1)

1)

a) $\log_2 n^2 + 1 = O(n)$ $g(n) \leq c f(n)$

$2\log_2 n + 1 \leq c.n$

$\frac{2\log_2 n}{2} \leq \frac{cn-1}{2}$

$\log_2 n \leq \frac{cn-1}{2}$

Because any function that grows faster than the logarithm

So, it is true.

b) $\sqrt{n(n+1)} = \Omega(n)$

$f(n) \geq c.g(n)$ for all $n \geq k$

where $c > 0$ and $k > 0$

$\lim_{n\to\infty} \frac{\sqrt{n(n+1)}}{n} \Rightarrow \lim_{n\to\infty} \frac{n(n+1)}{n} \Rightarrow \underbrace{\lim_{n\to\infty} \frac{n}{n}}_{1} + \underbrace{\lim_{n\to\infty} \frac{1}{n}}_{0} = 1$

Rule: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L$ if $L = \infty$ then $f(n) \in \Omega(g(n))$

So, it is false.

c) $n^{n-1} = \Theta(n^n)$

$c_1.g(n) \leq f(n) \leq c_2.g(n)$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} \Rightarrow \lim_{n\to\infty} \frac{n^{n-1}}{n^n} = \lim_{n\to\infty} \frac{1}{n} \rightarrow 0$

Rule: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L$ if $L = C$ $f(n) \in \Theta(n)$

So, it is false

2)

2) $n^2, n^3, n^2\log, \sqrt{n}, \log n, 10^n, 2^n, 8^{\log_2 n}$

- $\lim_{n\to\infty} \frac{10^n}{2^n} = \lim_{n\to\infty} \left(\frac{10}{2}\right)^n \Rightarrow \infty$ $\quad \underline{10^n > 2^n}$

Hospital

- $\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{2\sqrt{n}}{n} \Rightarrow \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$ $\quad \underline{\sqrt{n} > \log n}$

- $\lim_{n\to\infty} \frac{n^2}{n^3} \Rightarrow \lim_{n\to\infty} \frac{1}{n} = 0$ $\quad \underline{n^3 > n^2}$

- $\lim_{n\to\infty} \overset{\text{Le Hospital}}{\frac{n^3}{n^2\log n}} = \lim_{n\to\infty} \frac{n}{\log n} \Rightarrow \lim_{n\to\infty} \frac{1}{\frac{1}{n}} = \lim_{n\to\infty} n = \infty$ $\quad \underline{n^3 > n^2\log n}$

- $\lim_{n\to\infty} \frac{10^n}{n^3} \overset{\text{Le Hospital}}{=} \lim_{n\to\infty} \frac{\ln 10 \cdot 10^n}{3n^2} = \lim_{n\to\infty} \frac{\ln^2 10 \cdot 10^n}{6n} = \lim_{n\to\infty} \frac{\ln^3 10 \cdot \overset{5^n \cdot 2^n}{10^n}}{\frac{6}{3.2}} \Rightarrow \lim_{n\to\infty} \ln^3 10 \cdot 5^n \cdot 2^{n-1} = \infty$ $\quad \underline{10^n > n^3}$

- $\lim_{n\to\infty} \frac{2^n}{n^2} \overset{\text{Le Hospital}}{=} \lim_{n\to\infty} \frac{2^n \ln 2}{2n} = \lim_{n\to\infty} \frac{\ln^2 2 \cdot 2^n}{2} = \infty$ $\quad \underline{2^n > n^2}$

- $\lim_{n\to\infty} \frac{n^3}{\log n} = \lim_{n\to\infty} \frac{3n^2}{\frac{1}{n}} = \lim_{n\to\infty} 3n^3 = \infty$ $\quad \underline{n^3 > \log n}$

- $\lim_{n\to\infty} \frac{2^n}{n^3} = \lim_{n\to\infty} \frac{2^n \ln 2}{3n^2} = \lim_{n\to\infty} \frac{\ln^2 2 \cdot 2^n}{6n} = \lim_{n\to\infty} \frac{\lim^3 2 \cdot 2^n}{6} \Rightarrow \lim_{n\to\infty} \frac{\lim^3 2 \cdot 2^{n-1}}{3} = \infty$ $\quad \underline{2^n > n^3}$

- $\lim_{n\to\infty} \frac{n^2\log n}{\log n} = \lim_{n\to\infty} n^2 = \infty$ $\quad \underline{n^2\log n > \log n}$

