md Sarwar Hossain
id: 161044121

(1)

(a)

$f(n) = \log_2 n^2 + 1$

$g(n) = n$

According to definition of Big(O),

$O(g(n)) = \{ f(n)$: There exists a positive constant n_0 & c such that, $f(n) \leq cg(n)$ for all $n \geq n_0$

Now,

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

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

$\Rightarrow \frac{2\log n + 1}{n} \leq c$ or,

As $n \to \infty$ Left hand sind will be 0. So, $c \geq 0$

If we take $c = 2$ and $n_0 = 1$

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

for, $n = 1$, $0 + 1 \leq 2$ is valid

for $n = 2$, $1 + 1 \leq 4$ is valid

for $n = 8$, $3 + 1 \leq 16$ is valid

for all $n \geq n_0$, $2\log_2 n + 1 \leq cn$ is valid where $c = 2$ & $n_0 = 1$ because logarithmic value will decrease much faster.

So, $\log_2 n + 1 = O(n)$ is true.

(b)

$f(n) = \sqrt{n(n+1)}$

$g(n) = n$

According to definition of Omega Ω,

$\Omega(g(n)) = \{ f(n)$: There exists a positive constant n_0 and c such that $c\,g(n) \leq f(n)$ for all $n \geq n_0$

$cn \leq \sqrt{n^2 + n}$

$\Rightarrow (cn)^2 \leq n^2 + n$

Let's assume $c = 1$,

from the equation, $(cn)^2 \leq n^2 + n$ as $c = 1$ so it won't have any effect on $(cn)^2$ as, $1^2 = 1$.

Now,

$n^2 \leq n^2 + n$ is always true for any $n \geq n_0$ when $n_0 = 1$ as the right hand side will be always ahead by n value.

So, we can say that, for all $n \geq n_0$ where $n_0 = 1$ there is a constan $c = 1$ for which the equation,

$cn \leq \sqrt{n(n+1)}$ is satisfied.

So, $\sqrt{n(n+1)} = \Omega(n)$ is true.

(c) $f(n) = n^{n-1}$

$g(n) = n^n$

According to definition of Theta $\Theta(n)$,

$\Theta(g(n)) = \{ f(n)$: There exists a positive constant c_1, c_2 and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$

Let's find for Big O (n),

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

Let's assume, $c_2 = 1$ & $n_0 = 1$

Now, for $n = 1$,

$1 \leq 1$ which is valid,

for $n = 2$,

$2^1 \leq 2^2$. whic is valid,

Since for any value of $n \geq n_0$, $n^{n-1} \leq n^n$ is valid when constant $c_2 = 1$, we can say, there is a upper bound for,

$n^{n-1} \leq c_2 n^n$ for constant $c_2 = 1$ and $n \geq n_0$ where $n_0 = 1$

so, $f(n) \leq c_2 g(n)$ is true,

lets find, $\Omega(n)$,

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

$$c_1 \leq \frac{n^{n-1}}{n^n}$$

$$c_1 \leq \frac{1}{n}$$

As, n goes to infinity the right hand side will be lower than the constant c_1. So there cannot be any constant for which,

$c_1 n^n \leq n^{n-1}$ for $n \geq n_0$.

so, $c_1 n^n \leq n^{n-1}$ is false.

So, we can say,

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

(3) we will group and compare the functions using limit method.

Group 1: $\lim_{n\to\infty} \frac{n^2}{n^3} = 0$ so, $n^2 < n^3$

Group 2: $\lim_{n\to\infty} \frac{n^2 \log n}{\sqrt{n}} = \infty$ so, $\sqrt{n} < n^2 \log n$

Group 3: $\lim_{n\to\infty} \frac{\log n}{10^n} = 0$ so, $\log n < 10^n$

Group 4: $\lim_{n\to\infty} \frac{2^n}{8^{\log n}} = \infty$ so, $8^{\log n} < 2^n$

Group 5: Group 5 is between Group 1 & Group 2

$\lim_{n\to\infty} \frac{n^3}{\sqrt{n}} = \infty$ $n^3 > \sqrt{n}$, $\lim_{n\to\infty} \frac{n^2}{\sqrt{n}} = \infty$ $n^2 > \sqrt{n}$

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

So, $\sqrt{n} < n^2 < n^2 \log n < n^3$

Group 5: Group 6 is between Group 3 & 4

$$\lim_{n\to\infty} \frac{10^n}{8^{\log n}} = \infty \quad 10^n > 8^{\log n}$$

$$\lim_{n\to\infty} \frac{\log n}{8^{\log n}} = 0, \quad 8^{\log n} > \log n$$

$$\lim_{n\to\infty} \frac{2^n}{10^n} = 0, \quad 10^n > 2^n$$

So, $10^n > 2^n > 8^{\log n} > \log n$

Finally: between Group 5 & 6,

$$\sqrt{n} < n^2 < n^2 \log n < n^3 \quad \& \quad \log n < 8^{\log n} < 2^n < 10^n$$

$$\lim_{n\to\infty} \frac{n^3}{\log n} = \infty, \quad n^3 > \log n, \quad \lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = 0, \quad \log n < \sqrt{n}$$

