1. [*8 marks*] Give a proof from first principles (not using limits) the following statements:

(a) [*4 marks*]

$$n^{2.7} - 100n^{2.4} + 1000 \in \omega(n^{2.5})$$

(b) [*4 marks*] Let $f(n)$ and $g(n)$ be positive-valued functions. Then:

$$\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$$

(a) Let $c > 0$

$$n^{2.7} - 100 \cdot n^{2.4} + 1000 > c \cdot n^{2.5} \Rightarrow n^{0.2} - 100 \cdot n^{-0.1} + 1000 > c$$

$$\Rightarrow (n^{0.1})^2 - \frac{100}{n^{0.1}} + 100 > (n^{0.1})^2 - \frac{100}{n^{0.1}} > n^{0.1} - \frac{100}{n^{0.1}}$$

$$n^{0.1} - \frac{100}{n^{0.1}} > c \Rightarrow n > \left(\frac{c + \sqrt{c^2+400}}{2}\right)^{10}$$

Thus, if we choose $n_0 = \left(\frac{c+\sqrt{c^2+400}}{2}\right)^{10}$,

for any $c > 0$ we can prove that

$$n^{2.7} - 100 \cdot n^{2.4} + 1000 > c \cdot n^{2.5} \text{ for } n \geq n_0.$$

Therefore, we show that $n^{2.7} - 100 \cdot n^{2.4} + 1000 \in \omega(n^{2.5})$

(b) We want to find such $c_1, c_2 > 0$ that

$$c_1 \cdot [f(n) + g(n)] \leq \max\{f(n), g(n)\} \leq c_2 \cdot [f(n) + g(n)] \text{ for } n \geq n_0$$

We can express $\max\{f(n), g(n)\}$ as

$$\max\{f(n), g(n)\} = \frac{f(n)+g(n)}{2} + \left|\frac{f(n)-g(n)}{2}\right|$$

- $\frac{f(n)+g(n)}{2} + \left|\frac{f(n)-g(n)}{2}\right| \geq \frac{f(n)+g(n)}{2} = \frac{1}{2} \cdot [f(n)+g(n)]$

  Thus, c_1 we will choose $\frac{1}{2}$.

- $\frac{f(n)+g(n)}{2} + \left|\frac{f(n)-g(n)}{2}\right| \leq \frac{f(n)+g(n)}{2} + \frac{f(n)+g(n)}{2} = 1 \cdot [f(n)+g(n)]$

  Thus, c_2 we will choose 1

Since we can find such c_1 and c_2, we can conclude that

$$\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$$

2. [*12 marks*] For each pair of functions $f(n)$ and $g(n)$, fill in the correct asymptotic notation among Θ, o, and ω in the statement $f(n) \in \sqcup\, (g(n))$. Formal proofs are not necessary, but provide brief justifications for all of your answers. (The default base in logarithms is 2.)

(a) $f(n) = (8n)^{250} + (3n + 1000)^{500}$ vs. $g(n) = n^{500} + (n + 1000)^{400}$

(b) $f(n) = n^{1.5} 2^n$ vs. $g(n) = (n)^{100} 1.99^n$.

(c) $f(n) = (256)^{n/4}$ vs. $g(n) = (125)^{n/3}$

(d) $f(n) = 2^{\log(n)\cdot\log(n)}$ vs. $g(n) = n^{2012}$

(a) $f(n) \geq 3^{500} \cdot n^{500}$, $f(n) \leq (8^{500} + 1003^{500}) \cdot n^{500}$

Thus, $f(n) \in \Theta(n^{500})$

$g(n) \geq n^{500}$, $g(n) \leq (1 + 1001^{500})\, n^{500}$

Thus, $f(n) \in \Theta(n^{500})$

Therefore, $f(n) \in \Theta(g(n))$

(b) $\dfrac{f(n)}{g(n)} = \dfrac{(2/1.99)^n}{n^{98.5}} \longrightarrow \infty$, as $n \to \infty$.

Thus $f(n) \in \omega(g(n))$

(c) $f(n) = 4^n \qquad g(n) = 5^n$

$\dfrac{f(n)}{g(n)} = (\frac{4}{5})^n \longrightarrow 0$, as $n \to \infty$

Thus, $f(n) \in o(g(n))$

(d) $\log f(n) = \log(n) \cdot \log(n) \qquad \log g(n) = 2012 \cdot \log n$

$\dfrac{\log f(n)}{\log g(n)} = \dfrac{\log(n)}{2012} \longrightarrow \infty$, as $n \to \infty$

Therefore, $\log f(n) \in o(\log g(n))$

And hence, $f(n) \in o(g(n))$

3. [*10 marks*] Analyze the following pseudocodes and give a tight Θ bound on the running time as a function of n. Carefully show your work.

