Algorithms

Selection

Selection Problem.

Given an array A and a positive integer k, find the k^{th} smallest element in A.

The algorithm we will discuss is called **quickselect**.

For example: median
If n is even, it is the n/2 element
If n is odd, it is the (n-1)/2 element

Do you have a good solution for this problem?

Selection

Selection Problem.

Given an array A and a positive integer k, find the k^{th} smallest element in A.

The algorithm we will discuss is called **quickselect**.

Warm Up.

Suppose that we are given an algorithm for the selection problem that runs in $O(n)$ time where n is the size of A.

How would you use it to find the median of the array?

Find the $\lceil n/2 \rceil^{th}$ smallest number in the array.

Suppose that we are given an algorithm to find the median in $O(n)$ time.

How would you use it to solve the selection problem?

Does a linear time algorithm for the approximate median suffice?

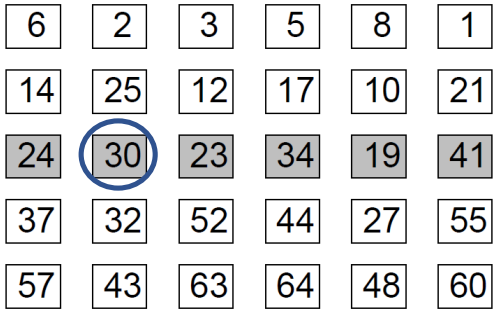
Selection

Step 1. Split the array into groups of 5 elements (ignore leftover elements).

Step 2. Find the median in each group. *The total time this takes is $O(n)$.*

Step 3. Recursively find the median of the group medians.

Example. $A =$ $14, 57, 24, 6, 37, 32, 2, 43, 30, 25, 23, 52, 12, 63, 3, 5, 44, 17, 34, 64, 10, 27, 48, 8, 19, 60, 21, 1, 55, 41, 29, 11, 58, 39.$

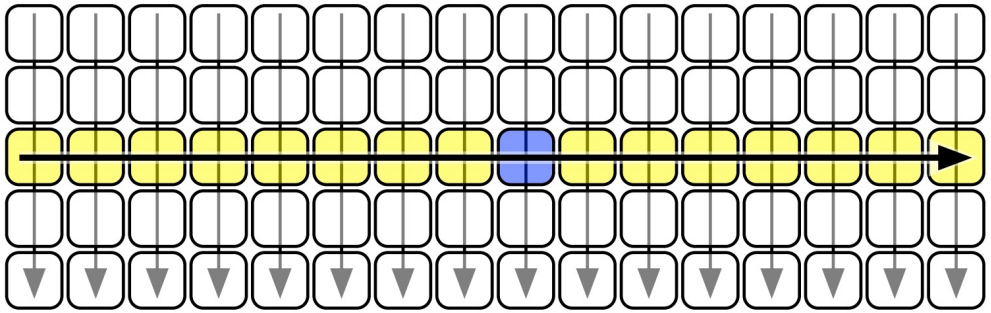


group medians

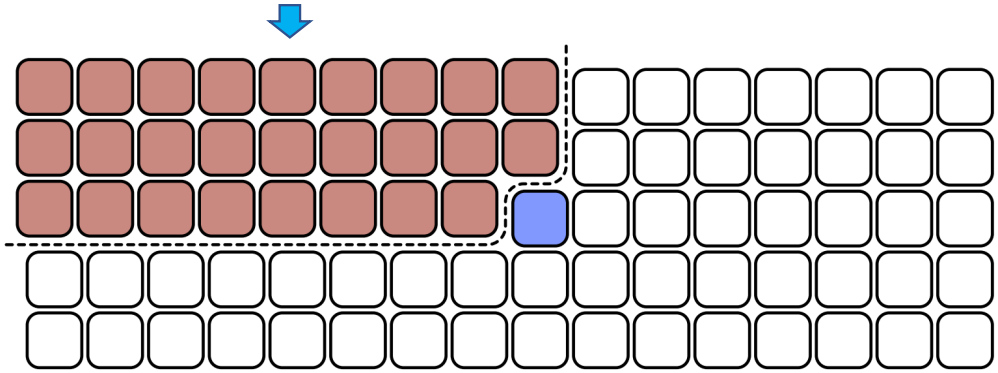
*recursively find the median of the medians (**mom**)*

