Tutorial for homework 1

Problem 1 (0.1 point). (a) The sum of arithmetic progression $a_k, a_{k+1}, \dots, a_l$ (where $a_i = a_k + (i-k)\cdot d$, $i = k+1, \dots, l$) may be expressed by formula

$$\sum_{j=k}^{l} a_j = \frac{a_k + a_l}{2}(l-k+1).$$

The sum of geometric progression $b_1, b_2, b_3, \dots, b_k$ (where $b_i = b_1 \cdot q^{i-1}$, $i = 2, \dots, k$ and $q \neq 1$) may be expressed by formula

$$\sum_{i=1}^{k} b_i = b_1 \frac{1-q^k}{1-q}.$$

Using these formulas as well as formula

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6},$$

which may be easily proved by mathematical induction, find the sum $f(n) = \sum_{k=u(n)}^{v(n)} g(k)$.

(b) Find asymptotics of $f(n)$, i.e. constants a and b such that $f(n) \sim an^b$, when $n \to \infty$. If $f(n)$ grows exponentially, then find constants a and b such that $f(n) \sim ab^n$.

Remark. $f(n) \sim g(n)$ (“f is asimptotically equal to g”) if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1.$$

Example 1. Find the sum $f(n) = 1^2 + 3^2 + 5^2 + \dots + n^2$, where $n = 2k+1$ is an odd number. We have:

$$\begin{aligned}
f(n) &= \sum_{k=0}^{(n-1)/2} (2k+1)^2 = \sum_{k=0}^{(n-1)/2} (4k^2+4k+1) = 4\sum_{k=0}^{(n-1)/2} k^2 + 4\sum_{k=0}^{(n-1)/2} k + \sum_{k=0}^{(n-1)/2} 1 \\
&= 4\frac{\frac{n-1}{2}(\frac{n-1}{2}+1)n}{6} + 4\frac{n-1}{4}\left(\frac{n-1}{2}+1\right) + \left(\frac{n-1}{2}+1\right) \\
&= \frac{(n-1)(n+1)n}{6} + \frac{(n-1)(n+1)}{2} + \frac{n+1}{2} \\
&= \frac{n^3 - n + 3n^2 - 3 + 3n + 3}{6} = \frac{n^3 + 3n^2 + 2n}{6} = \frac{n(n+1)(n+2)}{6}.
\end{aligned}$$

When $n = 2k+1 \to \infty$, we have $f(n) \sim \frac{n^3}{6}$, i.e., $a = \frac{1}{6}$, $b = 3$.

Problem 2 (0.3 point). Let us consider program code fragment depending on parameter n.

(a) Supposing that each operation (assignment, arithmetical, comparison etc.) has weight 1, find an exact number of operations $L(n)$. Number of operations should be counted for the worst case of initial data (maximum taken on all possible data of size n).

(b) Find asymptotics of $L(n)$, i.e. constants a and b such that $L(n) \sim an^b$, when $n \to \infty$.

(c) A small constant c is given. Indicate an example of initial data that requires exactly $L(c)$ steps and enumerate all these steps.

(d) Find order of growth for the program execution time $T(n)$, i.e., find a constant d such that $T(n) = \Theta(n^d)$, when $n \to \infty$. Counting time $T(n)$ we consider that execution of different operations (assignment, arithmetical, comparison etc.) takes different time: operation of type i requires time c_i.

Indications

1. $f(n) = \mathrm{O}(g(n))$ (or $f(n) \preccurlyeq g(n)$) (we say that "f has no higher asymptotical growth order than g") if $\exists N \in \mathbb{N}$ and $\exists c > 0$: $f(n) \leqslant cg(n)\ \forall n \geq N$;

2. $f(n) = \Theta(g(n))$ (or $f(n) \asymp g(n)$) (we say that "f and g have the same asymptotical growth order") if $f(n) = \mathrm{O}(g(n))$ and $g(n) = \mathrm{O}(f(n))$.

3. Counting steps we consider that both assignment and arithmetical operation require 1 step, i.e., instruction $a := 1$ takes 1 step and instruction $a := b+c$ also takes 1 step. However, instruction $a := b + c - d$ takes 2 steps. Instruction $A[i + j] := b + c$ also takes 2 steps: (1) we count index value $k = i + j$, and (2) we assign to $A[k]$ value $b + c$.

4. Execution of cycle **for** of length k, i.e. instruction **for** $i := 1$ **to** k **do** requires $2(k + 1)$ steps because each time adding $i := i + 1$ and comparison $i \leq k$? are executed. At the end of cycle once more adding $i := k + 1$ and comparison $k + 1 \leq k$? will be executed but the body of cycle will not be executed anymore.

