HOMEWORK

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

$\log_2 n^2+1 \leq c.n \qquad n \geq n_0$

$c=2 \qquad n_0 = n > 0$

$\log_2 n^2+1 \leq 2.n$ → grows faster so its true.

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

$\sqrt{n^2+n} \geq = c.n$

$c=1 \qquad n_0 = n > 1$

$\sqrt{n^2+n} \geq n$ → left side value grows faster. So true.

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

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

$c_1 = \frac{1}{4} \qquad n_0 = n > 0 \qquad c_2 = 1$

$n = 2$

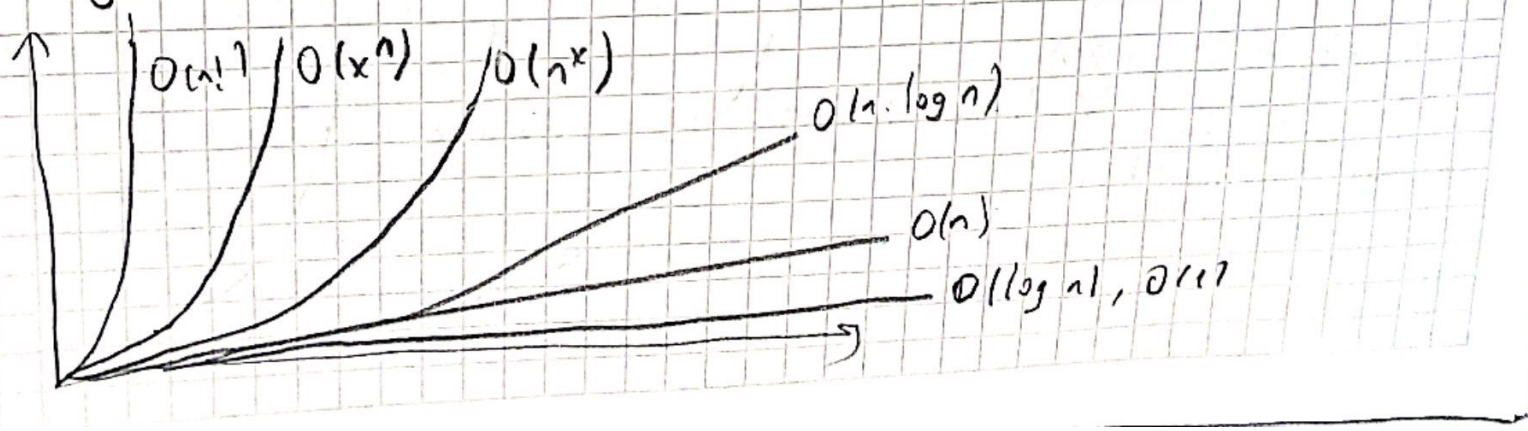
$\frac{1}{4} n^n \leq n^{n-1} \leq 1.n^n$ → this equation is true. So true.

2) in Limit $6 < \log.\log n < \log^a n < n^b < c^n < n! < n^n$

so exponential expressions grows faster because when n increase they increase too much. which is bigger, it is bigger. $8^{\log_2 n}$ becomes smaller for 2^n for increasing n values. then n^x values are bigger.

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

in limit method, if $f(x)$'s degree bigger than $g(x)$, than it grows faster.



3) $T_1(n)$ = loop inside $T_2(n)$ = loop work time

look first inside of loop then inside of ifs.

if statement → $\Theta(1)$
return statement → $\Theta(1)$
for $(i=2; i<n; i++)$ → $O(n)$
assignment → $\Theta(1)$

a) if → $\Theta(1)$
else → $\Theta(1)$
for → $T_2(w)$ best = $\Theta(1)$
$T_2(w)$ worst = $O(\log n)$ because i increasing i times
so $\Omega(1)$ or $\boxed{O(\log n)}$

$O(\log n) + \Theta(1) = \boxed{O(\log n)}$

b) if → $\Theta(1)$ Size of Array = n
else if → $\Theta(1)$
if → $\Theta(1)$
for → $O(n)$

$O(n) + \Theta(1) = \boxed{O(n)}$

c) return statements = $\Theta(1)$

d) int → $\Theta(1)$
assign → $\Theta(1)$
for → T best → $\Theta(1)$
T worst → $O(n)$
return → $\Theta(1)$