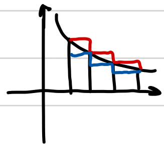
(a) [*5 marks*]

```
for i = 1 to n do
    A[i] = true
for i = 1 to n do
    j = i
    while j ≤ n do
        A[j] = false
        j = j + i
```

$\sum_{i=1}^{n} 1 + \sum_{i=1}^{n}\left(1+\left\lfloor \frac{n}{i} \right\rfloor \times 2\right) = n + n + 2n \cdot \sum_{i=1}^{n} \frac{1}{i}$.

For $\sum_{i=1}^{n} \frac{1}{i}$, since $\int_1^n \frac{1}{x} dx = \ln n$

$\sum_{i=1}^{n} \frac{1}{i+1} \le \int_1^n \frac{1}{x} \le \sum_{i=1}^{n} \frac{1}{i}$



Thus, $\sum_{i=1}^{n} \frac{1}{i} \in \Theta(\log n)$

Therefore, $T(n) = 2n + 2n \cdot \log n \in \Theta(n \cdot \log n)$

(b) [*5 marks*] The following is a sorting algorithm that sorts an array A of n integers, where each integer $e_i \in A$ is $0 \le e_i \le m-1$. Go through the code and verify that this algorithm indeed sorts A correctly.

```
1. for i = 0 to m − 1 do
2.     counts[i] = 0
3. for i = 0 to n − 1 do
4.     counts[A[i]] ++
5. k = 0
6. for i = 0 to m − 1 do
7.     for j = 0 to counts[i] − 1 do
8.         A[k] = i, k = k + 1
```

Line 1 & 2: initialize counts[] with m 0's.

Line 3 & 4: go through A[] and increament the corresponding count in counts[]

Now, we know that for $i = 0, 1, \dots, m-1$, there are counts[i] i's in A[]. Therefore, we can rearrange A[] based on counts[]. as line 5 – 8.

4. [*12 marks*] Given a string $s = a_1a_2...a_n$ of length n, where $a_1a_2...a_n \in \{0,1\}$, decide whether s is the kth *power* of a sub-string t, i.e., $s = t^k$, for some $k > 1$ and string t. Here, t^k denotes the string t repeated k times. For example, 01000100, 10101010, and 000000, are all perfect powers (e.g. $01000100 = 0100^2$) but 01000110 is not.

Give an algorithm that solves this problem in $O(n^{3/2})$ time. Describe your algorithm, provide the pseudocode, and analyze the run-time of your algorithm.

Hint: Observe that if $s = t^k$, and t has length ℓ, then $n = \ell k$. This implies that ℓ and k cannot both be greater than $\sqrt{n}$.

```
Check Pattern (s, l, k)
 // s: string, l: length of the pattern, k: times of repetition
   for i = 1 to k-1 do:
      if s[0 : l-1] != s[i·l : i·l+l-1] then:
          return false
   return true
main:
  Algo (s):
  // n = len(s)
   for l = 1 to ⌊√n⌋ do
      if  n % l = 0 then    // l divides n.
          k = n / l
          if  CheckPattern (s, l, k) = true then
                return true
   for k = 1 to ⌊√n⌋ do
      if  n % k = 0 then    // k divides n.
          l = n / k
          if  CheckPattern (s, l, k) = true then
                return true
   return false
```

Analysis:

CheckPattern(s, l, k) will loop $(k-1)$ times to check if there exists a pattern with length l in s. And for each substring comparison, it will take l time. Thus, this helper function is in $O(l(k-1))$.

Then, in the main function, we will call the helper in two loops. Since, from hint, we know l and k cannot both be greater than $\sqrt{n}$, we can check $\sqrt{n}$ times on l and then on k to cover all the cases. Thus, in total, Algo(s) will take $O(2\sqrt{n} \cdot l(k-1)) = O(\sqrt{n} \cdot l \cdot k)$. Since $n = lk$, the time complexity of my algorithm will be $O(\sqrt{n} \cdot n) = O(n^{3/2})$