Example 2. An array of integer numbers $A[1 : n]$, constant $c = 2$ and a fragment of sorting program INSERTION_SORT code are given:

```
for j := 2 to n do
    key := A[j]
    i := j − 1
    while i > 0 and A[i] > key do
        A[i + 1] := A[i]
        i := i − 1
    A[i + 1] := key
```

(a) At first let us estimate number of steps for each program code line:

for $j := 2$ **to** n **do**	$2n$
$\quad key := A[j]$	$n-1$
$\quad i := j-1$	$n-1$
$\quad$ **while** $i > 0$ **and** $A[i] > key$ **do**	$3\sum_{j=2}^{n} t_j + n - 1$
$\qquad A[i+1] := A[i]$	$2\sum_{j=2}^{n} t_j$
$\qquad i := i-1$	$\sum_{j=2}^{n} t_j$
$\quad A[i+1] := key$	$2(n-1)$

where t_j denotes how many times the body of cycle **while** is executed (this number depends on j). Verification of cycle conditions **while** $i > 0$ **and** $A[i] > key$ requires 3 steps (2 comparisons plus logical operation **and**). However, in last verification (when $i = 0$) after first step (checking $0 < 0$?) we have at once that cycle conditions are not satisfied and finish execution of the cycle (in opposite case the value $A[0]$ should be undefined).

It is easy to see that for INSERTION_SORT algorithm the worst case appears when initial data is ordered in reverse (i.e., decreasing) order. In this case for each i starting from $i = j-1$ and finishing with $i = 1$ we will have $A[i] > key$. Therefore, in the worst case the body of the cycle will be executed $j-1$ times.

Now let us count total number of steps:

$$\begin{aligned} L(n) &= 2n + 5(n-1) + 6\sum_{j=2}^{n} t_j = 7n - 5 + 6\sum_{j=2}^{n}(j-1) = 7n - 5 + \frac{6n(n-1)}{2} \\ &= 3n^2 + 4n - 5. \end{aligned}$$

(b) When $n \to \infty$, we receive $L(n) \sim 3n^2$ (i.e., $a = 3$, $b = 2$), since

$$\lim_{n\to\infty} \frac{3n^2 + 4n - 5}{3n^2} = \lim_{n\to\infty}\left(1 + \frac{4}{3n} - \frac{5}{3n^2}\right) = 1.$$

(c) Let us find the worst initial data for $n = 2$. We can choose any two numbers such that $A[2] > A[1]$. Let $A = [7, 3]$. Since $L(2) = 3 \cdot 4 + 8 - 5 = 15$, this data will require 15 steps:

$j := 2$
$2 \leq 2$? YES
$key := 3$
$i := 1$
$1 > 0$? YES
$7 > 3$? YES
YES **and** YES (=YES)
$1 + 1 = 2$
$A[2] := 7$
$i := 0$
$0 > 0$? NO

$0 + 1 = 1$
$A[1] := 3$
$j := 3$
$3 \leq 2$? NO

(d) Finally let us count the total program code execution time $T(n)$. Let constant c_i denote execution time of code line i. Then we have:

for $j := 2$ **to** n **do**	$c_1 n$
$\quad key := A[j]$	$c_2(n-1)$
$\quad i := j - 1$	$c_3(n-1)$
$\quad$ **while** $i > 0$ **and** $A[i] > key$ **do**	$c_4 \sum_{j=2}^{n} t_j + c_5(n-1)$
$\qquad A[i+1] := A[i]$	$c_6 \sum_{j=2}^{n} t_j$
$\qquad i := i - 1$	$c_7 \sum_{j=2}^{n} t_j$
$\quad A[i+1] := key$	$c_8(n-1)$

As above, in the worst case $t_j = j - 1$, so $\sum_{j=2}^{n} t_j = \sum_{j=2}^{n}(j-1) = \frac{n(n-1)}{2}$. Let us count $T(n)$:

$$\begin{aligned} T(n) &= c_1 n + (c_2 + c_3 + c_5 + c_8)(n-1) + (c_4 + c_6 + c_7)\left(\frac{n^2}{2} - \frac{n}{2}\right) \\ &= \frac{c_4 + c_6 + c_7}{2} n^2 + \left(c_1 + c_2 + c_3 + c_5 + c_8 - \frac{c_4}{2} - \frac{c_6}{2} - \frac{c_7}{2}\right) n - (c_2 + c_3 + c_5 + c_8) \\ &= An^2 + Bn - C. \end{aligned}$$

So $T(n) = \Theta(n^2)$, since from one hand we can find a constant $D > 0$ and a natural number N_1 such that $T(n) \leq Dn^2$ for each $n > N_1$, and from other hand we can find a constant $E > 0$ and a natural number N_2 such that $n^2 \leq ET(n)$ for each $n > N_2$. It means that program code execution time $T(n)$ has growth order $d = 2$ (see the initial formulation of task (d)). Algorithms having their complexity order 2 are also called "quadratic algorithms".