$\boxed{P1}$

1) $O(\log_2 n^2 + (\log_2 n)^2 + \log_2 n)$

$O(2\log_2 n + (\log_2 n)^2 + \log_2 n)$

$= O(\log_2 n)^2$

2) $O(n^2 + (n+1)^2 + (\frac{n}{2})^2)$

$O(n^2 + n^2 + 2n + 1 + \frac{n^2}{4})$

$O(\frac{9n^2}{4} + 2n + 1)$

$= O(n^2)$

3) $O(\sqrt[3]{n} + \log_2 n)$

$= O(\sqrt[3]{n})$

4) $O(1+2+3+\ldots+1000)$ Sum of first 1000 integers is 500500

$= O(1)$

5) $O(1+3+5+\ldots+(2n+1))$ Sum of first n odd numbers

$= O(n^2)$

$1 = 1^2$

$1+3 = 2^2$

$1+3+5 = 2^3$

$\boxed{P2}$

1)
```
int y = 0;
for(int i = 1; i < n; i *= 2){
    y++;
}
```
- i is doubled each time ($i *= 2$) which indicates a base 2 logarithmic behavior

$= O(\log_2 n)$

2)
```
int y = 0;
for(int i = 0; i < n; i++){
    for (int j = n; j > i; j--){
        y++;
    }
}
```
- The outer loop runs n times because it starts from 0 and goes up n-1
- The inner loop starts from n and decrements until it reaches i. As i increases the number of time the inner loops runs will decrease

$n + (n-1) + (n-2) + \ldots + 2 + 1$

$= \dfrac{n(n+1)}{2}$

$= O(n^2)$

3) int y = 0;

```
for ( int i = 1 ; i < n ; i++) {
    for ( int j = 1 ; j < i*i ; j++) {
        y++;
    }
}
```

- Outer loop = $O(n)$
- Inner loop will run i^2 times
  $1^2 + 2^2 + 3^2 + \ldots + (n-1)^2$
- sum of squares = $\frac{n(n+1)(2n+1)}{6}$
= $O(n^3)$

4) int y = 0;

```
for (int i = 1; i < n, i++) {
    for( int j = 1, j < sqrt(i), j++) {
        y++;
    }
}
```

$1 \quad 1.5$

- Outer loop : $O(n)$
- Inner loop depends on sqrt(i)
  $\sqrt{1} + \sqrt{2} + \sqrt{3} + \ldots + \sqrt{n-1}$
- Sum grows slower than $n^{1.5}$ $\qquad \sqrt{1 + \ldots + n} = n^{1.5}$
= $O(n^{1.5})$

5)
```
int y = 0;
if (x>0) {
    for (int i = n; i > 1; i--) {
        for (int j = i; j > 1; j = j / 3) {
            y++;
        }
    }
} else {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n / (i+1); j++) {
            y++;
        }
    }
}
```

If part

- Outer loop runs n times : $O(n)$
- Inner loop runs logarithmically to base 3 depending on i : $O(\log_3(i))$
- Together : $O(n \times \log_3(i))$

Else part

- Outer loop runs n times : $O(n)$
- Inner loop : as i grows, $n(i+1)$ decreases
- Combined effect is less than $O(n^2)$, but more than $O(n \times \log(n))$ so use upper bound : $O(n^2)$

$\boxed{P3}$

1. Find an element in the list based on its value
   - ArrayList: O(n), traverse the entire list

   - Singly Linked List: O(n), traverse through the list, one element by one

   - Doubly Linked List: O(n), traverse entire list, backward pointers don't help the case.
2. Inserting an element at the beginning of list
   - ArrayList: O(n), shift all elements to the right, linear time

   - Singly Linked List: O(1), adjust the head pointer

   - Doubly Linked List: O(1), prepend a node and adjust the front and back pointers in constant time.
3. Removing an element from the end of the list
   - ArrayList: O(1) if array implementation keeps track of size or last element. Otherwise it is O(n).
   - Singly Linked List: O(n), traverse the list to find second last element to remove the last element
   - Doubly Linked List: O(1) if it keeps a tail pointer.

4. Inserting an element at the middle of the list
   - ArrayList: O(n). move half of elements

   - Singly Linked List: O(n), traverse half of the list

   - Doubly Linked List: O(n), traverse half of list even though there are two pointers