At this point we have,

$$\log n < \sqrt{n} < n^2 < n^2 \log n < n^3$$

Again,

$$\lim_{n\to\infty} \frac{n^3}{8^{\log n}} = 1, \text{ so, } n^3 = 8^{\log n}, \quad \lim_{n\to\infty} \frac{n^3}{2^n} = 0, \text{ so, } n^3 > 2^n$$

So, the answer is,

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

(3)

(a)

```
for (int i = 2 ; i < n; i++)
        if (i % 2 == 0)
                count++;
        else
                i = (i-1) * i
```

for the else block i is increasing as $i^2 - i$

so we are rewriting the for loop as,

int = 2 ; i < n ; x where $x = i^2 - i$

We can further ignore $-i$ as i^2 grows much faster.

when i > n, the loop will stop execution,

Since i increases much faster we say,

$i = 2^{2^k}$

Now, $2^{2^k} = n$

$\Rightarrow 2^k = \log n$

$\Rightarrow k = \log(\log n)$

So, the time complexity of this function, $O(\log(\log n))$.

(b)

	step / freq / total
first_element = arr(0)	1 / 1 / 1
second_element = arr(0)	1 / 1 / 1
for (i = 0; i < arraySize; i++)	2 / n+1 / 2n+2
if (array(i) < firstelement){	1 / n / n
second_element = first_element;	1 / n / n
first_element = array(i);	1 / n / n
}else if (array(i) < second_element){	1 / n / n
if (array(i) != first_element)	1 / n / n
second_element = array(i)	1 / n / n
}	Total = 8n+4

$f(n) = 8n + 4$

As the loop will always run for n times. We can best represent the running time using $\Theta(n)$.

(c)

```
int f3 (int array[]){
  return array[0] * array[2];
}
```

The best & worst both the time complexity for this function will be at constant time. Since retrival of data from array takes constant time. so we can best represent the algorith using $\Theta(1)$.

(d)

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

i value increase in a order of,

0, 5, 5+5, 5+5+5 or we can say 5i

Now the loop will stop when, $5i \geq n$ considering on the equal case we get,

$5i = n$

$i = \frac{n}{5}$, we can ignore the constants time complexitie in the function. As there are no other stop condition. we can say time complexity is, $\Theta(n)$.

(e)

```
for (i=0 ; i<n ; i++) ——— n
    for (j= 1 ; j<i; j = j* 2) — n log (i)
        print (" ");
```

Time complexity of this function will depend upon $n \log(i)$ where $i = n$. So we can say time complexity of this function is $O(n \log n)$.

(f)

P_4 function has time complexity $\Theta(n)$
P_5 has time complexity $O(n \log n)$
P_3 has time complexity $\Theta(1)$

```
P_6 (array, n)
    if (P_4(array, n)) > (1000) — O(n)
        P_5 (array, n) —
    else print (P_3 (array) * P_4 (array_n))
```

The if condition has time complexity of $\Theta(n) + O(n \log n)$
The else condition has time complexity of $\Theta(1) + \Theta(n)$
So, the best case is for the else case, $\Omega(n)$ and worst case is, $O(n \log n)$.
We can best describe the time complexity to be $O(n \log n)$

(g)

```
int P_7 (int n){
    int i= n;
    while (i>0) {
        for (j=0 ; j<n ; j++)
            print ( );
        i/2;
    }
}
```

The while loop will run approximately of $\frac{n}{2^k}$ times.
So, the while loop has a complexity of $\frac{n}{2^k} = 1$ or, $\log(n)$.

The inner loop will run for $n * \log(n)$ times.
So, the time complexity will be $O(n \log n)$

(h)

```
P_8 (int n)
    while (n > 0)
        for (j = 0; j < n; j++)
            printf();
    n = n/2;
```

The outer while loop runs for $\frac{n}{2^k}$ times. The time complexity of outer loop is, $\log(n)$.
Now the inner for loop will run for $\frac{n}{2^k}$ at each iteration and bounded by $\log(n)$ running time. So we can say $\int_0^{\log n} \frac{n}{2^k} dk = n$ or the running time will be $O(n)$.

Time complexity, $O(n)$.