*How many elements in the array are guaranteed to be $\leq$ **mom**?*

Selection



elements smaller than ***mom***



Imagine that:

- *Each group of* 5 *is sorted vertically*
- *Groups are sorted in increasing order of their medians*

Prove using the fact that : $(n-4)/5 \leq \left\lfloor \frac{n}{5} \right\rfloor \leq n/5$

 # (elements that are $\leq$***mom****)*

$\geq 3\lfloor n/5 \rfloor/2 \quad > n/4$ *for* $n > 24$

$\Longrightarrow$ *# (elements that are* $\geq$***mom****)*

$\leq 3n/4$ *for* $n > 24$

Similarly, # (elements that are $\leq$***mom****)*

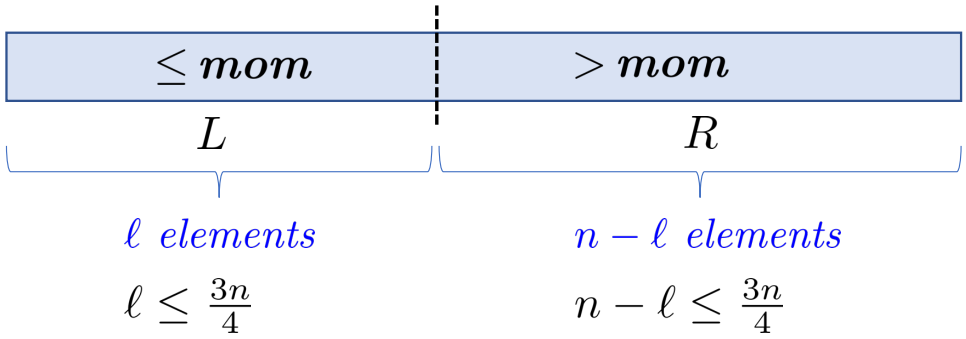
$\leq 3n/4$ *for* $n > 24$

Thus, ***mom*** is an approximate median.

Selection

Step 4. Split the array into two parts - L containing the elements that are $\leq \boldsymbol{mom}$ and R containing the remaining elements.

Let $\ell = |L|$.



How much time does it take to split the array? $O(n)$

Recall that we need to search for the k^{th} smallest element in the array.

Step 5. If $k \leq \ell$: return the k^{th} smallest element in L.

Else: return the $(k - \ell)^{th}$ smallest element in R.

Selection

Task. Find the k^{th} smallest element in an array A of size n.

Step 0. If A has ≤ 24 elements, sort A and return $A[k]$.

Step 1. Split the array into groups of 5 elements (ignore leftover elements).

Step 2. Find the median for each group.

Step 3. Recursively find the median of the group medians.

Step 4. Split the array into two parts - L containing the elements that are $\leq$ ***mom*** and R containing the remaining elements.
Let $\ell = |L|$.

Step 5. If $k \leq \ell$: return the k^{th} smallest element in L.
Else: return the $(k - \ell)^{th}$ smallest element in R.

How many recursive calls?

Selection

Represents k , or n/2

DSelect(array A, length n, order statistic i)

1. Break A into groups of 5, sort each group
2. C = the n/5 "middle elements"
3. p = DSelect(C,n/5,n/10) [recursively computes median of C]
4. Partition A around p
5. If j = i return p j is the position of mom p
6. If j < i return DSelect(1st part of A, j-1, i)
7. [else if j > i] return DSelect(2nd part of A, n-j, i-j)

What is the difference between these 2 recursive calls?

Selection

Task. Find the k^{th} smallest element in an array A of size n.

