Q1)

Q1)

a) $\log_2 n^2 + 1 \stackrel{?}{=} O(n)$ (TRUE)

$f(n) \leq c.g(n)$

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

$2\log_2 n + 1 \leq cn$ it must be true while c and no are positives $n \geq no$

for $c=2$;

$1 \leq \underbrace{2n - 2\log_2 n}$

for all positive n values n will allways bigger than $\log_2 n$

so This is True

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

$c.g(n) \leq f(n)$

$c.n \leq \sqrt{n.(n+1)}$

$$\frac{c^2 n^2}{n} \leq \frac{n^2+n}{n}$$

$c^2 n \leq n+1 \Rightarrow \underbrace{n(c^2-1) \leq 1}_{\text{not possible}} \Rightarrow$ False

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

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

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

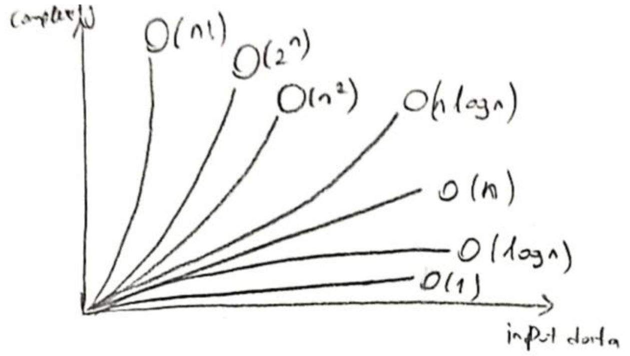
$$\boxed{c_1 n \leq \boxed{1}} \boxed{\leq c_2 n}$$

This side is true

This side is false so it is False

Q2)

Q2)



According to the graph we can group the given function like that:

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

- $\lim_{n\to\infty} \frac{2^n}{10^n}$

$= \lim_{n\to\infty} \frac{\cancel{2^n}}{\cancel{2^n}\cdot 5^n} = \frac{1}{\infty}$

$= 0$

This means that

$\boxed{O(10^n) > O(2^n)}$

- $\lim_{n\to\infty} \frac{n^2}{n^3}$

$= \lim_{n\to\infty} \frac{1}{n} = 0$

$O(n^3) > O(n^2)$

- $\lim_{n\to\infty} \frac{\sqrt{n}}{n^2}$

$= \lim_{n\to\infty} \frac{1}{n\sqrt{n}} = 0$

$O(n^2) > O(\sqrt{n})$

- $8^{\log_2 n} = n^{\log_2 2^3}$

$= n^3$

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

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

- $\lim_{n\to\infty} \frac{n^2 \cancel{\log n}}{\cancel{\log n}}$

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

This means that:

$O(n^2 \log n) > O(\log n)$

- $\lim_{n\to\infty} \frac{\sqrt{n}}{n^2 \log n} = 0$

$= O(n\log n) > O(\sqrt{n})$

- $\lim_{n\to\infty} \frac{n}{\log n} = \infty$

$O(\sqrt{n}) > O(\log n)$

final:

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

Q3)

a)

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

$\rightarrow \Theta(\log n)$

$\Theta(1)$

$\rightarrow \Theta(1)$

$= \Theta(\log n)$

✓

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];
                        }
                }
}
```

c. time

$\rightarrow \Theta(n)$

constant time

c)

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

$\Rightarrow T_b(N) = T_w(N) = \Theta(1)$

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;
}
```

$\rightarrow$ c time

$\Theta(n)$

$\rightarrow$ c time

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]);
```

$\rightarrow \Theta(n)$

iter1: $j = 1 = 2^0$

iter2: $j = 2 = 2^1$

⋮

iter P: $j = P = 2^{P-1} = n$

$P = \log_2 n + 1$

$= \Theta(\log n)$

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

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))
}
```

$\rightarrow \Theta(n)$

$\rightarrow \Theta(n \log n)$

$\rightarrow \Theta(n)$

$T_b(n) = \Theta(n)$

$T_w(n) = \Theta(n \log n)$

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

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;
        }
}
```

$\Theta(n)$

1. $i = i$
2. $i = i \cdot 2^{-1}$
3. $i = i \cdot 2^{-2}$

$\vdots$

$n \quad i = i \cdot 2^{-(p-1)}$

$$p = -\log_2 n + 1$$
$$= \Theta(\log n)$$

$$= \Theta(n \cdot \log n)$$

h)

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

$\Theta \log n$

$\Theta \log n$

$\Rightarrow \Theta \log^2 n$

i)

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

1

1 1

$$T(n) = T(n-1) + 3$$
$$= T(n-2) + 6$$

continue k times

$$T(n) = T(\underbrace{n-k}_{0}) + 3.k$$

$n = k$

$$\Rightarrow T(n) = 3n+1$$
$$\boxed{= O(n)}$$

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;
    }
}
```

