MUHAMMET ÇAĞRI YILMAZ

1901042694

QUESTION 1

1) For each of the following statements, specify whether it is true or not, and prove your claim. Use the definition of asymptotic notations.

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

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

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

ANSWER

1)

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

$\log_2 n^2 \leq n-1$

$2\log_2 n \leq n-1$ $\qquad \log_2 n \leq \frac{n-1}{2}$

$n \leq 2^{\frac{n-1}{2}}$

$c=1$ $\qquad n \geq 8 = n_0$ $\qquad$ it is true

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

$\sqrt{n.n+1} \geq c_1 n$

if $c_1 = 1$

$\sqrt{n.n+1} - n \geq 0$ $\qquad \sqrt{n}\,(\sqrt{n+1} - \sqrt{n}) \geq 0$

$\sqrt{n} \geq 0$ it is true $\qquad$ True

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

$c_1 n^n \geq n^{n-1} \geq c_2 n^n$

$c_1 n^n \geq n^{n-1}$

$c_1 \geq \frac{1}{n}$ $\qquad$ c_1 is constant, false

QUESTION 2

2) Order the following functions by growth rate and explain your reasoning for each of them. Use the limit method.

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

ANSWER

Subject: Date : ...../...../.....

2.

$$\lim_{n\to\infty} \frac{10^n}{2^n} = \left(\frac{10}{2}\right)^n \to 5^n = \infty \qquad \lim_{n\to\infty} \frac{10^n}{8^{\log_2 n}} = \infty,\quad 8^{\log_2 n} \to n^3$$

$$\lim_{n\to\infty} \frac{10^n}{n^3} = \infty \qquad \lim_{n\to\infty} \frac{10^n}{n^2 \log n} = \infty \qquad \frac{10^n}{n^2} = \infty$$

$$\lim_{n\to\infty} \frac{10^n}{\sqrt{n}} = \infty \qquad \lim_{n\to\infty} = \frac{10^n}{\log n} = \infty$$

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

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

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

$$\lim_{n\to\infty} \frac{8^{\log_2 n}}{n^3} = \frac{2^{3\log_2 n}}{n^3} = \frac{n^3}{n^3} = 1 \text{, the same}$$

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

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

$$\lim_{n\to\infty} \frac{8^{\log_2 n}}{\sqrt{n}} = \frac{n^3}{\sqrt{n}} = n^{5/2} = \infty$$

$$\lim_{n\to\infty} \frac{8^{\log_2 n}}{\log n} = \frac{n^3}{\log n} = \infty$$

I don't look n^3 because n^3 and $8^{\log_2 n}$ are the same

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

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

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

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

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

$$\lim_{n\to\infty} = \frac{\sqrt{n}}{\log n} = \text{$\sqrt{n}$ grows more than $\log n$} \;\to\; = \infty$$

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

QUESTION

3) What is the time complexity of the following programs? Use most appropriate asymptotic notation. Explain by giving details.

a)

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

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)

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

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;
}
```

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]);
}
```

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))
}
```

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;
        }
}
```

h)

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

i)

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

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;
      }
}
```

ANSWER

3-)

a)

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

There is a if else statement. In if statement and else statement. in else statement. i does not increase linearly. it increases i^2. therefore, $\Theta(\log_2 n)$

b)

```
int p_2 (int my_array[]){     T(n)
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]!=first_element){
        second_element= my_array[i];
    }
}
}
```

All statements that are in loop take constant time $\Theta(1)$. loop runs n time other statements run constant time as well. $\Theta(1)+\Theta(1)+\Theta(n)=\Theta(n)$

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

c) int p_3 (int array []){ T(n)
return array[0] * array[2];
}

This is a multipication. It runs constant time $\Theta(1) = T(n)$

d) int p_4 (int array[], int n){ T(n)

```
a      int sum=0;
   for (int i=0; i<n; i=i+5)
b              sum += array[i] * array[i]
c        return sum;
   }
```

a and c are constant. they are running constant $\Theta(1)$. b is also constant $\Theta(1)$ but it works a loop n times
$\Theta(n) + \Theta(1) + \Theta(1) = \Theta(n)$
$T(n) = \Theta(n)$

e) void p_5 (int array [], int n){ T(n)

