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

$f(n) = O(g(n))$ positive constent c and n_0

$0 \leq f(n) \leq c.g(n)$ for all $n \geq n_0$

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

$= \underbrace{2\log_2 n}_{\text{constant}} + \underbrace{1}_{\text{Cost}} \leq cn$

$0 \leq \log_2 n \leq c.n$ for all $n \geq n_0$, There exists positive constents, c and n_0, st: $c \geq 1$, $n_0 \geq 1$ $\Rightarrow$ $\log_2 1 \leq 1.1$ **True**

$\underline{n=1}$ $0 \leq 1$ ✓

---

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

$0 \leq c.g(n) \leq f(n)$ for all $n \geq n_0$

$c.n \leq \underbrace{\sqrt{n(n+1)}}_{\text{that gives } n+x}$

$c.n \leq n+x$ = so it sees for all n values we can find c which make correct that statement, so this is true.

$n_0=2$, $c=1$ $1 \times 2 \leq 2,4$ = $2 \leq 2,4$ ✓ **True**

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

$0 \leq c_1.g(n) \leq f(n) \leq c_2.g(n)$ for all $n \geq n_0$

$c_1.n^n \leq n^{n-1} \leq c_2.n^n$

$c_1.n^n \leq n^{n-1}$ $n^{n-1} \leq c_2.n^{n-1}.n$

$c_1.\cancel{n^{n-1}}.n \leq \cancel{n^{n-1}}$ $\cancel{n^{n-1}} \leq c_2.\cancel{n^{n-1}}.n$

$c_1.n \leq 1$ X $1 \leq c_2.n$ ✓

↳ for that part, we can say that is false statement

$n \geq n_0$ && $c_1, c_2, n_0 \geq 0$

$c_1.n \leq 1$

if $n=2$

$c.2 \leq 1$

c has to be positive, so there is no c value, that provide statement is correct. So that is false

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

$n^3 = 8^{\log_2 n} = O(n^{\log_2 8}) = O(\underline{n^3})$

$\lim_{n\to\infty} \frac{n^3}{n^2} = n = \infty \quad O(n^3) = O(8^{\log_2 n}) > O(n^2)$

$\lim_{n\to\infty} \frac{n^2 \lg n}{n^3} = \frac{\lg n}{n} \cong 0 \quad O(n^3) > O(n^2 \lg n)$

$\lim_{n\to\infty} \frac{10^n}{2^n} = \frac{2^n \cdot 5^n}{2^n} = 5^n \to \infty = O(10^n) > O(2^n)$

$\lim_{n\to\infty} \frac{n^3}{2^n} \cong 0 \quad O(2^n) > O(n^3)$

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

$O(10^n) > O(2^n) > O(n^3) = O(8^{\log_2 n}) > O(n^2 \lg n) > O(n^2) > O(\sqrt{n}) > O(\lg n)$

$T(1) = 1$

CamScanner ile tarandı

3) What is the time complexity of the following programs? Use most appropriate asymptotic notation. Explain by giving details.

a)

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

$i^2 - i$

if i is even = increase 1

if odd = $i^2 - i$

$i_{n+1} = i_n(i_n - 1)$

$i_n^2 - i_n$ → this bigger than that

i^{2^k} → k times run → it increases until the value n

$x^{2^k} = n$ find k

$\log x^{2^k}$ find k take log

$k = \log(\log n) \Rightarrow \theta(\log(\log n))$

i
2
3
6
7
42
⋮

b)

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

size of array = n

n times

→ 1 compare

→ 1 compare

n

$\theta(n)$

CamScanner ile tarandı

c)

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

Just multiplication.

$\Theta(1)$

d)

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

$\frac{n}{5}$ times $\Theta\left(\frac{n}{5}\right) = \Theta(n)$

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]);
}
```

$i/2$ $n \Rightarrow n * \lg n$

$= \Theta(n.\lg n)$

f)

