Week 2

Display replies in nested form

Settings

The cut-off date for posting to this forum is reached so you can no longer post to it.

Week 2

by Romana Riyaz (Instructor) - Thursday, 20 June 2024, 10:26 AM

Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five sub-problems of half the size, recursively solving each sub-problem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two sub-problems of size n-1 and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine sub-problems of size n/3, recursively solving each sub-problem, and then combining the solutions in O(n2) time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

For your discussion post, explain in your own words why you chose a particular algorithm. Include one or two examples to explain your thought process to show what is occurring and how the methodology works. Demonstrate your understanding of the intricacies of the algorithm you chose. Use APA citations and references for any sources used.

Post your response to these problems and then respond to at least 3 of your peers. Discuss whether each peer's solution is correct and why?

184 words

Permalink

Re: Week 2

by Cherkaoui Yassine - Friday, 28 June 2024, 4:06 PM

Running Times of Algorithms

1. Algorithm A:

- Divide: 5 sub-problems of size n/2.

- Combine: Linear time, O(n).

- Recurrence Relation: T(n) = 5T(n/2) + O(n).

- Solution: Using the Master Theorem, T(n) = O(n^log2(5)) ≈ O(n^2.32).

2. Algorithm B:

- Divide: 2 sub-problems of size n-1.

- Combine: Constant time, O(1).

- Recurrence Relation: T(n) = 2T(n-1) + O(1).

- Solution: This is a recursive call that translates to T(n) = O(2^n).

3. Algorithm C:

- Divide: 9 sub-problems of size n/3.

- Combine: Quadratic time, O(n^2).

- Recurrence Relation: T(n) = 9T(n/3) + O(n^2).

- Solution: Using the Master Theorem, T(n) = O(n^2).

- Choice of Algorithm:

Chosen Algorithm: Algorithm A

Explanation: Algorithm A, with a time complexity of O(n^2.32), is significantly better than Algorithm B's exponential time complexity O(2^n). While Algorithm C has a similar quadratic time complexity to Algorithm A, the constant factors involved in practical scenarios often make Algorithm A more efficient due to lower constants in its linear combination step.

Example 1: Sorting an array: Consider sorting an array of integers. If we use Algorithm A, the array is split into five smaller arrays of half the size, sorted individually, and then merged back together in linear time. This approach efficiently handles large datasets by reducing the problem size significantly at each step.

Example 2: Matrix Multiplication: In matrix multiplication, Algorithm A can be applied to divide the matrices into smaller sub-matrices, recursively multiply them, and then combine the results efficiently. This reduces the overall computational complexity compared to a naïve approach.

259 words

Permalink Show parent

Re: Week 2

by Romana Riyaz (Instructor) - Friday, 28 June 2024, 8:59 PM

Cherkaoui,
Thank you for your submission. You've done a good job analyzing the running times of the algorithms and applying the Master Theorem to determine their complexities. Your choice of Algorithm A is well-justified, given its more favorable time complexity compared to Algorithm B's exponential growth and Algorithm C's quadratic complexity. The examples provided, such as sorting an array and matrix multiplication, effectively illustrate how Algorithm A can be applied in practical scenarios to handle large datasets efficiently. Your detailed breakdown and clear explanation showcase a strong understanding of algorithm analysis.
Best,
Romana
93 words

Permalink Show parent

Re: Week 2

by Benjamin Chang - Sunday, 30 June 2024, 11:35 PM

Hi Cherkaoui,

Thank you for sharing this excellent topic this week. I agree with your concerns in the post, and your responses are all correct. Especially you compare the algorithm A and C to say because a has lower constants in its linear, that is precisely to explain it! Great job! We hope to see another post from you next week.