- $\lim_{n\to\infty} \frac{\sqrt{n}}{n^2\log n} = \frac{1}{n^3\ln n} = 0$ $\quad \underline{n^2\log n > \sqrt{n}}$

- $\underline{n^3 = 8^{\log_2 n}}$

$$10^n > 2^n > n^3 = 8^{\log_2 n} > n^2\log n > n^2 > \sqrt{n} > \log n$$

3)

a)

```
int p_1 ( int my_array[]){
        for(int i=2; i<=n; i++){
                if(i%2==0){ O(1)
                        count++; O(1)
                } else{ → O(1)
                        i=(i-1)i; O(1)
                }
        }
}
```

O(n)

Time Complexity : O(n)

Space Complexity:O(1)

b)

```
int p_2 (int my_array[]){
        first_element = my_array[0]; O(1)
        second_element = my_array[0]; O(1)
        for(int i=0; i<sizeofArray; i++){
                if(my_array[i]<first_element){ O(1)
                        second_element=first_element; O(1)
                        first_element=my_array[i]; O(1)
                }else if(my_array[i]<second_element){ O(1)
                        if(my_array[i]!= first_element){ O(1)
                                second_element= my_array[i]; O(1)
                        }
                }
}
```

O(n)

Time Complexity : O(n)

Space Complexity:O(1)

c)

```
int p_3 (int array[]) {
        return array[0] * array[2]; → O(1)
}
```

Time Complexity : O(1)

Space Complexity:O(1)

d)

```
int p_4(int array[], int n) {
        Int sum = 0   O(1)
        for (int i = 0; i < n; i=i+5)
                sum += array[i] * array[i];   O(1)   O(n)
         return sum;   O(1)
}
```

Time Complexity : O(n)

Space Complexity:O(1)

e)

```
void p_5 (int array[], int n){
        for (int i = 0; i < n; i++)
                for (int j = 1; j < i; j=j*2)
                        printf("%d", array[i] * array[j]);   O(1)   O(logn)   O(n)
}
```

Time Complexity : O(nlogn)

Space Complexity:O(1)

g)

```
int  p_7( int n ){
        int i = n;   → O(1)
        while (i > 0) {
                for (int j = 0; j < n; j++)
                        System.out.println("*");   O(n)   O(logn)
                                  O(1)
                 i = i / 2;
         }
}
```

Time Complexity : O(nlogn)

Space Complexity:O(1)

h)

```
int  p_8( int n ){
        while (n > 0) {
                for (int j = 0; j < n; j++)
                        System.out.println("*");
                n = n / 2;
        }
}
```

O(1) O(n) O(logn)

Time Complexity : O(nlogn)

Space Complexity:O(1)

i)

```
int p_9(n){
        if (n = 0)
                return 1
        else
                return n * p_9(n-1)
}
```

O(n)

Time Complexity : O(n)

Space Complexity:O(1)

j)

```
int p_10 (int A[ ], int n) {
        if (n == 1)
                return;
        p_10 (A, n − 1);
        j = n − 1;
        while (j > 0 and A[j] < A[j − 1]) {
                SWAP(A[j], A[j − 1]);
                j = j − 1;
        }
}
```

O(1

→ O(n)

Time Complexity : O(n)

Space Complexity:O(1)

4)

4)

a) Because O notation provides an upper bound not a lower bound. Saying the running time of algorithm A is at most $O(n^2)$ would be correct. So it is meaningles to say 'at least'.

b)

I. $\lim_{n \to \infty} \frac{2^{n+1}}{2^n} \Rightarrow \frac{2^n . 2}{2^n} \Rightarrow \lim_{n \to \infty} 2 = 2$

Rule: $\lim_{N \to \infty} \frac{f(N)}{g(N)} \rightarrow f(N) = \Theta(g(N)) = c != 0 \Rightarrow f(N) = \Theta(g(N))$

So, $2^{n+1} = \Theta(2^n)$ is true.

II. $2^{2n} = \Theta(2^n)$

$\lim_{n \to \infty} \frac{2^{2n}}{2^n} = \lim_{n \to \infty} 2^n = \infty$

Rule = $\lim_{N \to \infty} \frac{f(N)}{g(N)} = \infty \rightarrow g(N) = o(f(N))$

So, $2^{2n} = \Theta(2^n)$ is false.