$\Theta(1)$ $T(N-1)$ $\Theta(1)$ $\Theta(n)$

$T(1) = \Theta(1)$

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

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

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

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

$T(n) = T(n-3) + 3n - 3$

$T(n) = T(n-k) + k \cdot n - k$

$n - k = 1$

$k = n - 1$

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

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

$= \Theta(n^2)$

Q4)

Q4/

a) Big-O notation is used for showing the upper bound of an algorithm. When we talk about upper bounds don't use "at least" instead we use "at most".

The statement should be:

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

b)

1. $2^{n+1} \stackrel{?}{=} \Theta(2^n)$ (True)

$$c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$$

$$\frac{c_1 \cdot \cancel{2^n}}{\cancel{2^n}} \le \frac{2^{n+1}}{2^n} \le \frac{c_2 \cdot \cancel{2^n}}{\cancel{2^n}}$$

$$c_1 \le 2 \le c_2$$

There are c1 and c2 values that provides the equations (c1 = 1, c2 = 2 ...)

So This is True

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

$$\frac{c_1 \cdot 2^n}{2^n} \le \frac{2^{2n}}{2^n} \le \frac{c_2 \cdot 2^n}{2^n}$$

$$c_1 \le \boxed{2^n \le c_2} \rightarrow$$ 2^n cannot be bounded by a constant so False

III. $f(n) = O(n^2)$
$g(n) = \Theta(n^2)$ $>$ $f(n) \times g(n) \stackrel{?}{=} \Theta(n^4)$

(False)

formula:

$$C_1 \cdot h(n) \leq K(n) \leq C_2 h(n)$$

This formula must be ~~possible~~ True for $f(n) \times g(n)$

$$f(n) \leq C_1 \cdot n^2$$

$$C_3 n^2 \leq g(n) \leq C_2 n^2$$

if we product inequalities:

$$C_3 n^2 \leq f(n) \times g(n) \leq C_2 \cdot n^4 \Rightarrow n^2 \neq n^4$$ so wrong form

$\Rightarrow$ False

Q5)

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

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

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

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

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

continue k times

$$\boxed{T(n) = 2^k \cdot T\left(\frac{n}{2^k}\right) + kn}$$

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

$\frac{n}{2^k} = 1 \qquad n = 2^k$

$k = \log_2 n$

$T(n) = n + T(1) + n\log_2 n$

$T(n) = n + 1 + n\log_2 n$

$T(n) = n\log_2 n \quad = O(n\log n)$

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

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

$$\Downarrow$$

$$T(n) = 2\left(2T(n-2) + 1\right) + 1$$

$$= 4T(n-2) + 3$$

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

$$T(n) = 4\left(2T(n-3) + 1\right) + 3$$

$$= 8T(n-3) + 7$$

⋮ continue k times

$$\boxed{T(n) = 2^k T(n-k) + 2^k - 1}$$

$n - k = 0 \quad n = k$

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

$$T(n) = 2^n - 1 \implies O(2^n)$$

Q6)

```java
int[] arr0 = new int[10];
int[] arr1 = new int[100];
int[] arr2 = new int[1000];
int[] arr3 = new int[10000];

//initialize arrays
for(int i=0; i<arr0.length;++i) {
    arr0[i] = i;
}
for(int i=0; i<arr1.length;++i) {
    arr1[i] = i;
}
for(int i=0; i<arr2.length;++i) {
    arr2[i] = i;
}
for(int i=0; i<arr3.length;++i) {
    arr3[i] = i;
}
```

First I created 3 arrays those have different size and initialized with a for loop.

```java
System.out.println("10 size arr: ");  //#1
startTime = System.nanoTime();
findPairs(arr0,10);
endTime = System.nanoTime();
timeElapsed = endTime - startTime;
System.out.println("Execution time in milliseconds: " + timeElapsed / 1000000);
System.out.println("-----------------------------------");

System.out.println("100 size arr: "); //#2
startTime = System.nanoTime();
findPairs(arr1,10);
endTime = System.nanoTime();
timeElapsed = endTime - startTime;
System.out.println("Execution time in milliseconds: " + timeElapsed / 1000000);
System.out.println("-----------------------------------");

System.out.println("1000 size arr: "); //#3
startTime = System.nanoTime();
findPairs(arr2,10);
endTime = System.nanoTime();
timeElapsed = endTime - startTime;
System.out.println("Execution time in milliseconds: " + timeElapsed / 1000000);
System.out.println("-----------------------------------");

System.out.println("10000 size arr: "); //#4
startTime = System.nanoTime();
findPairs(arr3,10);
endTime = System.nanoTime();
timeElapsed = endTime - startTime;
System.out.println("Execution time in milliseconds: " + timeElapsed / 1000000);
```