```
int p_6(int array[], int n) {
        If (p_4(array, n)) > 1000)
                p_5(array, n)
        else printf("%d", p_3(array) * p_4(array, n))
}
```

If case $= \underbrace{\Theta(n)}_{p-4} + \underbrace{\Theta(1)}_{\text{comparison}} + \Theta(n.\lg n) = \Theta(n.\lg n)$

Else $= \underbrace{\Theta(n)}_{p-4} + \Theta(1) + \underbrace{\Theta(1)}_{p-3} + \underbrace{\Theta(n)}_{p-4} + \underbrace{\Theta(1)}_{\text{multiplication}} = \Theta(2n) = \Theta(n)$

Best case $= \Theta(n)$

Worst case $= \Theta(n.\lg n)$

$T6(n) = O(T4(n) + O(1) + T5(n)), (T3(n) + T4(n) + O(1))$

$T6(n) = O((\Theta(n) + O(1) + \Theta(n.\lg n)), (\Theta(1) + \Theta(n) + O(1)))$

g)

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

$n, \frac{n}{2}, \frac{n}{4} \dots \frac{n}{2^k} = 1$ $n = 2^k$ $k = \lg n$

$\rightarrow O(1)$

n times

$\lg n = \Theta(n.\lg n)$

h)

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

$n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8} \dots \frac{n}{2^k} = 1$ $n = 2^k$, $k = \lg_2 n = \Theta(\lg n)$

this loop runs $n, \frac{n}{2}, \frac{n}{4} \dots \frac{n}{2^k}$

$\left(n + \frac{n}{2} + \frac{n}{4} \quad \frac{n}{2^k}\right)$

$\lg n = k$

$n = 2^k$

$2^0 + 2^1 + 2^2 \quad 2^k = 2^{k+1} - 1$

$2^{\lg n + 1} - 1 \cong n = \Theta(n)$

If $n=0$ then return 1 → $O(1)$ times

$T(0)=1$

If $n \neq 0$ then $T(n) = T(n-1)$ + multiplication gives constant time

$$T(n) = T(n-1) + O(1)$$
$$T(n-1) = T(n-2) + O(1)$$
$$T(n-2) = T(n-3) + O(1)$$

k times

$$T(n) = T(n-k) + k.O(1)$$

$n-k=0$

$n=k$

$$T(n) = T(0) + n \cdot O(1)$$
$$T(n) = O(1) + \theta(n) = \theta(n)$$

i)

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

$T(n)$ = # of swap operation

$$T(1) = 0$$
$$T(n) = T(n-1) + n$$
$$T(n) = T(n-2) + n - 1 + n$$
$$T(n) = T(n-3) + n-2 + n-1 + n$$
$$T(n) = T(n-k) + k.n - \frac{(k-1)k}{2}$$

$n-k=1$, $n-1=k$

$$T(n) = \underline{T(1)} + (n-1).n - \frac{(n-2).(n-1)}{2}$$
$$T(n) = 0 + n^2 - \frac{n^2-3n+2}{2} = \theta(n^2)$$

4)

4) a) O(f) means presents upper bound, It means the algorithm can be at most $O(n^2)$. That's why we can't say <u>at least</u> n^2.

4) b) -1)

$2^{n+1} = \Theta(2^n)$

$c_1 2^n \leq 2^{n+1} \leq c_2 . 2^n \longrightarrow$

$c_1 . \cancel{2^n} \leq \cancel{2^n} . 2$

$c_1 \leq 2$ ✓

$2^{n+1} = \Theta(2^n)$

$\cancel{2^n} . 2 \leq c_2 \cancel{2^n}$

$2 \leq c_2$ ✓

$2^{n+1} = \Theta(2^n)$

4) b) -2)

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

$c_1 . 2^n \leq 2^{2n} \leq c_2 . 2^n$

$c_1 . \cancel{2^n} \leq \cancel{2^n} . 2^n$

$n \geq n_0$

there should be c ✓

$\longrightarrow 2^{2n} \leq c_2 . 2^n$

$2^n . 2^n \leq c_2 . \cancel{2^n}$

2^n increases faster then c_2 no ✗

$2^{2n} \neq \Theta(2^n)$

4) b) -3)

$f(n) = O(n^2)$ $\quad g(n) = \Theta(n^2)$ $\quad = f(n) * g(n) = \Theta(n^4)$

$f(n) =$ is less then or equal to n^2

so $f(n)$ could be $\Theta(n)$ $\Rightarrow$ $g(n) = \Theta(n^2)$ $= f(n) * g(n) = \Theta(n) * \Theta(n^2) = \underline{\Theta(n^3)}$

so the statement not true

5) a) $T(n) = 2T\left(\frac{n}{2}\right) + n$ $\quad T(1)=1$

$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2} \Rightarrow T(n) = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$

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

$n = 2^k = k = \log n$

$T(n) = 2^{\log n} T\left(\frac{n}{2^{\log n}}\right) + \log n.n$

$T(n) = n \cdot T\underbrace{(1)}_{1} + \log n.n$

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

$T(n) = \theta(n.\log n)$

5) b) $T(n) = 2T(n-1) + 1$ $\quad T(0)=0$

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

$T(n-1) = 2T(n-2) + 1 \Rightarrow 2(2T(n-3)+1)+1 = 4T(n-3)+3 \Rightarrow = 2(4T(n-3)+3)+1 = 8T(n-3)+7$

$T(n-2) = 2T(n-3)+1$ $\quad T(n-3) = 2T(n-4)+1 \Rightarrow T(n-2) = 2(2T(n-4)+1)+1$

$T(n-2) = 4T(n-4)+3$, $T(n-1) = 2(4T(n-4+3)+1 = T(n-1) = 8T(n-4)+7$

$T(n) = 2(8T(n-4)+7)+1 \Rightarrow \boxed{16T(n-4)+15}$

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

$n-k=0$

$k=n$

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

6) void iterative(int array[ ], int given Sum){

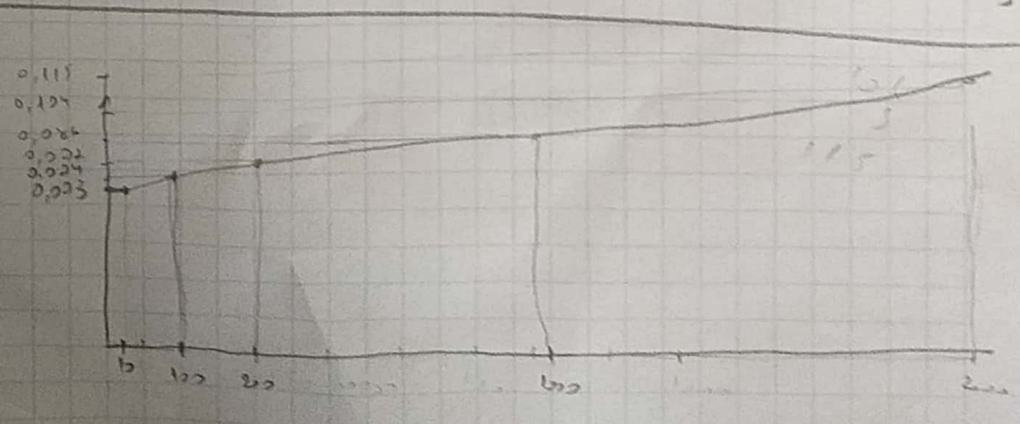
```
for (int i=0 < ; i< array.length ; i++){
    for( int j=i+1 , j< array.length ; j++){
        if ((array[i] + array[j]) = given Sum){


