

Vistula, IT Faculty, 2014

Algorithms and Complexity

Dmitry A. Zaitsev

<http://daze.ho.ua>

Lecture 5:

Time and space complexity of algorithms

Time and space complexity of TM

Time complexity $T(M)$ – the number of M steps.

Space complexity $S(M)$ – the maximal width of M working zone.

Complexity is different for different input data w .

We study dependence of complexity on input data size $n = |w|$

Complexity is still different for different input data w of the same size n .

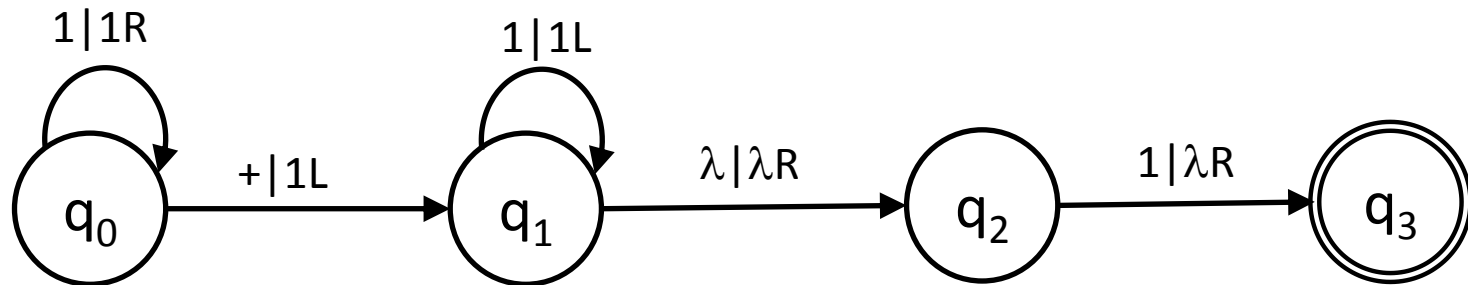
We estimate complexity in the worst (best) cases and on average.

We are absorbed in asymptotic estimations when $n \rightarrow \infty$

We denote $g(n) = O(f(n))$, if there is n_0 and C that for $n > n_0$
 $g(n) \leq C \cdot f(n)$. Other symbols $o(f)$ and $\Omega(f)$.

An example of TM complexity evaluation

Addition of natural numbers in unary numeral system



$$x + y = z$$

Space: $x+y+1$

Time: $2x+2$

Suppose n – initial with of the working zone

Space: $O(n)$

Time: $O(n)$

Time and space complexity of a program

Time complexity $T(M)$ – the number of operations.

Space complexity $S(M)$ – the used memory size.

Measuring memory: in bits, in bytes, in machine words

Measuring time of operations: linear scale, logarithmic scale

Measuring input data size: in bits, using some parameters

Tasks:

- Choice of data size parameters (better one – n)
- Calculating (estimating) number of operations depending on n
- Decide whether it is required to use logarithmic scale

Linear complexity – find a minimal element of an array

```
int amin( int * a, int n )           (n+1)W
{
    int i, im=0;                      2W, 1I
    for( i=1; i<n; i++ )              3I
    {
        if (a[im]<a[i])                1I
            im=i;                      1I
    }
    return(im);                        1I
}
```

(n-1) times

Time complexity: $4(n-1)+2 \approx 4n = O(n)$

Space complexity: $n+3 \approx n = O(n)$

Square complexity – sort an array

<code>int asort(int * a, int n)</code>	<code>(n+1)W</code>	
<code>{</code>		
<code> int i, x;</code>	<code>2W</code>	
<code> for(i=0; i<n-1; i++)</code>	<code>3I</code>	
<code> {</code>		
<code> im=i+1+amin(a+i+1,n-i-1);</code>	<code>(7 + n)I, 2W</code>	} (n-2) times
<code> x=a[i];</code>	<code>1I</code>	
<code> a[i]=a[im];</code>	<code>1I</code>	
<code> a[im]=x;</code>	<code>1I</code>	
<code> }</code>		
<code> return(im);</code>	<code>1I</code>	
<code>}</code>		

Time complexity: $1 + ((10 + n - 1) + (10 + n - 2) + \dots + (10 + 1)) = 1 + 10(n - 1) + ((n - 1) + (n - 2) + \dots + 1) \approx 10n - 9 + (n^2 / 2) = O(n^2)$

Space complexity: $n + 5 + 2 \approx n = O(n)$

Exponential complexity – exhaustive search in triangle of numbers

Space complexity: $2n^2 + 2n \approx n^2 = O(n^2)$

Time complexity:

- input triangle, print triangle – n^2
- fill in a path, calculate sum – n
- number of paths: suppose there are $np(k)$ paths for a triangle if height k than $np(k+1) = 2np(k)$; $np(n) = 2^{n-1} = O(2^n)$

Time complexity: $O(2^n)$

n	5	10	20	30	50	100	1000
n^2	25	10^2	$4 \cdot 10^2$	$9 \cdot 10^2$	$2.5 \cdot 10^3$	10^4	10^6
2^n	32	10^3	10^6	10^9	10^{15}	10^{30}	10^{301}

Triangle of numbers

Greedy	Exhaustive search	Dynamic programming
n	2^n	n^2

Time complexity for $n=100$

10	10^{30}	10^2
-----------	-----------------------------	--------------------------

Logarithmic scale – calculate m^n

```
int mpown( int m, int n )
{
    int p=1;
    for( i=1; i<=n; i++ )
    {
        p*=m;
    }
    return(p) ;
}
```

It seems that

- Space complexity $C=O(1)$
- Time complexity $O(n)$

But the result is growing rapidly!

It can exceed int

Suppose m consists of $k = \lfloor \log_2 m \rfloor + 1 \approx \log_2 m$ bit
then m^2 requires $\log_2 m^2 = 2 \cdot \log_2 m = 2k$ bits
and m^n requires $n \cdot \log_2 m$ bits

Thus

Space complexity $O(n \cdot \log_2 m)$

Time complexity $O(n^2 \cdot \log_2 m)$

Usually it is supposed that
 $n \leq N$, $m \leq N$ or $N = \max(n, m)$

and

Space complexity $O(N \cdot \log_2 N)$

Time complexity $O(N^2 \cdot \log_2 N)$