Java Algorithms

Randomized Algorithms

1. **Modular Arithmetic**
    a. Input: n-bit integers **x**, **y**, **N** ≥ 0
    b. Output: x^y **mod** N
    c. If y = 0, return 1
    d. **z** = mod-exp(**x**, **y/2**, **N**)
    e. If **y** is even, return ($\mathbf{z}^2$ **mod N**)
    f. If **y** is not even, return ($\mathbf{xz}^2$ **mod N**)
    g. Example: LeetCode 372
    h. Code example:

```
int modularExp(int x, int y, int N) {
    int res = 1;

    x = x % N;
    while (y > 0) {
        // If y is odd, multiply x with result
        if (y % 2 == 1)
            res = res * x % N;

        // y must be even now
        y = y / 2;
        x = x * x % N;
    }
    return res;
}
```

2. **Multiplicative inverses**
    a. It is another name for reciprocal.
    b. Try to find another number that times current number can be 1 after mod by the modulo.
    c. The multicative inverse is **unique**.
    d. Example:
        i. N = 14
        ii. The multicative inverse of **1** is **1** because (**1***1) **mod 14** = 1
        iii. The multicative inverse of **2** does not exist because both 2 and 14 are even.
        iv. The multicative inverse of **3** is **5** because (**3***5) **mod 14** = 1

v. The multicative inverse of **4** does not exist because both 2 and 14 are even.
vi. The multicative inverse of **5** is **3** because (**3***5) **mod 14** = 1

e. Code example

```
int multiplicativeInverse(int x, int N) {
   x = x % N;
   for (int i = 1; i < N; i++) {
       if ((x * i) % N == 1)
           return i;
   }

   // If the multiplicative inverse does not exist.
   return -1;
}
```

3. **Greatest Common Divisor (GCD) Euclid Algorithm**
   a. **GCD** of two numbers is the largest number that divides both of them.
   b. Euclid's rule: **gcd(x, y) = gcd(x mod y, y)**
   c. input : integers **x**, **y** where **x** ≥ **y** ≥ 0
   d. Output: gcd(x, y)
   e. If y = 0, return x
   f. If y ≠ 0, return **Euclid(y, x mod y)**
   g. Code example:

```
int euclidAlgorithm(int x, int y) {
   if (x == 0)
       return y;

   return euclidAlgorithm(y%x, x);
}
```

4. **Extended Euclid's algorithm**
   a. The formula with coefficients: **a*x + b*y = gcd(x, y)**.
   b. Input: **x, y, a, b**
   c. Output: the **gcd d** and coefficients **a** and **b**.
   d. For example
   e. Use:
      i. The extended Euclid's algorithm is particularly useful when **x** and **y** are **coprime** (or gcd is 1). Since **x** is the modular multiplicative

inverse of "**x mod y**", and **y** is the modular multiplicative inverse of "**y mod x**".

ii. The computation of the modular multiplicative inverse is an essential step in RSA public-key encryption method.

f. Code example:

```
int[] extEuclidAlgorithm(int x, int y) {
   if (y == 0)
       return new int[] {x, 1, 0};

   int[] values = extEuclidAlgorithm(y, x % y);
   int d = values[0];
   int a = values[2];
   int b = values[1] - (x / y) * values[2];
   return new int[] {d, a, b};
}
```

5. **Fermat's little theorem**
   a. If **P** is **prime** then for every **1 ≤ Z ≤ P-1**:
      i. $\mathbf{Z^{P-1} \equiv 1 \bmod P}$
   b. Statement: if **P** is a **prime** number, then for any integer Z, the number $\mathbf{Z^P}$ - **Z** is an multiple of **P**.
   c. For example:
      i. P = **17**
      ii. Z = **2**
      iii. $\mathbf{2^{17-1} \equiv 1 \bmod 17}$
      iv. 65536 % **17** ≡ 1
      v. (655536-1) is a multiple of 17
   d. Basis of RSA algorithm
   e. Fermat's Last Theorem
      i. According to Fermat's Last Theorem, no three positive integers a, b, c satisfy the equation $\mathbf{a^2 + b^2 = c^2}$, for any integer value of n greater than 2.
   f. The theorem (Fermat's little) can help find the modular inverse.
   g. Code example:

```
int calculateModInverse(int x, int y) {
   RA1 ra1 = new RA1();
   if (ra1.gcd.euclidAlgorithm(x, y) != 1)
       return -1;
   else {
       // Based on Fermat's little theorem, if a and m
```

```
            // are relatively prime, then modulo inverse is
            // x^(y-2) mod y.
            return power(x, y - 2, y);
        }
    }

    private int power(int x, int y, int m) {
        if (y == 0)
            return 1;

        int p = power(x, y / 2, m) % m;
        p = (p * p) % m;

        return (y % 2 == 0) ? p : (x * p) % m;
    }
```

