General instruction: This assignment includes two parts, written and programming. Please write/type your answers neatly so they can be readable. Please submit a single **PDF** file for the written part and a **zip** file for the programming part **before 11:59 P.M. on September 25, 2023**. File name format: P1_YourCaseID_YourLastName.zip (or pdf).

The special office hour for this assignment will be from **6:00 to 7:00 P.M. on September 20** in Zoom. You can also send an email to wxy215@case.edu for written problems and ajn98@case.edu for coding problems.

Written Problems (50 pts)

P1. Simplify (as much as possible) the following Big-O notation and explain. (15 pts)

1. $O(log_2 n^2 + (log_2 n)^2 + log_2 n)$ $= O(2\cancel{\log_2 n} + (\log_2 n)^2 + \cancel{\log_2 n}) = \boxed{O((\log_2 n)^2)}$ highest degree log
2. $O(n^2 + (n + 1)^2 + (n/2)^2)$ $= \boxed{O(n^2)}$, highest degree polynomial
3. $O(\sqrt[3]{n} + log_2 n)$ $= O(n^{\frac{1}{3}} + \cancel{\log_2 n}) = \boxed{O(n^{\frac{1}{3}})}$, n^c grows faster than $\log_c n$
4. $O(\cancel{1 + 2 + 3 + ... + 1000})$ $= \boxed{O(1)}$, it's constant
5. $O(\cancel{1 + 3 + 5 + ...} + (2n + 1))$ $\boxed{O(n^2)}$, arithmetic sequence is $O\left(\frac{n((2n+1)+(1))}{2}\right) = O(n^2)$ ← highest term

P2. Provide a tight Big-O notation and explain for each part of the pseudocode. (23 pts)

1. (2 pts)

```
int y = 0;
for (int i = 1; i < n; i *= 2) {
      y++;
   }
```

$\boxed{O(\log n)}$

i grows exponentially by a factor of 2. For instance if $n = 32$ and k represents the current iteration:

k	i
1	1
2	2
3	4
4	8
5	16
~~6~~	~~32~~

or as $i < n \rightarrow 2^k < 32 \rightarrow k \log_2 2 < \log_2 32$

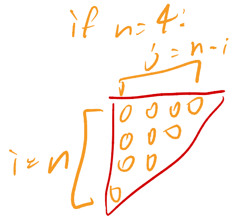
2. (3 pts)

```
int y = 0;
for (int i = 0; i < n; i++) {
  for (int j = n; j > i; j--) {
    y++;
  }
}
```

$\boxed{O(n^2)}$

The iterations can be visualized like a triangle

if $n = 4$:



$\rightarrow O\left(\frac{n^2}{2}\right) = O(n^2)$

3. (5 pts)

```
int y = 0;
  for (int i = 1; i < n; i ++) {
    for (int j = 1; j<i*i; j++){
      y++;
    }
  }
```

$\hookrightarrow j < i^2$

$\boxed{O(n^3)}$

Makes a triangle again except there's another dimension brought by i^2

if $n=4$:

visualization

$$i=(n-1)\left[\begin{array}{l} \overbrace{(0\,0\,0)^2}^{j\sim(n-1)^2} \\ (0\,0)^2 \\ (0)^2 \end{array}\right. \rightarrow O\left(\frac{(n-1)(n-1)^2}{2}\right) = O(n^3)$$

4. (5 pts)

```
int y = 0;
  for (int i = 1; i < n; i ++) {
    for (int j = 1; j<sqrt(i); j++){
      y++;
    }
  }
```

$\boxed{O(n^{\frac{3}{2}})}$

Makes yet another triangle

if $n=4$

$$i=(n-1)\left[\begin{array}{l} \overbrace{\sqrt{0\,0\,0}}^{j=\sqrt{n-1}} \\ \sqrt{0\,0} \\ \sqrt{0} \end{array}\right. \rightarrow O\left(\frac{(n-1)(n-1)^{\frac{1}{2}}}{2}\right) = O(n^{\frac{3}{2}})$$

5. (8 pts)

```
int y = 0;
if (x > 0) { // x is a random number
  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++;
    }
  }
}
```

$\boxed{O(n \log n)}$

Worst case is the same for whatever value x is

when $n=9$

$$i=n\left[\begin{array}{l} \overbrace{0\,0\,0 \cdots 0\,0\,0\,0}^{j=\log_3 n} \\ 0\,0\,0 \\ 0 \end{array}\right. \rightarrow O(n\log_3 n) = O\left(n\frac{\log n}{\log 3}\right) = O(n \log n)$$

when $n=8$

$$i=n\left[\begin{array}{ll} \overbrace{0\,0\,0\,0\,0\,0\,0\,0}^{j=1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}\approx \log n+\gamma} & n \\ 0\,0\,0\,0 & \frac{n}{2} \\ 0\,0\,0 & \frac{n}{3} \\ 0\,0 & \frac{n}{4} \\ 0\,0 & \frac{n}{5} \\ 0 & \frac{n}{6} \\ 0 & \frac{n}{7} \\ 0 & \frac{n}{8} \end{array}\right. \rightarrow O(n\cdot(\log n+\gamma)) = O(n \log n)$$

P3. In the context of array lists, linked lists, and doubly linked lists, explain their worst-case big O time complexities for the following operations. You need to provide answers for all 4 operations across all 3 types of lists, totaling 12 in all. (12 pts)

1. Finding an element in the list based on its value
2. Inserting an element at the beginning of the list
3. Removing an element from the end of the list
4. Inserting an element at the middle of the list

1. Worst case big O time complexity of finding an element in the list based on its value:
   a. Array: O(n); start at the beginning of the array and iterate over each value until you find the element
   b. Linked list: O(n); same linear search as an array from left to right
   c. Doubly linked list: O(n); same linear search as an array from left to right
2. Worst case big O time complexity of inserting an element at the beginning of the list:
   a. Array: O(n); create a new array of max size + 1, the first element is set, then the original array is copied into the new array
   b. Linked list: O(1); create a new head node and point it to the previous one
   c. Doubly linked list: O(1); perform the same operation as linked list
3. Worst case big O time complexity of removing an element from the end of the list
   a. Array: O(n); create a new array of max size - 1, and copy all the elements except for the last one
   b. Linked list: O(n); iterate to the second to last element and set the node pointer to null
   c. Doubly linked list: O(1); fetch the second element from the right and set the node pointer to null
4. Worst case big O time complexity of inserting an element at the middle of the list:
   a. Array: O(n); create a new array of max size + 1, copy the elements up to array.length/2, add the new element at the next index, then continue copying the rest of the elements
   b. Linked list: O(n); assuming there's a field that updates the size of the list, iterate to the node before the middle and create a new node that is pointed to by the current node, then point the new node to the next node.
   c. Doubly linked list: O(n); perform the same operation as the linked list