G52ADS 2014-15: Coursework TWO (7%). Tue. 4th Nov, 2014.

VERSION: EVEN
Use this version only if the last digit of your student ID number is EVEN, i.e. 0, 2, 4, 6 or 8

Please fill in the appropriate letter on your mark-sheet.

Questions are NOT negative marked - that is, incorrect answers are not penalised, they just receive zero marks.

QUESTION NUMBER	ANSWER (from A,…,E)	MARK (leave empty)	OVERALL MARK
1	C	1	14
2	E	1	
3	E	1	
4	C	1	
5	C	1	
6	E	1	
7	A	2	
8	C	2	
9	E	2	
10	A	2	

Note: when writing powers will often use "^" as it is easier to type, and also somewhat easier to read. That is, n^3 means n^3 etc. (This is the way it is done in LaTeX, which is the standard typesetting system for writing maths equations.)

Question 1. [1 mark]

The runtime of a program is $\Theta(n^3)$. On input of size n=20 it takes 10 seconds. What is the best estimate of the time it takes at input size n=40?

A. 20 seconds

B. 40 seconds

C. 80 seconds

D. 160 seconds

E. 180 seconds

The quick argument you should use is that the input size doubled, and so the runtime should increase by a factor of $2^3 = 8$, hence giving 10*8 = 80.

The longer version is:
The best estimate is to assume that the runtime is $f(n) = k\, n^3$ for some constant k.

Then we have
f(20) = k 20^3 = 10

f(40) = k 40^3
= k 2^3 20^3
= 2^3 f(20) // using the n=20 case
= 8 * 10 = 80

Note that this is only a "best estimate" and is not a guarantee. The reason for this caveat is that the actual runtime might be something that includes lower powers of n. These would not change the $\Theta(n^3)$ behaviour but could change the result. The estimate is the "best" in the sense it is the best one can do without being given more information.

Question 2. [1 mark]

The runtime of a program is $\Theta(3^n)$. On input of size n=20 it takes 10 seconds. What is the best estimate of the time it takes at input size n=40?

A. 11 seconds

B. 20 seconds

C. 40 seconds

D. 80 seconds

E. 34867844010 seconds

Quick argument: 3^40 = 3^20 * 3^20 , hence desired runtime is 3^20 *10. 3^20 is certainly larger than 8, and hence E is the only possible answer (no calculator required as E is the only viable answer).

Longer version is as for Q2.

The best estimate is to assume that the runtime is $f(n) = k\,3^n$ for some constant k.

Then we have
f(20) = k 3^20 = 10

f(40) = k 3^40
= k 3^20 2^20
= 3^20 f(20) // using the n=20 case
= 3^20 * 10

NOTE: The reason for including these questions is to give an example of the difference.

n^3 is a POWER LAW and doubling the size is a big difference. It is NOT called an exponential growth.

3^n is an EXPONENTIAL growth, as this terms is used for the case when the 'n' is in the exponent. Doubling the size will generally make an enormous difference.

Question 3. [1 mark]

The runtime of a program is $\Theta(3^n)$. On input of size n=20 it takes 10 seconds. What is the best estimate of the time it takes at input size n=22?

A. 11 seconds

B. 20 seconds

C. 40 seconds

D. 80 seconds

E. 90 seconds

Similar to Q2. But 3^22 = 3^20 3^2 = 9 3^20. So it gets 9 times slower. Hence 9*10.

NOTE: Again compare this with Q1. With roughly the same increase in time of 10 secs to 80-90 seconds, then a power law allows us to increase from n=20 to n=40, but the exponential only allows to increase to n=22. If an algorithm has exponential scaling, then the run time increases much faster than a power law.

For anyone interested (not needed for this module!), this has practical impact on many problems in scheduling, timetabling and optimisation in general, where the even the very best algorithms are often exponential. E.g. see http://en.wikipedia.org/wiki/Travelling_salesman_problem and it is also the reason that scheduling lectures and examinations is hard.

Question 4. [1 mark]

The function $f(n) = (n^2 (\log n) + n (\log n)^4)$ is which one of the following?

A. $O(n^3 \log n)$ but not $O(n^3)$

B. $O(n^3)$ but not $O(n^2 \log n)$

C. $O(n^2 \log n)$ but not $O(n \log n)$

D. $O(n^2)$ but not $O(n)$

E. $O(n \log n)$ but not $O(n)$

Logs are dominated by powers, so the leading term is $n^2 (\log n)$. This is trivially big-Oh of itself, but is definitely not O(n log n).

