COMP 3711 Design and Analysis of Algorithms
Spring 2016 Midterm Exam

1. **Time Complexity** (15 pts)

   We have two algorithms, A and B. Let $T_A(n)$ and $T_B(n)$ denote the time complexities of algorithm A and B respectively, with respect to the input size n. Below are 5 different cases of time complexities for each algorithm. Complete the last column of the following table with "A", "B", or "U", where:

   - "A" means that algorithm A is faster;
   - "B" means that algorithm B is faster;
   - "U" means that we do not know which algorithm is faster.

Case	$T_A(n)$	$T_B(n)$	Faster
1	$\Theta(n^{2.1})$	$\Theta(n^2 \log^3 n)$	
2	$\Theta(n^9)$	$\Theta(2^{\sqrt{n}})$	
3	$O(n^2)$	$O(n \log n)$	
4	$\Omega(n^3)$	$O(n^2)$	
5	$O(n^2)$	$\Theta(n^{2.31})$	

2. **Recurrences** (12 pts)

   Solve the following recurrences. A correct answer gives you full credits; otherwise, showing the steps may gain you partial credits. Please give the answer using the $\Theta()$ notation. You may assume that n is a power of a for any constant $a > 1$ for your convenience.

   (a) $T(1) = 1$; $T(n) = 2T(n/2) + n$ for all $n \geq 2$.
   (b) $T(1) = 1$; $T(n) = 4T(n/2) + n^2$ for all $n \geq 2$.
   (c) $T(n) = 1$ for $n \leq 3$; $T(n) = T(n/4) + T(3n/4) + 1$ for all $n \geq 4$.
   (d) $T(1) = 1$; $T(n) = T(n/2) + \log n$ for all $n \geq 2$.

3. **Divide-and-conquer** (15 pts)

   We say that an array $A[1...n]$ is k-sorted if the array can be divided into k blocks with n/k elements in each block, such that the elements in each block are larger than the elements in earlier blocks, and smaller than elements in later blocks. The elements in each block need not be sorted. For example, the array $[\underline{5, 2, 3}, \underline{7, 6, 8}, \underline{10, 11, 12}, \underline{20, 14, 13}]$ is 4-sorted. Design and analyze an $O(n \log k)$ algorithm that k-sorts any given array. You may assume that both n are k are powers of 2, and all elements in A are distinct. [Hint: You may use the deterministic linear-time selection algorithm as a black box.]

4. **Max-heap** (14 pts)

   Given an array $A[1..10]$ with initial content in the table below, show the content of the array after each max-heap operation. Note that each operation is applied on the array

resulting from the previous operation. Recall that Increase-Key(i, k) increases the key stored in $A[i]$ to k, and Decrease-Key(i, k) decreases the key stored in $A[i]$ to k.

Array index	1	2	3	4	5	6	7	8	9	10
Initial content	4	5	1	12	9	10	16	7	3	8
Build-Max-Heap										
Increase-Key(10, 11)										
Decrease-Key(2, 2)										
Extract-Max										
Insert(13)										

5. **Decision tree** (15 pts)

   Describe an algorithm to find the median of an array $A[1..3]$ of distinct numbers using at most 3 comparisons. Instead of writing pseudocode, describe your algorithm using a decision tree. Recall that a decision tree is a binary tree where each internal node contains a comparison of the form "$A[i] < A[j]$?", and each leaf contains an index into the array that points to the median.

6. **Randomized algorithm** (15 pts)

   Suppose you only have access to a function FairCoin that returns a single random bit, chosen from $\{0, 1\}$ with equal probability. Design an algorithm that generates a random number uniformly from $\{0, 1, 2\}$, i.e., your algorithm should return each of $0, 1$, and 2 with probability $1/3$. [Hint: Generate two random bits first, then decide what to do next.] How many calls to FairCoin does your algorithm make in expectation? Please compute the exact number, not just an asymptotic result. [Hint: Is this a hint?]

7. **Huffman coding** (14 pts)

   Find an optimal Huffman code for the following alphabet with the given frequencies. Note that the optimal Huffman code is not unique; any one of them is acceptable.

a	b	c	d	e	f	g
5	1	2	8	4	3	9