        }
    }
}
}
```

It check every 2 pairs whether they are equal or not the given Sum. For that it use 2 loop which size's equal n and n-1. Therefore, time complexity will be $\theta(n^2)$

input size	time1	2	3	4	5	6	7	8	9	10
10	0,07	0,09	0,06	0,07	0,06	0,07	0,10	0,09	0,06	0,06
20	0,09	0,07	0,09	0,07	0,07	0,05	0,06	0,07	0,08	0,09
50	0,07	0,10	0,07	0,06	0,06	0,07	0,08	0,06	0,07	0,09
100	0,08	0,06	0,09	0,09	0,06	0,08	0,06	0,08	0,06	0,08
200	0,10	0,10	0,06	0,05	0,07	0,06	0,11	0,07	0,06	0,09
1000	0,09	0,08	0,09	0,07	0,07	0,06	0,11	0,10	0,11	0,08
2000	0,09	0,13	0,12	0,07	0,11	0,10	0,13	0,10	0,09	0,10
10000	0,14	0,11	0,15	0,11	0,10	0,10	0,10	0,12	0,09	0,13

input size	avarage time
10	0,073
20	0,074
50	0,073
100	0,074
200	0,077
1000	0,086
2000	0,104
10000	0,115



7)

```
findPairRec ( int array , int givenSum , int n ) {

    int i = array length - n
    if ( n = 0 ) {
        return
    }
    else {
        for ( int j = i+1 ; j < array.length ; ++j )      n times
            if ( array [ i ] + array [ j ] = = sum )
                println ( array [ i ] , array [ j ] );
        }
        findPairRec ( array , sum , n-1 );                T(n-1)
    }
}
```

$T(n) = n + T(n-1)$, $\quad T(0) = 0$

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

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

$T(n) = n + (n-1) + \dots (n-(k-1)) + T(n-k)$

$\underline{k = n}$

$T(n) = n + (n-1) + \dots 2 + 1 + T(0)$

$T(n) = \frac{n(n+1)}{2} = \frac{n^2+n}{2} = T(n) = \underline{\underline{\Theta(n^2)}}$