https://doi.org/10.53656/math2024-4-2-ont

Science in Education
Научно-методически статии

ON THE TIME COMPLEXITY OF AN ALGORITHM

Krassimir Manev
Journal Mathematics and Informatics – Sofia (Bulgaria)

Abstract. The difficulty of estimation the time complexity of an algorithm in the worst case is not always the same as the mathematical complexity of the solved task. In this paper we demonstrate a mathematically clear task proposed in a programming contest for Grade 6 students. The task has efficient algorithmic solution but estimation of its time complexity in the worst case is really difficult. We provide corresponding reasoning of the estimated complexity and propose some explanation of it understandable for pupils of that age.

Keywords: programming contests; algorithm; subsequence cutting; time complexity of algorithm

1. Introduction

The ability of a programmer to estimate the time complexity of an algorithm is very important. This is especially valid for participants in the international competitions in programming (like ICPC[1] or IOI[2]), as well as in the national competitions, because one of the requirement for successful solving of a competition's task is the time for execution of the solution (i.e. the corresponding program) for single test case to be less than the *time limit* defined by the author of the task. That is why in the process of training of contestants (even of earliest age) their coaches must teach them the bases of the time complexity of algorithms theory[3].

Unfortunately the mathematical hardness of a task, which is always adequate to the age of the contestants in these competitions, as well as the difficulty of some efficient (i.e. quick) algorithm to solve it, is not always comparable with the difficulty of estimation the time complexity of the algorithm in the worst case. Let us consider the following task from the Winter programming competition that took place in Bulgaria in 2012 year, included in the task set for age group D (Grade 6):

Task "Programmer"[4,5]. Write a program that finds the sum

$$\lfloor N/1 \rfloor + \lfloor N/2 \rfloor + \cdots + \lfloor N/N \rfloor$$

for positive integer $N, N \leq 10^9$, where $\lfloor X \rfloor$ denotes the integer part of X.

We do not know the speed of the grading computer for that competition but the time limit of 0.1 sec, defined by the author, clearly shows that the trivial $O(N)$ solution given at Listing 1 will not pass the time limit for tests with large N.

Listing 1. Trivial $O(N)$ solution

```
#include <iostream>
using namespace std;
int main()
{  int N, i; long long sum = 0;
   cin >> N;
   for(i = 1; i <= N; i++)  sum + = N / i;
   cout << sum;
   return 0;
}
```

As the results of the competition show all 26 contestant submitted this trivial solution which have been evaluated with 40 points out of 100. Nobody has tried to make some reasoning and to find the fast algorithm.

2. Subsequence cutting

The general algorithmic scheme **called by us** *Subsequence cutting* is a well-known technique for decreasing the number of steps of an algorithm which has to examine the elements of a sequence in order to calculate some property of the sequence depending on some characteristic of the elements. The approach is able to get down the number of steps of the algorithm when there are subsequences, all elements of which have same characteristic. In such case it could be enough for the algorithm not to examine the elements of a subsequence one by one but to treat whole subsequence in one step.

A typical algorithm implementing the Subsequence cutting scheme is the well known *Binary search* in sorted sequence of N elements. After comparing the searched value with the middle element of the sequence the algorithm eliminates the half of the sequence in the worst case – when searched value is not equal to the middle element. During the second step it eliminates one quarter of the elements (in the worst case) and so on. The time complexity of this kind of subsequences cutting is well known – $O(\log N)$.

In our sample task there are consecutive integers for which the quotient $\lfloor N/i \rfloor$ of the integer division is the same for all i in the subsequence. Obviously for half of the elements – these from $\lfloor N/2 \rfloor + 1$ to N – the quotient is 1 and the algorithm has to add their count to the sum. Next, for one third of the remaining $N - \lfloor N/2 \rfloor$ elements – these from $N - \lfloor N/2 \rfloor - \lfloor N/3 \rfloor + 1$

to $N - \lfloor N/2 \rfloor$ – the quotient is 2 and the algorithm has to add their count multiplied by 2 to the sum, etc. The process will continue to some moment when the current subsequence for last time is composed of 2 elements with the same quotient and then each of the remaining elements will have a specific quotient.

Let, for example, $N = 100$. For numbers from 51 to 100 the quotient is 1; for numbers from 34 to 50 quotient is 2; for numbers from 26 to 33 the quotient is 3; for numbers from 21 to 25 it is 4; for numbers from 17 to 20 – 5; for numbers 15 and 16 – 6; for numbers 13 and 14 – 7. For the other numbers from 12 to 1 the quotient is specific – namely 8, 9, 10, 11, 12, 14, 16, 20, 25, 33, 50 and 100.

