EXAM II

FINAL EXAM

PROBLEM 1:

State the asymptotic relationship between the functions $f(n)$ and $g(n)$, as in $f(n) = X\big(g(n)\big)$, where X may be O, Ω, Θ. You must justify your answer by showing your work using the $\lim_{n\to\infty} \frac{f(n)}{g(n)}$

$f = O(g)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$
$f = \Omega(g)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$
$f = \Theta(g)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0, \infty$

1a) $f(n) = 9n^2 + 4n - 2,\ g(n) = 3n^2 - 3$

$\lim_{n\to\infty} \frac{f(n)}{g(n)}$	*formula*
$\lim_{n\to\infty} \frac{9n^2 + 4n - 2}{3n^2 - 3}$	*substitute values*
$\lim_{n\to\infty} \frac{9n^2 ~~+ 4n - 2~~}{3n^2 ~~- 3~~}$ $\lim_{n\to\infty} \frac{9n^2}{3n^2}$	*remove smaller order terms*
$\lim_{n\to\infty} \frac{9~~n^2~~}{3~~n^2~~}$ $\lim_{n\to\infty} \frac{9}{3}$	*cancel like terms*
3	*solve*

Since $3 < \infty$, the statement $f = O(g)$ is true.
Since $3 > 0$, the statement $f = \Omega(g)$ is true.
Since $0 < 3 < \infty$, the statement $f = \Theta(g)$ is true.

1b) $f(n) = 6 \times \log_2(n)$, $g(n) = 5n \times \log_2(n)$

$\lim_{n\to\infty} \frac{f(n)}{g(n)}$	*formula*
$\lim_{n\to\infty} \frac{6 \times log_2(n)}{5n \times log_2(n)}$	*substitute values*
$\lim_{n\to\infty} \frac{6 \times \cancel{log_2(n)}}{5n \times \cancel{log_2(n)}}$ $\lim_{n\to\infty} \frac{6}{5n}$	*cancel like terms*
$\frac{6}{5} \times \lim_{n\to\infty} \frac{1}{n}$	*limit rule* $\lim_{x\to a}(c \times f(x)) = c \times \lim_{x\to a}(f(x))$
$\frac{6}{5} \times \frac{\lim_{n\to\infty}(1)}{\lim_{n\to\infty}(n)}$	*limit rule* $\lim_{x\to a} \frac{f(x)}{g(x)} = \frac{\lim_{x\to a} f(x)}{\lim_{x\to a} g(x)}$
$\frac{6}{5} \times \frac{1}{\lim_{n\to\infty}(n)}$	*limit rule* $\lim_{x\to a}(c) = c$
$\frac{6}{5} \times \frac{1}{\infty}$	*limit rule* $\lim_{n\to\infty}(x) = \infty$
$\frac{6}{5} \times \frac{1}{\infty}$ $\frac{6}{5} \times 0$ 0	*solve*

Since $0 < \infty$, the statement $f = O(g)$ is true.
Since $0 \not> 0$, the statement $f = \Omega(g)$ is false.
Since $0 \not< 0 < \infty$, the statement $f = \Theta(g)$ is false.

1c) $f(n) = \log_4(n)$, $g(n) = \log_2(n)$

$\lim_{n\to\infty} \frac{f(n)}{g(n)}$	*formula*
$\lim_{n\to\infty} \frac{log_4(n)}{log_2(n)}$	*substitute values*

$\lim_{n\to\infty} \frac{log_{2^2}(n)}{log_2(n)}$	*base conversion*
$\lim_{n\to\infty} \frac{\frac{1}{2} \times log_2(n)}{log_2(n)}$	*log rule* $\log_{a^b} x = \frac{1}{b} \times \log_a x$
$\lim_{n\to\infty} \frac{\frac{1}{2} \times \cancel{log_2(n)}}{\cancel{log_2(n)}}$ $\lim_{n\to\infty} \frac{\frac{1}{2}}{1}$ $\lim_{n\to\infty} \frac{1}{2}$	*cancel like terms*
$\lim_{n\to\infty} \frac{1}{2}$ $\frac{1}{2}$	*limit rule* $\lim_{x\to a}(c) = c$

Since $\frac{1}{2} < \infty$, the statement $f = O(g)$ is true.

Since $\frac{1}{2} > 0$, the statement $f = \Omega(g)$ is true.

Since $0 < \frac{1}{2} < \infty$, the statement $f = \Theta(g)$ is true.