Step 0. If A has ≤ 24 elements, sort A and return $A[k]$. $O(1)$ *time.*

Step 1. Split the array into groups of 5 elements (ignore leftover elements).

Step 2. Find the median for each group. $O(n)$ *time.*

Step 3. Recursively find the median of the group medians. $T(\lfloor n/5 \rfloor)$ *time.*

Step 4. Split the array into two parts - L containing the elements that are $\leq$ ***mom*** and R containing the remaining elements. $O(n)$ *time.*
Let $\ell = |L|$.

Step 5. If $k \leq \ell$: return the k^{th} smallest element in L.
Else: return the $(k-\ell)^{th}$ smallest element in R. $\leq T(3n/4)$ *time.*

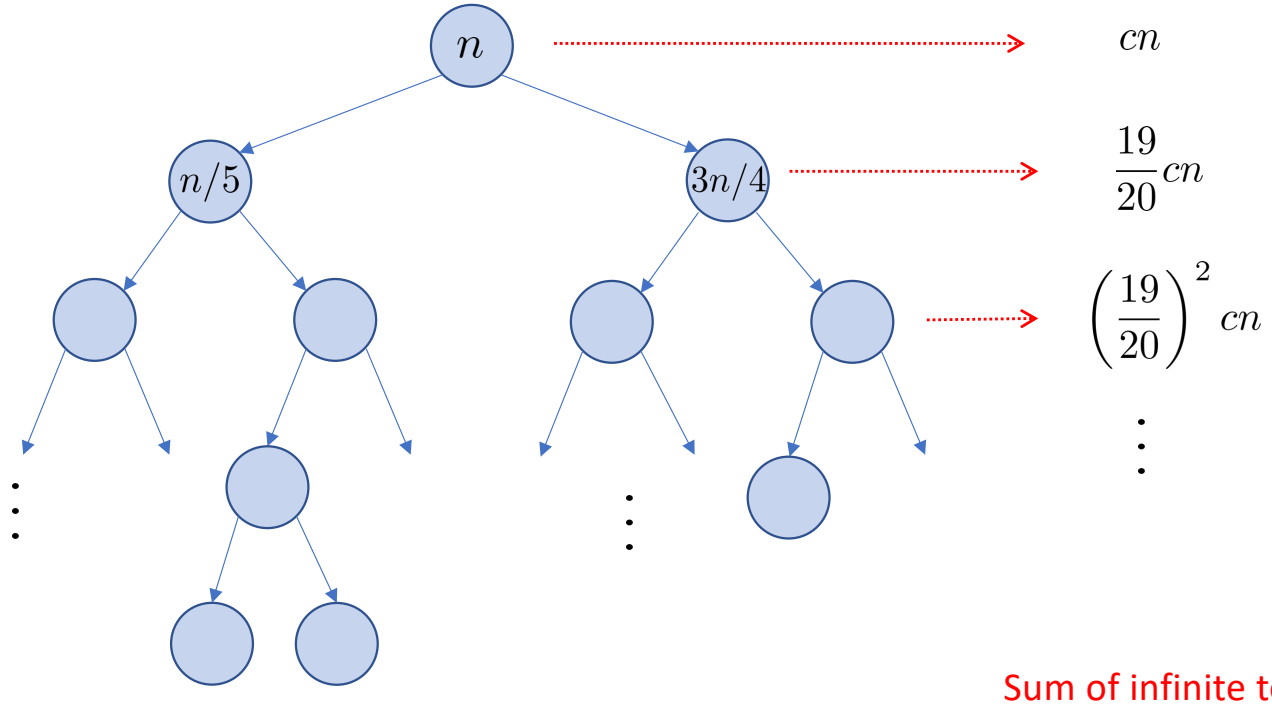
Recurrence relation on running time $T(n)$:

$$\left.\begin{array}{l} T(n) = O(1) \quad \text{for } n \leq 24 \\ T(n) \leq T(n/5) + T(3n/4) + cn \quad \text{for } n > 24 \end{array}\right\} \Longrightarrow T(n) = O(n)$$