```
    for (int i=0; i<n; ++i){
        for (int j=1; j<i; j=j*2)
            Printf("%d", array[i]*array[i]);
    }
```

inner loop: $\underbrace{1 \times 2 \times 2 \times 2 \ldots}_{k}$ $j = n$

$2^k j = n$ $k = \log_2 n$

outer loop: $\underbrace{1+2+3+4}_{k}$ $k = n$

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

f) int p_6 (int array[], int n){

```
    if (p_4 (array, n) > 1000)            ①
              p_5 (array, n)              ①
    else
        printf("%d", p_3 (array) * p_4 (array, n))   ③
```

There is an if else statement and we need to find best and worst case

Subject :

Date :....../....../......

① → $\Theta(n)$. 2 → $\Theta(\log n \cdot n)$

③ → $\Theta(1.n) = \Theta(n)$ $T_{Best}(n) = \Theta(n)$

$T_{worst}(n) = \Theta(n) + \Theta(n \log n) = \Theta(\log n \, n)$

g)

```
int p_7 (int n){      T(n)
    int i=n;          a
while(i>0){
  for(int j=0; j<n; j++)
        System.out.println("*");
     i=i/2;
  }
}
```

a is assign and operator
it takes constant time $\Theta(1)$
inner loop runs n time
but outer loop works $\log_2 n$
$T(n) = \Theta(n.\log_2 n)$

h)

```
int p_8 (int n){   T(n)
    while(n>0){
      for(int j=0; j<n; j++);
          n=n/2;
    }
}
```

this is not the same the other question;
this question, we decrease n
Now we are looking inner loop
j goes to : n., n/2, n/4, n/8

$$n\underbrace{\left(1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}\right)}_{2}$$

$T(n) = 2 \cdot n = T(n) = O(n)$

i)

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

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1)+1 & n > 0 \end{cases}$$

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

Assume $n = k$

$T(0) + n$

$T(n) = O(n+1)$

$T(n) = O(n)$

Subject :

j

```
int p_10 (int A[], int n){   T(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;
    }
}
```

$$T(n)=\begin{cases} 1 & , n=1 \\ T(n-1)+n+1, & n>1 \end{cases}$$

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

1. step

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

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

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

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

3. step

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

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

4. step

k. step

$T(n)=T(n-k)+k.n+((-(k^2)+3n)/2))$

let assume $n-k=1$

$k=n-1$

$T(1)+n^2-n \quad \frac{-n^2+n-1}{2}$

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

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

QUESTION

4)

a) Explain what is wrong with the following statement. “The running time of algorithm A is at least $O(n^2)$”.

b) Prove that clause true or false? Use the definition of asymptotic notations.

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

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

III. Let $f(n)=O(n^2)$ and $g(n)= \Theta(n^2)$. Prove or disprove that: $f(n) * g(n) = \Theta(n^4)$

ANSWER

4-)

a) It is wrong statement because Big oh notation provides an "upper bound" for the function. If this statement was "The running time of algorithm A is at max $O(n^2)$", it would be true.

b-)

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

$2^{n+1} = \Theta(2^n)$ $\quad c_1 2^n \geq 2^{n+1} \geq c_2 2^n, \; c_1, c_2 > 0$

Now I assign 1 to n $\quad n=1$

$c_1 2 \geq 4 \qquad 4 \geq c_2 \cdot 2$

$c_1 \geq 2 \qquad 2 \geq c_2$

$n \geq n_0 = 1, \; c_1 = c_2 = 2$ True

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

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

$c_1 2n \geq 2^{2n} \geq c_2 \cdot 2n, \; c_1, c_2 > 0$

$c_1 \geq \frac{2^{2n-1}}{n}$, false since c_1 is constant $\quad$ False

3. $f(n) = O(n^2)$ means that $f(n) \leq cn^2$

$g(n) = \Theta(n^2)$ means that there is a strict when we multiply them $(f(n) \cdot (g(n))$ loses certanity so this is Big oh (n^4) not teta $\quad$ False

QUESTION

5) Solve the following recurrence relations. Express the result in most appropriate asymptotic notation. Show details of your work.

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

b) T(n) = 2T(n – 1) + 1,   T(0)=0

ANSWER

5)

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

$T(n) = 2T(n/2) + n$ 1. step

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

$T(n) = 2[2T(n/2^2) + \frac{n}{2}] + n$

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

$T(n/4) = 2T(n/2^3) + n/4$

$T(n) = 4[2T(n/2^3) + \frac{n}{4}] + 2n$ 3. step

$T(n) = 2^3 T(n/2^3) + 3n$ 3. step

k. step

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

Let assume $\frac{n}{2^k} = 1$ $\log_2 n = k$

$2^{\log_2 n} T(1) + n.\log_2 n = n.\log n$

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

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

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

1. step

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

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

2. step

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

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

3. step

k. step

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

$2^k - 1$

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

let assume $n = k$

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

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

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

6-)

QUESTION

6) In an array of numbers (positive or negative), find pairs of numbers with the given sum. Design an iterative algorithm for the problem. Test the algorithm with different size arrays and record the running time. Calculate the resulting time complexity. Compare and interpret the test result with your theoretical result.

ANSWER

Assume that n is size of array.

İf statement runs constant time because this is only a increment operation

Inner loop runs n times Θ(n*1) = Θ(n);

Outter loop runs n times Θ(n*n) = Θ(n^2).

You can see code below.