1d) $f(n) = 2n^{0.9}$, $g(n) = 6\sqrt{n}$

$\lim_{n\to\infty} \frac{f(n)}{g(n)}$	*formula*
$\lim_{n\to\infty} \frac{2 \times n^{0.9}}{6\sqrt{n}}$	*substitute values*
$\lim_{n\to\infty} \frac{2 \times n^{0.9}}{2 \times 3\sqrt{n}}$	*factor*
$\lim_{n\to\infty} \frac{\cancel{2 \times} n^{0.9}}{\cancel{2 \times} 3\sqrt{n}}$ $\lim_{n\to\infty} \frac{n^{0.9}}{3\sqrt{n}}$	*cancel like terms*
$\lim_{n\to\infty} \frac{n^{0.9}}{3 \times n^{\frac{1}{2}}}$	*exponent rule* $a^{\frac{1}{2}} = \sqrt{a}$

$\lim_{n\to\infty} \frac{1}{3} \times \frac{n^{0.9}}{n^{\frac{1}{2}}}$ $\lim_{n\to\infty} \frac{1}{3} \times n^{0.9-\frac{1}{2}}$	*exponent rule* $\frac{x^a}{x^b} = x^{a-b}$
$\lim_{n\to\infty} \frac{1}{3} \times n^{\frac{9}{10}-\frac{1}{2}}$ $\lim_{n\to\infty} \frac{1}{3} \times n^{\frac{9}{10}-\frac{5}{10}}$ $\lim_{n\to\infty} \frac{1}{3} \times n^{\frac{4}{10}}$ $\lim_{n\to\infty} \frac{1}{3} \times n^{\frac{2}{5}}$	*subtraction*
$\frac{1}{3} \times \lim_{n\to\infty} \left(n^{\frac{2}{5}}\right)$	*limit rule* $\lim_{x\to a}\big(c \times f(x)\big) = c \times \lim_{x\to a}\big(f(x)\big)$
$\frac{1}{3} \times \left(\lim_{n\to\infty}(n)\right)^{\frac{2}{5}}$	*limit rule* $\lim_{x\to a}(f(x)^b) = \left(\lim_{x\to a}\big(f(x)\big)\right)^b$
$\frac{1}{3} \times (\infty)^{\frac{2}{5}}$ ∞	*limit rule* $\lim_{n\to\infty}(x) = \infty$

Since $\infty \not< \infty$, the statement $f = O(g)$ is false.
Since $\infty > 0$, the statement $f = \Omega(g)$ is true.
Since $0 < \infty \not< \infty$, the statement $f = \Theta(g)$ is false.

PROBLEM 2:

Using the master theorem for solving recurrences, state the Big-O value for the following recurrences. If it is inappropriate to use the master method, then state this fact instead.

$$\boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + f(n)\ where\ a \geq 1\ and\ b > 1}$$

$f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$	$T(n) = \Theta(n^{\log_b a})$
$f(n) = \Theta\left(n^{\log_b a}\right)$	$T(n) = \Theta(n^{\log_b a} \log_2 n)$
$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$ $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n	$T(n) = \Theta(f(n))$

2a) $T(n) = 27T\left(\frac{n}{3}\right) + 6n^4 + 2n + 2$

$a =$	27
$\frac{n}{b} =$	$\frac{n}{3}$
$f(n) =$	$6n^4 + 2n + 2$ $O(n^4)$
$n^{\log_b a} =$	$n^{\log_b a}$ $n^{\log_3 27}$ n^3

Case 3

$f(n) = O(n^4)$ is polynomially greater than $n^{\log_b a} = n^3$

$T(n) = \Theta(n^4)$: The work completed in the sub-roots, which are not leaves, of the recursion tree is the primary determinant of the computational complexity. As $f(n)$ is polynomially greater than $n^{\log_b a}$, our regularity condition is met.

2b) $\mathrm{T}(n) = 2\mathrm{T}\left(\frac{n}{4}\right) + 10\sqrt{n} + 1$

$a =$	2
$\frac{n}{b} =$	$\frac{n}{4}$
$f(n) =$	$10\sqrt{n} + 1$ $O\left(n^{\frac{1}{2}}\right)$
$n^{\log_b a} =$	$n^{\log_b a}$ $n^{\log_4 2}$ $n^{\frac{1}{2}}$

Case 2

$f(n) = O(n^{\frac{1}{2}})$ and $n^{\log_b a} = n^{\frac{1}{2}}$ are polynomially equivalent

$T(n) = \Theta(n^{\frac{1}{2}} \times \log_2 n)$: The work is evenly distributed between the leaves and sub-roots of the recursion tree.