6. **Euler's theorem**
    a. The **Euler's totient function**, or **phi (φ)** function is defined as the number of positive integers ≤ **n** that are relatively prime to **n**.
    b. For example:
        i. The co-prime number of **3** and ≤ **3** are **1, 2**, so φ(3) = 2.
        ii. The co-prime number of **4** and ≤ **4** are **1, 3**, so φ(4) = 2.
        iii. The co-prime number of **5** and ≤ **5** are **1, 2, 3, 4**, so φ(5) = 4.
        iv. The co-prime number of **7** and ≤ **7** are **1, 2, 3, 4, 5, 6**, so φ(7) = 6.
        v. The co-prime number of **15** and ≤ **15** are **1, 2, 4, 7, 8, 11, 13, 14**, so φ(15) = 8
    c. Relationship: when **n** is a prime number, e.g., 2, 3, 5, 7, **φ(n) = n-1**

7. **RSA Idea**
    a. Make use of Fermat's theorem and Euler's theorem.
    b. The idea of RSA is based on the fact that it is difficult to factorize a large integer.
    c. The **public key** consists of two numbers where one number is multiplication of **2** large **prime** numbers.
    d. The **private key** is also derived from the same **2 prime** numbers.

8. **RSA Algorithm Protocol**
    a. Generate public key
        i. Select 2 **prime** numbers **p** and **q**.
        ii. Assign **n** = **p***q**.
        iii. Need another number **e** where 1 ≤ **e** ≤ **φ(n)** and **e** is relatively prime to (p-1)*(q-1)

iv. Let **e** small so encryption is easier.
v. Public key will be made of **n** and **e**.
vi. Code example:

```
int[] getPublicKeyPair(int p, int q) {
   int e = 2;
   int phi = (p-1)*(q-1);
   while (e < phi) {
       if (euclidAlgorithm(e, phi) == 1)
           break;
       else
           e++;
   }
   return new int[] {p*q, e};
}
```

b. Generate private key
   i. Calculate **d ≡ e^{-1} mod (p-1)(q-1)**
   ii. Use extended Euclid's algorithm to find **d.**
   iii. Do not give private key **d** to anyone!
   iv. Code example:

```
double getPrivateKey(int p, int q, int e) {
   return multiplicativeInverse(e, (p-1)*(q-1));
}
```

c. Encrypting
   i. Input: message **m**.
   ii. Output: encrypted message **y**.
   iii. Use formula **y ≡ m^e mod N**.
   iv. Code example:

```
double encrypt(Double message, int[] publicKeyPair) {
   // Encryption c = (msg ^ e) % n
   int n = publicKeyPair[0];
   int e = publicKeyPair[1];
   return modularExp(message.intValue(), e, n);
}
```

d. Decrypting
   i. Input: encrypted message **y**.

ii. Output: original message **m**.
iii. Use formula $\mathbf{y^d \bmod N = m}$.
iv. Code example:

```
double decrypt(Double message, int n, Double d) {
   return modularExp(message.intValue(), d.intValue(), n);
}
```

e. RSA pitfalls
   i. If the message **m** is not prime of **n**, such that **gcd(m, n) > 1**. Otherwise, other people may be able to break the crypto system.
   ii. The message **m** is not too large.
   iii. Send the same message **m** too many times.

9. Fermat's Test

a. Used to test if the number is **prime**.
b. If **r** is prime then for all $\mathbf{z} \in \{1, \ldots, r-1\}$, $\mathbf{z^{r-1} \equiv 1 \bmod r}$.
c. Lemma: if **r** has nontrivial Fermat witness, then ≥ ½ the $\mathbf{z} \in \{1, \ldots, r-1\}$ are Fermat witness.

10. Simple Primality Test

a. Choose **z** randomly from {1, …, r-1}
b. Compute $\mathbf{z^{r-1} \equiv 1 \bmod r}$.
c. If $\mathbf{z^{r-1} \equiv 1 \bmod r}$, then output **r** is prime, else **r** is composite.
d. Code example:

```
boolean isPrimeUsingFermat(int n, int k) {
   // Handle edge cases where n = 1, 2, 3, 4
   if (n <= 1 || n == 4) return false;
   if (n <= 3) return true;

   // Try k times
   while (k > 0) {
       // Pick a random number in [2..n-2]
       int x = 2 + (int)(Math.random() % (n - 4));

       // Fermat's little theorem
       if (modularExp(x, n - 1, n) != 1)
           return false;

       k--;
   }
   return true;
}
```

11. **Balls into Bins - Mathematical problem**
    a. Input: **n** identical balls and **n** bins $\mathbf{B_1, B_2, \ldots, B_n}$.
    b. Each ball is thrown into a random bin (independent of other balls).
    c. Let load(i) = balls assigned to $\mathbf{B_i}$.
    d. Problem: how large is the max load?
        i. Answer: **O(log log n)**.
        ii. If choose d ≥ 2, get **O((log log n) / log d)**.
    e. Reference: http://facstaff.bloomu.edu/dcoles/Java/book/chapters/7_arrays/4.html