(i)

```
P_9 (n)
    if (n = 0)
        return 1
    else
        return n * P_9 (n-1)
```

$T(0) = 1$

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

Now,

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

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

$= T(n-k) + kC$ ——— (1)

According to base condition,

$n - k = 0$

$k = n$

$T(n) = T(0) + nC$ from eq (1)

$= 1 + nC$

So, the time complexity is $O(n)$.

(j)

```
P_10 (int A[], int n)
    if(n==1)
        return;
    P_10 (A, n-1);
    j = n-1;
    while( j>0 && A[j] <A[j-1]) {
        swap (a[j], A[j-1]);
        j = j-1;
```

These function has two part. One is recursive and after the recursive part a while loop. The running time will be the result of iterative and recursive part.

For the recursion part,

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

$T(1) = 1$

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

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

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

$= T(n-k) + kc$ —①

Now, $n-k=1$

$k = n-1$

substituting in ①,

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

$= 1 + nc - c$

Or, we can say $O(n)$.

Now, the while loop will depend on n value. And swap will be constant time.

For n value of 5, j will will , 4, 3, 2, 1 & recursion will act as an outer loop. So, the time complexity will be $O(n^2)$.

(4)

(a) From the definition of Big Oh (O) we know that Big O gives us the upper bound of running time of an algorithm. When we say time complexity is $O(n^2)$, we will mean the minimum running time or lower bound of the algorithm is $O(n^2)$. This contradicts the main definition of Big O. That's why the statement is wrong.

(b)

(i) $f(n) = 2^{n+1}$

$g(n) = 2^n$

$\Theta(g(n))$ means, $\{ f(n)$: there exists a positive constant c_1, c_2 & n_0 such that, $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0 \}$

case Big O,

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

$c_2 \geq \frac{2^n \cdot 2}{2^n}$

$c_2 \geq 2$

if $c_2 = 2$, $2^n \cdot 2 \leq 2 \cdot 2^n$ is always true for all $n \geq n_0$ where $n_0 = 0$.

So, 2^{n+1} is $O(2^n)$

Now for Big Ω,

$c_1 2^n \leq 2^n \cdot 2$

$c_1 \leq 2$

for $c_1 = 2$,

$2 \cdot 2^n \leq 2^n \cdot 2$ is always true for all $n > n_0$ where $n_0 = 0$

So, 2^{n+1} is $\Omega(2^n)$

So, we can say, 2^{n+1} is $\Theta(2^n)$ is true.

(ii)

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

$g(n) = 2^n$

for $\Theta(g(n))$ we have to prove, $c_1 g(n) \leq f(n) \leq c_2 g(n)$.

where c_1, c_2 & n_0 are positive constant and $n \geq n_0$

Case Big O,

$$2^{2n} \leq c_2 2^n$$

$$\Rightarrow c_2 \geq \frac{2^{2n}}{2^n}$$

$$\Rightarrow c_2 \geq 2^{2n} \cdot 2^{-n}$$

$$\Rightarrow c_2 \geq 2^n$$

So,

As, $n \to \infty$, right hand side will go to infinity.

So, there can't be any such constant c_2.

Thus we can say, 2^{2n} in $\Theta(2^n)$ is false.

(iii) $f(n) = O(n^2)$

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

$F(n) = f(n) * g(n)$

$G(n) = \Theta(n^4)$

$\Theta(G(n))$ iff $(F(n))$ such that there is c_1 & c_2 & n_0 positive constant and $c_1 G(n) \leq F(n) \leq c_2 G(n)$

For Big O: $F(n) \leq c_2 G(n)$ or, $f(n) * g(n) \leq c_2 n^4$ — (i)

If we take $c_2 = 1$ & $n_0 = 1$ then for any value of $n \geq n_0$ eqn (i) is valid. So, $F(n)$ is $O(n^4)$

For Big Ω: $c_1 G(n) \leq F(n)$ or, $c_1 n^4 \leq f(n) * g(n)$ — (ii)

If we again take, $c_1 = 1$ & $n_0 = 1$ then for any value $n \geq n_0$, eqn (ii) is valid.

So, $F(n)$ is $\Omega(n^4)$

Thus, $F(n)$ is $\Theta(n^4)$

so, we can say, $f(n) * g(n)$ is $\Theta(n^4)$ is true.

(5)

(a)

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

$T(1) = 1$

Now,

$$T(n) = 2T(n/2) + n$$
$$= 2^2 T\left(\frac{n}{2^2}\right) + 2n$$
$$= 2^3 T\left(\frac{n}{2^3}\right) + 3n$$
$$= 2^k T\left(\frac{n}{2^k}\right) + kn \quad \text{—(1)}$$

Now, $T\left(\frac{n}{2^k}\right) = T(1)$

or, $\frac{n}{2^k} = 1$, $n = 2^k$

So, $k = \log n$

From eqn. (1),

$2^k T(1) + kn$

$\Rightarrow n + n \log n$. $[\because n = 2^k]$

So, Time complexity of $T(n)$ is $O(n \log n)$.