So we have the following quick algorithm: while a subsequence of at least two elements with equal quotient is identified, add to sum the number of elements in the subsequence multiplied by the common quotient. Then for each of the other elements add its quotient to the sum. The coded algorithm is shown on Listing 2.

Listing 2. Quick solution

```
#include <iostream>
using namespace std;
int main()
{  long long N, K, sum = 0, i = 1;
   cin >> N; K = N;
   while(1)
   {  K = N / i - N / (i + 1); if(K == 1) break;
      sum += K * i; i++;
   }
   for(int j = N / i; j >= 1; j--) sum += N / j;
   cout  << sum << endl;
   return 0;
}
```

It is clear that, because of the cutting of some very long subsequences, this algorithm is much faster then the linear one. The problem is to estimate the asymptotic behaviour of its time complexity in the worst case.

3. Time complexity of the algorithm

Let us consider the two stages of the algorithm (see fig. 1). On the first stage the algorithm eliminates on each step all elements having the same quotient when they are dividing N – namely $1, 2, \ldots, i(N)$, where $i(N)$ is the

largest quotient for which there is more than one number with such quotient. So, on this stage the algorithm eliminated:

$$N-\left\lfloor\frac{N}{2}\right\rfloor+\left\lfloor\frac{N}{2}\right\rfloor-\left\lfloor\frac{N}{3}\right\rfloor+\cdots+\left\lfloor\frac{N}{i(N)}\right\rfloor-\left\lfloor\frac{N}{i(N)+1}\right\rfloor=N-\left\lfloor\frac{N}{i(N)+1}\right\rfloor \quad (1)$$

numbers, making $i(N)$ steps.

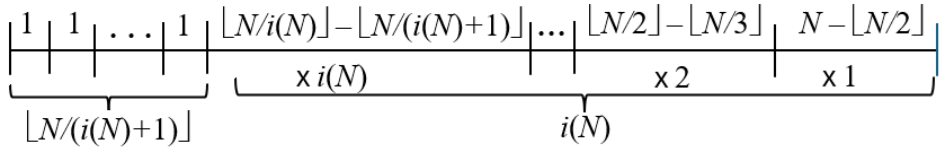

Figure 1. Stages of the algorithm

On the second stage the algorithm examines each of the remaining elements one by one, thus making $\left\lfloor\frac{N}{i(N)+1}\right\rfloor$ more steps. Finally, for the time complexity $T(N)$ of the algorithm we obtain the estimation:

$$T(N)\leq C\left(i(N)+\frac{N}{i(N)+1}\right). \quad (2)$$

for some positive constant C.

By the definition of $i(N)$ we have the inequality:

$$\left\lfloor\frac{N}{i(N)}\right\rfloor-\left\lfloor\frac{N}{i(N)+1}\right\rfloor>1\Rightarrow N/x-N/(x+1)>1, \quad (3)$$

eliminating the integer part notation and introducing the real variable x instead of the searched integer $i(N)$ because we will find the asymptotic behaviour of the time complexity, not its exact value. The exact function of time complexity will be a bit different and we will estimate the difference in the next section.

From the inequality (3) we obtain:

$$x^2+x-N<0\Rightarrow i(N)\leq x=\frac{\sqrt{1+4N}-1}{2}=\sqrt{1/4+N}-\frac{1}{2}. \quad (4)$$

Replacing $i(N)$ in (2) we obtain:

$$\begin{aligned} T(N) &\leq C(\sqrt{1/4+N} - \tfrac{1}{2} + \tfrac{N}{\sqrt{1/4+N}+\frac{1}{2}}) = C(\tfrac{1/4+N-1/4+N}{\sqrt{1/4+N}+\frac{1}{2}}) < \\ &< \tfrac{2CN}{\sqrt{1/4+N}} < \tfrac{2CN}{\sqrt{N}} = 2C\sqrt{N} = O(\sqrt{N}). \end{aligned} \tag{5}$$

4. Discussion

It is clear that the necessary mathematics for the calculations above is not understandable for the contestants of age 12-13 years. So it is necessary to find some understandable for them reasons for the estimation.

