1. By the definition of big-O notation, $f(n) = O(n)$ means that there exist positive constants c_1, n_1 so that $f(n) \leq c_1 n$ for all $n \geq n_1$, and $g(n) = O(n^2)$ means that there exist positive constants c_2, n_2 so that $f(n) \leq c_2 n^2$ for all $n \geq n_2$. If we set n' as $\max\{n_1, n_2\}$, we can say that for any $n \geq n'$, $f(n) \leq c_1 n$ and $g(n) \leq c_2 n^2$.

   (a) For any $n \geq n'$, $f(n) + g(n) \leq c_1 n + c_2 n^2 \leq (c_1 + c_2) n^2$. Thus, $f(n) + g(n) = O(n^2)$.

   (b) For any $n \geq n'$, $f(n) \cdot g(n) \leq c_1 n \cdot c_2 n^2 = c_1 c_2 n^3$. Hence, $f(n) \cdot g(n) = O(n^3)$.

2. We use Master Theorem for the first two subproblems, and use the recursion tree method for the last one.

   (a) $f(n) = n^{2.5}$ where $2.5 > \log_b a = 2$ and $4f(n/2) \leq \frac{1}{\sqrt{2}} f(n)$. The third case of Master Theorem applies. We thusly have $T(n) = O(f(n)) = O(n^{2.5})$.

   (b) $f(n) = n^3$ where $3 = \log_b a = 3$. The second case of Master Theorem applies. We thusly have $T(n) = O(n^3 \log n)$.

   (c) Say $T(n) \leq d$ for $n \leq 20$ where d is a constant.

$$
\begin{aligned}
T(n) &= T(n-13) + n \\
&= T(n-26) + n - 13 + n \\
&= \cdots \\
&\leq \lceil n/13 \rceil n + d \\
&= O(n^2).
\end{aligned}
$$

3. The key observation is that if there is a number x in A that appears more than $n/2$ times, then the median equals x.

   (a) Algorithm 1 is the pseudocode.

```
1  x ← median(A);
2  count ← 0;
3  for y ∈ A do
4  |  if x equals y then
5  |  |  count ← count + 1;
6  |  end
7  end
8  if count > n/2 then
9  |  output "Yes";
10 else
11 |  output "No";
12 end
```

Algorithm 1: decide(A, n)

(b) It takes $O(n)$ time to find the median (Line 1) and $O(n)$ time to iterate the loop (Lines 3-7). The rest of lines needs $O(1)$ time. Hence, this algorithm runs in $O(n)$ time.

(c) Yes, this problem can be solved in $O(n) = o(n \log n)$ time because

$$\lim_{n\to\infty} \frac{n}{n \log n} = 0.$$

4. We use the Young tableau for this problem.

(a) Algorithm 2 is the pseudocode.

```
ℓ ← 1;
r ← the index of the last prime in S;
found ← "No";
while ℓ ≤ r do
|  if S[ℓ] + S[r] equals k then
|  |  found ← "Yes";
|  |  break;
|  else
|  |  if S[ℓ] + S[r] > k then
|  |  |  r ← r − 1;
|  |  else
|  |  |  ℓ ← ℓ + 1;
|  |  end
|  end
end
output found;
```

Algorithm 2: test_GC(k)

(b) In each iteration of the while loop (Lines 4-15), either r decreases by 1 or ℓ increases by 1. Therefore, the number of iteration is $O(|S|) = O(n/\log n)$. Combining that each iteration needs $O(1)$ operations, the total running time is $O(n/\log n)$.

(c) Yes, this algorithm runs in $O(n/\log n) = o(n)$ time because

$$\lim_{n\to\infty} \frac{\frac{n}{\log n}}{n} = 0.$$

5. The key observation is that if $S[x] > x$, then $S[x+i] > x+i$ for any $i > 0$. Analogously, if $S[x] < x$, then $S[x-i] < x-i$ for any $i < 0$.

(a) Algorithm 3 is the pseudocode. The initial call is find$(S, 1, n)$

```
if ℓ > r then
    output "No";
end
x ← ⌊(ℓ + r)/2⌋;
if S[x] equals x then
    output "Yes";
    return;
else
    if S[x] > x then
        return find(S, ℓ, x − 1);
    else
        return find(S, x + 1, r);
    end
end
```

Algorithm 3: find(S, ℓ, r)

(b) The correctness of Algorithm 3 relies on the correctness of the key observation. Because S is a sorted array of n distinct integers, $S[x+i] \geq S[x] + i$ for any $x, i \geq 0$. If we know that $S[x] > x$, then it implies that $S[x+i] \geq S[x] + i > x + i$ for any $x, i \geq 0$. Thus, $S[k] \neq k$ for $k \geq x$. Similarly, if $S[x] < x$, then $S[k] \neq k$ for any $k \leq x$. To sum up, we can always safely reduce the problem size into a half without missing the possible candidate $S[i]$ that has value i.

(c) It takes $O(1)$ time to reduce the problem size into a half. This process can be repeated by at most $\lceil \log_2 n \rceil$ times. The total running time is therefore $O(\log n)$.