1) Give the best possible bound for each function below using big-O notation and justify your answer.

(a) \( t(n) = 2n^2 + 3n - 5 \).

\text{ANSWER: } t(n) \text{ is } O(n^2). \text{ To show this we must show constants } c \text{ and } n_0 \text{ such that } t(n) \leq cn^2 \text{ for all } n \geq n_0. \\
First, t(n) = 2n^2 + 3n - 5 \leq 2n^2 + 3n.

And \( 3n \leq 3n^2 \) for all \( n \geq 1 \) (since dividing both sides by 3n gives \( n \geq 1 \)).
Therefore \( t(n) \leq 2n^2 + 3n^2 = 5n^2 \) for all \( n \geq 1 \). Therefore \( t(n) \) is \( O(n^2) \).

(b) \( t(n) = 8 \log_2(n) + n - 5 \)

\text{ANSWER: } t(n) = O(n).

\text{Proof: } t(n) = 8 \log_2(n) + n - 5 \leq n + 8 \log_2 n \text{ and } \\
8 \log_2 n \leq 8n \text{ for } n \geq 1, \text{ since } n \geq \log_2 n \text{ for all } n \geq 1.

Therefore \( t(n) \leq 9n \) for all \( n \geq 1 \). Therefore \( t(n) = O(n) \).

(c) \( t(n) = 4n \log_2 n + 4n + 10 \)

\text{ANSWER: } t(n) = O(n \log_2 n).

\text{Proof: } 4n \leq 4n \log_2 n \text{ for all } n \geq 2.

\( 10 \leq 10n \log_2 n \text{ for all } n \geq 2 \)

Therefore \( t(n) = 4n \log_2 n + 4n + 10 \leq 18n \log_2 n \text{ for all } n \geq 2. \\
Therefore \( t(n) \) is \( O(n \log_2 n) \).

(d) \( t(n) = \sum_{i=1}^{2n} i \)

\text{ANSWER: } t(n) = O(n^2).

\text{Proof: } t(n) = \sum_{i=1}^{2n} i = 2n(2n + 1)/2 = n(2n + 1) = 2n^2 + n.

Clearly, \( n \leq n^2 \) for all \( n \geq 1 \) (and \( n \leq -1 \), but this does not affect our answer). Therefore \( t(n) = 2n^2 + n \leq 2n^2 + n^2 = 3n^2 \) for all \( n \geq 1. \\
Therefore \( t(n) = O(n^2) \).

\textbf{Extra Credit: } t(n) = \sum_{i=1}^{\log_2 n} 2^i

\text{ANSWER: } \sum_{i=1}^{\log_2 n} 2^i = 2^{\log_2 n+1} - 1 = 2n - 1 \text{ which is } O(n) \text{ for } c = 2 \text{ and } n_0 = 1.
2) Show that \( t(n) = 2n + 1 \) is \( \Theta(n) \).

**ANSWER:** To show that \( t(n) \) is \( \Theta(n) \) we need to show \( t(n) \) is \( O(n) \) and \( t(n) \) is \( \Omega(n) \).

It is easy to see that \( t(n) = 2n + 1 \) is \( O(n) \) since \( 2n + 1 \leq 3n \) for \( n \geq 1 \). Similarly, \( t(n) = 2n + 1 \geq 2n \) for all \( n \geq 1 \), so \( t(n) \) is \( \Omega(n) \).

Therefore we have shown that \( t(n) \) is \( \Theta(n) \).

3) Show that \( t(n) = 2n + 1 \) is not \( O(\log_2 n) \).

**ANSWER:** Proof by contradiction. Assume that \( t(n) = 2n + 1 \) is \( O(\log_2 n) \).

Then by the definition of big-O, there exist positive constants \( c \) and \( n_0 \) such that \( 2n + 1 \leq c \log_2 n \) for all \( n \geq n_0 \).

Solving for \( c \), we get: \( c \geq \frac{2n + 1}{\log_2 n} \). But the right-hand side is an increasing function and therefore cannot remain smaller than a constant as \( n \) increases to infinity. Therefore this cannot be true.

Therefore \( t(n) \) is not \( O(\log_2 n) \).

4) Prove: If \( f(n) \) is \( O(n) \) and \( g(n) = f(n) + 100 \) then \( g(n) \) is \( O(n) \).

**ANSWER:** Assume \( f(n) \) is \( O(n) \) and let \( g(n) = f(n) + 100 \).

By the definition of big-O, this means that \( f(n) \leq cn \) for all \( n \geq n_0 \) for some positive constants \( c \) and \( n_0 \).

Then \( g(n) = f(n) + 100 \leq cn + 100 \).

To show \( g(n) \) is \( O(n) \) we need to show that \( g(n) \) satisfies the definition if \( O(n) \). That is, we must show some constants \( c' \) and \( n'_0 \) such that \( g(n) \leq c'n \) for all \( n \geq n'_0 \).

We know that \( g(n) \leq cn + 100 \). Since \( 100 \leq 100n \) we can also say:

\( g(n) \leq cn + 100n = (c + 100)n \). This is true for all \( n \geq n_0 \). Therefore \( g(n) \) is \( O(n) \) or \( c' = c + 100 \) and \( n'_0 = n_0 \).

5) Consider the following code:

```plaintext
read n
i = 1
while (i <= n) {
  i = 2*i
  print i
}
```

(a) If \( n = 50 \) what does the code print out?
ANSWER: 2 4 8 16 32 64
(b) As a function of \( n \), how many times is the loop executed?
   (i.e., how many numbers does the code print out?) Do some small examples
to figure this out.
ANSWER: \([\log_2 n] + 1\).
6) Give recurrence relation for running time of the following functions.
   For part (a) count the number of multiplications. For part (b) count the
   number of comparisons (\( \text{tmp} < A[\text{start}] \)). For part (c) count the number
   of comparisons if \( \text{start} == \text{end} \).

(a) int myst(int n)
   {
     if (n == 0) return 1;
     if (n == 1) return 2;
     else return myst(n-2)*4;
   }

ANSWER: \( t(0) = t(1) = 0 \)
   \( t(n) = 1 + t(n-2) \quad \text{(for all} \ n > 1) \)

(b) int find_min(int A[], int start, int end)
   {
     int tmp;

     if (start == end) return A[start];
     else {
       tmp = find_min(A, start + 1, end);
       if (tmp < A[start]) return tmp;
       else return A[start];
     }
   }

ANSWER: \( t(1) = 0 \)
   \( t(n) = t(n-1) + 1 \quad \text{(for all} \ n > 1) \)
(c) int process(int A[], int start, int end)
{
    int mid;

    if (start == end) {
        printf("%d\n", A[start]);
        return;
    } else {
        mid = (start + end)/2;
        process(A, start, mid);
        process(A, mid+1, end);
    }
}

ANSWER: t(1) = 1
        t(n) = 1 + 2t(n/2)

You may assume for part (c) that n is a power of 2.