Note: This exam has 5 questions for a total of 100 points. Answer all questions. Write all your answers on the blue books.

**Question I** (25 points)

**Note:** The Master Theorem for recurrences is given on page 3 of this examination.

(a) Use the Master Theorem to find the asymptotic solution to the following recurrence:

\[ T(n) = T(\lceil 3n/4 \rceil) + 7n \]

Show work. (15 points)

(b) Find the largest integer \( a \) such that the asymptotic solution to the recurrence

\[ T(n) = aT(\lfloor n/3 \rfloor) + 2n^2 \]

is \( T(n) = O(n^2) \). Be sure to indicate how you arrived at your answer. (10 points)

**Question II** (20 points total)

(a) Let \( S = \{1, 2, \ldots, n\} \). We construct a subset \( S' \) of \( S \) using the following random experiment. \( S' \) is initially empty. For each element \( i \) of \( S \), \( 1 \leq i \leq n \), we toss a fair coin and add element \( i \) to \( S' \) only if the result of the toss is HEAD. Find the expected size of \( S' \). (8 points)

(b) Let \( A[1..n] \) be an array. Suppose we initialize each element of \( A \) with an integer value drawn uniformly randomly from the range \([1..n^4]\). Assume also that the random choices for the \( n \) elements are independent. Prove that the probability that the values in \( A \) are all distinct is at least \( 1 - \frac{1}{2n^2} \). (12 points)

**Question III** (20 points total)

For both parts of this question, assume that array \( A \) is a MAX-HEAP with \( n \geq 3 \) elements. In your answers to this question, you may use, if necessary, any of the standard MAX-HEAP operations, namely MAX-HEAPIFY, HEAP-MAXIMUM, MAX-HEAP-INSERT, HEAP-EXTRACT-MAX and HEAP-INCREASE-KEY. (For your reference, pseudocode descriptions of these operations appear on page 4 of this examination. In your answers, there is no need to show the pseudocode for these standard operations.)

(a) We want to implement a new operation called SECOND-MAX on \( A \). This operation simply returns the second largest value stored in the heap without modifying the heap. We want the operation to run in \( O(1) \) time in the worst-case. Show the pseudocode for implementing the operation and indicate why the running time is \( O(1) \). (8 points)

(over)
We want to implement a new operation called EXTRACT-SECOND-MAX on \( A \). This operation removes and returns the second largest value stored in the heap. We want this operation to run in \( O(\log n) \) time in the worst-case. Show the pseudocode for implementing the operation and indicate why the running time is \( O(\log n) \). (12 points)

**Question IV** (20 points total)

**Note:** Outlines of \textbf{Partition} and \textbf{Quicksort} are given on page 3 of this examination.

(a) Suppose we have an array \( A[1..n] \) which has been initialized so that \( A[i] = x_i, 1 \leq i \leq n \), where \( x_1 > x_2 > x_3 > \ldots > x_n \). What is the value returned by the call \texttt{Partition}(\( A, 1, n \))? Also, show the values in \( A \) after this call to \texttt{Partition}. (8 points)

(b) Given an array \( S \) containing \( n \) numbers, define a middle element of \( S \) to be a value \( x \in S \) such that at most \( \lceil n/2 \rceil \) values in \( S \) are less than or equal to \( x \) and at most \( \lceil n/2 \rceil \) are greater than or equal to \( x \). Suppose you are given an algorithm \( A \) which for any array \( S \) with \( t \) elements returns the index \( j \) of a middle element of \( S \) in \( O(t) \) time. (You can also assume that \( A \) does not modify the array \( S \).) Show how \textbf{Quicksort} can be modified using \( A \) so that the worst-case running time of the resulting version of \textbf{Quicksort} is \( O(n \log n) \).

Your answer must include the pseudocode for the modified version, a recurrence for the running time \( T(n) \) of the resulting algorithm and an explanation of why the solution to the recurrence is \( O(n \log n) \). (12 points)

**Hint:** Use Algorithm \( A \) to modify the \texttt{Partition} function.

**Question V** (15 points)

When we discussed the \( \Omega(n \log n) \) lower bound for comparison-based sorting algorithms, we assumed that each internal node of the tree has two outcomes labeled ‘\( \leq \)’ and ‘\( > \)’. Suppose we modify the decision tree for a sorting algorithm so that each internal node has three outcomes, namely ‘\( < \)’, ‘\( > \)’ and ‘\( = \)’. Prove that even under this modified form of decision trees, the \( \Omega(n \log n) \) lower bound holds for comparison-based sorting algorithms.
Statement of Master Theorem:

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function and let $T(n)$ be defined on nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n)$$

where $n/b$ may be either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

**Part 1:** If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

**Part 2:** If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a \log n})$.

**Part 3:** If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and if $a f(n/b) \leq c f(n)$ for some constant $c < 1$ and for sufficiently large $n$, then $T(n) = \Theta(f(n))$.

---

**Pseudocode for Quicksort and Partition**

Quicksort (A, p, r) // Sort A[p .. r] into ascending order.

1. if (p < r)
   1.1 q = Partition(A, p, r)
   1.2 Quicksort(A, p, q-1)
   1.3 Quicksort(A, q+1, r)

Partition (A, p, r) // It is assumed that p <= r.

1. x = A[r]; i = p-1; // x is the pivot value.
2. for j = p to r-1 do
   if (A[j] <= x) {
     i = i + 1
   }
4. return i+1.
Pseudocode for Operations on a Max-Heap:

(a) Pseudocode for Heapify:

Heapify (A, i)
1. l = Left(i);  r = Right(i)
2. if ( (1 <= heap_size(A) and (A[l] > A[i]) )
     then m = l else m = i
3. if ((r <= heap_size(A) and (A[r] > A[m])
     then m = r
4. if (m != i) 
    then
       4.1 Exchange A[i] with A[m].
       4.2 Heapify(A,m)

(b) Pseudocode for Heap-Extract-Max:

Heap-Extract-Max(A)
1. if (heap_size[A] = 0) then print "Error: Heap underflow" and stop.
2. max = A[1]
4. heap_size[A] = heap_size[A]-1
5. Heapify(A,1)

(c) Pseudocode for Heap-Increase-Key:

Heap-Increase-Key(A, x, k) // The value of A[x] must be changed to k.
1. if (k < A[x])
    then print "Error: New key value smaller than current" and stop.
2. A[x] = k
3. while ( (i > 1) and (A[Parent(i)] < A[i]) ) do
   3.1 Exchange A[i] and A[Parent(i)]
   3.2 i = Parent(i)

(d) Pseudocode for Heap-Insert:

Heap-Insert(A, k) // k : Key to be inserted.
1. heap_size[A] = heap_size[A] + 1
2. A[heap_size[A]] = -infinity
3. Heap-Increase-Key(A, heap-size[A], k)