1. **A) $\log_2 n$** ; B) Sqrt(N); C) n; D) n^2; E) 2^n
   #Setting up the equation
   - 1 min = 60 seconds
   - 1 microsecond = $1x10^{-6}$ seconds;
     - 1 second = 1,000,000 microseconds;
       - 60 seconds = 60,000,000 microseconds

   #Solving the equations

a. **$\text{Log}_2 n \leq 60{,}000{,}000$ ($\log_b a = c \rightarrow b^a = c$; b = 2, a = n & c = 60,000,000)**
   $N \leq 2^{60{,}000{,}000}$
b. $\sqrt{N} \leq 60{,}000{,}000$ (Square both sides to get rid of √)
   $N \leq (60{,}000{,}000)^2$
c. $N \leq 60{,}000{,}000$
d. $N^2 \leq 60{,}000{,}000$ (Square root to solve for N)
   $N \leq 7745.966$ (round up to 7746)
e. $2^n \leq 60{,}000{,}000$ (Take the logarithm of both sides to isolate N)
   $\log_2(2^n) \leq \log_2(60{,}000{,}000)$
   $N \leq \log_2(60{,}000{,}000)$
   25.8 (26 round up)

$\text{Log}_2(n)$ allows for the biggest n

2. Justify the following statements using the "Big-OH" definition

A) (n+100)^3 is O(n^3), and n^3 is O((n+100)^3);
   - For (n+100)^3 to be in O(n^3), there must be positive constants C and n0 such that c*(n^3) >= (n+100)^3 for all n>=n0. When n0=101 and c=8, this statement is true. Therefore, $(n+100)^3$ is in $O(n^3)$. Attached is a proof that $8*(n)^3 >= (n+100)^3$ for all n >= 101

   - For n^3 to be in O((n+100)^3), there must be a positive constants C and n0 such that c*((n+100)^3) >= n^3 for all n>=n0. When n0=1 and c=1, this statement is true, therefore n^3 is in $O(n+100)^3$. Attached is a proof that $n^3 <= (n+100)^3$ for all n>=1.

B) n^4 is NOT O(n^2);
   - Suppose n^4 is $O(n^2)$. Then there must exist positive constants C and N_0 such that $C*N^2>=N^4$ for all $N>=N_0$
     $N^4 \leq c*n^2 \rightarrow n^2 \leq c$
     The above inequality cannot be satisfied since c must be a constant. As n approaches infinity, no matter the size of c, there will always exist an $n > n_0$ for which $n^2 > c$. Since there does not exist a C and an N_0 for which the statement, $C>=N^2$ $(C*N^2>=N^4)$ for all $N>=N_0$, is true, N^4 is not $O(N^2)$

C) If f1^(n) is O(g1^(n)) and f2^(n) is O(g2^(n)), then f1^(n)*f2^(n) is O(g1^(n)*g2^(n))

- By the definition of big O notation, f1(n) is O(g1(n)) if and only if there exists a constant C1 and n1 such that for all n >= n1, f1(n) <= C1 * g1(n). Similarly there exists constants C2 and n2 such that for all n >= n2, f2(n) <= C2 * g2(n)
- Multiplying these two equations we get: $f_1(n) * f_2(n) \le c_1(g1(n)) * c_2(g2(n))$ for all n>=n1 OR for all n>=n2. We can set our final n0 to the larger of n1 and n2 and rearrange the equation. Resulting in $f_1(n)*f_2(n) \le c_1*c_2*(g1(n)) * (g2(n))$ for all n $>= n_0$
- Now let $C_3 = C_1*C_2$ and $g_3(n) = g1(n)*g2(n)$
- With this, we can set up our equation, f1(n) f2(n) <= C3 * g3(n) for all n $\ge n_0$.

3.Give the asymptotic (“Big-Oh”) running time complexity of the following algorithm, show all the work that you have done

- Algorithm: Foo(A[], int n)
  Input: an Array A, an integer n

  X = 0 → **O(1)**
  For (i=0; i≤ n-1; i++) { → **O(N)**
      For (j=i; j≤n-1; j++) { → O(N/2) → **O(N)**
          X = x + A[j] → **O(1)**
      }
  For (k=0; k≤ n-1; k++) { → **O(N)**
      For (j=0; j≤ n-1; j++) { → **O(N)**
          X = x + A[j]*A[k] → **O(1)**
          }
      }
  }

  Equation = 1*N*(N+N*(N+1)) → $N*(N+N^2)$ → N^2+N^3 → N^3 → **O(N^3)**