Benjamin
62 words

Permalink Show parent

Re: Week 2
by Moustafa Hazeen - Monday, 1 July 2024, 6:30 AM

Cherkaoui provides a thorough analysis of the algorithms' complexities and correctly identifies Algorithm A as the preferable choice due to its superior theoretical efficiency compared to Algorithms B and C, especially highlighting the practical implications of constant factors in the combine step. Examples provided (sorting an array and matrix multiplication) effectively illustrate the application of Algorithm A's divide-and-conquer strategy. Well done!
61 words

Permalink Show parent

Re: Week 2
by Akomolafe Ifedayo - Tuesday, 2 July 2024, 10:39 PM

Hi Cherkaoui, great work on your submission. Your solutions were well-detailed and engaging to go through. You provided the running times of each algorithm and also determined their complexities. Keep it up.
32 words

Permalink Show parent

Re: Week 2
by Fadi Al Rifai - Wednesday, 3 July 2024, 9:50 PM

Hi Cherkaoui,
Thank you for your thoughtful contribution to a detailed explanation of options A, B, and C were clear and concise, and I like your description of recurrence relations and time complexity.
Keep it up.
36 words

Permalink Show parent

Re: Week 2
by Siraajuddeen Adeitan Abdulfattah - Thursday, 4 July 2024, 2:11 AM

Hi Cherkaoui,

Good submission in response to be questions asked. Your choice of Algorithm A seems justified and rightly so, you have help me understood this concept better through your explanation. Having made a mistake in my submission with regards to Algorithm B. Your example explaining why Algorithm A is a better choice is quite clear and self explanatory.
59 words

Permalink Show parent

Re: Week 2
by Nour Jamaluddin - Thursday, 4 July 2024, 4:09 AM

Good job,
You understand the concepts very well. Regarding the first and third algorithms, I preferred to use the references and support your results. Your examples are helpful.
Thank you.
30 words

Permalink Show parent

Re: Week 2
by Chong-Wei Chiu - Thursday, 4 July 2024, 8:56 AM

Hello, Cherkaoui Yassine. Thank you for sharing your point of view about this topic. You clearly state each step of calculation, including divide, combine and recurrence relation. Besides, your example is easy to understand. However, you should add the reference you used in this post and that would make your post more complete.
53 words

Permalink Show parent

Re: Week 2
by Benjamin Chang - Saturday, 29 June 2024, 10:15 PM

- Algorithm A solves issues by employing a recursive approach, where it divides the problems into five sub-problems that are half the size. It then solves each sub-problem recursively and combines the results in linear time.
- Algorithm B solves problems of size n by recursively solving two sub-problems of size $n - 1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine sub-problems of size $n / 3$, recursively solving each sub-problem, and then combining the solutions in $O (n 2)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

- In analyzing Algorithm A, I will **use the master theorem** because it is effective for solving recurrence relations in this context. According to Dasgupta, S., Papadimitriou, C.H., & Vazirani, U.V. (2016), the master theorem applies to any recurrence relation of the form $T (n) = a T (n / b) + c n k$ with $T (1) = c$, and the following relationships hold as shown in the reference (p. 508). The master theorem solves recurrences of the form $T (n) = a T (n / b) + f (n)$. For Algorithm A, which divides the problem into five sub-problems of half the size, we have $a = 5$, $b = 2$, and $f (n)$ is $O (n)$. Calculating log

25gives approximately 2.32. Thus, the running time of Algorithm A is $O (n$ log25) . To verify and prove this result, we can use another method, such as the recursion tree. By analyzing the recursion tree, we see that the branch factor is 5 and can check the depth of the tree at each level. Finally, we find that $T (n)$ is $O (n$ log25) , consistent with the master theorem.

- For Algorithm B, we should use a different method because the recurrence $T (n - 1)$ is different. I personally prefer using a **recursion tree** to analyze this algorithm, as it allows us to see the different levels from top to bottom. According to GeeksforGeeks (2021), each node in the recursion tree represents the cost incurred at various levels of recursion. By counting the total number of nodes and summing up the costs at all levels in the tree, we can determine

  the overall complexity.

Image from Shaffer, C. A. (2010). A Practical Introduction to data Structures and algorithm Analysis third edition (C++ Version) (3rd ed.).

