homework 3

Name: Yifan Zhang

University ID: yz113

Problem 1

> Solve the following recursions using "Unrolling" and "Guess+Verify". Boundary condition : $T(n) = 1$ for all $n \leq 1$.
>
> 1. $T(n) = 2T(\frac{n}{2}) + c$.
> 2. $T(n) = 2T(\frac{n}{2}) + c\sqrt{n}$.
> 3. $T(n) = 2T(\frac{n}{2}) + cn^2$.

1. $T(n) = 2T(\frac{n}{2}) + c$
   - Unrolling

     Analyzing the first few levels: At the first level of recursion, we have $T(n)$ that takes time c plus the time spent in all subsequent recursive calls. At next level, we have two problems that each of them takes time c, for a total of $2c$.

     At level j of recursion, we will have 2^j subproblems that each of them takes time c, for a total of $2^j c$.

     Summing over all levels of recursion, we will have $\log_2 n$ levels. Summing these levels up, we got a total running time of $O(nc - c)$.
   - Guess+Verify

     Suppose that $T(n)$ will take time $nc - c$.

     This holds for boundary case, because $T(2) = 2T(1) + c = 2 + c$.

     Now suppose, by induction, that $T(m) = mc - c$ for all values of m less than n, and we want to establish this for $T(n)$. Here is the full calculation:

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + c \\ &= 2(\frac{n}{2}c - c) + c \\ &= nc - 2c + c \\ &= nc - c \end{aligned}$$

2. $T(n) = 2T(\frac{n}{2}) + c\sqrt{n}$.
   - Unrolling

Analyzing the first few levels: At the first level of recursion, we have $T(n)$ that takes time $c\sqrt{n}$ plus the time spent in all subsequent recursive calls. At next level, we have two problems that each of them takes time $c\sqrt{\frac{n}{2}}$, for a total of $2c\sqrt{\frac{n}{2}} = c\sqrt{2n}$.

At level j of recursion, we will have 2^j subproblems that each of them takes time $c\sqrt{\frac{n}{2^j}}$, for a total of $2^j c\sqrt{\frac{n}{2^j}} = c\sqrt{2^j n}$.

Summing over all levels of recursion, we will have $\log_2 n$ levels. Summing these levels up, we got a total running time of $c(\sqrt{2}n - \sqrt{n})(\sqrt{2}+1)$, namely, $O(c(\sqrt{2}n - \sqrt{n}))$.

- Guess+Verify

  Suppose that $T(n)$ will take time $c(\sqrt{2}n - \sqrt{n})(\sqrt{2}+1)$.

  This holds for boundary case, because $T(2) = 2T(1) + c\sqrt{n} = c\sqrt{2} + 2$.

  Now suppose, by induction, that $T(m) = c(\sqrt{2}m - \sqrt{m})(\sqrt{2}+1)$ for all values of m less than n, and we want to establish this for $T(n)$. Here is the full calculation:

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + c\sqrt{n} \\ &= 2c(\sqrt{2}\frac{n}{2} - \sqrt{\frac{n}{2}})(\sqrt{2}+1) + c\sqrt{n} \\ &= c(\sqrt{2}n - \sqrt{2n} + 2n - 2\sqrt{n} + \sqrt{n}) \\ &= c((\sqrt{2}+1)\sqrt{2}n - (\sqrt{2}+1)\sqrt{n}) \\ &= c(\sqrt{2}n - \sqrt{n})(\sqrt{2}+1) \end{aligned}$$

3. $T(n) = 2T(\frac{n}{2}) + cn^2$.

   - Unrolling

     Analyzing the first few levels: At the first level of recursion, we have $T(n)$ that takes time cn^2 plus the time spent in all subsequent recursive calls. At next level, we have two problems that each of them takes time $c(\frac{n}{2})^2$, for a total of $2c(\frac{n}{2})^2 = c\frac{n^2}{2}$.

     At level j of recursion, we will have 2^j subproblems that each of them takes time $c(\frac{n}{2^j})^2$, for a total of $2^j c(\frac{n}{2^j})^2 = c\frac{n^2}{2^j}$.

     Summing over all levels of recursion, we will have $\log_2 n$ levels. Summing these levels up, we got a total running time of $2cn^2 - cn$, namely, $O(cn^2 - cn)$.

   - Guess+Verify

     Suppose that $T(n)$ will take time $2cn^2 - cn$.

     This holds for boundary case, because $T(2) = 2T(1) + cn^2 = 4c + 2$.

     Now suppose, by induction, that $T(m) = 2cm^2 - cm$ for all values of m less than n, and we want to establish this for $T(n)$. Here is the full calculation:

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + cn^2 \\ &= 2(2c(\frac{n}{2})^2 - c\frac{n}{2}) + cn^2 \\ &= cn^2 - cn + cn^2 \\ &= 2cn^2 - cn \end{aligned}$$

Problem 2.