(b) $T(0) = 0$

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

$= 4T(n-2) + 2$ $\quad = 2T(n-2) + 1$

$= 8T(n-3) + 3$

$= 2^k T(n-k) + k$ —(1)

From base condition,

$n - k = 0$ or, $n = k$

From eqn (1), $2^n \cdot 0 + n = n$

So, time complexity of $T(n)$ is $O(n)$.

⑥
```
int pairfromSum (int array[], int size, int sum){
    for(int i=0; i< size; i++){
        for(int j= i+1; j <size;j++){
            if(array[i]+ array[j] == sum)
                return 1;
        }
    }
    return 0;
}
```

The outer for loop will run for n times.
The inner for loop will run for $(n-1), (n-2) \ldots 1$ times.
So, the time complexity in $O(n^2)$.

After testing the algorithm for array of size, 10, 100 & 1000 the recorded running time approximately, 15 microsecond, 400 microsecond and 5400 microsecond in worst case scenerio when the pair is not found.

So, theoritically if n = 10 running time = 100
n = 100 running time = 10000

So, from n = 100 takes 10 times more time than n = 10
From recorded result, n = 100 took 400/15 or, 26 times more than n = 10

Again, n = 1000 should take 10 times more time than n = 100
From recorded time we got, n = 1000 in 13'5 times more than n = 100.
So, the results are pretty approximate.

```
sarwar@sarwar-ThinkPad-T470s:~/Desktop/playground$ ./a.out
pair not found in array size 100
elapsed time size 100 : 402.000000
pair not found in array size 10
elapsed time for array size 10 : 15.000000
pair not found in array size 1000
elapsed time 1000 : 5405.000000
```

(7)

```
pairfromsumrecursive(int array[], int size, int index, int sum){
    if(!(index < 0)){
        pairfromsumrecursion(array, size, index-1, sum);
        for(int i= index+1; i<size; i++){
            if(array[i] + array[index] = sum){
                printf("pair found at %d %d\n", i, index);
            }
        }
    }
}
```

For the recursive part,

$T(-1) = 0$

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

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

$= T(n-k) + kc$

$n - k = -1$

$\Rightarrow k = n+1$

So, $T(n-k) + kc$

$\Rightarrow n+1 + (n+1)c$

we can say time complexity is $O(n)$.

Now for the while loop i value depend on n. where the i for will run for $n(n+1)/2$ times.

So, the time complexity of the function will be $O(n^2)$.

Edit Selection View Go Run Terminal Help

C test.c

home > sarwar > Desktop > playground > C test.c > main()

```
49      //array 100
50      init = clock();
51      (pairFromSum(array_100,100,sum))?printf("pair found in array size 100\n") : printf("pair not found in array size 100\n");
52      end = clock();
53      time_size_100 = (double)(end - init);
54      printf("elapsed time size 100 : %f\n", time_size_100 );
55      //printf("size_100 took %d percent more time than size_10\n",(int)((time_size_100/time_size_10)*100));
56
57
58      //array size 10
59      init= clock();
60      (pairFromSum(array_10,10,sum))?printf("pair found in array size 10\n") : printf("pair not found in array size 10\n");
61      end = clock();
62      time_size_10 = (double)(end - init);
63      printf("elapsed time for array size 10 : %f\n", time_size_10 );
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

bash - playground

```
sarwar@sarwar-ThinkPad-T470s:~/Desktop/playground$ ./a.out
pair not found in array size 100
elapsed time size 100 : 591.000000
pair not found in array size 10
elapsed time for array size 10 : 47.000000
pair not found in array size 1000
elapsed time 1000 : 10623.000000
sarwar@sarwar-ThinkPad-T470s:~/Desktop/playground$ ./a.out
pair not found in array size 100
elapsed time size 100 : 97.000000
pair not found in array size 10
elapsed time for array size 10 : 4.000000
pair not found in array size 1000
elapsed time 1000 : 1342.000000
sarwar@sarwar-ThinkPad-T470s:~/Desktop/playground$ ./a.out
pair not found in array size 100
elapsed time size 100 : 680.000000
pair not found in array size 10
elapsed time for array size 10 : 34.000000
pair not found in array size 1000
elapsed time 1000 : 8610.000000
sarwar@sarwar-ThinkPad-T470s:~/Desktop/playground$ ./a.out
pair not found in array size 100
elapsed time size 100 : 623.000000
pair not found in array size 10
elapsed time for array size 10 : 17.000000
pair not found in array size 1000
elapsed time 1000 : 10578.000000
sarwar@sarwar-ThinkPad-T470s:~/Desktop/playground$
```