Emirhan Burak Yilmaz
1901042659

1-)

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

$2\log_2 n + 1 \leq 2n$

for $n=1$ $\quad 1 \leq 2$

for $c=2, n_0=1$. it's true.

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

$\sqrt{n^2+n} \geq n$

$n^2+n \geq n^2$

for $c=1, n_0=1$ it's true.

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

$n^{n-1} = O(n^n)$

$\frac{n^n}{n} \leq n^n$

for $c=1, n_0=1$ it's true

$n^{n-1} = \Omega(n^n)$

$\frac{n^n}{n} \geq cn^n$

$n^n \geq cn^{n+1}$

this is not true for all n.

statement is false it should be $n^{n-1} = O(n^n)$

2-)

We know logarithmic functions grows slowest and then linear functions grows slower than exponential functions. So intuitivly we can start with comparison $\log n$ and $\sqrt{n}$,

- $\lim_{n\to\infty} \frac{\sqrt{n}}{\log n} = \frac{1/2 . n^{-1/2}}{1/(\ln 2 . n)} = \frac{\ln 2 . n^{1/2}}{2} = \infty$ $\quad \sqrt{n} = \Omega(\log n)$ // $\log n = O(\sqrt{n})$

- $\lim_{n\to\infty} \frac{n^2}{\sqrt{n}} = n^{3/2} = \infty$ $\quad \sqrt{n} = O(n^2)$

- $\lim_{n\to\infty} \frac{n^2 \log n}{n^2} = \log n = \infty$ $\quad n^2 = O(n^2 \log n)$

- $\lim_{n\to\infty} \frac{n^3}{n^2 \log n} = \frac{n}{\log n} = \frac{1}{1/(\ln 2 . n)} = \infty$ $\quad n^2 \log n = O(n^3)$

- $\lim_{n\to\infty} \frac{8^{\log_2 n}}{n^3} = \frac{\ln 8 . 8^{\log_2 n}}{3n^2} = \frac{(\ln 8)^3 . 8^{\log_2 n}}{3!} = \infty$ $\quad n^3 = O(8^{\log_2 n})$

after 2 L'Hopital

- $\lim_{n\to\infty} \frac{2^n}{8^{\log_2 n}} = \frac{2^n}{2^{3\log_2 n}} = 2^{n/3\log_2 n} = 2^{\lim_{n\to\infty} \frac{n}{3\log_2 n}}$

$\lim_{n\to\infty} \frac{n}{3\log_2 n} = \frac{1}{3/(\ln 2 . n)} = \frac{\ln 2 . n}{3} = \infty$ $\quad 8^{\log_2 n} = O(2^n)$

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

$\Rightarrow$ $\log n, \sqrt{n}, n^2, n^2\log n, n^3, 8^{\log_2 n}, 2^n, 10^n$ (increasing time complexity)

3-)

a.) for(____)
if(__)
____
else (___)
____

This is not an linear increasing (more like logarithmic) so it's upper bound should be an linear function. On the other hand since this is like an logarithmic function it's lower bound should be logn.

$T_a(n) = O(n)$
$= \Omega(n\log n)$

b.)

$\Theta(1)$ [____
____

$\Theta(n)$ [for (___) $\Theta(n)$
$\Theta(1)$ [if(___) $\Theta(1)$
____ $\Theta(1)$
else (___) $\Theta(1)$
____ $\Theta(1)$

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

c.) $T_c(n) = \Theta(1)$

d.) ____ $\Theta(1)$

$\Theta(n)$ [for (___)
____ $\Theta(1)$
____ $\Theta(1)$

# of iterations	i
1	0
2	5
3	5.2
⋮	⋮
k	5.(k-1)

$5(k-1) \geq n$

$k \geq \frac{n}{5} + 1 \rightarrow T_{for}(n) = \Theta(n)$

$T_d(n) = \Theta(n)$

e.)

for (———) $\theta(n)$ — $\sum_{i=1}^{n} \log i = \log n!$

for (———) $\theta(\log n)$

——— $\theta(1)$

# of iteration	3
1	2^0
2	2^1
⋮	⋮
k	2^{k-1}

$2^{k-1} = i \quad (3 < i)$

$k = \log_2 i + 1$

$T_e(n) = \sum \theta(n \log n) \quad \sum_{i=1}^{n} \log_2 i + n = \log_2 n! + n$

$T_e(n) = \theta(\log n!)$

f.)

if (———) $\theta(n)$

——— $\theta(n.\log n)$

——— $\theta(n)$

$T_{fw}(n) = \theta(n \log n)$ → worst case

$T_{fb} = \theta(n)$ → best case

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

$= \Omega(n)$

g.)

