Analysis of Algorithm | Set 4 (Solving Recurrences)

In the previous post, we discussed analysis of loops. Many algorithms are recursive in nature. When we analyze them, we get a recurrence relation for time complexity. We get running time on an input of size n as a function of n and the running time on inputs of smaller sizes. For example in Merge Sort, to sort a given array, we divide it in two halves and recursively repeat the process for the two halves. Finally we merge the results. Time complexity of Merge Sort can be written as $T(n) = 2T(n/2) + cn$. There are many other algorithms like Binary Search, Tower of Hanoi, etc.

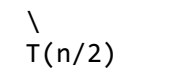
There are mainly three ways for solving recurrences.

1) Substitution Method: We make a guess for the solution and then we use mathematical induction to prove the the guess is correct or incorrect.

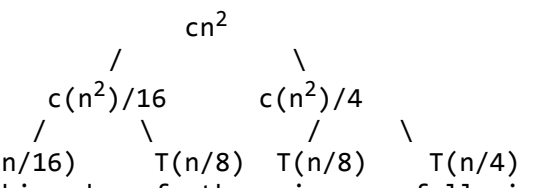
```
For example consider the recurrence T(n) = 2T(n/2) + n

We guess the solution as T(n) = O(nLogn). Now we use induction
to prove our guess.

We need to prove that T(n) <= cnLogn. We can assume that it is true
for values smaller than n.

T(n) = 2T(n/2) + n
    <= cn/2Log(n/2) + n
    =  cnLogn - cnLog2 + n
    =  cnLogn - cn + n
    <= cnLogn
```

2) Recurrence Tree Method: In this method, we draw a recurrence tree and calculate the time taken by every level of tree. Finally, we sum the work done at all levels. To draw the recurrence tree, we start from the given recurrence and keep drawing till we find a pattern among levels. The pattern is typically a arithmetic or geometric series.

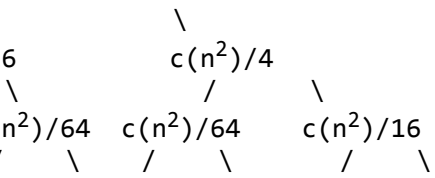
For example consider the recurrence relation
$T(n) = T(n/4) + T(n/2) + cn^2$



If we further break down the expression T(n/4) and T(n/2), we get following recursion tree.



Breaking down further gives us following




```
To know the value of T(n), we need to calculate sum of tree
nodes level by level. If we sum the above tree level by level,
we get the following series
T(n)  = c(n^2 + 5(n^2)/16 + 25(n^2)/256) + ....
The above series is geometrical progression with ratio 5/16.

To get an upper bound, we can sum the infinite series.
We get the sum as (n^2)/(1 - 5/16) which is O(n^2)
```

3) Master Method:
Master Method is a direct way to get the solution. The master method works only for following type of recurrences or for recurrences that can be transformed to following type.

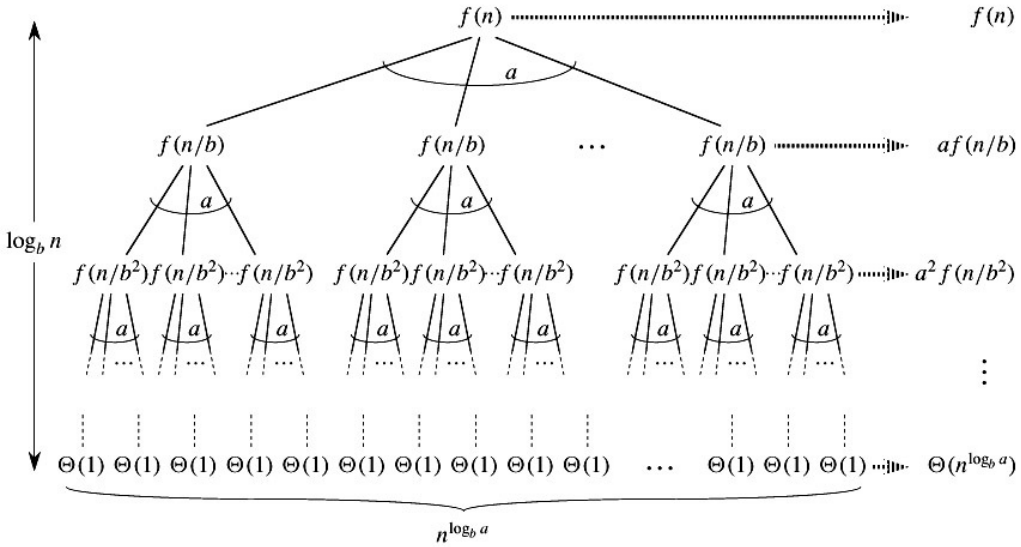
```
T(n) = aT(n/b) + f(n) where a >= 1 and b > 1
```