2c) $\mathrm{T}(n) = 2\mathrm{T}(n-1) + 8n + 4$

$a =$	2
$\frac{n}{b} =$	$\frac{n-1}{1}$
$f(n) =$	$8n + 4$ $O(n)$
$n^{\log_b a} =$	$n^{\log_b a}$ $n^{\log_1 2}$ $undefined$

No Master Theorem Case Applicable

We can not compare the $f(n)$ and $n^{\log_b a}$ as $n^{\log_b a}$ is *undefined*. As such, the master theorem can not be applied here.

2d) $\mathrm{T}(n) = 4\mathrm{T}\left(\frac{n}{4}\right) + n \times \log_2(n) + 3$

$$a = 4$$

$$\frac{n}{b} = \frac{n}{4}$$

$$f(n) = n \times \log_2(n) + 3$$
$$O(n\log_2(n))$$

$$n^{\log_b a} = n^{\log_b a}$$
$$n^{\log_4 4}$$
$$n$$

Case 3

$f(n) = O(n\log_2(n))$ is polynomially greater than $n^{\log_b a} = n$

$T(n) = \Theta(n\log_2(n))$: The work completed in the sub-roots, which are not leaves, of the recursion tree is the primary determinant of the computational complexity. As $f(n)$ is polynomially greater than $n^{\log_b a}$, our regularity condition is met.

2e) $\mathrm{T}(n) = 27\mathrm{T}\left(\frac{n}{3}\right) + \frac{n^2}{9}$

$$a = 27$$

$$\frac{n}{b} = \frac{n}{3}$$

$$f(n) = \frac{n^2}{9}$$
$$O(n^2)$$

$$n^{\log_b a} = n^{\log_b a}$$
$$n^{\log_3 27}$$
$$n^3$$

Case 1

$n^{\log_b a} = n^3$ is polynomially greater than $f(n) = O(n^2)$

$T(n) = \Theta(n^3)$: The work completed in the sub-roots, which are not leaves, of the recursion tree is the primary determinant of the computational complexity. As $f(n)$ is polynomially greater than $n^{\log_b a}$, our regularity condition is met.

PROBLEM 3:

For the following algorithm, (3A) give a precise line count for every numbered line of code in terms of n and (3B) give the efficiency of the algorithm in terms of Big-O.

CODE

```
void sort(int A[], int n) {
1       int i = 0, j, s;
2       while(i < n-1){
3           s = i;
4           j = i+1;
5           while(j < n){
6               if(A[j] < A[s]){
7                   s = j;
                }
8               j += 1;
            }
9           A[i] = A[s];
10          i += 1;
        }
    }
```

FREQUENCY COUNTS AND BIG O NOTATION

Line	*Code*	*Count* (3A)	$O(\cdot)$ (3B)
	`void sort(int A[], int n) {`	-----	-----
1	`int i = 0, j, s;`	1	$O(1)$
2	`while(i < n-1){`	n	$O(n)$
3	`s = i;`	$n-1$	$O(n)$
4	`j = i+1;`	$n-1$	$O(n)$
5	`while(j < n){`	$\frac{n(n-1)}{2}$	$O(n^2)$
6	`if(A[j] < A[s]){`	$\frac{n(n-1)}{2} - 1$	$O(n^2)$
7	`s = j;`	$x = if(A[j] < A[s])$ $\sum_{i=0}^{n-2} 1_x$ *best* 0	$O(1)$
		worst $\frac{n(n-1)}{2} - 1$	$O(n^2)$
	`}`	-----	-----
8	`j += 1;`	$\frac{n(n-1)}{2}$	$O(n^2)$
	`}`	-----	-----
9	`A[i] = A[s];`	$n-1$	$O(n)$
10	`i += 1;`	$n-1$	$O(n)$
	`}`	-----	-----
	`}`	-----	-----

Line 1: `int i = 0, j, s;`

- Count: 1
  - This line initializes three integer variables `i`, `j`, and `s` and is invoked only once during program execution; therefore, the count is 1.
- $O(\cdot)$: $O(1)$
  - The time complexity is constant, $O(1)$, because this line has a fixed execution time.