It is obvious that, because of the repeated subsequence cutting, the complexity $T(N)$ of the algorithm is better than the linear $O(N)$ – mathematically expressed by $T(N) = o(N)$. The contestants know the process of subsequence cutting of the Binary search whose time complexity is $O(\log N)$. This algorithm eliminates $1/2$ of the elements first, then $1/4$ of the elements, $1/8$ and so on. Our algorithm is obviously slower because it starts with cutting $1/2$ of the elements, but on the second step eliminates $\frac{1}{3} \cdot \frac{1}{2} = \frac{1}{6} < \frac{1}{4}$ of the elements, and leaves $1 - \frac{1}{2} - \frac{1}{6} = \frac{1}{3}$ of them. On the third step it eliminates $\frac{1}{4} \cdot \frac{1}{3} = \frac{1}{12} < \frac{1}{8}$ and so on. So $T(N) = \omega(\log N)$. Which is this function growing faster than $\log N$ and slower than N? Let us use the fillowing:

Lemma (Cormen et al. 2022). $\log N = o(N^p)$ for each real p.

Because $T(N)$ is growing more slowly than N^1 and faster than $\log N$ it is natural to expect that $T(N) = O(N^p)$, for some $p < 1$. It is reasonable to suppose that $p = 1/2$, i.e. $T(N) = O(\sqrt{N})$. The function $\sqrt{N}$ is not totally unknown for contestants. They have met intuitively this function when they searched the divisors of N and realised that there is no sense to check a number $d, d^2 > N$ for being a divisor of N because they have found it together with the smaller divisor N/d.

Listing 3. Counting number of steps

```
#include <iostream>
using namespace std;
int main()
{  long long N, K, sum=0, i=1, steps=0;
   cin>>N; K=N;
   while(1)
   {  K = N / i - N / (i + 1); if(K == 1) break;
      sum += K * i; i++; steps++;
```

```
    }
    for(int j = N / i; j >= 1; j--) sum += N / j;
    cout << N << " " << steps + N / i << endl;
    return 0;
}
```

For checking the hypothesis we will use the program Excel which is known to the contestants from the school classes in Information technologies. It is trivial to calculate the real number of steps that the algorithm made, using an additional variable `steps` to find this count (Listing 3).

Obtained counts of steps are shown in the second row of Table 1 for large values of N in order to show more clear the asymptotic behavior. In the third row of Table 1 the values of the function $2\sqrt{N}$ are shown (because we have no clear estimation of the constant C).

Table 1. Comparing of real and estimated number of steps

N	5E+6	1E+7	5E+7	1E+8	5E+8	1E+9	5E+9	1E+10
prog	4724	6679	14949	21116	47374	67025	149841	212030
$2\sqrt{N}$	4472	6325	14142	20000	44721	63246	141421	200000
T*(N)	4745	6710	15005	21220	47449	67104	150048	212200

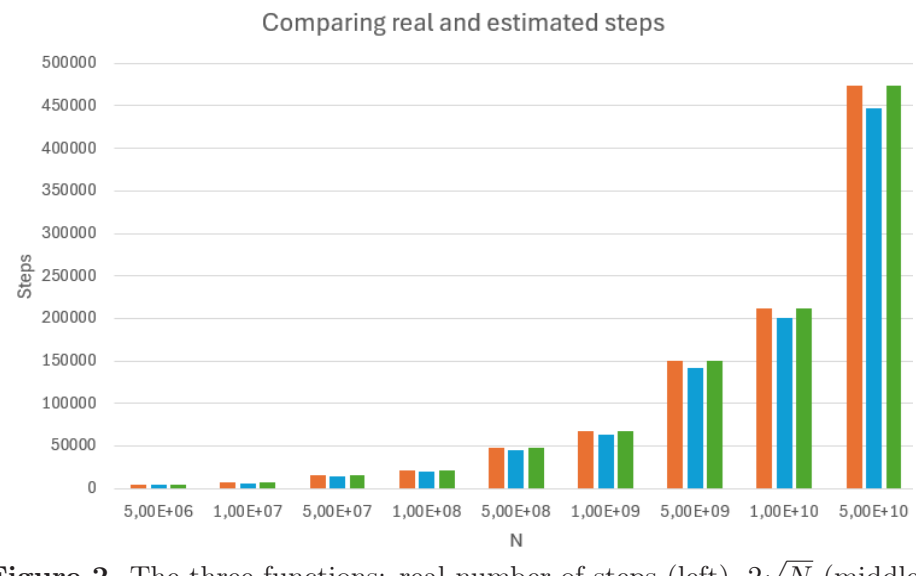

Figure 2. The three functions: real number of steps (left), $2\sqrt{N}$ (middle) and improved estimation $T^*(N)$ (right)

The effect of missing constant C, transforming the results of the integer division to rational values and elimination of some small constants that made the calculation easier is visible – values of the estimation are smaller than the real. But the behavior of both functions is asymptotically the same. This means that the constant in front of $\sqrt{N}$ is not 2 but a bit higher. For experiment the corresponding values of the function $T^*(N) = 2.125\sqrt{N}$ are shown in the last row of Table 1. The values of the three functions are visualized on a diagram (fig. 2) showing that $T^*(N)$ is a very good approximation of the real counts of steps.

