Solution for homework CS112

Analysis of complexity of recursive algorithms.

P. H. Phước[1] N. X. Bách[2]

[1]22521156
KHTN2022

[2]22520093
KHTN2022

October 2023

Table of Contents

Table of Contents

Task 1

Problem

Calculate the complexity of the following recurrence relations:

a)

$$\begin{cases} T(1) = 4 \\ T(n) = 3T(n-1), \forall x > 1 \end{cases}$$

b)

$$\begin{cases} T(1) = 1 \\ T(n) = 2T(\frac{n}{2}) + \frac{n}{2}, \forall x > 1 \end{cases}$$

c)

$$\begin{cases} T(1) = 1 \\ T(n) = 7T(\frac{n}{4}) + n^2, \forall x > 1 \end{cases}$$

Task 1

a)

$$\begin{cases} T(1) = 4 \\ T(n) = 3T(n-1), \forall x > 1 \end{cases}$$

$$\begin{gathered} T(n) = 3T(n-1) \\ T(n-1) = 3T(n-2) \\ T(n-2) = 3T(n-3) \\ \dots \\ T(2) = 3T(1) = 12 \end{gathered}$$

$$\begin{gathered} T(n) = 3T(n-1) \\ T(n) = 9T(n-2) \\ T(n) = 27T(n-3) \\ \dots \\ T(n) = 3^i T(n-i) \end{gathered}$$

Task 1

a)

$$\begin{cases} T(1) = 4 \\ T(n) = 3T(n-1), \forall x > 1 \end{cases}$$

$$\begin{gathered} T(n) = 3T(n-1) \\ T(n-1) = 3T(n-2) \\ T(n-2) = 3T(n-3) \\ \ldots \\ T(2) = 3T(1) = 12 \end{gathered}$$

$$\begin{gathered} T(n) = 3T(n-1) \\ T(n) = 9T(n-2) \\ T(n) = 27T(n-3) \\ \ldots \\ T(n) = 3^i T(n-i) \end{gathered}$$

Replace $i = n - 1$, we have:

$$T(n) = 3^{n-1}T(n - n + 1) = 3^{n-1}T(1) = 3^{n-1}4$$

Task 1

b)

$$\begin{cases} T(1) = 1 \\ T(n) = 2T(\frac{n}{2}) + \frac{n}{2}, \forall x > 1 \end{cases}$$

We will find the upper limit of n (Big O) by assuming that $n = 2^k$.

$$T(n) = 2T(\frac{n}{2}) + \frac{n}{2}$$
$$T(\frac{n}{2}) = 2T(\frac{n}{4}) + \frac{n}{4}$$
$$T(\frac{n}{4}) = 2T(\frac{n}{8}) + \frac{n}{8}$$
$$\ldots$$
$$T(2) = 2T(1) + 1 = 3$$

$$T(n) = 2T(\frac{n}{2}) + \frac{n}{2}$$
$$T(n) = 2 * (2T(\frac{n}{4}) + \frac{n}{4}) + \frac{n}{2}$$
$$T(n) = 4T(\frac{n}{4}) + \frac{n}{2} + \frac{n}{2}$$
$$T(n) = 4 * (2T(\frac{n}{8}) + \frac{n}{8}) + n$$
$$T(n) = 8T(\frac{n}{8}) + \frac{3n}{2}$$
$$\ldots$$

Task 1

b)

$$\begin{cases} T(1) = 1 \\ T(n) = 2T(\frac{n}{2}) + \frac{n}{2}, \forall x > 1 \end{cases}$$

$$T(n) = 2^i T(\frac{n}{2^i}) + \sum_{j=1}^{i} \frac{n}{2}$$

Replace $i = log(n)$, we have:

$$T(n) = 2^{log(n)} T(\frac{n}{2^{log(n)}}) + \sum_{j=1}^{log(n)} \frac{n}{2}$$

$$T(n) = nT(1) + \frac{nlog(n)}{2}$$

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

According to remove the constant and max rules, we got:

$$T(n) \approx O(nlog(n))$$

Task 1

c)

$$\begin{cases} T(1) = 1 \\ T(n) = 7T(\frac{n}{4}) + n^2, \forall x > 1 \end{cases}$$

According to the Master theorem, we got:

$$a = 7, b = 4, d = 2$$
$$\Rightarrow a < b^d$$
$$\Rightarrow T(n) = n^d = n^2$$

Table of Contents

Task 2

Problem

Given the Python code as follows:

```python
def Search(val, left = 0, right = len(b) - 1):
    if left > right:
        return -1
    mid = (left + right) // 2
    if b[mid] == val:
        return mid
    elif b[mid] > val:
        return Search(val, left, mid - 1)
    else:
        return Search(val, mid + 1, right)
```

Assuming that $b[]$ is a sorted increasing array.

a) What the code above is doing and what it outputs ?

b) Determine the base case and the recursive case.

c) Set up the recurrence relations and calculate the complexity.

Task 2

a) What the code above is doing and what it outputs ?
Explain:
The code above is a recursive version of the binary search algorithm. It'll find the index that has a value equal to the `val` variable and return it.

Task 2

b) Determine the base case and the recursive case.

The base case:

```
if left > right:
    return -1

if b[mid] == val:
    return mid
```

Task 2

b) Determine the base case and the recursive case.

The recursive case:

```
elif b[mid] > val:
    return Search(val, left, mid - 1)
else:
    return Search(val, mid + 1, right)
```

Task 2

c) Set up the recurrence relations and calculate the complexity.