OUTER WHILE LOOP

Line 2: `while(i < n-1) {`

- Count: n

Outer While Loop Code Analysis

```
while(i < n-1){
    s = i;
    j = i+1;
    while(j < n){
        if(A[j] < A[s]){
            s = j;
        }
        j += 1;
    }
    A[i] = A[s];
    i += 1;
}
```

  - We know $i = 0$ *(line 1),* assume $n - 1 = 3$

Iteration	i	*Condition*		*Result*
1	$i = 0$	$0 < 3$	true	loop executes
2	$i = 1$	$1 < 3$	true	loop executes
3	$i = 2$	$2 < 3$	true	loop executes
4	$i = 3$	$3 < 3$	false	loop terminates

  - When $n - 1 = 3$, the while loop condition is executed for 4 iterations, or n. This is because the loop condition must perform an additional check for when the condition fails:
$$(n - 1) + 1 = n$$
However, the inside of the while loop is executed for 3 iterations, $n - 1$, because the loop body is only executed when the condition is true. The exact count value varies depending on the value of n.
- $O(\cdot)$: $O(n)$
  - The time complexity $O(n)$ is equivalent to the count of n, and represents the linear relationship between the input size and the execution time.

MIDDLE WHILE LOOP

Line 5: `while(j < n) {`

- Count: $\frac{n(n-1)}{2}$

Middle While Loop Code Analysis

```
while(j < n){
    if(A[j] < A[s]){
        s = j;
    }
    j += 1;
}
```

- We know $j = i + 1$ (*line 4*), assume $i = 0$ and $n = 3$

Iteration	*i*	*Condition*		*Result*
1	$j = 0 + 1 = 1$	$1 < 3$	true	loop executes
2	$j = 0 + 2 = 2$	$2 < 3$	true	loop executes
3	$j = 0 + 3 = 3$	$3 < 3$	false	loop terminates

- When $n = 3$ and $i = 0$, the middle while loop is executed for 3 iterations, $n - i$. This includes an additional loop check for when the condition fails. However, the inside of the while loop is executed for 2 iterations ($n - i - 1$) because the loop body is only executed when the condition is true. The exact count value varies depending on the value of n.
- Since this loop resides within the outer while loop, and i increments by 1, we can use the formula above to solve for sigma to find the count.

Outer While Loop Code Analysis

```
while(i < n-1){
    s = i;
    j = i+1;
    while(j < n){
        if(A[j] < A[s]){
            s = j;
        }
        j += 1;
    }
    A[i] = A[s];
    i += 1;
}
```

- For $i = 0$ to $n - 2$
  - $i = \{0, 1, 2, \dots, n-2\}$
  - We know that $i = 0$ from line 1, and the outer loop is executed as long as $i < n - 1$, or $i \leq n - 2$

- Table Q3-A: Outer Loop Breakdown

$\sum_{i=0}^{n-2} \boldsymbol{middleloop}$	*The outer loop is equal to the sum of the middle loops for $i = 0$ to $n - 2$.*
$\sum_{i=0}^{n-2} n - i$	*Since we already found the middle count formula, we can insert it directly into sigma*
$\sum_{i=0}^{n-2} n - i$ $\sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} i$	*Breakdown Sigma*

table Q1 continued below

- Observe first sigma $\sum_{i=0}^{n-2} n$
  - Since n is a constant, and in this summation i is the only thing incrementing, the value remains the same and repeats for the entire series.

i	*Value*
$i = 0$	n
$i = 1$	n
$i = 2$	n

  - Therefore, to find the count for this summation, we simply need to multiply the value by the size of the series

$value \times series\ size$	*Formula*
$(n) \times series\ size$	*Input known value*
$series\ size\ = (0, 1, 2, \dots, n-2)$ $series\ size\ = n - 1$	*Determine Series Size* *A series $\{1,2 \dots, n\}$ has n terms*
$(n) \times (n-1)$	*Input Series Size*
$n^2 - n$	*multiply*

- Observe Second sigma $\sum_{i=0}^{n-2} i$
  - For this sigma, the terms exhibit a consistent increment of 1, indicating an arithmetic pattern. As each term i is incremented by 1 compared to the previous term, we can intuitively recognize that this is an arithmetic series and is equivalent to the following formula:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