> An array A of length N contains all the integers from 0 to N except one (in some random order). In this problem, we cannot access an entire integer in A with a single operation. The elements of A are represented in binary, and the only operation we can use to access them is "fetch the jth bit of $A[i]$". Using only this operation to access A, give an algorithm that

> determines the missing integer by looking at only O(N) bits. (Note that there are $O(NlogN)$ bits total in A, so we can't even look at all the bits). Assume the numbers are in bit representation with leading 0s.

We use recursion for this problem.

First we access the 2nd bit(because the first bit is always 0). Then we will know if numbers start with 1 miss a integer or numbers start with 0 miss a integer. Then, we throw the half part witch is not miss an integer, and recursively run this until we find the integer.

This will take $O(n + n/2 + n/4 + \dots) = O(n)$.

Problem 3.

We use recursion for this problem.

1. Suppose that current graph G' is $m \times m$. Find the smallest node (i, j) on row $m/2$. If this node is a local minimum node, then output this node. If this is not a local minimum node, then it means that $x(i, j+1)$ or $x(i, j-1)$ is/are smaller than $x(i, j)$. If $x(i, j+1) > x(i, j)$, then we throw rows $[m/2, m]$, otherwise we throw rows $[0, m/2]$ because there must exist a local minimum in the smaller part. Now we have a $(\frac{m}{2} \times m)$ graph.
2. Then we find the smallest node (k, l) on column $m/2$, (at this time we only find the smallest one in $\frac{m}{2}$ nodes because we have thrown half rows of graph). If this is not a local minimum node, then it means that $x(k, l+1)$ or $x(k, l-1)$ is/are smaller than $x(k, l)$. If $x(k, l+1) > x(k, l)$, then we throw columns $[m/2, m]$, otherwise we throw columns $[0, m/2]$. Now we have a $(\frac{m}{2} \times \frac{m}{2})$ graph. Then we recursively go to step 1 and use this $(\frac{m}{2} \times \frac{m}{2})$ graph as new input.

The time complexity is:

$$
\begin{aligned}
T(n) &= n + 2 \times n \times \frac{1}{2} + 2 \times n \times (\frac{1}{2})^2 \dots 2 \times n \times (\frac{1}{2})^{\log n} \\
&= n + n + n \times \frac{1}{2} + n \times (\frac{1}{2})^2 \dots n \times (\frac{1}{2})^{\log n} \\
&= n + 2 \times n - 1 \\
&= O(n)
\end{aligned}
$$

Problem 4.

1. Show how to solve this problem in $O(nlogn)$ time.

   We use recursion for this problem.

   1. We divide current array into 2 same size sub-array, and use these two sub-array as next recursion's input. Next recursion will return 2 hash map, keys are items in the sub-array, the values is how many time it occurs in its sub-array.
   2. Return a new hash map, the key is every distinct key in those 2 hash map, and the value is how many time it occurs in current array. If an item occur on both hash map, we simply add the 2 value. Note that this will only take $O(m)$, suppose that we have only have m items in current array.

   Every level will take $O(n)$ time and we will have $O(logn)$ levels. Overall, the time complexity is $O(nlogn)$

2. Can you give a linear-time algorithm?

   Yes we can. we need to use a hash table T with a hash function H.

We scan from the first element to the last element. For every element e, we find that if $T(H(e))$ is exist. If $T(H(e))$ is not exist, we create this pair and let its value be one, otherwise, we add one to it. If we find that after adding one, $T(H(e'))$ is larger than $\frac{n}{2}$, then we output e' as a majority element.

The time complexity is $O(n)$.

Problem 5.

We will use recursion to handle this problem.

For a Hadamard matrices H_k times a column vector of length $n = 2k$,

$$\begin{aligned} H_k v &= \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix} \begin{bmatrix} v_{0:n/2} \\ v_{n/2:n} \end{bmatrix} \\ &= \begin{bmatrix} H_{k-1} & v_{0:n/2} + v_{n/2:n} \\ H_{k-1} & v_{0:n/2} - v_{n/2:n} \end{bmatrix} \end{aligned}$$

Now it is divided into two matrix-vector product $H_{k-1}(v_{0:n/2} + v_{n/2:n})$ and $H_{k-1}(v_{0:n/2} - v_{n/2:n})$, and we recursively do this until $H_{k'-1} = H_0$.

We will have $logn$ levels and every level will take $O(n)$ time. Overall, this algorithm's time complexity is $O(nlogn)$

Problem 6.

First we transform the F_j to this:

$$F_j = Cq_j(\sum_{i<j} \frac{q_i}{(j-i)^2} + \sum_{i>j} \frac{q_i}{-(j-i)^2})$$

So the next step is obvious, we can use the FFT to compute the vector in the bracket for every j in $O(nlogn)$, after that, for every F_j, we can simply multiply Cq_j and corresponding element in the vector we can get from the FFT.

The vector in the bracket can be considered as two vector's convolution:

$$A = [q_1, q_2, \ldots, q_n]$$

$$B = [-\frac{1}{(n-1)^2}, -\frac{1}{(n-2)^2}, \ldots, -1, 0, 1, \frac{1}{(n-2)^2}, \frac{1}{(n-1)^2},]$$

From the definition of F_j,

$$F_j = Cq_j \sum_{1 \le i \le n} A_i B_{(n-1+j)-i}$$

and the right part $\sum_{1 \le i \le n} A_i B_{(n-1+j)-i}$ can be easily get from FFT in $O(nlogn)$. As a result, the overall time complexity is $O(nlogn)$.