Selection

$$T(n) = O(1) \quad \text{for } n \leq 24$$

$$T(n) \leq T(n/5) + T(3n/4) + cn \quad \text{for } n > 24$$



$$T(n) \leq cn + \alpha(cn) + \alpha^2(cn) + \cdots \quad \textit{where } \alpha = \tfrac{19}{20}$$

$$= cn \cdot (1 + \alpha + \alpha^2 + \cdots) \leq cn \cdot \left(\frac{1}{1-\alpha}\right) = 20 \cdot cn = O(n)$$

Sum of infinite terms

Exercise: integer Multiplication

For more infor watch:

https://www.youtube.com/watch?v=JCbZayFr9RE&list=PLEGCF-WLh2RLHqXx6-GZr_w7LgqKDXxN_&index=4

Integer Multiplication

Suppose that we want to multiply two n digit numbers.

Our computer can multiply two numbers with a small number of digits in constant time.

However, if n is large, the CPU cannot multiply them directly.

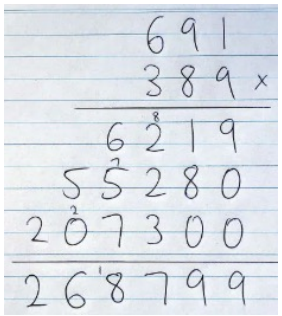
Let's assume that the CPU can multiply two 1 digit numbers in $O(1)$ time.

Also assume that adding two 1 digit numbers takes $O(1)$ time.

How much time does it take to add two k digit numbers?

How much time does it take to multiply two n digit numbers?

Integer Multiplication



This is how we do it by hand.

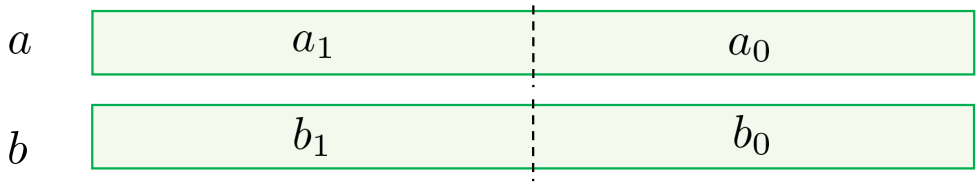
Takes $O(n^2)$ time for two n digit numbers.

Can we do it faster?

Idea. Let a and b be the two n digit numbers.

Assume that n is a power of 2 for convenience.

This will not affect the asymptotic running time.



$a = 10^{n/2}a_1 + a_0, \quad b = 10^{n/2}b_1 + b_0.$

a_0, a_1, b_0 and b_1 are $(n/2)$-digit numbers.

Integer Multiplication

$$a = 10^{n/2}a_1 + a_0, \quad b = 10^{n/2}b_1 + b_0.$$

a_0, a_1, b_0 and b_1 are $(n/2)$-digit numbers.

$$a \cdot b = 10^n(a_1 \cdot b_1) + 10^{n/2}(a_0 \cdot b_1 + a_1 \cdot b_0) + (a_0 \cdot b_0)$$

We need to compute four products of $(n/2)$ digit numbers.

Each of these products has at most n digits.

Multiplying a number by 10^k takes only $O(k)$ time.

There are three additions, which take $O(n)$ time.

Recurrence relation for the running time $T(n)$:

$$T(n) \leq 4\,T(n/2) + cn \qquad c\text{: some constant}$$

$$T(1) \leq c$$

Unfortunately this solves to $T(n) = O(n^2)$.

Integer Multiplication

$$a = 10^{n/2} a_1 + a_0, \quad b = 10^{n/2} b_1 + b_0.$$

a_0, a_1, b_0 and b_1 are $(n/2)$-digit numbers.

$$a \cdot b = 10^n (a_1 \cdot b_1) + 10^{n/2} (a_0 \cdot b_1 + a_1 \cdot b_0) + (a_0 \cdot b_0)$$

Idea for Improvement. Three multiplications suffice.