- Convert $\sum_{i=1}^{n} i$ *to* $\sum_{i=0}^{n-2} i$

$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$	*sigma formula*
$\sum_{i=0}^{n-2} i = \frac{n-1((n-1)+1)}{2}$	*since our series starts at 0 and ends at n-2, we need to replace n-1 for n in formula**
$\sum_{i=0}^{n-2} i = \frac{n-1(n)}{2}$ $\sum_{i=0}^{n-2} i = \frac{n(n-1)}{2}$	*simplify*

** A series $\{1,2 \dots, n\}$ has n terms, therefore, our series $\{0, 1,2 \dots, n-2\}$ has n-1 terms*

- Table Q3-B: Outer Loop Breakdown

table Q1 continued from above

$\sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} i$	*sigma notation*
$(n^2 - n) - \frac{n(n-1)}{2}$	*replace with formulas found above*
$\frac{2 \times (n^2 - n)}{2 \times 1} - \frac{n(n-1)}{2}$ $\frac{2n^2 - 2n}{2} - \frac{n^2 + n}{2}$ $\frac{2n^2 - 2n - n^2 + n}{2}$	*combine fractions*
$\frac{2n^2 - 2n - n^2 + n}{2}$ $\frac{2n^2 - n^2 - 2n + n}{2}$ $\frac{n^2 - n}{2}$	*subtract*
$\frac{n(n-1)}{2}$	*factor*

- $O(\cdot): O(n^2)$
  - The time complexity of $O(n^2)$ is equivalent to the count of $\frac{n(n-1)}{2}$. In Big O notation, the focus is on the dominant term as it has the greatest impact on the overall time complexity. In this case, the dominant term of $\frac{n(n-1)}{2} = \frac{n^2-n}{2}$ is n^2, and the $\frac{-n}{2}$ is less significant, especially as n^2 grows larger. Therefore, the time complexity is simplified to $O(n^2)$.

Line 8: and `j += 1;`

- Count: $\frac{n(n-1)}{2} - 1$
  - This line increments the value of the variable `j` by 1 and executes once for each iteration of the Middle While Loop.
  - Although the middle while condition (*line 5*) executes $\frac{n(n-1)}{2}$ times due an additional iteration when the loop condition fails, it is important to note that the statements *inside* the loop execute only when the condition passes. Therefore, the count for this line is $\left(\frac{n(n-1)}{2}\right) - 1$.
- $O(\cdot): O(n^2)$
  - The time complexity of $O(n^2)$ is equivalent to the count of $\left(\frac{n(n-1)}{2}\right) - 1$. In Big O notation, the focus is on the dominant term as it has the greatest impact on the overall time complexity. In this case, the dominant term of $\left(\frac{n(n-1)}{2}\right) - 1 = \frac{n^2+n-2}{2}$ is n^2, and the $\frac{n-2}{2}$ is less significant, especially as n^2 grows larger. Therefore, the time complexity is simplified to $O(n^2)$.

INNER IF STATEMENT

Line 6: `if(A[j] < A[s]){`

- Count: $\frac{n(n-1)}{2} - 1$
  - This line compares the elements `A[j]` and `A[s]` and executes once for each iteration of the Middle While Loop.
  - Although the middle while condition (*line 5*) executes $\frac{n(n-1)}{2}$ times due an additional iteration when the loop condition fails, it is important to note that the statements *inside* the loop execute only when the condition passes. Therefore, the count for this line is $\left(\frac{n(n-1)}{2}\right) - 1$.
- $O(\cdot): O(n^2)$
  - The time complexity of $O(n^2)$ is equivalent to the count of $\left(\frac{n(n-1)}{2}\right) - 1$. In Big O notation, the focus is on the dominant term as it has the greatest impact on the overall time complexity. In this case, the dominant term of $\left(\frac{n(n-1)}{2}\right) - 1 = \frac{n^2+n-2}{2}$ is n^2, and the $\frac{n-2}{2}$ is less significant, especially as n^2 grows larger. Therefore, the time complexity is simplified to $O(n^2)$.

