1. The following list of functions lists the exact number of steps required by a variety of algorithms, in the best case and in the worst case, respectively. For each of them, give tight upper and lower bounds in terms of the Big-$\mathcal{O}$ and Big-Ω notations, respectively. If these bounds match, then specify the bound using the Θ notation.

   Note that a tight upper (resp. lower) bound on a function $f(n)$ is a function $g(n)$ such that $f(n) = \mathcal{O}(g(n))$ (resp. $f(n) = \Omega(g(n))$) and $f(n) \neq o(g(n))$ (resp. $f(n) \neq \omega(g(n))$)

   1. $2n^2 + n\log n$ and $5n^3 + 2n - 1132$
   2. $10 \cdot 3^n + 6n\log n$ and $3 \cdot 2^n + 100 \cdot 3^n + 29n\log n$
   3. $79n + 11n\log n$ and $4n\sqrt{n} + 9n(\log n)^4$
   4. 42 and $16n^{\log^2 n} + 2n2^{\log^3 n}$

2. In each of the following questions, the functions $f(n), g(n)$, for $n \in \mathbb{Z}^+$, contain arbitrary constants a, b, c and d. For each pair, determine if $f(n) = \mathcal{O}(g(n))$ irrespective of the value of the constants. If not, specify the conditions to be imposed on the constants under which $f(n) = \mathcal{O}(g(n))$. Assume that the logarithms are given in base 2, unless specified otherwise.

   For example, if $f(n) = an^c$ and $g(n) = bn^d$, then $f(n) = \mathcal{O}(g(n))$ if and only if $c \leq d$ (Why?).

   1. $f(n) = a\log_c n$ and $g(n) = b\log_d n$
   2. $f(n) = a(\log_c n)^2$ and $g(n) = b\log_d n$
   3. $f(n) = an\log_c n$ and $g(n) = bn^{1+\epsilon}$, where $\epsilon > 0$ is a small constant
   4. $f(n) = 3^{\log_c n}$ and $g(n) = 2^{\log_d n}$
   5. $f(n) = 3^{an}$ and $g(n) = 2^{bn}$
   6. $f(n) = 3^{a\log_c n}$ and $g(n) = 2^{b\log_d n}$
   7. $f(n) = (\log n)^{\log n}$ and $g(n) = 2^{(\log_d n)^2}$
   8. $f(n) = n^c$ and $g(n) = (\log n)^{b\log_d n}$
   9. $f(n) = a\log_c n$ and $g(n) = b\log^* n$
   10. $f(n) = n$ and $g(n) = (\log n)^{\log^* n}$

3. For each of the following questions, determine if it is *always true*, *always false* or neither.

   1. $f = \mathcal{O}(g)$ and $g = \mathcal{O}(f)$
   2. $f = \mathcal{O}(g)$ and $g = \Omega(f)$
   3. $f = \mathcal{O}(g)$ and $g = o(f)$
   4. $f \neq \mathcal{O}(g)$ and $g \neq \mathcal{O}(f)$
   5. $f = \Omega(g)$ and $g = \Omega(f)$
   6. $c = \Theta(1)$
   7. $af(n) = \mathcal{O}(f(n))$; a constant
   8. $f(n) = \mathcal{O}\left(f(n)^2\right)$
   9. $f = \mathcal{O}(g)$ and $2^{f(n)} = \mathcal{O}\left(2^{g(n)}\right)$
   10. $f = o(g)$ and $\log f(n) = \mathcal{O}(\log g(n))$
   11. $f(n) + g(n) = \Theta(max(f(n), g(n)))$
   12. $f(n) + \mathcal{O}(f(n)) = \Theta(f(n))$

   If your answer is neither always true nor always false, then give an example to show when it is true and an example for when it is false. If possible, also make minor modifications to the statement to make it either always true or always false.

4. Show that $\sum_{i=1}^{n} i^k = \mathcal{O}\left(n^{k+1}\right)$.

5. Show that if $f(n) = \mathcal{O}(h(n))$ and $g(n) = \mathcal{O}(h(n))$ then $f(n)g(n) = \mathcal{O}\left(h(n)^2\right)$.

6. For each of the following recurrences, solve it to give both an upper bound and a lower bound on $T(n)$; using the recursion tree method would generally be easier, though not required. You may also try to solve these using the Master method.

