

- Modify recurrence,

$$T(n) = T(n/5) + T(7n/10) + O(n)$$

Solution, solve using Akra-Baazi method or using induction

- Resulting in linear time complexity algorithm

Activity 1


- (1) Solve following recurrence using substitution method:

- a. $T(n) = \begin{cases} nT(n-1), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$
- b. $T(n) = \begin{cases} T(n-1) + \log n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$
- c. $T(n) = \begin{cases} 3T(n/2) + n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$

1. Recursive relations normally reflect the runtime of recursive algorithms.

Substitution method for solving recurrence comprises two steps,

- Guess the form of the solution. **Requires substantial experience (guess the form of the answer to apply it, use some heuristics) or you can use recursion trees to generate good guesses**
- Use mathematical induction to find the constants and show that the solution works.

a. $T(n) = \begin{cases} nT(n-1), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$  Given base case

Base case: $T(1) = 1$

Recursive case: where $n > 1$,

$$T(n) = n \cdot T(n-1)$$

Expand $T(n-1)$ using same relation,

$$T(n-1) = (n-1) \cdot T(n-2)$$

Expand until base case $T(1)$ is found,

$$T(n) = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 2 \cdot 1 \cdot T(1)$$

Substitute the base case,

Since $T(1) = 1$, this value can be used

$$T(n) = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 2 \cdot 1 \cdot 1$$

Simplify,

$$T(n) = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 2 \cdot 1$$

Closed form solution $T(n)$, factorial is the product of all positive integers from 1 to n , which is $n!$

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

$$T(n) = n!$$

Time complexity,

$$O(n!)$$

The function is the factorial of n , where $T(n)$ grows at a rate (very fast) that is proportional to $n!$. Additionally this exponential growth is not practical for large values.

b. $T(n) = \begin{cases} T(n-1) + \log n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$

Base case: $T(1) = 1$

Recursive case: where $n > 1$,

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

Expand $T(n-1)$ using same relation,

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

Expand until base case $T(1)$ is found,

$$\begin{aligned} T(n) &= T(n-1) + \log(n) \\ &= (T(n-2) + \log(n-1)) + \log(n) \\ &= (T(n-3) + \log(n-2) + \log(n-1)) + \log(n) = \dots \end{aligned}$$

Continued,

$$T(n) = T(1) + \log(2) + \log(3) + \dots + \log(n)$$

Simplify,

$$T(n) = 1 + \log(2) + \log(3) + \dots + \log(n)$$

Sum logs,

Logarithmic rule: $\log(a) + \log(b) = \log(a \cdot b)$,

$$T(n) = \log(2 \cdot 3 \cdot \dots \cdot n) + 1$$

$\log(2 \cdot 3 \cdot \dots \cdot n)$ is factorial

Simplify,

$$T(n) = \log(n!) + 1$$

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

$$T(n) = \log(n!) + 1$$

Time complexity,

$$O(\log(n!) + 1) = O(n \log n)$$

c. $T(n) = \begin{cases} 3T(n/2) + n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$

Base case: $T(1) = 1$

Recursive case: where $n > 1$,

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

Expand $T(n/2)$ using same relation,

$$T(n/2) = 3T\left(\frac{n/2}{2}\right) + n/2$$

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

(3)

Expand until base case $T(1)$ is found,

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

$$= 3[3T(n/4) + n/2] + n$$

(3²)

$$= 9T(n/4) + 3n/2 + n$$

Substitute,

$$T(n/4) = 3T\left(\frac{n/4}{2}\right) + n/4$$

Substitute for recursive implementation,

$$T(n/2) = 9[3T(n/8) + n/4] + 3n/2 + n$$

(3³)

$$= 27T(n/8) + 9n/4 + 3(n/2) + n$$

Substitute,

$$T(n/8) = 3T\left(\frac{n/8}{2}\right) + n/8$$

Substitute,

$$T(n/8) = 27\left(3T\left(\frac{n/8}{2}\right) + n/8\right) + (9/4)n + (3/2)n + n$$

(3⁴)

$$T(n/8) = 81T(n/16) + 27n/8 + (9/4)n + (3/2)n + n$$

Find Pattern,

$$3, 3^2, 3^3, 3^4 \dots 3^k$$

Simplify using k implementations,

$$= 3^k T\left(\frac{n}{2^k}\right) + n \left[\frac{3}{2} + \left(\frac{3}{2}\right)^2 + \left(\frac{3}{2}\right)^3 + \left(\frac{3}{2}\right)^4 + \dots + \left(\frac{3}{2}\right)^{k-1} \right]$$