Line 7: and `s = j;`

If Statement Analysis

```
if(A[j] < A[s]){
    s = j;
}
```

When analyzing the time complexity of lines inside an if statement, it's essential to consider both the best-case and worst-case scenarios

$x = if(A[j] < A[s]))$

$\sum_{i=0}^{n-2} \mathbf{1}_x$

	Count	$O(\cdot)$
best case	0	$O(1)$
worst case	$\frac{n(n-1)}{2} - 1$	$O(n^2)$

- *best case scenario:* `A[j]` is *always* greater than `A[s]`
  - Count: 0
    - In this case, the code inside the if statement will never execute.
  - $O(\cdot): O(1)$
    - The time complexity of the best-case scenario is $O(1)$ because it takes constant time regardless of the input size.
- *worst case scenario:* `A[j]` is *always* less than `A[s]`
  - Count: $\frac{n(n-1)}{2} - 1$
    - In this case, the condition evaluates to true for every iteration of the loop.
    - Therefore, the count of line 7 would be equal to the count of line 6, since the inside of the statement will execute every time if condition is checked, which is $\frac{n(n-1)}{2} - 1$.
  - $O(\cdot): O(n^2)$
    - The time complexity of $O(n^2)$ is equivalent to the count of $\left(\frac{n(n-1)}{2}\right) - 1$. In Big O notation, the focus is on the dominant term as it has the greatest impact on the overall time complexity. In this case, the dominant term of $\left(\frac{n(n-1)}{2}\right) - 1 = \frac{n^2+n-2}{2}$ is n^2, and the $\frac{n-2}{2}$ is less significant, especially as n^2 grows larger. Therefore, the time complexity is simplified to $O(n^2)$.
- *indicator function:* comparison result between `A[j]` and `A[j+1]`
  - Since the time complexity of line 7 can range from $O(1)$ in the best case to $O(n^2)$ in the worst case, depending on the input and the values in the array `A`, we will use the Indicator Function to convey this mathematically.

Note: Since our array is notated by a capital A, we will use x in the Indicator Function to avoid confusion.

$$\underset{\mathbf{1}_x}{x = if(A[j] < A[s])}$$

Where $\mathbf{1}_x$ evaluates to 1 when x is *true* and line 7 will be *executed*
Where $\mathbf{1}_x$ evaluates to 0 when x is *false* and line 7 will be *skipped*

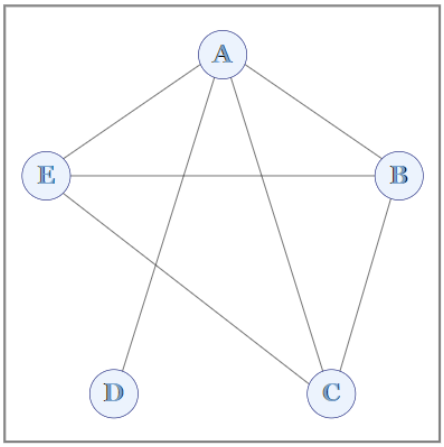
- Therefore, to find the count for line 7, we use sigma notation to find the summation of the Indicator Function. This allows us to express the count mathematically and account for the varying execution of line 7 based on the input and the values in array `A`.

$$\underset{\sum_{i=0}^{n-2} \mathbf{1}_x}{x = if(A[j] < A[s])}$$

  - $i = 0$
    - The lower bound of i represents the starting point of the summation.
    - By setting $i = 0$, we ensure that the summation begins at the first element of the array A.
    - Set by line 2 where $i = 0$
  - $n - 2$
    - The upper bound of i represents the ending point of the summation.
    - By setting the upper bound to $n - 2$, we ensure that the summation ends at the final element of the array `A`.
    - Set by line 2 where $i < n - 1$, or $i \leq n - 2$

