CSE 222 Homework 2

Ali Kaya
1901042618

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

$2\log_2 n + 1 = O(n) \Rightarrow 2\log_2 n + 1 \leq cn \quad \forall n \geq n_0$

$c = 10, \ n_0 = 1 \qquad 2\log_2 n + 1 \leq 10n \quad n \geq 1$

$2.0 + 1 \leq 10$ ✓ true

b) $\sqrt{n(n+1)} = \Omega(n) \Rightarrow \sqrt{n^2+n} \geq cn \quad \forall n \geq n_0$

$c = 1, \ n_0 = 5 \qquad \sqrt{n^2+n} \geq n \quad n \geq 5$

$\sqrt{30} \geq 5$ ✓ true

c) $n^{n-1} = \Theta(n^n) \Rightarrow c_1 n^n \leq n^{n-1} \leq c_2 n^n \quad n \geq n_0$

$c_1 = 1, \ c_2 = 5 \qquad n^n \leq n^{n-1} \leq 5n^n \quad \forall n \geq 1$

$n_0 = 1 \qquad 1 \leq 1 \leq 5$

$n = 2 \qquad 4 \leq 2 \leq 20$ Ω notation doesn't qualified ✗ false

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

$\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = 0$ $\log n$ grows slower than $\sqrt{n}$

$\lim_{n\to\infty} \frac{\sqrt{n}}{n^2} = 0$ $\sqrt{n}$ grows slower than n^2

$\lim_{n\to\infty} \frac{n^2}{n^2\log n} = 0$ n^2 grows slower than $n^2\log n$

$\lim_{n\to\infty} \frac{n^2\log n}{n^3} = 0$ $n^2\log n$ grows slower than n^3

$\lim_{n\to\infty} \frac{8^{\log_2 n}}{2^n} = 0$ $8^{\log_2 n}$ grows slower than 2^n

$\lim_{n\to\infty} \frac{2^n}{10^n} = 0$ 2^n grows slower than 10^n

Growth rate:

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

$\lim_{n\to\infty} \frac{n^3}{8^{\log_2 n}} = \lim_{n\to\infty} \frac{n^3}{n^{\log_2 8}}$ n_3 and $8^{\log n}$ have same growth rate

$= \lim_{n\to\infty} \frac{n^3}{n^3} = 1$

3 7

2,3,6,7,42,43, 1806, 1807

3) a)

```
int p-1 (int my-array[]) {
    for (int=2 , i<=n , i++){
        if(i%2==0) {          -> Θ(1)
            count++;          -> Θ(1)
        } else {
            i=(i-1)i ;        -> Θ(1)
        }
    }
}
```

$\Theta(\log N)$

b)

```
int p-2 (int my_array[]){
    first_element = my_array[0];
    second-element = my-array[0];        } Θ(1)
    for (int i=0 ; i<sizeofArray ; i++){ -> Θ(N)
        if(my_array[i] < first_element){ -> Θ(1)
            second_element = first-element;
            first_element = my-array[i];    } Θ(1)       } Θ(1)
        }else if(my-array[i] <second_element){ -> Θ(1)
            if(my-array[i] != first element){ -> Θ(1)
                second_element = my-array[i]; -> Θ(1)   } Θ(1)   } Θ(1)
            }
        }
    }
}
```

$\Theta(N)$

- $T_{Best}(n) = \Theta(N)$
- $T_{worst}(n) = \Theta(N)$

- $T(n) = O(N)$
  $= \Omega(N)$ $\rightarrow \Theta(N)$

c)

```
int p-3 (int array[]){
    return array[0] * array[2]; -> Θ(1)
}
```

- $\underline{T(n) = \Theta(1)}$

d)
```
int p_4(int array[], int n){
    int sum = 0;                          → Θ(1)
    for(int i = 0 ; i < n ; i = i+5)      → Θ(1/5 n)
        sum += array[i] * array[i];       → Θ(1)
    return sum;
}
```
$\Theta(N)$ } $\Theta(N)$

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

e)
```
void p_5(int array[], int n){
    for(int i = 0 ; i < n ; i++)          → Θ(N)
        for(int j = 1 ; j < i ; j = j*2)  → O(logN)
            printf("%d", array[i] * array[j]);  → Θ(1)
}
```
$O(\log N)$ } $O(n\log N)$

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

f)
```
int p_6(int array[], int n){
    if( p_4(array, n) > 1000)             → Θ(N)
        p_5(array, n);                    → O(nlogN)
    else printf("%d", p_3(array) * p_4(array, n));  → Θ(N)
}
```
$p\_3(array)$: $\Theta(1)$, $p\_4(array, n)$: $\Theta(N)$

- $T_{Best} = \Theta(N) + \min(O(n\log N), \Theta(N)) \longrightarrow T_{Best} = \Theta(N)$
- $T_{worst} = \Theta(N) + \max(O(n\log N), \Theta(N)) \longrightarrow T_{worst} = O(n\log N)$
- $T(n) = O(n\log N)$
  $= \Omega(N)$

g)
```
int p_7(int n){
    int i = n;                            → Θ(1)
    while(i > 0){                         → Θ(logN)
        for(int j = 0 ; j < n ; j++)      → Θ(N)
            System.out.println("*");      → Θ(1)
        i = i/2;                          → Θ(1)
    }
}
```
$\Theta(N)$ } $\Theta(N\log_2 N)$