Remark. Have in mind that the number of steps made by the algorithm is a discrete (integer) function of the integer variable N and a better approximation must not be expected.

5. Other solutions and a related problem

From the archive of the competition[4] we may see the solution proposed by the author of the task (Listing 4).

Listing 4. Solution of the author

```
#include <iostream>
using namespace std;
int main()
{  int i = 1, N, j; long long s = 0, k;
   cin >> N;
   while(i <= N)
   {  k = N / i;
      j = N / k;
      s = s + k * (j - i + 1);
      i = j + 1;
   }
   cout << s << endl;
}
```

Because of the complex relation between variables `i` and `N` it is difficult to estimate the number of steps of the `while` loop of this algorithm. In addition, the general idea of sequence cutting, which will be helpful for solving other tasks, is not clearly visible in this solution.

In different internet places[6,7] the mathematical solution

$$a_N = \sum_{i=1}^{N} \left\lfloor \frac{N}{i} \right\rfloor = \sum_{i=1}^{k} \left\lfloor \frac{N}{i} \right\rfloor - k^2,$$

where $k = \sqrt{N}$ could be found. This leads to another algorithm with obviously same estimation of the time complexity – $O(\sqrt{N})$, which implementation is shown on Listing 5. But, to guess this formula, and to reason the statement is extremely difficult and is going behind the scope of this paper.

Listing 5. Another $O(\sqrt{N})$ solution

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{  int i, N; long long s = 0, k;
   k=sqrt(N);
   cin >> N;
   for(i = 1; i <= k; i++) s += N/i;
   cout << 2 * s - k * k << endl;
}
```

The sequence $a_N = \sum_{i=1}^{N} \left\lfloor \frac{N}{i} \right\rfloor$ is well known. Various publications about this sequence could be found in (Sloane 2024, seq. A006218).

6. Conclusion

The ability of students that participate in competitions in programming to estimate time complexity of the used algorithms is very important. Sometimes such estimation is difficult even for relatively simple algorithms. The coaches must not avoid providing the necessary explanations in such case, because would question the possibilities and importance of the theory. It is obvious that such explanation will by difficult too and have to be presented to the young programmers in a form adequate of their age.

The computer, with its computational power and various helpful applications, is the inevitable instrument that can help the coach to overwhelm the difficulties.

NOTES

1. International Collegiate Programming Contest (ICPC). https://icpc.global/
2. International Olympiad in Informatics (IOI). https://ioinformatics.org/
3. For competitions in programming the memory complexity of algorithms is also important but for limited amount of tasks.

4. The statement of the task is simplified. See the text of the author of the task K. Kirilova-Lupanova (in Bulgarian) and proposed by her solution at https://arena.olimpiici.com/api/public/problems/960/pdf.
5. Different form of the task could be found in the archive of UvAOJ, https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=27&page=show_problem&problem=2521
6. https://math.stackexchange.com/questions/487401/sum-of-floor-of-harmonic-progression-sum-i-1n-lfloor-frac-ni-rfloor-2-sum
7. https://cs.stackexchange.com/questions/130728/sum-of-series-n-1-n-2-n-3-n-4-n-n

REFERENCES

CORMEN, T.H., LEISERSON, C.E., RIVEST, R.L., STEIN, C., 2009. *Introduction to algorithms*, 3rd edition. MIT press, Cambridge, Massachusetts.

SLOANE, N.J.A., 2024. *The on-line encyclopedia of integer sequences.* https://oeis.org/A006218

✉ **Dr. Krassimir Manev**
ORCID iD: 0000-0002-3275-374X
WoS ResearcherID: IAM-7959-2023
Journal Mathematics and Informatics
125, Tsarigradsko shose Blvd., bl. 5
1164 Sofia, Bulgaria
E-mail: kmanev@azbuki.bg