Each time, the sequence we got splits into 2 subsequences which have approximate length. Depending on the condition, we can follow the right or left subsequence.
$\Rightarrow$ The size of subproblem is $\frac{n}{2}$
Time to calculate `mid` is $O(1)$

Task 2

c) Set up the recurrence relations and calculate the complexity.

Each time, the sequence we got splits into 2 subsequences which have approximate length. Depending on the condition, we can follow the right or left subsequence.

$\Rightarrow$ The size of subproblem is $\frac{n}{2}$

Time to calculate `mid` is $O(1)$

Then we have this recurrence relation:

$$\begin{cases} T(1) = 1 \\ T(n) = T(\frac{n}{2}) + O(1), \forall x > 1 \end{cases}$$

Task 2

c) Set up the recurrence relations and calculate the complexity.

$$\begin{cases} T(1) = 1 \\ T(n) = T(\frac{n}{2}) + O(1), \forall x > 1 \end{cases}$$

Assuming that $n = 2^k$, then we have:

$$T(n) = T(\frac{n}{2}) + 1 \Rightarrow T(n) = T(\frac{n}{4}) + 2$$
$$\Rightarrow T(n) = T(\frac{n}{8}) + 3$$
$$...$$
$$T(n) = T(\frac{n}{2^{log(n)}}) + log(n)$$
$$T(n) = T(1) + log(n)$$
$$\Rightarrow T(n) = 1 + log(n) \approx O(log(n))$$

Table of Contents

Task 3

Problem

Calculate the minimum time to print N paper for 2 printers. Supposing each printer costs 1 minute to print one side of a paper. Consider the following recursive algorithm:

- If $n \leq 2$, print 1 or 2 papers at the same time on 1 or 2 printers.
- If $n > 2$, print 2 random papers at the same time on 2 printers and continue the process for $n - 2$ remaining papers.

a) Set up recurrence relations to calculate the complexity of the algorithm above.
b) Explain why the algorithm above didn't give the best answer to the problem.
c) Provide a recursive algorithm to give the best answer. Set up the recurrence relations and calculate the complexity.

Task 3

a) According to the hypothesis, when we have $n \leq 2$, then the time it's complete is 2 minutes. If we have more, we just print 2 parallel papers and continue until the base case. Call $F(n)$ as the minimum time to print n paper. we have:

$$\begin{cases} F(1) = F(2) = 2 \\ F(n) = F(n-2) + 2, \forall x > 2 \end{cases}$$

Task 3

a) According to the hypothesis, when we have $n \leq 2$, then the time it's complete is 2 minutes. If we have more, we just print 2 parallel papers and continue until the base case. Call $F(n)$ as the minimum time to print n paper. we have:

$$\begin{cases} F(1) = F(2) = 2 \\ F(n) = F(n-2) + 2, \forall x > 2 \end{cases}$$

Based on the formula above, we have the recurrence relations to calculate the complexity:

$$\begin{cases} T(1) = T(2) = 0 \\ T(n) = T(n-2) + 1, \ \forall x > 2 \end{cases}$$

Task 3

b) We can notice that the amount of time to print all odd number of paper is also equal to an even number.

Examples

For $n = 4$, according to the algorithm above, first we print 2 side of 2 paper, which cost 2 minute. Then we continue to print 2 side of the 2 remaining papers. The total cost is 4 minute.

Examples

For $n = 3$, first we print 2 side of 2 paper, which cost 2 minute. After that, we print 2 side of the remaining paper. The total cost is also 4 minute.

In general, for all odd number n ,$n > 1$, we can see that time we need to print n and $n + 1$ paper is the same.

$\Rightarrow$ Is there any way better than this ?

Task 3

c) Design algorithm:

- If n is an even number, we can process as the algorithm above, just print 2 side of 2 random paper and continue process for $n - 2$ remaining papers.

Task 3

c) Design algorithm:

- If n is an even number, we can process as the algorithm above, just print 2 side of 2 random paper and continue process for $n - 2$ remaining papers.
- If $n = 1$, the fix time to print is 2 minutes.

Task 3

c) Design algorithm:

- If n is an even number, we can process as the algorithm above, just print 2 side of 2 random paper and continue process for $n - 2$ remaining papers.
- If $n = 1$, the fix time to print is 2 minutes.
- If $n > 1$ and n is an odd number, we can print 2 papers. The printer one print one paper with 1 side and the printer two print the other for 2 side. Then, the printer one print the remain side of the paper before, the printer two print the other. Now we have fulfill the timeline by just use 3 paper. The remain $n - 3$ paper can be processed as even case.

Task 3

Call $G(n)$ as the minimum time to print n paper, we have:

$$\begin{cases} G(1) = G(2) = 2 \\ G(3) = 3 \\ G(n) = G(n-2) + 2, \forall x > 3 \end{cases}$$

Task 3

Call $G(n)$ as the minimum time to print n paper, we have:

$$\begin{cases} G(1) = G(2) = 2 \\ G(3) = 3 \\ G(n) = G(n-2) + 2, \ \forall x > 3 \end{cases}$$

And base on that, we have the following recurrence relations to calculate the complexity:

$$\begin{cases} T(1) = T(2) = T(3) = 0 \\ T(n) = T(n-2) + 1, \ \forall x > 3 \end{cases}$$

Thank for listening !

Do you have any quesion ?