And then called the functions in main. The functions will find pairs whose sum is 10 (in 10, 100,1000, and 10000 sized array)

This function takes an array and sum value that is going to be found.

```java
//iterative approach
public static void findPairs(int[] arr,int sum) {

    for(int i=0; i<arr.length; i++) {
        for(int j=i; j<arr.length;++j) {
            if(arr[i]+arr[j]==sum) {
                System.out.print(arr[i] + "-" + arr[j]+", ");
            }
        }
    }
    System.out.println();
}
```

Time Complexity of findPairs function:

for(i=0; i< n; i++) → n times

for(j=i j<n; ++j) → ? times

statements

i	j	no of time.
0	0, 1, 2, 3, ..., n	n times works
1	1, 2, ..., n	n-1 times works
2	2, 3, ..., n	n-2 times works
⋮	⋮	⋮
n	n	1 time works

Total : $n + (n-1) + \dots + 1$

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

```
10 size arr:
1-9, 2-8, 3-7, 4-6, 5-5,
Execution time in milliseconds: 0.2469
----------------------------------
100 size arr:
0-10, 1-9, 2-8, 3-7, 4-6, 5-5,
Execution time in milliseconds: 0.3121
----------------------------------
1000 size arr:
0-10, 1-9, 2-8, 3-7, 4-6, 5-5,
Execution time in milliseconds: 6.9523
----------------------------------
10000 size arr:
0-10, 1-9, 2-8, 3-7, 4-6, 5-5,
Execution time in milliseconds: 15.3716
```

The difference is shown clearer in bigger sized arrays.

Q7)

```java
int[] arr0 = new int[10];
int[] arr1 = new int[50];
int[] arr2 = new int[100];

//initialize arrays
for(int i=0; i<arr0.length;++i) {
    arr0[i] = i;
}
for(int i=0; i<arr1.length;++i) {
    arr1[i] = i;
}
for(int i=0; i<arr2.length;++i) {
    arr2[i] = i;
}
```

```java
System.out.println("10 size arr: ");  //#1
startTime = System.nanoTime();
findPairsRecursively(arr0,10,0,1);
endTime = System.nanoTime();
timeElapsed = endTime - startTime;
System.out.println("Execution time in milliseconds: " + timeElapsed / 1000000);
System.out.println("-----------------------------------");

System.out.println("50 size arr: "); //#2
startTime = System.nanoTime();
findPairsRecursively(arr1,10,0,1);
endTime = System.nanoTime();
timeElapsed = endTime - startTime;
System.out.println("Execution time in milliseconds: " + timeElapsed / 1000000);
System.out.println("-----------------------------------");

System.out.println("100 size arr: "); //#3
startTime = System.nanoTime();
findPairsRecursively(arr2,10,0,1);
endTime = System.nanoTime();
timeElapsed = endTime - startTime;
System.out.println("Execution time in milliseconds: " + timeElapsed / 1000000);
System.out.println("-----------------------------------");
```

Best case : Θ(1) -> Base case

Worst case : Θ(n)

Time complexity : O(n)

```java
public static void findPairsRecursively(int[] arr, int sum, int first, int next) {
    if(first>=arr.length-1) return;
    else {

        if (next >= arr.length) {
            next = first+1;
            findPairsRecursively(arr, sum, first+ 1, next);
            return;
        }
        if (arr[first] + arr[next] == sum)
            System.out.println(arr[first] + " - " + arr[next]);


        findPairsRecursively(arr, sum, first, next+ 1);
    }

}
```

```
10 size arr:
1 - 9
2 - 8
3 - 7
4 - 6
5 - 5
Execution time in milliseconds: 0.2107
-----------------------------------
50 size arr:
0 - 10
1 - 9
2 - 8
3 - 7
4 - 6
5 - 5
Execution time in milliseconds: 0.5001
-----------------------------------
100 size arr:
0 - 10
1 - 9
2 - 8
3 - 7
4 - 6
5 - 5
Execution time in milliseconds: 1.0666
-----------------------------------
```

HACI HASAN SAVAN

1901042704

HMW2