$\Theta(1) + O(n) = \boxed{O(n)}$

e) if double for then must be $O(n^2)$
printf → $\Theta(1)$
First loop → best → $\Theta(1)$
worst → $\Theta(n) = O(n)$ because j increase
Second loop → best → $\Theta(1)$
worst → $O(\log_2 n)$ because j increase 2x

$\boxed{T_n = O(n) \cdot O(\log_2 n)}$

f) inside of if → O(n)
if → Θ(1)
else → Θ(1)
$T_n = O(n \cdot \log_2 n) + 1$
$T_n = O(n \cdot \log_2 n)$

g) println → Θ(1)
for → Best → Θ(1)
worst → Θ(n) → O(n)
while → Best → Θ(1)
worst → $O(\log_2 n)$ because i decreasing 2x
$T_n = O(n) \cdot O(\log_2 n) = O(n \cdot \log_2 n)$

h) println → Θ(1)
for → Best Θ(1)
worst O(n)
while → Best Θ(1)
worst $O(\log_2 n)$ because n decreasing 2x
so $T_n = O(n) \cdot O(\log_2 n) = O(n \cdot \log_2 n)$

i) its recursion. so $T = O(n)$

j) recursion = O(n)
while loop = O(n)
$T = O(n^2)$

4) a) We need both upper bound and lower bound. But in this sentence we only have one. So its meaningless.

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

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

$c_1 = 1 \quad c_2 = 3 \quad n_0 = n \geq 0$

$2^n \leq 2^{n+1} \leq 3 \cdot 2^n$

in this case its true

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

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

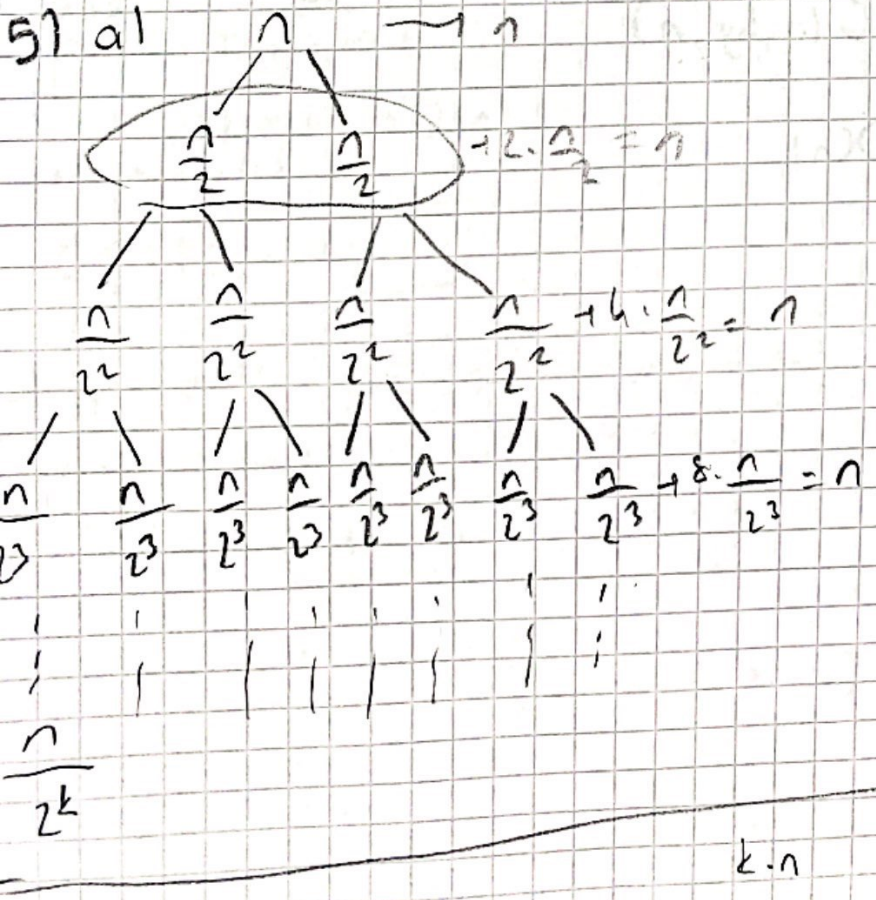
2^{2n} never can be bigger than 2^n so this is false.