Question 5. [1 mark]

The function $f(n) = n^{200} + 2^n + n^{201} / (\log n)$

has which one of the following big-Oh behaviours? (If multiple expressions are correct then pick the one that gives the slowest growth – i.e. the best description of the growth).

A. $O(n^{201})$

B. $O(n^{200})$

C. $O(2^n)$

Exponentials dominate any power – even with a large power. Hence, for large values of n, then the 2^n term dominates.

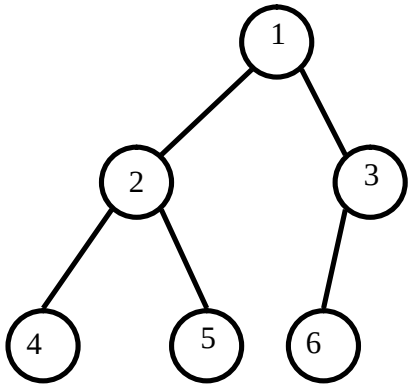
Question 6. [1 mark]

Consider a **doubly**-linked list with references to both ends (i.e. to the `head' and the `tail'). Which of the following operations **cannot** be done in O(1) time?

A. Insert at the head
B. Delete at the head
C. Insert at the tail
D. Delete at the tail
E. None of the above (i.e. all of operations A-D can be done in O(1) time)

In a doubly-linked list then everything is efficient at both ends.
If you answered D, then you were probably thinking about singly-linked lists, and needed to read the question more carefully. I was being kind in putting the "doubly" in boldface ☺.

Question 7 [2 marks]



Consider the rooted binary tree above.
Which of the following is the result of performing a **post**-order traversal?

A. 4, 5, 2, 6, 3, 1

B. 1, 2, 3, 4, 5, 6

C. 1, 2, 4, 5, 3, 6

D. 4, 5, 6, 2, 3, 1

E. 4, 2, 5, 1, 6, 3

Direct from the lectures.

Question 8 [2 marks]

The following code is an implementation of insertion sort of an array `A' of integers which is intended to put them into non-decreasing order:

```
void sort( int A[] ) {
     int i, j, temp;
     for ( j=1 ; j < A.length ; j++ ) {
       temp = arr[ j ];
       i = j;
       while( i > 0 && A[i-1] > temp ) {
           A[i] = A[i-1];
           i--;
       }
       A[i] = temp;

       // Assertion X

   }
}
```

Which of the following would be correct and the most appropriate for assertion X?

A. i > 0

B. The entries A[0] … A[j-1] are sorted. That is, A[s] <= A[t] for all 0 <= s < t < j

C. The entries A[0] … A[j] are sorted. That is, A[s] <= A[t] for all 0 <= s < t <= j

D. The entries A[0] … A[j] are in their final position.

That is, for each s in 0 … j then A[s] <= A[h] for all s < h < n

E. j > 0

The assertion is AFTER the case j has been handled and so it will cover the range 0..j. The way that insertion sort works is that the beginning of the list is sorted. However, unlike bubble or selection sort, it does not guarantee that they will not have to move again – so D is incorrect – as the very last element in the array might be the minimum and cause everything to move. Answers A and E are logically correct, but do not capture anything about how the algorithm works, and so are not “the most appropriate”.

Question 9. [2 marks]

The following code returns the **maximum** value in an integer array of size n >= 1.

```
int getMax( int A[] )
{
   int n = A.length;
   int m = A[0];
   for ( int i = 0 ; i < n ; i++ )
   {
          // assertion X
          if ( A[i] > m ) {
              m = A[i];
          }
    }
   return m;
}
```

Which of the following is the most appropriate and useful choice for assertion X

A. forall k such that 0 <= k < n. m >= A[k]
B. forall k such that 0 <= k <= i. m >= A[k]
C. m > A[i]
D. forall k such that 0 < k < i-1. m > A[k]
E. forall k such that 0 <= k < i. m >= A[k]

Similar to Q.8. Answer E just captures the most about the intention of how the algorithm works, which is that at position of the assertion then the entries 0..(j-1) will have been processed, and none of them will be larger than m.

Question 10. [2 marks]

The function $f(n) = (n^3 + 3n^2)$ is which one of the following?

A. $\Omega(n^3)$ but not $\Omega(n^4)$

B. $\Omega(n^2)$ but not $\Omega(n^3)$

C. $\Omega(n)$ but not $\Omega(n^2)$

D. $\Omega(1)$ but not $\Omega(n)$

The potential mistake in this question is to think that Omega captures the best case, and then mistakenly focus on the smallest term. However, it still captures the 'large n' behaviour, which is still the n^3 term. Hence, it is $\Omega(n^3)$.

END