

Solution 1:

Divide the program in fragments

5 new How to Calculate Time Complexity of an Algorithm - Solved Questions (With Notes): (1080p).mp4 - VLC media player

Media Playback Audio Video Subtitle Tools View Help

Questions - OneNote

Time Complexity Practice Set

1. Find the time complexity of the func1 function in the program show in program1.c as follows:

```
#include <stdio.h>

void func1(int array[], int length)
{
    int sum = 0;
    int product = 1;
    for (int i = 0; i < length; i++)
    {
        sum += array[i];
    }

    for (int i = 0; i < length; i++)
    {
        product *= array[i];
    }
}

int main()
{
    int arr[] = {3, 5, 66};
    func1(arr, 3);
    return 0;
}
```

Handwritten annotations on the code:

- f_1 next to the initialization of `sum` and `product`.
- f_2 next to the first loop.
- f_3 next to the second loop.
- Checkmarks next to the array declaration and the function call.

5 new How to Calculate Time Complexity of an Algorithm - Solved Questions (With Notes): (1080p).mp4 - VLC media player

Media Playback Audio Video Subtitle Tools View Help

Questions - OneNote

Time Complexity Practice Set

1. Find the time complexity of the func1 function in the program show in program1.c as follows:

```
#include <stdio.h>

void func1(int array[], int length)
{
    int sum = 0;
    int product = 1;
    for (int i = 0; i < length; i++)
    {
        sum += array[i];
    }

    for (int i = 0; i < length; i++)
    {
        product *= array[i];
    }
}

int main()
{
    int arr[] = {3, 5, 66};
    func1(arr, 3);
    return 0;
}
```

Handwritten annotations on the code:

- $f_1 = k_1$ next to the initialization of `sum` and `product`.
- $f_2 = k_2 n$ next to the first loop.
- $f_3 = k_3 n$ next to the second loop.
- Checkmarks next to the array declaration and the function call.

Handwritten derivation of time complexity:

$$\begin{aligned} T_n &= f_1 + f_2 + f_3 \\ &= k_1 + k_2 n + k_3 n \\ &\Rightarrow (k_2 + k_3) n \\ &= k_4 n \rightarrow O(n) \\ &O(\text{length}) \end{aligned}$$

Solution 2:

Time Complexity

2. Find the time complexity of the func function in the program from program2.c as follows:

```
void func(int n)
{
    int sum = 0;
    int product = 1;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("%d ", %d\n", i, j);
        }
    }
}
```

Handwritten analysis for Question 2:

$O(\text{length})$

$\rightarrow O(n)$

$\rightarrow O(n^2)$

$\rightarrow [n + n + n + \dots + (n-1)n]k_2$

$n k_2 (1 + 1 + \dots + 1) = k_2 n^2$

$O(n^2)$

3. Consider the recursive algorithm above, where the random(int n) spends one unit of time to return a random integer which is evenly distributed within the range [0,n][0,n]. If the average processing time is T(n), what is the value of T(n)?

```
int function(int n)
{
    int i;
    if (n <= 0)
    {
        return 0;
    }
    else
    {
        i = random(n - 1);
        printf("this\n");
        return function(i) + function(n - 1 - i);
    }
}
```

4. Which of the following are equivalent to $O(N)$? Why?

a) $O(N + P)$, where $P < N/2$

b) $O(2N-k)$

c) $O(N + \log N)$

Solution 3:

Time Complexity Practice Set

random integer which is evenly distributed within the range $[0, n]$. If the average processing time is $T(n)$, what is the value of $T(6)$? [-]

$random(6) \rightarrow [0, 6]$

$[0, 5]$

$O(n^2)$

n) $\rightarrow k=0$

```

function(n-1);
this(n);
function(i) + function(n-1-i);

```

are equivalent to $O(N)$? Why?

are $P < N/9$

code sums the values of all the nodes in a balanced binary search tree. What is its

```

node)
== NULL)
n 0;

```

Time Complexity Practice Set

random integer which is evenly distributed within the range $[0, n]$. If the average processing time is $T(n)$, what is the value of $T(6)$? [-]

$random(6) \rightarrow [0, 6]$

$[0, 5]$

$O(n^2)$

```

int function(int n)
{
    int i;  $\rightarrow k=0$ 
    if (n <= 0)
    {
        return 0;
    }
    else
    {
        i = random(n-1);  $\rightarrow 1$ 
        printf("this\n");
        return function(i) + function(n-1-i);
    }
}

```

4. Which of the following are equivalent to $O(N)$? Why?

- $O(N + P)$, where $P < N/9$
- $O(9N-k)$
- $O(N + 8 \log N)$
- $O(N + M^2)$

5. The following simple code sums the values of all the nodes in a balanced binary search tree. What is its runtime?

```

int sum(Node node)
{
    if (node == NULL)
    {
        return 0;
    }
}