Problem 7.

1. Proof
    1. $gcd(a, b) = 2gcd(a/2, b/2)$ if a, b are even.

        2 is a common divisor. Let's go from left to right

First we prove that $\frac{gcd(a,b)}{2}$ divides both $\frac{a}{2}$ and $\frac{b}{2}$ (Of course two even number's GCD is a even number. For and odd number x that divides both a and b, $2x$ will also be able to divides both a and b):

$$gcd(a,b)|a, gcd(a,b)|b \Rightarrow \frac{gcd(a,b)}{2}|\frac{a}{2}, \frac{gcd(a,b)}{2}|\frac{b}{2}$$

Then we prove that $\frac{gcd(a,b)}{2}$ is the greatest common divisor. Suppose there exist another number x which is $gcd(a/2, b/2)$ and x is larger than $\frac{gcd(a,b)}{2}$. In this case, we have:

$$\begin{aligned} x|\frac{a}{2}, x|\frac{b}{2} &\Rightarrow 2x|a, 2x|b \\ &\Rightarrow \text{2x should be the gcd(a, b)}, \\ &\text{because 2x is larger than gcd(a, b) and 2x devides both a and b} \end{aligned}$$

Proved by contradiction, there should be no x which can divide both $a/2$ and $b/2$ and larger than $\frac{gcd(a,b)}{2}$.

As a result, $\frac{gcd(a,b)}{2}$ can divide both $\frac{a}{2}$ and $\frac{b}{2}$, and there will not be any number larger than $\frac{gcd(a,b)}{2}$ which can divide both a and b. Namely, we have:

$$\frac{gcd(a,b)}{2} = gcd(\frac{a}{2}, \frac{b}{2}) \Rightarrow gcd(a,b) = 2gcd(\frac{a}{2}, \frac{b}{2})$$

2. $gcd(a,b) = gcd(a, b/2)$ if a is odd, b is even.

   It is obvious that 2 is not a common divisor. But if we have to prove, let's go from right to left.

   First we prove that $gcd(a, \frac{b}{2})$ divides b (Of course we know that $gcd(a,b)$ divides a. Besides, when a is odd, b is even, $gcd(a,b)$ is odd, because otherwise an even GCD will not divide an odd a):

   $$gcd(a, \frac{b}{2})|\frac{b}{2} \Rightarrow gcd(a, \frac{b}{2})|b$$

   Then we prove that $gcd(a, \frac{b}{2})$ is the greatest common divisor of a and b. Suppose that there exist a number x which is $gcd(a,b)$ and x is larger than $gcd(a, \frac{b}{2})$. In this case, we have:

   $$\begin{aligned} \text{a is odd; b is even; x is odd}, x|b &\Rightarrow x|2\frac{b}{2} \\ &\Rightarrow x|\frac{b}{2} \text{ , because of x is even} \\ &\text{x should be gcd(a, b/2),} \\ &\text{because x is larger than gcd(a, b/2)} \\ &\text{and x devides both a and b/2} \end{aligned}$$

   Proved by contradiction. As a result, $gcd(a, \frac{b}{2})$ can divide both a and b and there will not be any number larger than $gcd(a, \frac{b}{2})$ which can divide both a and b. Namely, we have:

   $$gcd(a, \frac{b}{2}) = gcd(a,b) \Rightarrow gcd(a,b) = gcd(a, \frac{b}{2})$$

3. $gcd(a,b) = gcd((a-b)/2, b)$ if a, b are odd and $a \geq b$.

   Because $a - b$ is even and b is odd, it is proved by sub-problem 2 that $gcd(a,b) = gcd(a, b/2)$. As a result, $gcd((a-b), b) = gcd((a-b)/2, b)$. The only thing that has not proved is $gcd((a-b), b) = gcd(a,b)$. It is obvious:

$$
\begin{aligned}
gcd(a,b)|a, gcd(a,b)|b &\Rightarrow gcd(a,b) \times m = a, gcd(a,b) \times n = b \\
&\Rightarrow a - b = gcd(a,b) \times m - gcd(a,b) \times n \\
&\Rightarrow a - b = gcd(a,b) \times (m - n) \\
&\Rightarrow gcd(a,b)|(a - b)
\end{aligned}
$$

As a result, we have $gcd(a,b) = gcd((a-b)/2, b)$ if a, b are odd and $a \geq b$.

2. b

We can simply add another rule base on sub-problem 1:

$$
\text{Rule 4:} gcd(a,b) = gcd(a/2, b), \text{if a is even, b is odd.}
$$

Then we can simply use these Rule 4 and other three rules from sub-problem 1 on a GCD algorithm, until we end with (0, b) or (a, 0), then let x be how many time we go into Rule 1 $gcd(a,b) = 2gcd(a/2, b/2)$, the $gcd(a,b)$ is $b \times 2^x$ or $a \times 2^x$.

The time complexity should be $O(n^2)$.