```java
static int checkPairs(int[] arr1, int value) {
    int count=0;
    for(int i = 0; i < (arr1.length - 1); ++i) {
        for(int j = (i + 1); j < arr1.length; ++j) {
            if (arr1[i] + arr1[j] == value) {
                ++count;
            }
        }
    }
    return count;
}
```

$T(n)$

$\Theta(n)$

$\Theta(1)$

$\Theta(n)$

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

Theoretically, Time complexity is Θ(n^2).

```java
int[] arr2 ;
arr2 = new int[10000] ;
arr2[0]=10;
arr2[1]=3;
arr2[2]=5;
arr2[3]=2;
arr2[4]=4;
arr2[5]=-3;
int i;
for(i=6;i< arr2.length;++i){
    arr2[i]=i+10;
}
long startTime = System.currentTimeMillis();
System.out.println(checkPairs(arr2,  value: 7));
long endTime   = System.currentTimeMillis();
long totalTime = endTime - startTime;
System.out.println("Running time "+totalTime);
```

I have 2 test cases one of them is that : Array size is 10000 the other one is that : Array size is 100000

When I run this program it gives 34 ms.

```java
int[] arr2 ;
arr2 = new int[100000] ;
arr2[0]=10;
arr2[1]=3;
arr2[2]=5;
arr2[3]=2;
arr2[4]=4;
arr2[5]=-3;
int i;
for(i=6;i< arr2.length;++i){
    arr2[i]=i+10;
}
long startTime = System.currentTimeMillis();
System.out.println(checkPairs(arr2,  value: 7));
long endTime   = System.currentTimeMillis();
long totalTime = endTime - startTime;
System.out.println("Running time "+totalTime);
```

However, It gives 4271 ms.

If I look theoretically, one of them 34 ms the other one will be 3400 ms.

But there are a lot of conditions that effect running time directly(compiler ,OS).

QUESTION

7) Write a recursive algorithm for the problem in 6 and calculate its time complexity. Write a recurrence relation and solve it.

ANSWER

```java
static void checkPairs(int[] arr1, int value,int temp) {        T(n)
    if (temp == 0) {  — 1
    } else {
        for (int j = temp ; j >=0; --j) {
            if (arr1[temp] + arr1[j] == value) {
                System.out.println(arr1[temp]+" + "+arr1[j]+" = "+value);  1      n
            }
        }
        checkPairs(arr1, value, --temp);   T(n-1)
    }
}
```

7-1

$$T(n) = \begin{cases} 1 & , n = 0 \\ T(n-1) + n & , n > 0 \end{cases}$$

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

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

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

$T(n) = T(n-4) + 4n - 6$ 4. step

$\vdots$

k

$T(n) = T(n-k) + kn - \frac{k \cdot k-1}{2}$

1. step

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

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

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

let assume $n = k$

$T(0) + n^2 - \frac{n^2 - n}{2}$

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

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