```

$= 6$

Time Complexity

3. Consider the recursive algorithm below, where the random(int n) spends one unit of time to return a random integer which is evenly distributed within the range [0, n]. If the average processing time is $T(n)$, what is the value of $T(6)$? $(-)$

random(6) \rightarrow [0, 6]

```
int function(int n)
{
    int i;  $\rightarrow k_1 = 0$ 
    if (n <= 0)
    {
        return 0;
    }
    else
    {
        i = random(n - 1);  $\rightarrow 1$ 
        printf("this is n");
        return function(i) + function(n - 1 - i);
    }
}
```

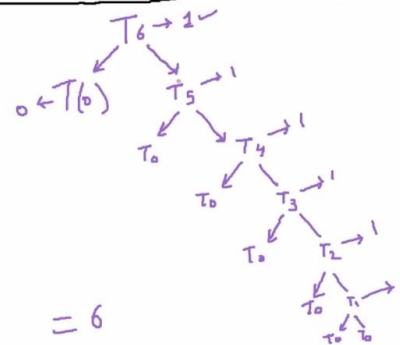
4. Which of the following are equivalent to $O(N)$? Why?

- $O(N + P)$, where $P < N/9$
- $O(9N - k)$
- $O(N + 8 \log N)$
- $O(N + M^2)$

5. The following simple code sums the values of all the nodes in a balanced binary search tree. What is its runtime?

```
int sum(Node node)
{
    if (node == NULL)
    {
        return 0;
    }
    return sum(node.left) + node.value + sum(node.right);
}
```

$$n k_2 \underbrace{(1 + 1 + \dots + 1)}_{n \text{ times}} = k_2 n^2$$



Solution 4:

Here M is variable not a constant

Time Complexity Practice Set

4. Which of the following are equivalent to $O(N)$? Why? (k is a constant)

- ✓ a) $O(N + P)$, where $P < N/9 \rightarrow O(N)$
- ✓ b) $O(N - k) \rightarrow O(N)$ (N is a constant)
- ✓ c) $O(N + 8 \log N) \rightarrow O(N)$ ($\log N$)
- ✗ d) $O(N + M^2) \rightarrow$

5. The following simple code sums the values of all the nodes in a balanced binary search runtime?

```
int sum(Node node)
{
    if (node == NULL)
```

Solution 5:

Time Complexity

4. Which of the following are equivalent to $O(N)$? Why? (It's a function)

- a) $O(N + P)$, where $P < N/9 \rightarrow O(N)$
- b) $O(N \cdot N) \rightarrow O(N^2)$
- c) $O(N + 8 \log N) \rightarrow O(N)$
- d) $O(N + M^2) \rightarrow O(N)$

5. The following simple code sums the values of all the nodes in a balanced binary search tree. What is its runtime? (n is the no of nodes)

```
int sum(Node node)
{
    if (node == NULL)
    {
        return 0;
    }
    return sum(node.left) + node.value + sum(node.right);
}
```

6. Find the complexity of the following code which tests whether a give number is prime or not?

```
int isPrime(int n){
    if (n == 1){
        return 0;
    }

    for (int i = 2; i * i < n; i++) {
        if (n % i == 0)
    }
```

$\Rightarrow O(n)$

$= 6$

Solution 6:

Time Complexity

```
{
    return 0;
}
return sum(node.left) + node.value + sum(node.right);
}
```

6. Find the complexity of the following code which tests whether a given number is prime or not?

```
int isPrime(int n){
    if (n == 1){
        return 0;
    }

    for (int i = 2; i * i < n; i++) {
        if (n % i == 0)
            return 0;
    }

    return 1;
}
```

Handwritten notes and diagrams:

k_1 and k_2 are marked next to the `if (n == 1)` and `for` loops respectively. The total complexity is noted as $k_1 + k_2$.

The complexity of the `for` loop is derived as $O(\sqrt{n})$. The loop runs from $i = 2$ to $i = \sqrt{n}$. The number of iterations is $\sqrt{n} - 1$. The total complexity is $O(\sqrt{n})$.

Diagram illustrating a binary tree structure with nodes labeled $0(n)$ and $1(n)$. The tree is rooted at $0(n)$ and branches into $1(n)$ and $0(n)$. The nodes are labeled $0(n)$ and $1(n)$. The tree is rooted at $0(n)$ and branches into $1(n)$ and $0(n)$. The nodes are labeled $0(n)$ and $1(n)$.

7. What is the time complexity of the following snippet of code?

Solution 7:

The screenshot shows a OneNote page titled "Questions - OneNote". The page content includes a question and a code snippet. Handwritten red annotations provide the solution.

7. What is the time complexity of the following snippet of code?

```
int isPrime(int n){  
    for (int i = 2; i * i < 10000; i++) {  
        if (n % i == 0)  
            return 0;  
    }  
    return 1;  
}  
isPrime(0);
```

Handwritten red annotations:

- Top right: $i = \sqrt{n}$ and $O(\sqrt{n})$ (crossed out).
- Next to the for loop: k_1 and $T_n = k_1 \rightarrow O(1)$.

It is running in constant time because "for" loop doesn't depend on n . It always run for 10000 times i.e. const. So it is $O(1)$