Part 2

a) $O(n)$ provide us find maximum complexity. so that we can't talk about at least $O(n^2)$.

—

b) We need to proof following equations

$c_1(f(n)+g(n)) >= 0$

$c_1(f(n)+g(n)) <= max(f(n), g(n))$

$max(f(n), g(n)) <= c_2(f(n)+g(n))$

$f(n)$ and $g(n)$ are asimptoticaly non-negative functions.

$max(f(n), g(n)) >= f(n)$ and $max(f(n), g(n))$. Thus we get

$max(f(n), g(n)) = f(n)+g(n)/2$ we can say $c_1 = 1/2$

$max(f(n), g(n)) <= f(n) + g(n)$ and $c2$ could be any positive number larger or equals to 1.

c) 1. $\lim\limits_{n\to\infty} \frac{2^{n+1}}{2^n} = \lim\limits_{n\to\infty} \frac{\cancel{2^n}.2}{\cancel{2^n}} = 2$

It is known that if $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = c \in R$ then $f(n) = \Theta(g(n))$.

2. $2^{2n} = (2^2)^n = (2^n)^2$ so $(2^2)^n = 4^n$

$2^n < 4^n$ so that this equation $O(4^n)$, Obviously rate of growth false.

3. —

Part-3

$3^n > n.2^n > 2^{n+1} = 2^n > 5^{\log_2 n} > n.\log_2 n > n^{1.01} > \sqrt{n} > \log n = (\log n)^3$

- $\lim_{n \to \infty} \frac{3^n}{n 2^n}$ Since $\left(\frac{3}{2}\right)^x$ grow asimptotically faster than the polynomial x

so $\lim_{n \to \infty} \frac{(3/2)^n}{n} = \infty$

- $\lim_{n \to \infty} \frac{n.2^n}{2^{n+1}} = \frac{n.\cancel{2^n}}{2.\cancel{2^n}} = \frac{n}{2} = \infty$

- $\lim_{n \to \infty} \frac{2^{n+1}}{2^n} = \frac{\cancel{2^n}.2}{\cancel{2^n}} = 2$. constant number doesn't matter in fast growing numbers

- $\lim_{n \to \infty} \frac{2^n}{5^{\log_2 n}}$ so $5^{\log_2 n} = n^{\log_2 5}$ we calculate $\log_2 5 = 2.25$ after that

$\lim_{n \to \infty} \frac{2^n}{n^{2.25}} = \infty$ 2^n grows faster than $n^{2.25}$.

- $\lim_{n \to \infty} \frac{5^{\log_2 n}}{n.\log_2 n} = \frac{n^{2.25}}{n} \cdot \frac{1}{\log_2 n} = \frac{n^{1.25}}{\log_2 n} = \infty$ because $n^{1.25}$ grows fa[ster] than $\log_2 n$

$\lim_{n \to \infty} \frac{n.\log_2 n}{n^{1.01}} = \frac{\log_2 n}{n^{0.01}} = \infty$

$\lim_{n \to \infty} \frac{n^{1.01}}{\sqrt{n}} = \frac{n^{1.01}}{n^{0.5}} = n^{0.51} = \infty$

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

- $\lim_{n \to \infty} \frac{\log n}{(\log n)^3} = \infty$

Part - 4

a) find_min (arr[ ])
implement a temp value and initialize array's first value
for i smaller than arr.length - 1 {
if temp bigger than than the array[i] element
sets array[i] to temp
}

There are just one case. We are doing 1 comparison and set at every step of the loop (n-1).

$T(n) = \theta(n-1) . \theta(1) + \theta(1) + \theta(1)$

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

$T(n) = O(n)$

b)

find_median (arr[ ])
implement temp;
for i smaller than arr.length - 1 $\quad \theta_w(n-1) \quad \theta_b(n-1)$
for j smaller than arr.length - 1 - i $\quad \theta_w(n-1-i) \quad \theta_b(n-1-i)$
if (arr[j] bigger than arr[j+1] $\quad \theta(1)$
temp = arr[i]
arr[i] = arr[i+1]
arr[i+1] = temp
$\theta(1)$ $\quad \theta(1)$

end
temp = arr[(arr.length + 1)/2] $\quad \theta(1)$

$T(n) = \theta(n-1) . \theta(n-1-i) . \theta(1) . \theta(1) + \theta(1)$

$T(n) = \theta(n^2)$

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

c) find_two_elements (arr[], value)
int v1, v2 ;

Pseudo

for i smaller than arr.length - 1 $\Theta_w(n-1)$ $\Theta_b(1)$
v1 = arr[i] ; $\Theta(1)$
for j = i+1 smaller than arr.length $\Theta_w(n-1)$ $\Theta_b(1)$
v2 = arr[j] ; $\Theta(1)$
if (v1+v2)/2 == value
i = arr.length ; $\Big\}\ \Theta(1)$

Analyze

$T_w = ((\Theta(1) + \Theta(1)) . \Theta(n-1) + \Theta(1)) . \Theta(n-1) = \Theta(n^2)$

$T_b = ((\Theta(1) + \Theta(1)) . \Theta(1) + \Theta(1))\ \Theta(1) = \Theta(1)$

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

$T(n) = \Omega(1)$

d) arr1[n] ;
arr2[n] ;
list[2n] ;
int i, j, k ;
int temp ;
for (k=0; i=0; i < arr1.length ; ++i, ++k) $\Theta(n)$
list[k] = arr1[i] ; $\Theta(1)$
for (j=0 ; j < arr2.length ; ++j, ++k) $\Theta(n)$
list[k] = arr2[j] ; $\Theta(1)$
$\Theta(1)$ $\Big\}\ \Theta(n)$ $\Big\}\ \Theta(n^2)$
temp = list[0] ;
for (i=0 ; i < list.length ; ++i) $\Theta(n)$
for (j = i+1 ; j < list.length ; ++j) $\Theta(n-1)$
if (list[i] < list[j])
swap list[i] and list[j] $\Big\}\ \Theta(1)$ $\Big\}\ \Theta(n^2)$

$T(n) = \Theta(n^2)$

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

Part - 5

a)
```
int p-1 (int array[])          T(n) = Θ(1)
{
    return array[0] * array[2].    Θ(1)
}
```

$T(n) = O(1)$ $\quad S(n) = n$

b)
```
int p-2 (int array[], int n)
{
    int sum = 0                           Θ(1)
    for (int i=0; i<n ; i=i+5)
            sum += array[i] * array[i]    Θ(n/5)
    return sum                            Θ(1)
}
```

$T(n) = \Theta(1) + \Theta(n/5) + \Theta(1) = \Theta n$

$T(n) = O(n)$ $\quad S(n) = n + 1 = n$

c)
```
void p-3 (int array[], int n)
{
    for(int i=0; i<n; ++i)
        for(int j=0; j<i; j=j*2)
            printf("%d", array[i] * array[j])
}
```

$T(n) = n.(1+2+3+\ldots+n) = n^2$

$T(n) = O(n^2)$ $\quad S(n) = n+3 = n$

d)
```
void p-4 (int array[], int n):
{   if (p-2(array, n)) > 1000)        Θ(n)
        p-3(array, n)                 Θ(n²)

    else
        printf(" ... ", p-1(array) * p-2(array, n))   Θ(1) + Θ(n)

}
```

$T_w(n) = \Theta(n^2)$

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

$T(n) = O(n^2)$ $\quad T(n) = \Omega(n)$ $\quad S(n) = n + 5 = n$