12. **Hashing setup**
    a. Example: unacceptable passwords
        i. Huge set **U** = possible passwords.
        ii. Maintain **S** ⊂ **U**.
        iii. Query for if **x** ∈ **U**, then is **x** ∈ **S**?
        iv. Hash table **H** = [0, 1, ... , n-1] where **H** is an array of linked lists.
        v. Hash function **h**: **U** → **H**.
        vi. Function **h(x)** random in {0, 1, …, n-1}.
        vii. **|U| = N** ≫ **|H| = n** ≫ **|S| = m**.
    b. Chain hashing
        i. **H[i]** = linked list of elements **x** ∈ **S** where **h(x) = i**.
        ii. Query time (is **x** ∈ **S**?): load at bin h(x).
        iii. Let **m = |S|** & **n = |H|**.
        iv. If m = n then max load is **O(log n)**, therefore query time = **O(log n)**.
    c. Operations needs to be implemented in hashmap:
        i. **get(K key)** : returns the value corresponding to the key if the key is present in HashTable.
        ii. **getSize()** : return the size of the HashTable.
        iii. **add()** : adds new valid key, value pair to the HashTable, if already present updates the value
        iv. **remove()** : removes the key, value pair
        v. **isEmpty()** : returns true if size is zero
    d. Implementation:
        i. https://www.geeksforgeeks.org/implementing-our-own-hash-table-with-separate-chaining-in-java/

13. **Load factor**
    a. If **n** be the total number of buckets we decided to fill initially.
    b. Say 10 and let's say 7 of them got filled now, so the load factor is 7/10=0.7

14. Bloom filters

a. Features
   i. Faster queries = **O(1)** time.
   ii. Less space, simpler
   iii. **But:** false positives with some small probability.
   iv. The false positive rate increases steadily as elements are added until all bits in the filter are set to 1, at which point all queries yield a positive result.
   v. Bloom filters never generate **false negative** result.
   vi. Deleting elements from filter is not possible.

b. Initialize **H** to all **0**.

c. Use random hash function **h: U → H**.

d. Have **k hash functions**: $\mathbf{h_1, h_2, \ldots, h_k}$.

e. Insert(x): **For i → k**, set $\mathbf{H[h_i(x)] = 1}$.

f. Query(x):
   i. if for all **i**, $\mathbf{H[h_i(x)] = 1}$ then output YES,
   ii. If for some **i**, $\mathbf{H[h_i(x)] = 0}$ then output NO.

g. Reference:
   i. https://www.geeksforgeeks.org/bloom-filters-introduction-and-python-implementation/

h. Code example:

```
class BloomFilter {

    private byte[] bitArray;
    private int size, hashCount;
    private MessageDigest md;

    public BloomFilter(int capacity, float falsePostiveProb) {
        // Size of bit array to use
        this.size = this.getSize(capacity, falsePostiveProb);

        // Number of hash functions
        this.hashCount = this.getHashCount(size, capacity);

        // Bit array of given size
        bitArray = new byte[this.size];

        // Initialize all bits as 0
        for (int i = 0; i < bitArray.length; i ++) {
            bitArray[i] = 0;
        }
```

```
        // Used for hashing
        try {
            md = MessageDigest.getInstance("MD5");
        } catch (NoSuchAlgorithmException e) {
            throw new IllegalArgumentException("MD5 not found");
        }
    }

    public int getSize(int count, float prob) {
        double m = -(count * Math.log(prob)/
                                    (Math.pow(Math.log(2), 2)));
        return (int) m;
    }

    private int getHashCount(int size, int count) {
        double k = (size/count) * Math.log(2);
        return (int) k;
    }

    public void add(Object data) {
        int[] keys = getCompleteHash(data);
        for (int i = 0; i < keys.length; i++) {
            this.bitArray[keys[i]] = 1;
        }
    }

    private int[] getCompleteHash(Object data) {
        int[] hashKeys = new int[hashCount];
        hashKeys[0] = getHash(data.hashCode());
        for (int i = 1; i < hashCount; i++)
            hashKeys[i] = (getHash(hashKeys[i - 1]));
        return hashKeys;
    }

    private int getHash(int i) {
        md.reset();
        byte[] bytes = ByteBuffer.allocate(4).putInt(i).array();
        md.update(bytes, 0, bytes.length);
        return Math.abs(new BigInteger(1, md.digest()).intValue())
                                        % (bitArray.length - 1);
    }

    public boolean contains(Object data) {
        int[] keys = getCompleteHash(data);
        for (int i = 0; i < keys.length; i++) {
```

```
            if (this.bitArray[keys[i]] != 1) {
                return false;
            }
        }
        return true;
    }
}
```