3) $\Theta(n^2) = \Theta(n^2) = \Omega(n^2)$

↳ so $O(n^2) \cdot O(n^2) = \boxed{O(n^4)}$

$\Omega(n^2) \leq \Theta(n^2) \leq O(n^2)$

5) a)



$T(n) = k \cdot n$

$T(1) = 1 \quad \frac{n}{2^k} = 1$

$n = 2^k$

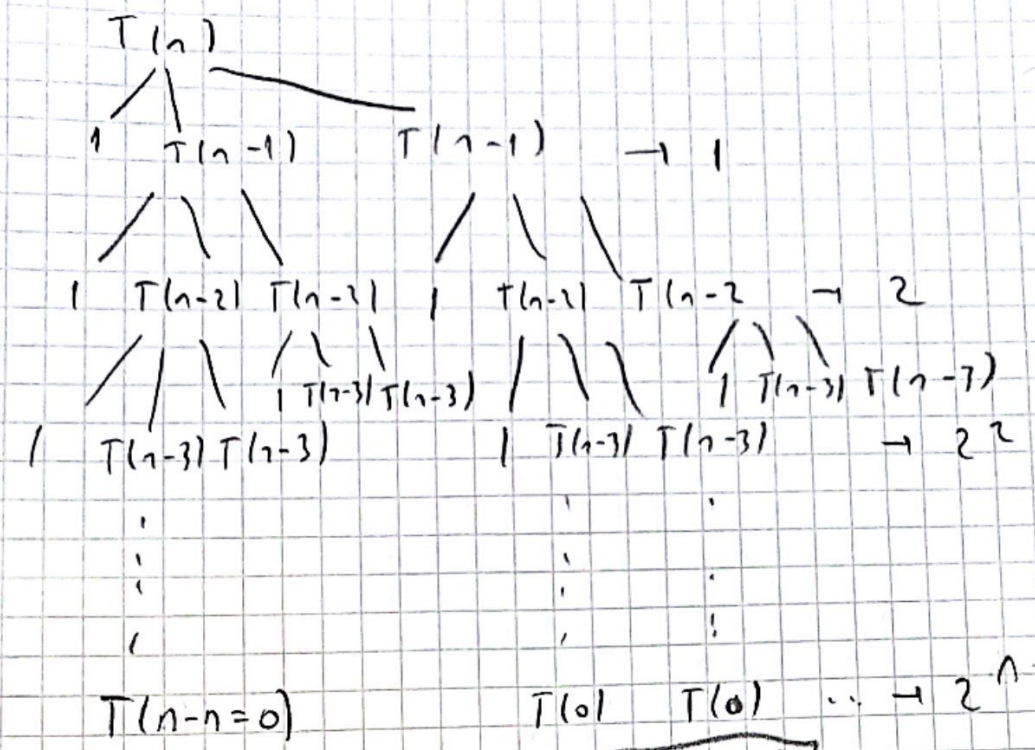
$k = \log_2 n$

$\boxed{T(n) = \log_2 n \cdot n}$

$\boxed{O(\log_2 n \cdot n)}$

k·n

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



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

c) Time taken for 5000 element array = 0.9 s

for 10000 element array = 1.15 s

for 20000 element array = 2.32 s

for 50000 element array = 5.05 s

for loop = $O(n)$

Second for loop = $O(n) \cdot O(n) = O(n^2)$

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

$\boxed{= O(n^2)}$

QUESTION 6 CODE:

```
#include <iostream>
#include <time.h>
#include <chrono>

using namespace std;

int main(){

    auto start = chrono::steady_clock::now();
    int number = 50000, /*number is length of array can be changed*/
    arr[number],
    sum;



    for (int i=0;i<number;i++){
        arr[i]=rand()%10001 + (-5000);
    }

    cout << "Please enter sum " << endl;
    cin >> sum;

    for (int i=0;i<number;i++){

        for (int j=i;j<number;j++){

            if (arr[i]+arr[j]==sum){
                cout << arr[i] << "+" << arr[j] << "=" << sum << endl;
            }
        }
    }
    auto end = chrono::steady_clock::now();
    auto diff = end - start;
    cout << "Array size:" << number <<  endl;
    cout << chrono::duration <double, milli> (diff).count() << " ms" << endl;
    return 0;

}
```