There are following three cases:
1. If f(n) = [Tex]\Theta \left(n^{{c}}\right)[/Tex] where c < [Tex]\log _{b}a[/Tex] then T(n) = [Tex]\Theta \left(n^{{\log _{b}a}}\right)[/Tex]

2. If f(n) = [Tex]\Theta \left(n^{{c}}\right)[/Tex] where c = [Tex]\log _{b}a[/Tex] then T(n) = [Tex]\Theta \left(n^{{c}}logn\right)[/Tex]

3.If f(n) = [Tex]\Theta \left(n^{{c}}\right)[/Tex] where c > [Tex]\log _{b}a[/Tex] then T(n) = [Tex]\Theta \left(f(n)\right)[/Tex]

How does this work?
Master method is mainly derived from recurrence tree method. If we draw recurrence tree of T(n) = aT(n/b) + f(n), we can see that the work done at root is f(n) and work done at all leaves is [Tex]\Theta \left(n^{{c}}\right)[/Tex] where c is [Tex]\log _{b}a[/Tex]. And the height of recurrence tree is [Tex]\log _{b}n[/Tex]



In recurrence tree method, we calculate total work done. If the work done at leaves is polynomially more, then leaves are the dominant part, and our result becomes the work done at leaves (Case 1). If work done at leaves and root is asymptotically same, then our result becomes height multiplied by work done at any level (Case 2). If work done at root is asymptotically more, then our result becomes work done at root (Case 3).

Examples of some standard algorithms whose time complexity can be evaluated using Master Method
Merge Sort: T(n) = 2T(n/2) + [Tex]\Theta(n)[/Tex]. It falls in case 2 as c is 1 and [Tex]\log _{b}a[/Tex] is also 1. So the solution is [Tex]\Theta(nLogn)[/Tex]

Binary Search: T(n) = T(n/2) + [Tex]\Theta(1)[/Tex]. It also falls in case 2 as c is 0 and [Tex]\log _{b}a[/Tex] is also 0. So the solution is [Tex]\Theta(Logn)[/Tex]

Notes:
1) It is not necessary that a recurrence of the form T(n) = aT(n/b) + f(n) can be solved using Master Theorem. The given three cases have some gaps between them. For example, the recurrence T(n) = 2T(n/2) + n/Logn cannot be solved using master method.

2) Case 2 can be extended for f(n) = [Tex]\Theta \left(n^{{c}}\log ^{{k}}n\right)[/Tex].
If f(n) = [Tex]\Theta \left(n^{{c}}\log ^{{k}}n\right)[/Tex] for some constant k >= 0 and c = [Tex]\log

_{b}a[/Tex], then T(n) = [Tex]\Theta \left(n^{{c}}\log ^{{k+1}}n\right) [/Tex]

Practice Problems and Solutions on Master Theorem.

References:
http://en.wikipedia.org/wiki/Master_theorem
MIT Video Lecture on Asymptotic Notation | Recurrences | Substitution, Master Method
Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Related Topics:

- Linearity of Expectation
- Iterative Tower of Hanoi
- Count possible ways to construct buildings
- Build Lowest Number by Removing n digits from a given number
- Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
- Find number of days between two given dates
- How to print maximum number of A's using given four keys
- Write an iterative O(Log y) function for pow(x, y)

Writing code in comment? Please use **ideone.com** and share the link here.

Vibhu Varshney • a month ago
How wil
T(n)=T(n/2)+n^2.2^n.logn
this ques. can be solved??
• Reply • Share ›

Techie Me • 2 months ago
Solving recurrence can be done using many ways. I love the recursion tree method and it comes handy for merge sort. Watch out the video about analysing Merge Sort..
https://www.youtube.com/watch?...
• Reply • Share ›

Prakhar Thakur • 2 months ago

how do you solve this recurrence:
T(n)=(n^1/2)T(n^1/2) +n

• Reply • Share ›

Aditya Goel • 2 months ago

Case 3 of master method also covers a regularity condition which is not given here - f(n) should satisfy the regularity condition that af(n/b) ≤cf(n) is true for some constant c< 1.

• Reply • Share ›

Shubham Bharti • 3 months ago

Use master methods, answer is thita(n^(log4,base 3))

• Reply • Share ›

sanjana ravichandran • 3 months ago

could anybodyplease say me how to solve T(n)=4T(n/3)+nlogn

• Reply • Share ›

Hospity • 5 months ago

Very interesting theorem.. Awesome post.. Great Thanks.

Hospital Management System

• Reply • Share ›

Guest • 5 months ago

At substitution method:
T(n) = 2T(n/2) + n
<= cn/2Log(n/2) + n
It should be
2c n/2 Log(n/2) + n
Missing 2.