Simplify summation,

$$= 3^k T\left(\frac{n}{2^k}\right) + n \left[\Sigma \left(\frac{3}{2}\right)^k \right]$$

Simplify apply \log_2 of both sides,

Exponential form,

$$\begin{aligned} 2^k &= n \\ \log_{2^k} &= \log_n \\ k &= \log_n \end{aligned}$$

$$= 3^{\log_n} T\left(\frac{n}{n}\right) + n \left[\frac{3^k}{2^k} \right]$$

Substitute,

$$= 3^{\log_n} T(1) + n \left[\frac{3^{\log_2 n}}{2^{\log_2 n}} \right]$$

Logarithmic rule: $a^{\log_b c} = b^{\log_a c}$

Simplify,

$$= n^{\log_{32}} + n \left[\frac{n^{\log_2 3}}{n^{\log_2 2}} \right]$$

$$= n^{\log_2 3} + n \left[\frac{n^{\log_2 3}}{n} \right]$$

$$= 2n^{\log_2 3}$$

Time complexity,

$$= O(n^{\log_2 3})$$

Activity 2

- (2) Solve following recurrence using Master Theorem:

$$a \quad T(n) = \begin{cases} T(\sqrt{n}) + \log n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

- The Master Theorem is a formula that solves recurrences when all the subproblems are the same size!
- There are two forms of The Master Theorem

- 1. Where the equation suits the recurrence of the form
- 2. Alternative Formulation is needed as the equation does not suit the recurrence of the form
- The recurrence requires three parameters (options),
- a: number of subproblems, b: factor by which input size shrinks, d: need to do n^d work to create all the subproblems and combine solutions.

$$a. \quad T(n) = \begin{cases} T(\sqrt{n}) + \log n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

$$O(n \log(n))$$

Equation does not suit the recurrence of the form, hence using formulation (2)
Alternative formulation is used

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Rewrite the recurrence,

$$\begin{aligned} T(n) &= T(\sqrt{n}) + \log(n) \\ &= T(n^{1/2}) + \log(n) \end{aligned}$$

As per textbook example [4],

Similar to recurrence relation $S(m) = S(m/2) + m$

Where, $S(m)$ denotes time complexity of $T(2^m)$

Substitute $n = 2^m$,

$$n^{1/2} = 2^{m/2}$$

$$m = \log(n)$$

$$T(2^m) = T(2^{m/2}) + \log(2^m)$$

Simplify,

$$\log(2^m) = m \log_2 2 = m = 1$$

$$T(2^m) = T(2^{m/2}) + m$$

$$O(n)$$

$$T(2^m) = S(m)$$

$$S(m) = S\left(\frac{m}{2}\right) + m \log(m)$$

Coefficient of T in the recurrence,

$$a: 1$$

Factor by which n is divided in the recurrence/ problem size,

$$b: 2$$

Asymptotically positive for all sufficiently large n / cost of outside the recursive call,
 $d: f(n) = 1$

Solution as per Master method,

$$T(n) = \begin{cases} O(n^d \log n), & \text{if } a = b^d \\ O(n^d), & \text{if } a < b^d \\ O(n^{\log_b a}), & \text{if } a > b^d \end{cases}$$

$$\begin{aligned} &\text{if } a = b^d \\ &1 \neq 2^1 \end{aligned}$$

Since $a = 1$ and $b = 2$, $a \neq b^d$ for any non-negative integer d

Case 2, where $d = 0$, as $\log n$ does not use n ,

$$\begin{aligned} &\text{if } a < b^d \\ &1 < 2^1 \end{aligned}$$

Solution as per the Master Theorem,

$$\begin{aligned} n &= 2^m \\ T(2^m) &= T(2^{(m/2)}) + m \\ T(n) &= O(n^d) = O(1) \end{aligned}$$

