D2: Complexity and Sorting

Edda Steinunn Rúnarsdóttir Birgitta Feldís Bjarkadóttir

September 8, 2017

1. *Give the simplest tilde approximations for the following quantities:*

1. $2(N+3)+10 \sim \mathbf{2N}$.
2. $2-1/N \sim \mathbf{2}$.
3. $(1-1/N)(1-2/N) \sim \mathbf{1}$.
4. $2N^3-15N^2-N \sim \mathbf{2N^3}$.
5. $lgN^2/lgN = lgN/lgN + lgN/lgN = 1+1 \sim \mathbf{2}$.
6. $2^{lgN} = N \sim \mathbf{N}$.

2. *Give the tilde time complexity of each of the following code fragments:*

(a)
```
int sum = 0;
for (i=0; i ¡ n; i++)
    for (j=0; j ¡ 5; j++)
        for (k=1; k ¡ n; k+=k)
            sum++;
```

Solution. *Outer loop runs* $\sim n$ *operations, first inner loop runs* ~ 5 *and second inner loop runs* $\sim lg(n)$

Therefore, the time complexity is $\sim \mathbf{5nlg(n)}$

(b)
```
int sum = 0;
for (i=0; i ¡ n*n; i++)
    for (j=0; j ¡ 2*n; j++)
        sum++;
```

Solution. *Outer loop:* n^2, *inner loop:* $2n$

Therefore, the time complexity is $n^2 * nN \sim \mathbf{2n^3}$

(c) *int sum = 0;*
for (int t = n; t ¿ 0; t /= 2)
for (int i = 0; i ¡ t; i++)
sum++;

Solution. *The outer loop runs* $\sim lg(n)$ *times, the inner loop runs* $\sim n + 1/2n + 1/4n + 1/8n + ... + 1/2lg(n)n$ *times which adds up to (almost)* $2n$

Therefore, the time complexity is $\sim \mathbf{2nlg(n)}$

(d) *int sum = 0;*
for (i=0; i ¡ n; i++)
for (j=0; j ¡ i; j+=2)
sum++;

Solution. *The outer loop loops* n *times, and the inner loop loops* $(n-1)/2$ *times*

Therefore, the time complexity is $n * (n-1)/2 \sim \mathbf{1/2n(n-1)}$

3. *Utilize the approximation* $2^{10} = 10^3$ *to approximate (without a calculator):*

(a) 2^{32} *as* $a * 10^b$

Solution. $2^{32} = (2^{10})^3 * 2^2 = (10^3)^3 * 2^2 = \mathbf{4 * 10^9}$.

(b) $lg(10^{18})$

Solution. $lg(10^{18}) = lg((10^3)^6) = lg((2^{10})^6) = \mathbf{lg(2^{60}) = 60}$.

4. *A given program runs in 2.1s on inputs of size 1000 but 16.8s on inputs of size 4000. Derive the time complexity of the program of the form* $T(n) = an^b$.

Solution. *Start with b: Slope b is defined as* $b = lg(T(N_1)/T(N_0))$*:*
Therefore we need to find the running time for $N = 2000$ *via linear equation using the two points. Running time* $T(2000)$ *is therefore calculated to be* 7 *s.*
$b = lg(7/2.1) = lg(3.33) = 2$
Then let's use $N = 1000$ *to estimate a:*
$T(1000) = a * 1000^2$
$a = 2.1/2000 = 1.05 * 10^{-3}$

Therefore, $\mathbf{T(N) = 1.05 * 10^{-3} * N^2}$

5. *Suppose we use insertion sort on a randomly ordered array where elements have only three values (small, medium, big). Is the running time linear, quadratic, or something else?*

Solution. *The running time is quadratic (that is, worst case and an average case time complexity of* $\sim \mathbf{1/4N^2}$*) because the elements are randomly ordered. Therefore, the insertion sort algorithm still has to loop through each in the array for comparison and then again to move it to the appropriate index.*

6. *A clerk at a shipping company is charged with the task of rearranging a number of large crates in order of the time they are to be shipped out. Thus, the cost of compares is very low (just look at the labels) relative to the cost of exchanges (move the crates). The warehouse is nearly full – there is extra space to hold any one of the crates, but not two. What sorting method should the clerk use and why (briefly)?*

Solution. *Since comparison costs are low relative to the high exchanges, an optimal sorting method should include as little of exchanges as possible. The clerk should choose selection sort because the exchanges in the worst case of that method are N (number of cases) which are the least possible number of exchanges to complete the task.*

7. *Suppose Mergesort is modified to divide the input into three parts, each with one third of the input. It then sorts each part recursively and combines using three-way merge. Explain why the order of growth of the overall running time is O(n log n).*

Solution. *Instead of dividing array into two, splitting N into N/2 to N/4 to N/8 and etc which is a process with $O(lg(N))$ time complexity (observe: log base is 2), array is divided from n to N/3 to N/9 etc, which is a process of $O(log(N))$ WITH LOG BASE 3. Just as in normal merge sort process, linear comparison is done to "sub arrays", which is an operation of time complexity $O(N)$, linear complexity. (although comparisons double, because a sub array of 2 only needs a single comparison while an array of 3 needs two comparisons - BUT this does not affect the order of growth - still linear).*

Therefore, the order of growth is indeed $\mathbf{O(Nlog(N))}$ ***with log base of 3.***

8. *Solve problem IV in final exam 2014. note that it is sufficient to identify methods 0-5, not 6 (Quicksort) or 7 (Heapsort).*

Solution.

Column 0: *0. Initial sequence*

Column 1: ***3. Insertion sort***

Column 2: ***4. MergeSort (top down)***

Column 3: **-**

Column 4: **-**

Column 5: ***2. Selection sort***

Column 6: ***5. MergeSort (bottom up)***

Column 7: *1. Sorted sequence*