Midterm Exam

Name: _____

Perfect score: 100 points - Available points: 112.
No notes or books are allowed.

A. Specify whether the following statements are true or false and **argue why**:

- For any pair of numbers a and N, you can find a multiplicative inverse x of a modulo N using the extended Euclid algorithm:

False. It is not true for *any pair* of a and N. It has to be that a and N are co-prime.

- If $a \neq b$, then $log_b N \neq \Theta(log_a N)$ [provide mathematical explanation]:

False.

$$\log_b N = \frac{\log_a N}{\log_a b} = \frac{1}{\log_a b} \cdot \log_a N$$

The term $\frac{1}{\log_a b}$ is a constant and therefore: $\log_b N = \Theta(\log_a N)$

- If algorithm A's running time is n^2 and algorithm B's running time is 4^{lgn}, then algorithm B's running time grows asymptotically faster:

False.

$$4^{\lg n} = (2^2)^{\lg n} = n^2,$$

therefore the two running times grow similarly in an asymptotic sense.

- If d divides both a and b, and $d = a \cdot x + b \cdot y$ for integers x and y, then $d = gcd(a, b)$.

True. By the first two conditions, d is a common divisor of a and b and so it cannot exceed the gretest common divisor; that is, $d \leq gcd(a, b)$. On the other hand, since $gcd(a, b)$ is a common divisor of a and b, it mut also divide $ax + by = d$, which implies $gcd(a, b) \geq d$. Putting these together: $d = gcd(a, b)$.

- The security of the RSA protocol is based on the fact that the "factoring" problem can be solved in polynomial time, while the "primality" problem is believed to be a hard, computationally intractable problem.

False. It is the opposite, i.e., "factoring" is believed to be a potentially intractable problem.

(20 points: 1 point for correct answer and 3 for correct reasoning per question)

B. Provide a recursive formula for the multiplication of two n-bit numbers that results in $O(n^2)$ running time. Argue why this is the correct running time. (6 points)

The formulation is

$$x \cdot y = \begin{cases} 2(x \cdot \lfloor y \rfloor), & \text{if } y \text{ even} \\ x + 2(x \cdot \lfloor y \rfloor), & \text{if } y \text{ odd} \end{cases}$$

This recursive formulation will be exeucuted n times. For each one of the n bits of number y, we check if it is 0 or 1 (i.e., whether it is an even or odd number), we perform left shift of the intermediate result and add x if y is odd.

Left shifts takes $O(1)$ time but addition takes linear time to the size of the involved numbers. The biggest possible number for $x \cdot y$ is in the order of $2n$-bits. So each addition in the worst case is: $O(2n) = O(n)$. So, since we have n steps, the overall running time for this algorithm is $O(n^2)$.

What is the complexity of modular addition and modular multiplication of two n-bit numbers and why? (6 points)

Modular addition is exactly like regular addition, but it needs subtraction if the result exceeds n-bits. Both addition and subtraction take $O(n)$. So modular addition completes after $O(n)$ time.

In modular multiplication, we may need to perform division after we finish the multiplication. Both of these operations cost $O(n^2)$, so the overall time needed to finish modular multiplication is $O(n^2)$.

Prove that if x and y are positive integers with $x \geq y$, then $gdc(x, y) = gcd(x \mod y, y)$. (8 points)

This is Eulcid's rule, and the proof is:

It is enough to show the slightly simpler rule $gcd(x, y) = gcd(x - y, y)$ from which the one stated can be derived by repeatedly subtracting y from x.

Any integer that divides both x and y must also divide $x - y$, so $gcd(x, y) \leq gcd(x - y, y)$. Likewise, any integer that divides both $x - y$ and y must also divide both x and y, so $gcd(x, y) \geq gcd(x - y, y)$.

In an RSA cryptosystem, $p = 5$ and $q = 13$. Find appropriate exponents d and e. (8 points)

$$N = p \cdot q = 65$$

Let

$$\phi = (p - 1) \cdot (q - 1) = 48$$

e should be chosen to be relative prime w.r.t. ϕ, and $d = e^{-1} \mod \phi$.

So we have multiple choices for e (a small prime) and d, two of those choices are:

- $e = 5$, and $d = 29$. For this guess of e, we can find d by using the Extended Euclid Algorithm.
- $e = 7$, and $d = 7$. For this guess of e, we can find d by using the Extended Euclid Algorithm.

Note: e cannot be 3 because 3 divides ϕ.

C. Describe an expression for the multiplication of two n-bit numbers that has better running time than $O(n^2)$. Use the Master Theorem to show its running time. (6 points)

Let x and y be the two n-bit numbers. Let x_L, y_L and x_R, y_R be the left and right $\frac{n}{2}$-bit numbers of x and y respectively. Then we have $x = 2^{\frac{n}{2}}x_L + x_R$, $y = 2^{\frac{n}{2}}y_L + y_R$. The multiplication of x and y is

$$\begin{aligned} xy &= (2^{\frac{n}{2}}x_L + x_R)(2^{\frac{n}{2}}y_L + y_R) \\ &= 2^n x_L y_L + 2^{\frac{n}{2}}(x_L y_R + x_R y_L) + x_R y_R \\ &= 2^n x_L y_L + 2^{\frac{n}{2}}((x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R) + x_R y_R \end{aligned}$$

Hence, we could divide x and y into 4 $\frac{n}{2}$-bit integers x_L, x_R, y_L, y_R and obtain the multiplication by computing 3 multiplications $x_L y_L$, $x_R y_R$ and $(x_L + x_R)(y_L + y_R)$. Because we divide the integers into half the size of the original ones, the size of each subproblem is $\frac{n}{2}$. There are totally 3 subproblems. After solving those 3 subproblems, we have to combine the result using addition, subtraction and left shift operation to obtain the final result xy. All of the operations could be done in $O(n)$ time. The recurrence equation for the time complexity is:

$$T(n) = 3T(\frac{n}{2}) + O(n)$$

Applying the Master theorem, we know the time complexity is $O(n^{\log_2 3})$.