Thus,

$$S(m) = O(m)$$

For $n = 2^m$,

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

Time Complexity in The Master Solution is given,

$$T(n) = O(n^d (\log(n))^k)$$

k is a non-negative constant

d is a non-negative integer such that $a = b^d$

$$T(n) = O(m^1 (\log^0 m))$$

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

Activity 3

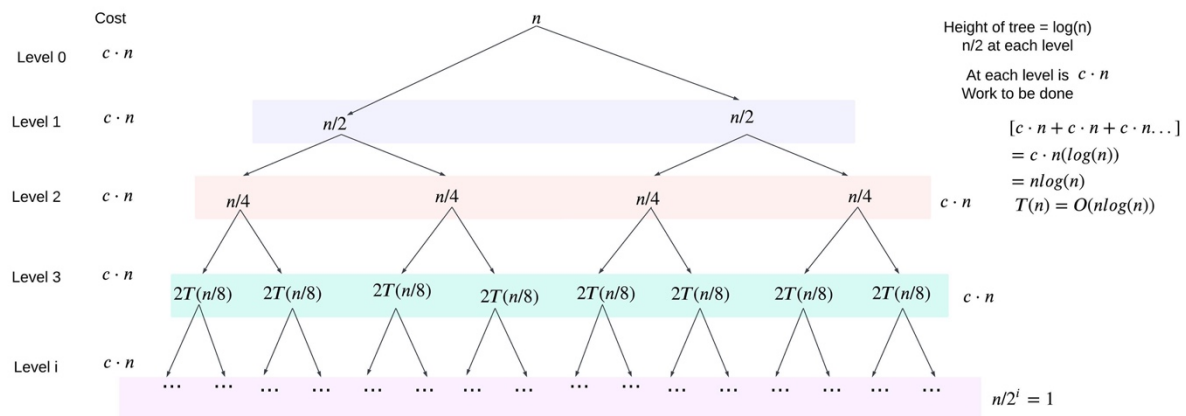
- (3) Solve following recurrence using recursion tree method:

$$a, T(n) = 2T(n/2) + cn$$

Recurrence relation,

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

Recursion Tree Method, break down recurrence relation into smaller subproblems (tree structure). At each level of tree, the subproblem is broken into smaller subprograms, each of size $n/2$, repeating until base case found $n = 1$.



The height of tree is $\log n$, dividing $n/2$ at each level, total work done at all levels (cn)

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

Expand,

$$\begin{aligned} T(n) &= 2[2T(n/4) + c(n/2)] + cn \\ &= 4T(n/4) + 2cn \end{aligned}$$

$$\begin{aligned} T(n) &= 4[2T(n/8) + c(n/4)] + 2cn \\ &= 8T(n/8) + 3cn \end{aligned}$$

Pattern identified!

Find base case where $n/2^i = 1$,

$$T(n) = 2^i T(n/2^i) + i \cdot cn$$

$$i = \log_2(n),$$

$$T(n) = 2^{\log_2 n} T(n/2^{\log_2 n}) + \log_2 n \cdot cn$$

Simplify,

$$T(n) = n \cdot T(1) + c \cdot n \cdot \log_2 n$$

Since $T(1)$ is a constant, total work at each level \cdot total height,
 $cn \cdot \log_2 n$

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

Activity 4

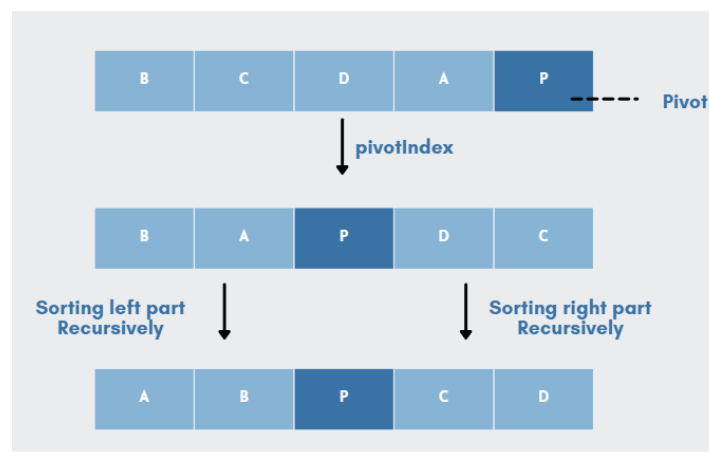
- (4) We had a detailed analysis of select(A,k) problem in this module. Based on what you learned from this problem, analyse the complexity of Quick Sort Algorithm. Note, quick sort will lead to unbalanced sub-problems (much like select(A,k)) problem. You are expected to use either Akra-Baazi or induction method to solve the recurrence.

Analyse the complexity of Quick Sort Algorithm,

Quick Sort algorithm based on divide and conquer approach, it is a fast algorithm '...uses $n \log(n)$ comparisons to sort an array of n elements...' [5]. Like Select(A, k) problem from Module 10 [6], Quick sort we can choose a pivot randomly and divide lists into two parts. It splits a large problem into smaller subproblems, best case time complexity $O(n \log(n))$ and space complexity of $O(\log(n))$. As Quick Sort has an $O(\log n)$ space complexity it is recommended when space is restricted [5].