We compute $a_1 \cdot b_1$, $a_0 \cdot b_0$ and $c = (a_1 + a_0) \cdot (b_1 + b_0)$.

Then, $a_0 \cdot b_1 + a_1 \cdot b_0 = c - (a_0 \cdot b_0 + a_1 \cdot b_1)$

This requires some more additions and a subtraction but they still take $O(n)$ time overall.

Recurrence relation for the running time $T(n)$:

$$\left.\begin{aligned} &T(n) \leq 3\,T(n/2) + cn \qquad c\text{: some constant} \\ &T(1) \leq c \end{aligned}\right\} \implies T(n) = O(n^{\log_2 3})$$

$\log_2 3 \approx 1.58$

Integer Multiplication

Remark. A computer uses base 2 arithmetic instead of bases 10.

So, the calculations look more like this:

$a = 2^{n/2}a_1 + a_0, \qquad b = 2^{n/2}b_1 + b_0.$

a_0, a_1, b_0 and b_1 are $(n/2)$-digit numbers.

$$a \cdot b = 2^n(a_1 \cdot b_1) + 2^{n/2}(a_0 \cdot b_1 + a_1 \cdot b_0) + (a_0 \cdot b_0)$$

All 10's are replaced by 2's

The running time remains the same.

Practice

Lecture 2 (substitution method), chapter 2.4,2.5

Practice

What is the running time of the following functions in terms of n?

```
1 def f(n):
2   if n<=1: return n
3   return 2*f(n//2)
```

$O(\log n)$

```
1 def f(n):
2   if n<=1: return n
3   return f(n//2) + f(n//2)
                  // : integer division
```

$O(n)$

```
1 def f(n):
2   if n<=1: return n
3   return f(n-1) + f(n-1)
```

$O(2^n)$

Solve these as a homework

Read the substitution method in Reccurence relations documents on brightspace

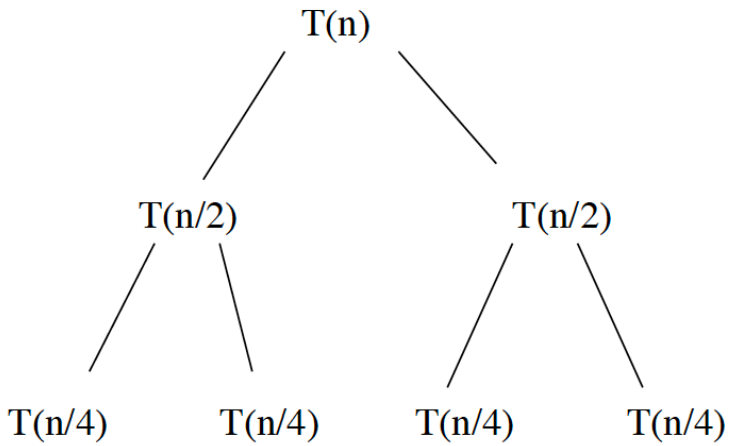
Practice

What is the running time of the following functions in terms of n?