- $T(n) = \Theta(N\log_2 N)$

h) int p-8 (int n) {

while (n > 0) { → $\Theta(\log N)$

for(int j=0 ; j < n ; j++) → $\Theta(\log N)$

System.out.println("*"); → $\Theta(1)$

n = n/2 ; → $\Theta(1)$

}

}

$\Theta((\log N)^2)$

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

i) int p-9 (n) {

if (n==0) → $\Theta(1)$

return 1 ; → $\Theta(1)$

else

return n * p-9 (n-1) ;

}

$\boxed{T(n) = \Theta(1),\ n=0}$

$T(n) = \Theta(1) + T(n-1)$

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

$k = n$

$T(n) = k.\Theta(1) + T(n-k)$

$T(n) = n.\Theta(1) - \Theta(1) + T(0)$

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

j) int p-10 (int A[] , int n) {

if (n==1) → $\Theta(1)$

return ; → $\Theta(1)$

p-10 (A ,n-1);

$T_x(n) = \Theta(n)$

j = n-1 ;

while (j > 0 and A[j-1]) { → It can be $\Theta(1)$ or $\Theta(n)$

SWAP (A[j] , A[j-1]) ;

j = j-1 ;

}

}

$T_{Best}(n) = \Theta(n) \cdot \min(\underline{\Theta(1)}, \Theta(n)) \Rightarrow T_{Best} = \Theta(n)$

$T_{worst}(n) = \Theta(n) \cdot \max(\Theta(1), \underline{\Theta(n)}) \Rightarrow T_{worst} = \Theta(n^2)$

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

$\Rightarrow$ If we say "at least", that means we want to indicate lower bound of the algorithm.; to indicate lower bound we need to use Ω (omega) notation: $\Omega(n^2)$.

b)

I $\rightarrow$ $2^{n+1} = \Theta(2^n)$ $\Rightarrow$ $C_1 . 2^n \leq 2^{n+1} \leq C_2 2^n$ $\forall n \geq n_0$

$C_1 = 1$ $\quad C_2 = 5$ $\quad n_0 = 1$ $\qquad 2^n \leq 2^{n+1} \leq 5.2^n$

$2 \leq 4 \leq 10$ ✓ True

II $\rightarrow$ $2^{2n} = \Theta(2^n)$ $\Rightarrow$ $C_1 . 2^n \leq 2^{2n} \leq C_2 2^n$ $\forall n \geq n_0$

$C_1 = 1$ $\quad C_2 = 4$ $\quad n_0 = 3$ $\qquad 2^n \leq 2^{2n} \leq 4.2^n$

$8 \leq 64 \leq 32$ ✗ False

III $\rightarrow$ $f(n) = O(n^2)$ and $g(n) = \Theta(n^2)$, $f(n) * g(n) =?\ \Theta(n^4)$

if we can say $f(n) = O(n^2)$ we can also say $f(n) = O(n^3)$ because O notation indicates upper bound

We cannot say: $f(n) * g(n) = \Theta(n^4)$ $\rightarrow$ because if $f(n) = O(n^3)$ and $g(n) = \Theta(n^2)$

✗ false

$f(n) * g(n) = O(n^5)$ $\qquad$ $O() * \Theta() = O($

5)

a) $T(n) = 2T(n/2) + n$ $\rightarrow$ $T(n) = 2T(\frac{n}{2}) + n$

$T(n) = 2\left[2T(\frac{n}{4}) + \frac{n}{2}\right] + n$ $\rightarrow$ $T(n) = 4T(\frac{n}{4}) + 2n$

$T(n) = 2\left[2\left(2T(\frac{n}{8}) + \frac{n}{4} + \frac{n}{2}\right]\right. + n$ $\rightarrow$ $T(n) = 8T(\frac{n}{8}) + 3n$

$\Rightarrow T(n) = 2^k . T(\frac{n}{2^k}) + k.n$ $\Rightarrow$ $2^k = n$, $k = \log_2 n$ $\Rightarrow$ $T(n) = n.T(1) + n.\log_2 n$

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

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

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

$T(n) = 2[2T(n-2) + 1] + 1$ $\rightarrow$ $T(n) = 4T(n-2) + 3$

$T(n) = 2[2(2T(n-3) + 1) + 1] + 1$ $\rightarrow$ $T(n) = 8T(n-3) + 7$

$\rightarrow T(n) = 2^k . T(n-k) + 2^k - 1$ $\rightarrow$ $k = n$ $\Rightarrow$ $T(n) = 2^n . T(0) + 2^n - 1$

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

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

Scanned with CamScanner

6)

```
public int findPairs (int[] arr, int sum)
    int pairNum = 0;
    for(int i=0; i<arr.length; i++) { → Θ(N)
        for (int j=i+1; j<arr.length; j++) { Θ(N),
            if( j != i && arr[i] + arr[j] = sum) { → Θ(1)
                pairNum ++;   → Θ(1)
            }
        }
    }
    return pairNum;
}
```

Θ(N), Θ(N^2)

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

when capacity of arr is 100000   run-time = 4,70 sn

" " " " " 200000   " = 19,40 sn

7)

```
public int findPairsR(int arr, int sum, int x, int y){
    int pairNum = 0;
    if(x == arr.length-2){
        if(arr[x] + arr[y] == sum) return pairNum+1;
        else return pairNum;
    }
    else if(y == arr.length-1){
        if(arr[x] + arr[y] == sum) return pairNum+1;
        else return pairNum + findPairsR(arr, sum, x+1, x+2);
    }
    else {
        if(arr[x] + arr[y] == sum) return 1+ findPairsR(arr, sum, x, y+1);
        else return pairNum + findPairsR(arr, sum, x, y+1);
    }
}
```