    1. $T(n) = T(n/2) + \Theta(1)$
    2. $T(n) = T(2n/3) + \Theta(n)$
    3. $T(n) = 2T(n/2) + \Theta(1)$
    4. $T(n) = 3T(n/3) + \Theta(n)$
    5. $T(n) = 4T(n/2) + \Theta(1)$
    6. $T(n) = 4T(n/2) + \Theta(n^2)$
    7. $T(n) = T(n-2) + \Theta(1)$
    8. $T(n) = T(n-1) + \Theta(n^2)$
    9. $T(n) = 2T(n-1) + \Theta(1)$
    10. $T(n) = T(n/3) + T(2n/3) + \Theta(n)$
    11. $T(n) = T(n/2) + T(n/4) + \Theta(1)$
    12. $T(n) = T(n/2 + \log n) + \Theta(n)$

        Hint: Use the fact that $n/2 < n/2 + \log n < 3n/4$ and that $T(n)$ is monotonically increasing
    13. $T(n) = T(\sqrt{n}) + \Theta(n)$
    14. $T(n) = \sqrt{n}T(\sqrt{n}) + \Theta(n)$

        Hint: Divide the equation by n and let $S(n) = \frac{T(n)}{n}$.
    15. $T(n) = 2T(n/2) + \frac{n}{\log n}$
    16. $T(n) = 2T(n-2) + 1$
    17. $T(n) = 3T(n-2) + n$

7. Solve each of the following recurrences using the characteristic root method and express the solution using the Big-$\mathcal{O}$ notation.

    1. $T(n) = T(n-1) + T(n-2)$
    2. $T(n) = 5T(n-1) - 6T(n-2)$
    3. $T(n) = 3T(n-1) - 2T(n-2)$
    4. $T(n) = 4T(n-1) - T(n-2) - 6T(n-3)$
    5. $T(n) = T(n-1) + T(n-2) + c$

        Hint: Why can't you directly apply the characteristic root method here? Think in terms of the recursion tree and how the term c relates to it.

8. For each of the following algorithms, compute the best upper and lower bounds you can. You may ignore the correctness or the purpose of the algorithms.

(a) COMBINE$(A[1..n], B[1..m])$

Input. Arrays A, B

Output. Array C in sorted order, containing all elements from A, B

```
1:  j = 1, k = 1
2:  for i = 1 to n + m do
3:      if A[j] ≤ B[k] then
4:          C[i] = A[j]
5:          j = j + 1
6:      else
7:          C[i] = B[k]
8:          k = k + 1
9:      end if
10: end for
11: return C
```

(b) ISPRIME(n)

Input. Integer $n \geq 4$

Output. YES if n is a prime number, NO otherwise.

```
1:  j = 1, k = 1
2:  for i = 2 to √n + 1 do
3:      if n  mod i == 0 then
4:          return NO
5:      end if
6:  end for
7:  return YES
```

(c) ULTISEARCH$(A[1..n], x)$

Input. An unsorted array A of integers, an integer x

Output. YES if x is an element of A, NO otherwise.

```
1:  mid = (1+n)/2
2:  if ultiSearch(A[1..mid − 1], x) == YES then
3:      return ultiSearch(A[1..mid − 1], x)
4:  else if ultiSearch(A[mid + 1..n], x) == YES then
5:      return ultiSearch(A[mid + 1..n], x)
6:  else if A[mid] == x then
7:      return YES
8:  end if
9:  return ultiSearch(A[1..mid − 1], x)
```

(d) WEIRDSEARCH$(A[1..n], x)$

Input. An unsorted array A of integers, an integer x

Output. YES if x is an element of A, NO otherwise.

```
1:  mid = (1+n)/2
2:  if A[mid] == x then
3:      return YES
4:  else if weirdSearch(A[1..mid − 1], x) == YES or
5:                          weirdSearch(A[mid + 1..n], x) == YES then
6:      return YES
7:  end if
8:  return NO
```

(e) $\underline{F(n)}$

Input. Integer $n \geq 0$

Output. The value of $F(n)$, where F is defined as $F(i) = F(i-1) + i - 1$ if i is odd, $F(i) = F(i-1) + F(i-2)$ if i is even and $F(0) = 0$.

```
1: if n == 0 then
2:     return 0
3: else if n mod 2 == 1 then
4:     return F(n − 1) + n − 1
5: end if
6: return F(n − 1) + F(n − 2)
```

(f) $\underline{F(n)}$

Input. Integer $n \geq 0$

Output. The value of $F(n)$, where F is defined as $F(i) = F(i-1) + 2F(i-2)$ if i is odd, $F(i) = 2F(i-1) + F(i-2)$ if i is even and $F(0) = F(1) = 0$.

```
1: if n == 0 or n == 1 then
2:     return 0
3: else if n mod 2 == 1 then
4:     return F(n − 1) + 2(n − 2)
5: end if
6: return 2F(n − 1) + (n − 2)
```