Suppose that we are given a sorted array of distinct integers $A[1, \ldots, n]$ and we want to decide whether there is an index i for which $A[i] = i$. Describe a divide-and-conquer algorithm that solves this problem and use the Master Theorem to argue that it should run in $O(logn)$ time. (8 points)

We could perform a binary search on the array as follows:

1. Initially set up the boundary of search as $left = 1, right = n - 1$;
2. Check left>right or not. If it is true, report false and halt, i.e., no such index i satisfying $A[i] = i$;
3. Compare $A[(left + right)/2]$ and the index $(left + right)/2$.
   (a) If they equal, report true and the index $(left + right)/2$, then halt;
   (b) If the former one is larger, update $right = (left + right)/2 - 1$. Go back to Step 2;
   (c) If the latter one is larger, update $left = (left + right)/2 + 1$. Go back to Step 2;

It is easy to see that the array is divided into two equal size subarrays and only one of them is selected for searching for the index. The time complexity is $T(n) = T(n/2) + O(1)$ since the comparison is done in constant time. From the master theorem, we know $T(n) = O(\log n)$.

When does the best-case of quicksort arise? Compute the running time of the algorithm in this case. (6 points)

The best-case happens if every time we pick the pivot as the median value of the current array. Then we could divide the array into two subarrays and each of them has half size of the original one. The time complexity is $T(n) = 2T(\frac{n}{2}) + O(n)$. The master theorem tells us that $T(n) = O(n \log n)$.

D. Describe an $O(n)$ expected running time algorithm that, given a set S of n distinct numbers and a positive integer $k \leq n$ determines the k numbers in S that are closest to the median of S. (12 points)

Let $m = \frac{n+1}{2}$ be the rank of the median. Using a linear time selection algorithm twice, we can find the two elements x_1 and x_2 with ranks $m - \frac{k-1}{2}$ and $m + \frac{k-1}{2}$. Then we go through S and find all elements x for which $x_1 \leq x \leq x_2$, which is a linear time operation. The overall running time is $O(n)$.

We are given n points in the unit circle, $p_i = (x_i, y_i)$, such that $0 < x_i^2 + y_i^2 \leq 1$ for $i = 1, 2, \ldots, n$. Suppose that the points are uniformly distributed that is the probability of finding a point in any region of the circle is proportional to the area of that region. Design a $\Theta(n)$ expected-time algorithm to sort the n points by their distance $d_i = \sqrt{(x_i^2 + y_i^2)}$ from the origin. [The computation of the distance of a point from the origin can be computed in constant time] [12 points]

We can apply bucket sort as long as the points are uniformly distributed on each bucket. The points are not uniformly distributed based on distance but based on area, so we have to make sure that points are assigned to each bucket in a uniform manner. Each bucket can correspond to a ring of minimum radius r_i and maximum radius r_{i+1}. All points in the corresponding ring can be assigned to the same bucket and we need to make sure that all rings have the same area, so that points are uniformly distributed. The area of a ring can be computed as: $\pi \cdot (r_{i+1}^2 - r_i^2)$, which should be equal to $\frac{\pi}{n}$ for all rings. If we set $r_i = \frac{\sqrt{i}}{\sqrt{n}}$, then the above requirement is satisfied.

Consequently, the algorithm works as follows:

- For every point p_i compute its distance d_i (constant time operation per point and linear overall).
- Identify the ring/bucket that each point should be assigned, i.e., it should be assigned to the bucket with index $\lfloor \frac{d_i}{\sqrt{n}} \rfloor$ (constant time operation per point and linear overall).
- Then perform bucket sort given these buckets (linear time operation).

E. What is the prefix-free property in compression codes? (4 points)

No code can be the prefix of another code.

Provide the Huffman encoding algorithm and argue its running time. (6 points)

Either the algorithm or a description of the algorithm was considered as correct answer.

Algorithm 1: Huffman Encoding Algorithm

```
1  Input: an array f[1,...,n] of frequencies.;
2  Output: An encoding tree with n leaves;
3  Lets H be a priority queue of integers, ordered by f;
4  for i = 1 to n do
5      insert(H,i)
6  for k=n+1 to 2n-1 do
       ; // This will run for n times :  O(n)
7      i = deletemin(H) ; //  O(logn)
8      j = deletemin(H) ; //  O(logn)
9      create a node numbered k with children i,j;
10     f[k] = f[i] + f[j];
11     insert(H,k) ; //  O(logn)
```

Running time: $O(nlogn)$ because a binary heap is used in order to build the tree. The actions insert(H,k) and deletemin(H) will take $O(logn)$ time. To build the tree we will call these functions n times. So the running time in total will be $O(nlogn)$

Consider an alphabet C where characters have frequencies $f[c], \forall c \in C$ and x, y are the two lowest frequency characters. Show that there exists an optimal prefix code for alphabet AC, where x and y have the same length and differ only in the last bit. [10 points]

Proof of Lemma 16.2 at CLRS page 389 and 390.