$\theta(n \log n)$ [while (———) $\theta(\log n)$

$\theta(n)$ [for (———) $\theta(n)$

——— $\theta(1)$

$T_g = n \log n$

# of iteration	i
1	n
2	$n/2$
3	$n/2^2$
⋮	⋮
k	$n/2^{k-1}$

$\frac{n}{2^{k-1}} \cong 1$

$n = 2^{k-1}$

$k = \log n + 1$

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

h.)

while (———) $\theta(\log n)$

for (———) $\theta(n)$

———

$T_h(n) = n \log n$

multiplization, substraction

i.) $T_i(n) = 2 + T_i(n-1)$, $T_i(0) = 1$

$T_i(n) = 2 + (2 + T_i(n-2))$

$T_i(n) = 2.k + T_i(n-k)$ set $k=n$

$T_i(n) = 2n + T_i(0) = 2n+1$

$T_i(n) = O(n)$

j.) $T_j(1) = 1$

best case

no loop

$T_{jb}(n) = T_{jb}(n-1) + 1$

$T_{jb}(n) = T_{jb}(n-2) + 1 + 1$

$T_{jb}(n) = T_{jb}(n-k) + k$ for $k = n-1$

$T_{jb}(n) = 1 + n - 1$

$T_{jb}(n) = \Theta(n)$

worst case

$T_{jw}(n) = T_{jw}(n-1) + n - 1$

$T_{jw}(n) = T_{jw}(n-2) + (n-2) + (n-1)$

$T_{jw}(n) = T_{jw}(n-k) + kn - \frac{k(k+1)}{2}$

$T_{jw}(n) = 1 + n^2 - \frac{n^2+n}{2}$

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

4.) $\Rightarrow T(n) = O(n^2)$, $T(n) = \Omega(n)$

a.) Big Oh notation is used to indicate set of functions which are upper bound of given function.

$f(x) = O(g(x))$ means grow rate of $f(x)$ never be bigger than $g(x)$. So big Oh used for "at most" or "upper bound".

So using big Oh to indicate lower band of function is incorrect.

b.)

I.) $2^{n+1} = \Theta(2^n)$ $g(n) = 2^n$, $f(n) = 2^{n+1}$

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

$2.2^n \leq 3.2^n$ ✓ $2.2^n \geq 2^n$ ✓

$c_2 = 3, n_0 = 0$ $c_1 = 1, n_0 = 0$

since $c_1.g(n) \leq f(n) \leq c_2.g(n)$, $2^{n+1} = \Theta(2^n)$

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

$f(n) = 2^{2n}$
$g(n) = 2^n$

it can also observe with limit rule,

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

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

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

$2^{2n} \geq 2^n$ ✓

$c=1, n_0=0$

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

$2^n \cdot 2^n \leq c \cdot 2^n$

no constant c which is bigger than 2^n so this is not true

since $2^{2n} \neq O(2^n)$, statement is False. It should be $2^{2n} = \Omega(2^n)$.

III.) $f(n) = O(n^2)$
$g(n) = \Theta(n^2)$

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

proof by counter example.

let $f(n) = n$, let $g(n) = n^2$

$n \leq n^2$

$f(n) = n = O(n^2)$ ✓

$f(n) * g(n) = n^3$

$n^3 = O(n^4)$

$n^3 \leq n^4$ ✓

$n^3 \neq \Omega(n^4)$

$n^3 \geq c \cdot n^4$ X

there is no such constant c.

=> So statement is false.

5-)

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

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

$T(n) = 2^k \cdot T(n/2^k) + k \cdot n$

to derive $T(1)$, $\frac{n}{2^k} = 1$

$2^k = n$

$k = \log_2 n$

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

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

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

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

$T(n) = 4.(2T(n-3)+1) + 2 + 1 = 2^3.T(n-3) + 2^2 + 2^1 + 2^0$

$T(n) = 2^k.T(n-k) + \underbrace{2^{k-1} + 2^{k-2} + \cdots + 2^0}$

recall

$1 + r + r^2 + r^3 + \cdots + r^{n-1} = \dfrac{(1-r^n)}{1-r}$

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

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

$T(n) = \cancel{2^n.T(0)} + 2^n - 1$

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

6.)

```
int pairsOfSumLoop( int arr[], int size, int sum) {
    int count=0;
    for(int i=0; i< size ; ++i) {          ] O(n)
        int target = sum - arr[i];          ] O(1)
        for(int j=i+1; j< size ; ++j)       ] O(n)
            if( arr[j] == target)
                ++count;                    ] O(1)
    }
    return count;
}
```

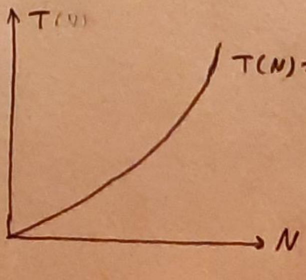
$O(n)$ $\Theta(n^2)$

N : problem size (length of the array)

T(N) : running time

N	T(N)	
1000	12.10^{-4} sec	(0.001100 sec)
10000	12.10^{-2} sec	(0.1280 sec)
100000	12 sec	(12.389 sec)
1000000	12.10^{2} sec	(1214.40 sec)

- Time complexity of the algorithm is $\Theta(n^2)$. It can also be observed with the given running times. Increasing problem size will effect the run time with quadratic increase for



7.)

```
int pairsOfSumRecursive( int arr[], int size, int sum) {
    int count = 0
    if( size > 1) {
        int target = sum - arr[size -1];
        for(int i = size-2; i>=0; --i)               θ(n)
            if( arr[i] == target)      ] θ(1)
                ++count;
        count += pairsOfSumRecursive( arr, size-1, sum);
    }
    return count;
}
```

$$T(n) = \overbrace{n-1}^{\text{from For Loop}} + T(n-1)$$

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

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

$$T(n) = nk - \frac{k.(k+1)}{2} + T(n-k) \quad \text{for } k = n-1$$

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

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

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