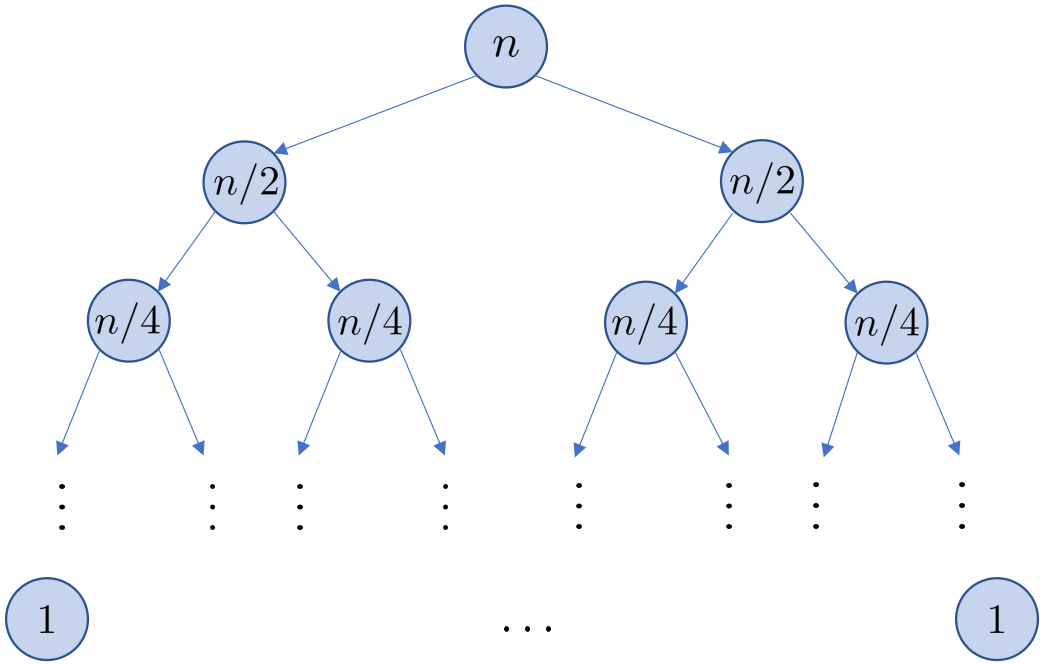
```
T(n)              1 def f(n):
C                 2     if n<=1: return n
T(n/2)+T(n/2)     3     return f(n//2) + f(n//2)
```

T(n)=T(n/2)+T(n/2)+c

Assume $n=2^k$



Recursion Tree



Practice

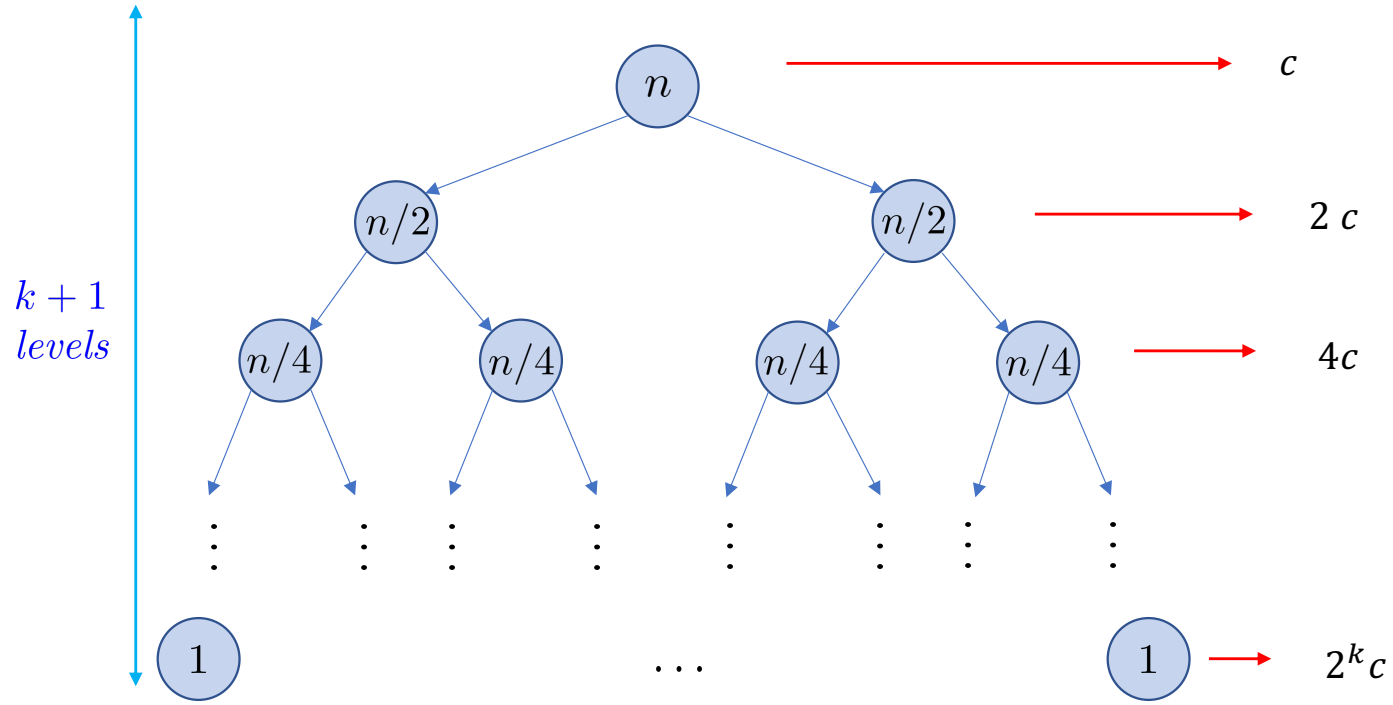
What is the running time of the following functions in terms of n?

```
T(n)              1  def f(n):
C                 2      if n<=1: return n
T(n/2)+T(n/2)     3      return f(n//2) + f(n//2)
```

