\begin{aligned}
f_1(n) &= (n+3)! \\
f_2(n) &= 2\log(6^{\log n^2}) + \log(\pi n^2) + n^3 \\
f_3(n) &= 4^{log_2 n} \\
f_4(n) &= 12\sqrt{n} + 10^{223} + \log 5^n \\
f_5(n) &= 10^{\lg 20} n^4 + 8^{229} n^3 + 20^{231} n^2 + 128 n \log n \\
f_6(n) &= \log n^{2n+1} \\
f_7(n) &= 101^{\sqrt{n}} \\
f_8(n) &= \log^2(n) + 50\sqrt{n} + \log(n) \\
f_9(n) &= n^n + 2^{2n} + 13^{124} \\
f_{10}(n) &= 14400
\end{aligned}
$$

- $f_1(n) = (n+3)! \in \Theta((n+3)!)$
- $f_2(n) = 2\log(6^{\log n^2}) + \log(\pi n^2) + n^3 = 2\log n^2 \log 6 + \log\pi + \log n^2 + n^3 = 4\log 6 \log n + \log\pi + 2\log n + n^3 \in \Theta(n^3)$
- $f_3(n) = 4^{log_2 n} = (2^2)^{log_2 n} = (2^{log_2 n})^2 = n^2 \in \Theta(n^2)$
- $f_4(n) = 12\sqrt{n} + 10^{223} + \log 5^n = 12\sqrt{n} + 10^{223} + n\log 5 \in \Theta(n)$
- $f_5(n) = 10^{\lg 20} n^4 + 8^{229} n^3 + 20^{231} n^2 + 128 n \log n \in \Theta(n^4)$
- $f_6(n) = \log n^{2n+1} = (2n+1)\log n \in \Theta(n \log n)$
- $f_7(n) = 101^{\sqrt{n}} \in \Theta(101^{\sqrt{n}})$
- $f_8(n) = \log^2(n) + 50\sqrt{n} + \log(n) \in \Theta(\sqrt{n})$
- $f_9(n) = n^n + 2^{2n} + 13^{124} = n^n + 4^n + 13^{124} \in \Theta(n^n)$
- $f_{10}(n) = 14400 \in \Theta(1)$

$f_{10} < f_8 < f_4 < f_6 < f_3 < f_2 < f_5 < f_7 < f_1 < f_9$

Task 3. Let n be an exact power of 2, $n = 2^k$.
Use mathematical induction over k to show that the solution of the recurrence involving positive constants $c, d > 0$

$$T(n) = \left\{ \begin{array}{ll} d, & \text{if } n = 2^0 = 1 \\ 2T(n/2) + cn & \text{if } n = 2^k \text{and } k \geq 1 \end{array} \right\}$$

is $T(n) = dn + cn \log(n)$

Hint: you may want to rewrite the above as $T(2^k) = d2^k + c2^k \log(2^k) = d2^k + c2^k \cdot k$.

Base case: we first prove the statement for k = 0. Here the first line in the definition of $T(n)$ applies: for k = 0 we have $T(2^0) = d = d2^0 + c2^0 \cdot 0$ For the inductive step, assume that the claim holds for $k \geq 0$, that is $T(2^k) = d2^k + c2^k \cdot k$ Then we show that it holds for $k + 1$:

- $T(2^{k+1}) = 2T(2^{k+1}/2) + c2^{k+1}$ (using second line of reccurence)
- $T(2^{k+1}) = 2T(2^k) + c2^{k+1}$ (apply rule: $2^{k+1}/2 = 2^k$)
- $= 2 \cdot (d2^k + c2^k \cdot k) + c2^{k+1}$ (using the assumption here)
- $= d2^{k+1} + c2^{k+1} * k + c2^{k+1}$ (as $2 \cdot 2^k = 2^{k+1}$)
- $= d2^{k+1} + c2^{k+1} \cdot (k + 1)$
- $= d2^{k+1} + c2^{k+1} \cdot \log(2^{(k+1)})$ (apply $k + 1 = \log(2^{k+1})$

Hence the statement also holds for $(k + 1)$. Proved by Induction.