Quick Sort Steps,

Using 3 steps, 1. Pick an element, 2. Divide the problem set, move smaller parts to the left of pivot and larger items to right. 3. Repeat steps to combine sub solutions stored.



Quick Sort Algorithm Overview, [5]

- The time complexity of Quick Sort,

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

$T(n)$ Time complexity of sorting array of size n

$T(k)$ and $T(n - k - 1)$ are the two recursive calls (also known as $(T(L) + T(R))$)

$O(n)$ Time partitioning, selecting pivot and rearranging elements

1. Best Case Complexity,

- Algorithm finds pivot in the middle element or near middle
- $O(n \log(n))$ time complexity

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

2. Worst Case Complexity,

- Algorithm finds pivot selecting largest or smallest element
- Acts similarly to the Select(A, k) problem
- $O(n^2)$ time complexity

$$T(n) = T(0) + T(n - 1) + O(n)$$

Quick Sort will lead to unbalanced subproblems if the pivot finds results in one subproblem that is larger than the other.

- The recurrence relation outline for Quick Sort,

$$T(n) = T(L) + T(R) + O(n)$$

$T(n)$ Time complexity of sorting array of size n

$T(L)$ or $T(R)$ Time sorting Left or Right side

$O(n)$ Time partitioning, selecting pivot and rearranging elements

Complexity of Quick Sort algorithm is based on $\text{len}(L)$ and $\text{len}(R)$

$$T(n) = \begin{cases} T(\text{len}(L)) + O(n), & \text{if } \text{len}(L) > k - 1 \\ T(\text{len}(R)) + O(n), & \text{if } \text{len}(L) < k - 1 \\ O(n), & \text{if } \text{len}(L) = k - 1 \end{cases}$$

Master Theorem Method Module 10 Recurrence

- Module 10 [6], used The Master Theorem, even though quick sort random pivot could leave to unequal sizes of lists. The module assumed that the pivot selected equal L and R sizes and resulted in linear time complexity $T(n) = O(n)$.

Assumed pivot selection,

$$\begin{aligned} 3n/10 &< \text{len}(L) < 7n/10 \\ 3n/10 &< \text{len}(R) < 7n/10 \end{aligned}$$

$$a: 1, \quad b = 10/7, \quad d = 1$$

- The recurrence relation,

$$\begin{aligned} T(n) &= T(7n/10) + O(n) \\ &\equiv O(n^d) = O(n) \end{aligned}$$

But we can't guarantee that the pivot selection will always bread '...not worse than 30% and 70%...' [6].

Median of the Median Method Module 10 Recurrence

- To find a median of list without computing the median
- The median breaks the list into two equal subproblems (lists)
- The recurrence **modified and solved using Akra-Baazi method** or using induction
- Resulting in linear time complexity $T(n) = O(n)$

This method of pivot selection guarantees,

$$\begin{aligned} 3n/10 &< \text{len}(L) < 7n/10 \\ 3n/10 &< \text{len}(R) < 7n/10 \end{aligned}$$

Modify recurrence due to divide and conquer strategy,

$$T(n) = T(n/5) + T(7n/10) + O(n)$$

Akra-Baazi Method/ Theorem Steps Module 10 Recurrence

Akra-Baazi is designed for problems which divide into subproblems with a fixed proportion on their size at each iteration, to analyse asymptotic behaviour [7].

Akra-Baazi Theorem,

$$T(n) = \sum_{i=1}^k a_i T(b_i n + h_i(n)) + g(n)$$

Where a_i and b_i are constants such that,

- $a_i > 0$
- $0 < b_i < 1$
- $g(n) \in O(n^c)$
- $h_i(x) \in O\left(\frac{n}{(\log n)^2}\right)$

Modify recurrence from Module 10

$$T(n) = T(n/5) + T(7n/10) + O(n)$$

Satisfy conditions $a_i > 0$,

$$a_1 = a_2 = 1$$

Satisfy conditions $0 < b_1, b_2 < 1$,

$$\begin{aligned} b_1 &= 3/10 \\ b_2 &= 4/5 \end{aligned}$$

Where a_i and b_i are constants such that,

$$\begin{aligned} a_1 &= a_2 = 1 \\ b_1 &= 3/10 \\ b_2 &= 4/5 \end{aligned}$$

Find p such that,

$$\sum_{i=1}^k a_i b_i^p = 1$$

Assume $p = 1$,

$$\sum_{i=1}^k a_i b_i^p = a_1 b_1 + a_2 b_2 = b_1 + b_2 = 1$$

Simplified form,

$$= a_1 b_1^{p_1} + a_2 b_2^{p_2} = 1$$

$$2 \left(\frac{1}{5}\right)^{p_1} + 2 \left(\frac{7}{10}\right)^{p_2} = 0.4^{p_1} + 0.2^{p_2} \approx 1$$