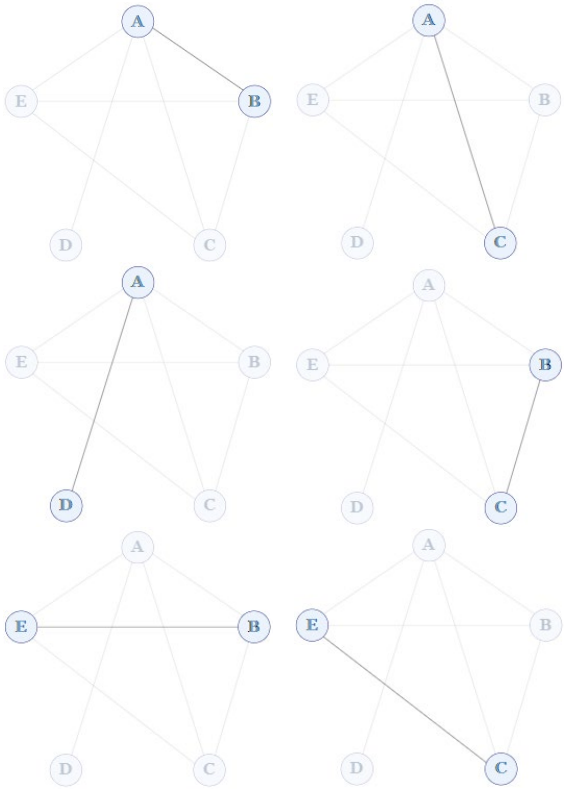
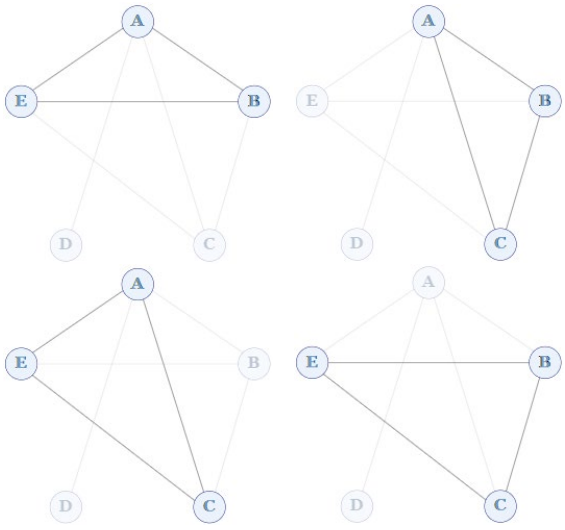
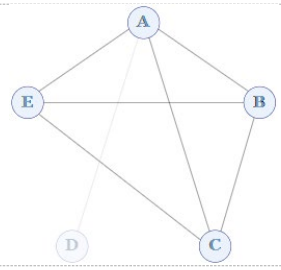
PROBLEM 4:

Identify all subgraphs that are Cliques or Independent Sets given the Graph Below. You can identify a subgraph by listing their vertices (e.g. ABD could be a potential clique or independent set).



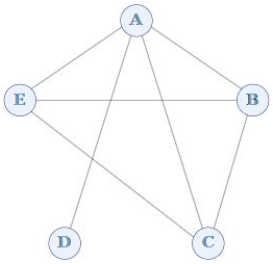
CLIQUES

Each vertex is directly connected to every other vertex, forming a fully connected induced subgraph.

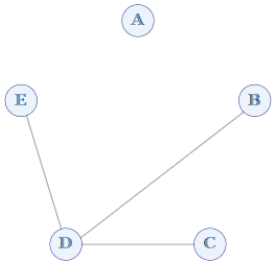
$\{A, B\}$, $\{A, C\}$, $\{A, D\}$, $\{B, C\}$, $\{B, E\}$, $\{C, E\}$,	
$\{A, B, E\}$, $\{A, B, C\}$, $\{A, C, E\}$, $\{B, C, E\}$	
$\{A, B, C, E\}$	

INDEPENDENT SETS

An induced subgraph with no edges, obtained by complementing a clique



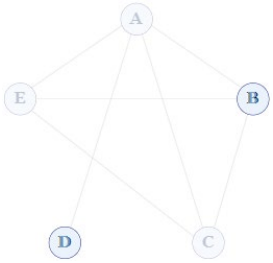
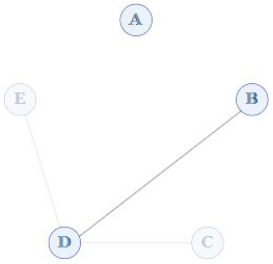
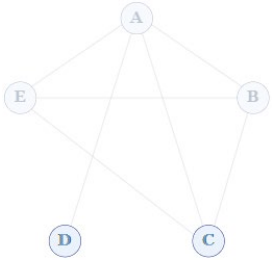
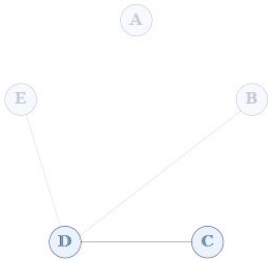
$G(V, E)$



$\bar{G}(V, \bar{E})$

To verify the independence of a set in graph $G(V, E)$

check if it forms a clique in the complement graph $\bar{G}(V, \bar{E})$.

	$G(V, E)$	$\bar{G}(V, \bar{E})$
$\{B, D\}$		
$\{C, D\}$		
$\{D, E\}$