• Reply • Share ›

gupta ➚ Guest • 4 months ago

very good

• Reply • Share ›

hanuman_patel • 7 months ago

Is this equation solve by the master theorem :: T(n) = 2T(n/2) + nlogn

• Reply • Share ›

Aditya Gaykar ➚ hanuman_patel • 5 months ago

Yes.

check http://people.csail.mit.edu/th...

• Reply • Share ›

Shubham Aggarwal ➚ Aditya Gaykar • 2 months ago

I do not understand how the case 2 was applied in this question!

• Reply • Share ›

Anurag • a year ago

In case two of master method.. it should be T(n) = theta(n^c logn)

10 • Reply • Share ›

moon ➚ Anurag • a year ago

yes it is.....plzz admin ...correct this..i wasted my precious 13 minutes on that

5 • Reply • Share ›

DS+Algo=Placement ➚ moon • 9 months ago

same here, then I checked out ur comment

1 • Reply • Share ›

GeeksforGeeks Mod ➚ DS+Algo=Placement • 8 months ago

All, thanks for pointing this out. We have updated the formula.

• Reply • Share ›

manish • a year ago

"T(n) = aT(n/b) + f(n) where a >= 1 and b > 1"
also a and b should be constants, otherwise master theorem not applied.
e.g. T(n) = 2^nT(n/2) + n^n.. what will solution in this case?

• Reply • Share ›

YouKnowWho ➚ manish • a year ago

This cannot be done using Master method. We have exponential number of subproblems being generated at each step.
Let us say T(1) = 1
Then,
T(n) = 2^n*(T(n/2)) + n^n
= 2^n*(2^(n/2)*T(n/4) + (n/2)^(n/2)) + n^n
<= 2^(n+ n/2 + n/4 + ...) + n^n + (n/2)^n + (n/4)^n + ...
and so on..
This would clearly be O(n^n) since the first term would become 2^(n*(1-(1/2)^log(n))/(1-0.5))) = 2^(2*(n-1))
The second term would boil down to n^n(1 + (1/2)^n +(1/4)^n + ...)
This would be equal to (n^n)*(1 - ((0.5)^nlogn)/(1 - 0.5^n) = n^n(1 - (nlogn)^-1)/(1-0.5^n)
= ((2n)^n)*(nlogn - 1)/(2^n - 1) <= 2n^n after some n0 thus, this would be in O(2n^n).
You could achieve much stricter upper bounds with more analysis.

• Reply • Share ›

hanuman_patel ➚ YouKnowWho • 7 months ago

Answer is :: T(n) = theta((n^n)*log n)=theta(n^n)

I think this is right just because i preferred corman book.

• Reply • Share ›

xxmajia • a year ago

Regarding "Binary Search: T(n) = T(n/2) + . It also falls in case 2 as c is 0 and is also 0. So the solution is " is INCORRECT.

1) Binary search should be T(N) = T(N/2) + O(1)
2) And T(N) = T(N/2) + O(N), you can image its quick select, which would be ≈O(2N) which is O(N)

1 • Reply • Share ›

GeeksforGeeks Mod ➚ xxmajia • a year ago

Thanks for pointing this out. We have corrected the recurrence for Binary Search.

1 • Reply • Share ›

Guest • a year ago

In master's method if f(n) contains some term of logn the is it possible to solve this by master's method?

• Reply • Share ›

GeeksforGeeks Mod ➚ Guest • a year ago

Prateek, the case 2 can be extended to handle Log in multiplication. We have updated the post and added this in notes as point 2.

1 • Reply • Share ›

Guest ➚ GeeksforGeeks • a year ago

But a/c CRLF if f(n) contains any logrithmic function then master's theorem is not applicable

• Reply • Share ›

-
  - Interview Experiences
  - Advanced Data Structures
  - Dynamic Programming
  - Greedy Algorithms
  - Backtracking
  - Pattern Searching
  - Divide & Conquer
  - Mathematical Algorithms
  - Recursion
  - Geometric Algorithms
-

- # Popular Posts

  - All permutations of a given string
  - Memory Layout of C Programs
  - Understanding “extern” keyword in C
  - Median of two sorted arrays
  - Tree traversal without recursion and without stack!
  - Structure Member Alignment, Padding and Data Packing
  - Intersection point of two Linked Lists
  - Lowest Common Ancestor in a BST.
  - Check if a binary tree is BST or not
  - Sorted Linked List to Balanced BST
-

- # Recent Comments

- lt_k

  i need help for coding this function in java...

  Java Programming Language · 1 hour ago

- Piyush

  What is the purpose of else if (recStack[*i])...

  Detect Cycle in a Directed Graph · 1 hour ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 1 hour ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 2 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 2 hours ago