Therefore, $p \approx 1$,

Satisfy conditions $g(n) = O(n^c)$ for constant c ,
Where $g(n) = O(n^1)$

There is only one level of recursion, for h_1, h_2 such that $h_i(n) = O(\frac{n}{(\log n)^2})$

Evaluate Akra-Baazi,

$$T(n) = O\left(n^p \left(1 + \int_1^n u^{-p} du\right)\right)$$

Substitute value $p = 1$,

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

Therefore, this recurrence based on Module 10 has worst time complexity of $O(n \log(n))$.

Additional Notes:

It does not result in Linear time complexity as the value of $p = 1$ is an approximate. The recurrence relation for time complexity of Quick Sort as stated above is

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

If the pivot point is selected at random than $k = (n - 1)/2$ on average therefore it is not possible to find a value of p that would make the time complexity linear, this is the same for induction. However, if we used 'median of medians', as shown above from Module 10 to select the pivot element it is possible to obtain a linear time complexity.

Additional Example – Difficult for recursion tree but easy for Akra Baazi

- This example is difficult for recursion trees but easy for Akra-Baazi theorem [7]

$$T(n) = T(3n/4) + T(n/4) + O(n)$$

Where a_i and b_i are constants such that,

$$a_i = 1, b_i = 3/4, h_i = 0$$

$$a_j = 1, b_j = 1/4, h_j = 0$$

$$g(n) = n$$

Find p such that,

$$\sum_{i=1}^k a_i b_i^p = 1$$

Example,

$$p = \sum_{i=1}^k a_i b_i^p = 1 \left(\frac{3}{4}\right)^p + 1 \left(\frac{1}{4}\right)^p = 1$$

$$\left(\frac{3}{4}\right)^p + 1 \left(\frac{1}{4}\right)^p \approx 1$$

$\approx p = 1$,

Close to zero, use 1 as approximation,

$$\left(\frac{3}{4}\right) + \left(\frac{1}{4}\right) - 1 = 0$$

Substitute values for p ,

$$\epsilon = 1 - p = 1 - 1 = 0$$

$$T(n) = O(n^{1+\epsilon})$$

Function grows asymptotically slower than $n^{1+\epsilon}$ for any positive constant ϵ

Evaluate Akra-Baazi,

$$T(n) = O\left(n\left(1 + \int_1^n \frac{1}{u} du\right)\right)$$

Substitute values,

Since $\epsilon = 0$,

$$\int_1^n \frac{1}{u} du = \log(n) - \log(1) = \log(n)$$

$$T(n) = O(n^{1+0} \log(n))$$

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

Since $\epsilon = 0$, any function that grows asymptotically slower than $n \log(n)$ would also be $O(n \log(n))$.

Bibliography

- [1] An insightful Techie, "Data Structures, Big 'O' Notations and Algorithm Complexity," 30 12 2017. [Online]. Available: <https://www.youtube.com/watch?v=L7nYZ19zPqw>. [Accessed 12 9 2023].
- [2] T. a. L. C. a. R. R. a. S. Coreman, "Chapter 4 ," in *C. Divide-and-Conquer*, Cambridge, Massachusetts London, England , The MIT Press , 2022, pp. 76-115.
- [3] Code Academy, "Asymptotic Notation," 2023. [Online]. Available: <https://www.codecademy.com/learn/cspath-asymptotic-notation/modules/cspath-asymptotic-notation/cheatsheet>. [Accessed 12 9 2023].
- [4] T. a. L. Coreman, "Divide-and-Conquer, Chapter 4," in *CLRS*, MIT Press Academic, 2022.
- [5] P. H, "An Overview of QuickSort Algorithm," Towards Data Science, 11 3 2022. [Online]. Available: <https://towardsdatascience.com/an-overview-of-quicksort-algorithm-b9144e314a72>. [Accessed 12 9 2023].

- [6] Deakin University, " Algorithm Analysis: The Select(A,k) Problem," n.d. [Online]. Available:
<https://d2l.deakin.edu.au/d2l/le/content/1316240/viewContent/7023569/View>.
[Accessed 12 9 2023].
- [7] J. Erickson, *Solving Recurrences*, Champaign, IL, United States : University of Illinois Urbana-Champaign, 2022.