Part (a):
The loop will end until i > n. For each iteration, i is powered by 2. Assume f(n) represents the current value of n after each turn. Thus,
$f(n) = f(n-1)^2$
Hence,
$f(n) = f(n-1)^2 = f(n-2)^3 = \ldots = (f(1)^2)^{n-1} = (2^2)^{n-1}$
Therefore, the time complexity for the function should be $\Theta(log(logn))$.

Part (b):
The outer loop will execute for n times. When i = $\sqrt{n}, 2\sqrt{n}, 3\sqrt{n} \ldots \sqrt{n}^2$, the inner loop will start to execute for i^3 times. Thus, the entire function will run

$$(\sqrt{n})^3 + (2\sqrt{n})^3 + (3\sqrt{n})^3 + \ldots + (\sqrt{n} * \sqrt{n})^2$$
$$= (\sqrt{n})^3 * (1^3 + 2^3 + 3^3 + \ldots + \sqrt{n}^3)$$

times. Thus the time complexity for the function will be O($(\sqrt{n})^3 * (1^3 + 2^3 + 3^3 + \ldots + \sqrt{n}^3)$) = O($n^3\sqrt{n}$).

Part (c):
Best case: A[k] will never equal to i and thus the if statement will never be executed. The time complexity should be O(n^2).
Worst case: every time A[k] equals to i, so the inner loop will be executed for $logn$ times each time. The total time complexity will then be O($nlogn$).
So overall, the time complexity should be O(n^2).

Part (d):
The outer loop will absolutely run n times and then consider for the worst case inside the outer loop, which is, for every n times, the value of i will equal to the value of size to trigger the if statement. As long as it starts to execute what is inside the if statement, there will run more "size" times because of the inside loop. And all the possible values of size form a geometric sequence for the next number is 1.5 times the former one. Thus, starting from 10, the total time the inner loop has run should be the sum of this geometric sequence. The maximum number of items in the sequence will be k < $\log_{\frac{3}{2}} 0.1n$. Therefore, by the sum formula of geometric sequence, the code should run $n + 10 * (\frac{3}{2}^{\log_{\frac{3}{2}} 0.1n} - 1) = n + 20 * (0.1n - 1) = 3n - 20$ times. Thus, the time complexity should be O(n).