Multiple-choice Problems

Problem 1. $\sum_{i=1}^{n} i^2 - i$ is

A)$O(n)$ B) $\Theta(n^3)$ C) $\Theta(n^4)$ D) $\Omega(n^2 - n)$ E) $\Theta(n^2)$

Problem 2. Let T(n)=4T($\lfloor \frac{n}{2} \rfloor$)+2n^2+n and T(0)=1. Then T(n) is

a) Θ(n^2 log n) b) Θ(n^2) c) Θ(n log n)
d) Θ(n log^2 n) e) None

Problem 3. Let T(n)=2T($\lfloor \sqrt{n} \rfloor$)+log(n) and T(0)=1. Then T(n) is

a) Θ(log n) b) Θ(log(n)× log(log(n))) c) Θ(log(log(n)))
d) Θ(log^2 n) e) Θ(n)

Problem 4. How many times do we call the procedure Proc() below?

```
for i:=1 to n do
    for j:=1 to n do
        for k:= 1 to n do
            if i<j<k then
                Proc(i, j, k);
```

a) O(n) b) 1^3+2^3+…+n^3 c) O(n^2) d) O(n^3) e) None

Problem 5. How many comparisons do we need in the worst case if we search for a number x in an arbitrary sequence of n numbers?

a) O(log n) b) O(log^2 n) c) n-1 d) n c) O(n log n)

Problem 6. How many comparisons would the Merge function used in merge sort in order to merge two sorted arrays 1, 2, 7, 11 (in order) and 3, 4, 5?

a) 2 b) 3 c) 4 d) 5 e) 6

Regular Problems:

Problem 7. Consider the following functions:

(func 0) 2^5
(func 1) $4\log(n^3)$
(func 2) 2^n
(func 3) $n + 5$
(func 4) $n!$
(func 5) $\log^3 n + 8$
(func 6) n . log n

Complete the table below. In the entry at row "$func\ i$" and column "$func\ j$", cross the best relation you can prove between "$func\ i$" and "$func\ j$", i.e. put ONLY ONE of O or Ω or Θ. (You do not need to write the proofs).

Functions	$func\ 0$	$func\ 1$	$func\ 2$	$func\ 3$	$func\ 4$	$func\ 5$	$func\ 6$
$func\ 0$	Θ	O	O	O	O	O	O
$func\ 1$	Ω	Θ	O	O	O	O	O
$func\ 2$	Ω	Ω	Θ	Ω	O	Ω	Ω
$func\ 3$	Ω	Ω	O	Θ	O	Ω	O
$func\ 4$	Ω	Ω	Ω	Ω	Θ	Ω	Ω
$func\ 5$	Ω	Ω	O	O	O	Θ	O
$func\ 6$	Ω	Ω	O	Ω	O	Ω	Θ

Note that all entries along diagonal are Θ. Also if entry (i, j) is O then entry (j, i) is Ω and vice versa. If entry (i, j) is Θ then so is entry (j, i)

Problem 8. Prove by induction that $\sum_{i=1}^{n} i \times i! = (n+1)! - 1$ for every $n \geq 1$.

Base Case: For n=1 we have LHS=1=RHS

Induction Hypothesis: For all $k < n$ we have $\sum_{i=1}^{k} i \times i! = (k+1)! - 1$

Inductive Step: For $k = n$ we have

$$\sum_{i=1}^{n} i \times i! = \left(\sum_{i=1}^{n-1} i \times i!\right) + n \times n! \qquad \text{By simplification}$$

$$= (n! - 1) + n \times n! \qquad \text{By Induction Hypothesis}$$

$$= (n+1)! - 1$$

Problem 9. The input to this problem is an array A of size n of positive real numbers a positive number M. Design an algorithm with running time $O(n)$ to find two indices i and j such that i<j and $\sum_{k=i}^{j} A[k] \leq M$. Your algorithm should find two indices that maximize j-i.

For any two indices i, j let $S(i,j) = \sum_{k=i}^{j} A[k]$.
For each $1 \leq i \leq n$ let r_i denote the maximum index such that $S(i, r_i) \leq M$, if such an index r_i exists. Note that r_i might not exists, for example if $A[i]$ is itself greater than M. In that case we set $r_i = 0$. Let $X = \{1 \leq i \leq n \mid r_i \neq 0\}$. In linear time find all the indices i such that $r_i = 0$, i.e., $A[i] > M$

Claim: For any two indices $i \leq i'$ such that $i, i' \in X$ we have $r_i \leq r_{i'}$.
Proof: Since $i, i' \in X$ we have $r_i \neq 0$ and $r_{i'} \neq 0$. Suppose $r_i > r_{i'}$. Since all numbers are positive we have $M \geq S(i, r_i) \geq S(i', r_i)$ as $i' \geq i$. This contradicts the maximality of $r_{i'}$.

By above claim we can compute the r_i value for each $1 \leq i \leq n$ in O(n) time by just a linear scan of the input as follows.

```
var = 1
for i=1 to n
    if i ∉ X
      while S(i, var) ≤ M
        var++
      r_i = var-1
    else
      r_i = 0
```

The required index i is the one that achieves $\max_{i \in X} (r_i - i)$ and the j is the corresponding r_i

Problem 10. The input to this problem is an array A of size n. Design an algorithm to find the minimum and the second minimum of array A. Your algorithm should use at most n+logn comparison.

See Section 6.11.2 of the book by Manber. There an algorithm is given for finding the maximum and second maximum numbers of an array A. By just maintaining min instead of max in each step the same algorithm works for this problem as well.

Problem 11. Suppose you are choosing between the following three algorithms:

- **a.** Algorithm A solves problems of size n by dividing them into four sub-problems of half the size, recursively solving each sub-problem, and then combining the solutions with cn^2 operations.
- **b.** Algorithm B solves problems of size n by recursively solving two sub-problems of size $n-1$ and then combining the solutions in constant time (c operations).
- **c.** Algorithm C solves problems of size n by dividing them into nine sub-problems of size $n/3$, recursively solving each sub-problem, and then combining the solutions with cn operations.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

For (a), we have $T(n) = 4T\left(\frac{n}{2}\right) + cn^2$. Note that $a = 4, b = 2$ and $k = 2$. Since $a = b^k$ we have $T(n) = O(n^2 \cdot \log n)$

For (b), we have $T(n) = 2T(n-1) + c$. This implies $T(n) = O(2^n)$

For (c), we have $T(n) = 9T\left(\frac{n}{3}\right) + cn$. Note that $a = 9,\ b = 3$ and $k = 1$. Since $a > b^k$ we have $T(n) = O(n^2)$

Clearly we would prefer Algorithm C

Problem 12. Explain the heapsort in details, write pseudo-code for it, and analyze its running time.

See Chapter 6 of the book by Cormen et. al.