T(n)=T(n/2)+T(n/2)+c

Assume $n=2^h$

Recursion Tree



$c + 2c + \cdots + 2^{\log(n)}c = [(1 - 2^{\log n})/-1]\, c = (n-1)c = O(n)$

Exercise 2.4.

- Take the following list of functions and arrange them in ascending order of growth rate. That is, if function g(n) immediately follows function f(n) in your list, then it should be the case that f(n) is O(g(n)).

$g_1(n) = 2^{\sqrt{\log n}}$

$g_2(n) = 2^n$

$g_4(n) = n^{4/3}$

$g_3(n) = n(\log n)^3$

$g_5(n) = n^{\log n}$

$g_6(n) = 2^{2^n}$

$g_7(n) = 2^{n^2}$

Solution

g1 < g3 < g4 < g5 < g2 < g7 < g6 , since : $2^{\sqrt{\log n}} < n(\log n)^3 < n^{\frac{4}{3}} < n^{\log n} < 2^n < 2^{n^2} < 2^{2^n}$

g1 grows slower than g3, you can easily see this comparing log(g1) and log(g2): $(log\ 2)^{\sqrt{\log n}} = \sqrt{\log n} \log 2 = \frac{1}{2} log2 \log n$ and $\log(n(log\ n)^3) = log\ n + \log(\log n)^3 = log\ n + 3\log log\ n$

g3 grows slower than g4. Divide both by "n" and we will have $(log\ n)^3$ against $(\ n)^{1/3}$. Now take a look at the item 2.8 of the book, it explains why $(log\ n)^3 < (\ n)^{1/3}$

g4 grows slower than g5, since $4/3 < log\ n$ when "n" tends to infinity

g5 grows slower than g2, since $n^{\log n} < 2^n$. The g2 exponent is way bigger than g5 exponent (in fact, $n \log n < n^2 < 2^n$ check item 2.9)

g2 grows slower than g7, since $2^n < 2^{n^2}$ as $n < n^2$

g7 grows slower than g6, since $2^{n^2} < 2^{2^n}$ (as $n^2 < 2^n$)

Exercise 2.5.

Assume you have functions f and g such that f(n) is O(g(n)). For each of the following statements, decide whether you think it is true or false and give a proof or counterexample.

- (a) log f(n) is O(log g(n)).
- (b) $2^{f(n)}$ is $O(2^{g(n)})$.
- (c) $f(n)^2$ is $O(g(n)^2)$.

Solution

We have for some n_0 and c, $f(n) \leq cg(n)$, for all $n \geq n_0$ (1)

(a) False, assume constant functions: $f(n) = 2$ and $g(n) = 1$.

(b) False (assignment problem)

(c) True, from (1) we have: $f(n)^2 \leq c^2 g(n)^2$, for all $n \geq n_0$. Our constant is c^2