III. Let $f(n) = O(n^2)$ and $g(n) = \Theta(n^2)$. Prove or disprove that: $f(n) * g(n) = \Theta(n^4)$

If $f(n) = O(n^2)$, $f(n)$ can be $\Theta(n^2)$, $\Theta(n)$, or $\Theta(1)$

Therefore, $f(n) * g(n)$ can be $\Theta(n^4)$, $\Theta(n^3)$ or $\Theta(n^2)$

As a result, $f(n) * g(n)$ can be $\Theta(n^4)$ but it is not certain.

5)

a)

$$T(n) = 2\boxed{T(n/2)} + n \xrightarrow{T(n)} T(n/2) = 2T(n/2^2) + \frac{n}{2}$$

$$T(n) = 2\left(2\boxed{T(n/4)} + \frac{n}{2}\right) + n \longrightarrow \underbrace{T(n/4) = 2T(n/2^3) + \frac{n}{4}}$$

$$T(n) = 2^2\,T(n/4) + n + n$$

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

$$T(n) = 2^3\,(T(n/8)) + 3n$$

$$T(n) = 2^k\,(T(n/2^k)) + kn$$

$$T(n/2^k) = T(1)$$

$$n/2^k = 1$$

$$n = 2^k$$

$$k = \log n$$

$$T(n) = n \times 1 + n\log n$$

$$T(n) = O(\log n)$$

b) $T(n) = 2T(n-1) + 1$, $T(0) = 0$

$$T(n) = 2(2T(n-2) + 1] + 1$$

$$T(n) = 2^2\,T(n-2) + 2 + 1$$

$$T(n) = 2^2(2T(n-3) + 1] + 3 \Rightarrow 2^3(T(n-3)) + 7$$

$$\Rightarrow 2^k\,T(n-k) + 2^{k-1} + 2^{k-2} + 2^{k-3} + 2^2 + 2^1 + 2^0$$

$$T(n-k) = T(0) = 0$$

$$n - k = 0$$

$$n = k$$

$$T(n) = 2^n\,T(0) + 2^k - 1$$

$$= 2^n \times 0 + 2^k - 1$$

$$2^k - 1 = 2^n - 1$$

$$T(n) = O(2^n)$$

6)

```java
public static void findPair(int[] numbers, int givenSum)
{
    for (int i = 0; i < numbers.length - 1; i++)
    {
        for (int j = i + 1; j < numbers.length; j++)
        {
            if (numbers[i] + numbers[j] == givenSum)
            {
                System.out.println("Pair  : (" + numbers[i] + "," + numbers[j] + ")");
                return;
            }
        }
    }
}

public static void main (String[] args)
{
    long startTime = System.nanoTime();
    int[] numbers = { 1, 2, 3, 4, 5, 6};
    int givenSum = 11;

    findPair(numbers, givenSum);
    long endTime   = System.nanoTime();
    long totalTime = endTime - startTime;
    System.out.println("Running Time   : "+totalTime);

}
```

Time Complexity : O(n^2)

```
Pair  : (2,6)
Pair  : (3,5)
Running Time    : 423213
```

```
Pair  : (5,6)
Running Time    : 1113151
```

7)

```java
public static void main (String[] args)
{
    long startTime = System.nanoTime();
    int[] numbers = { 1, 2, 3, 4, 5, 6};
    int givenSum = 11;

    findPair(numbers, givenSum,0,1);
    long endTime   = System.nanoTime();
    long totalTime = endTime - startTime;
    System.out.println("Running Time   : "+totalTime);

}

private static void findPair(int[] numbers,int expectedSum,int firstIndex,int nextIndex) {

    if (firstIndex >= numbers.length - 1) {
        return;
    }
    if (nextIndex >= numbers.length) {
        findPair(numbers, expectedSum, firstIndex + 1, firstIndex + 2);
        return;
    }

    if (numbers[firstIndex] + numbers[nextIndex] == expectedSum) {
        System.out.println("Pair  : (" + numbers[firstIndex] + "," + numbers[nextIndex] + ")");
    }
    findPair(numbers, expectedSum, firstIndex, nextIndex + 1);
}
```

Time Complexity : O(n)

```
Pair  : (5,6)
Running Time   : 393220
```

```
Pair  : (2,6)
Pair  : (3,5)
Running Time   : 444368
```