Q1.

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

statement is true but not useful

$logn < n$ it gives less information.

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

$\sqrt{n^2 + n} = \Omega(n) = n\sqrt{n}$

because of this statement is true and useful.

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

$c_1 * n^n \leq n^{n-1} \leq c_2 * n^n$ this statement is true

Q2.

$if \lim_{x \to \infty} \frac{f(x)}{g(x)} = \infty \quad it\ means\ f(x) > g(x)$

$if \lim_{x \to \infty} \frac{f(x)}{g(x)} = 0 \quad it\ means\ f(x) < g(x)$

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

Q3.

a) $\Theta(n)$

because for loop runs n times and the inside of the for loop both statements runs 1 time as a result 1*n = n ⇒ time complexity is $\Theta(n)$

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];
                        }
                }
}
```

$\theta(1)$

$\theta(1)$

$\theta(1)$

for loop $\theta(n)$

n+1

$O(n)$

c)

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

$\Theta(1)$

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;
}
```

Int sum = 0: $\theta(1)$

sum += array[i] * array[i]: $\theta(1)$ } for loop $\frac{n}{5}$ $\theta(n)$

} $\frac{n}{5}+1$

$\theta(n)$

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]);
}
```

for (int i = 0; i < n; i++): n

for (int j = 1; j < i; j=j*2): $\log n$

printf: $\theta(1)$ } $\log n \cdot 1$ } $\log n \cdot n$

$\Theta(n \log n)$

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 (p_4(array, n)) > 1000): $\Theta(n)$

p_5(array, n): $\theta(n \log n)$

p_3(array): $\theta(1)$, p_4(array, n): $\theta(n)$ → $\theta(n)$

$T(b) = \theta(n)$

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

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

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;
        }
}
```

int i = n: $\theta(1)$

while (i > 0): $\theta(\log n)$

for (int j = 0; j < n; j++): $\theta(n)$

System.out.println("*"): $\theta(1)$ } $\theta(n)$

i = i / 2: $\theta(1)$

} $\theta(n \log n)$

h)

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

while (n > 0): $\theta(\log n)$

for (int j = 0; j < n; j++): $\theta(n)$

System.out.println("*"): $\theta(1)$

n = n / 2: $\theta(1)$

} $\theta(n \log n)$

i)

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

$\theta(1)$

$\theta(n)$

$1 * n = \theta(n)$

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;
     }
}
```

$\theta(1)$

$\theta(n)$

$\theta(1)$

$\theta(1)$

because of func calling itself n times

$O(n^2)$

Q4.

a) "The running time of algorithm A is at least $O(n^2)$ "

<u>this is false</u> because Big-Oh notation shows the upper bound because of this instead of 'at least' there should be used 'at most'.

b)

I. $2^{n+1} = \Theta(2^n)$ this statement equals $2^n * 2$ because of this <u>this is true</u>

II. $2^{2n} = \Theta(2^n)$ this statement equals $2^{2^n} = 4^n$ because of this time complexity equals $\Theta(4^n)$ <u>this is false</u>

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 the running time of algorithm f(n) is at most $O(n^2)$ product of the f(n) and the g(n) equals $\Theta(n^4)$.

Q5.

a) $T(n) = 2T(n/2) + n,\ T(1) = 1$

$$T(n) = 2T(n/2) + n$$
$$T(n) = 2(2T(n/4) + n/2) + n$$

$$T(n) = 4T(n/4) + n/2 + n$$
$$T(n) = 4(2T(n/8) + n/4) + n/2 + n$$
$$T(n) = 8T(n/8) + n/4 + n/2 + n$$
$$T(n) = 2^k(T(n/2^k) - n(1/2^{k-1} + 1/2^{k-2} + 1/2^{k-3} \ldots + 1))$$

assume $n/2^k = 1$

$$T(n) = 2^k(1 + n(1 + 1))$$
$$T(n) = 2^k(1 + 2n)$$
$$T(n) = \Theta(n)$$

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

$$T(n) = 2T(n-1) + 1$$
$$T(n) = 2(2T(n-2) + 1) + 1$$
$$T(n) = 4T(n-2) + 3$$
$$T(n) = 4(2T(n-3) + 1) + 3$$
$$T(n) = 8T(n-3) + 4$$
$$T(n) = k^2T(n-k) + (k+1)$$

assume n-k = 0   n=k

$$T(n) = n^2T(0) + (n+1)$$
$$T(n) = n^2 * 0 + n + 1$$
$$T(n) = n + 1$$
$$T(n) = \Theta(n)$$

Q6.

```
for (int i=0;i<arr.length;i++){
    for (int j =i;j< arr.length;j++){

        if ((arr[i]+arr[j]) == x)
            System.out.println("the pair is " + arr[i] + " and " + arr[j]);
    }
}
```

$\Theta(n)$ $\Theta(n)$

teorical $\frac{\Theta(n^2)}{2}$

experimental

```
array 1 is  1 2 3 4 5
the pair is 1 and 4
the pair is 2 and 3
Elapsed Time in nano seconds: 56548
array 2 is  1 2 3 4 5 6 7
the pair is 1 and 6
the pair is 2 and 5
the pair is 3 and 4
Elapsed Time in nano seconds: 72910
array 3 is  1 2 3 4 5 6 7 8 9
the pair is 1 and 8
the pair is 2 and 7
the pair is 3 and 6
the pair is 4 and 5
Elapsed Time in nano seconds: 100443
```

T(5)<T(7)<T(9)

As the number of elements increases, so did the run time spent.

Q7.

```
public static void pairfinderrecurdsive(int array[], int initialindex, int nextindex, int sum) {
    int n=array.length; θ(1)
    if (initialindex >=  n - 1)
        return;

    if (nextindex >= n ) {
        pairfinderrecurdsive(array, initialindex: initialindex + 1, nextindex: initialindex + 2, sum); T(n)
        return;
    }

    if (array[initialindex] + array[nextindex] == sum)
        System.out.println("the pair is " + array[initialindex] + " and " + array[nextindex]); θ(1)

    pairfinderrecurdsive(array, initialindex, nextindex: nextindex + 1 , sum); T(n)
}
```

array gets smaller for every turn

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

experimental

```
array 1 is  1 2 3 4 5
the pair is 1 and 4
the pair is 2 and 3
Elapsed Time in nano seconds: 38593
array 2 is  1 2 3 4 5 6 7
the pair is 1 and 6
the pair is 2 and 5
the pair is 3 and 4
Elapsed Time in nano seconds: 57579
array 3 is  1 2 3 4 5 6 7 8 9
the pair is 1 and 8
the pair is 2 and 7
the pair is 3 and 6
the pair is 4 and 5
Elapsed Time in nano seconds: 80001
```