- So, the total cost at depth d d is $n - 1$, and at depth k k it is 0. If the branching factor is 2, then $T (n) = O (2^{n - 1})$. Therefore, we can say that $T (n)$ is finally $O (2 n)$, indicating that the algorithm has exponential running time.
- Algorithm C is similar to Algorithm A, so we can use **the master theorem** to calculate the recurrence relation $T (n) = 9 T (n / 3) + O (n^2)$. Since

log39 is 2, and n^2 is nlog39 , we find that T(n) is Θ (n2logn).

-

Therefore, we can see that Algorithm B has the worst performance due to its exponential running time $O (2^n)$. When comparing Algorithm A and Algorithm C, we find that n 2.32 (Algorithm A) grows faster than (n2logn). (Algorithm C). Thus, (

n2logn) is less than n 2.32, indicating that Algorithm C is slower than Algorithm A. Since Algorithm A performs better than both Algorithm C and Algorithm B, **I prefer Algorithm A**. What do you think, everyone?

Reference

ota, S., Papadimitriou, C.H., & Vazirani, U.V. (2006). *Algorithms.* Berkeley, CA: University of California Berkeley, Computer Science Division. Available at http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf

orGeeks. (2021, November 24). *How to solve time complexity Recurrence Relations using Recursion Tree method?* GeeksforGeeks. https://www.geeksforgeeks.org/how-to-solve-time-complexity-recurrence-relations-using-recursion-tree-method/

, C. A. (2010). *A Practical Introduction to data Structures and algorithm Analysis third edition (C++ Version)* (3rd ed.). https://people.cs.vt.edu/~shaffer/Book/C++3e20100119.pdf

729 words

Permalink Show parent

Re: Week 2
by Romana Riyaz (Instructor) - Sunday, 30 June 2024, 12:14 AM

Benjamin,
Thank you for your submission. You provide a thorough analysis of each algorithm, clearly explaining how you derived the running times. You effectively apply the Master Theorem to Algorithms A and C, showing a strong understanding of how it works. For Algorithm B, you correctly identify that it doesn't fit the Master Theorem and use a recursion tree method to analyze it, showing flexibility in your approach. You compare the running times of the algorithms and conclude which is more efficient, which is useful for understanding their relative performance. Ensure the mathematical notation is consistent and clear. Consider adding visuals like recursion trees or diagrams to illustrate the recursive breakdown of each algorithm. This can help readers better understand the analysis. Break down complex sentences into simpler ones to improve readability. For example: "Algorithm A divides the problem into five sub-problems of half the size, solves each sub-problem recursively, and combines the results in linear time." Be consistent in the way you describe each algorithm’s process before delving into the analysis.
Best,
Romana
174 words

Permalink Show parent

Re: Week 2
by Moustafa Hazeen - Monday, 1 July 2024, 6:31 AM

Benjamin's analysis is thorough and backed by solid references. The use of the Master Theorem for Algorithms A and C is appropriate and demonstrates a good understanding of recurrence relations. The explanation of Algorithm B's exponential complexity using the recursion tree method adds depth to the analysis. However, a minor improvement could involve clearer transitions between theoretical analysis and practical implications in real-world scenarios.
64 words

Permalink Show parent

Re: Week 2
by Cherkaoui Yassine - Monday, 1 July 2024, 3:41 PM

Benjamin, I wanted to drop you a quick note to say excellent job on your discussion post! Your answer is not only clear but also very well-written. It's evident that you put thought and effort into it, and it really shines through. Keep up the great work!
47 words

Permalink Show parent

Re: Week 2
by Akomolafe Ifedayo - Tuesday, 2 July 2024, 10:41 PM

Hi Benjamin, your submission was well-detailed as you provided the running time of each algorithm and also determined its complexity. Your work was well-detailed, keep it up.

27 words

Permalink Show parent

Re: Week 2
by Naqaa Alawadhi - Wednesday, 3 July 2024, 2:52 AM

Good job

2 words

Permalink Show parent

Re: Week 2
by Fadi Al Rifai - Wednesday, 3 July 2024, 9:50 PM

Hi Benjamin,
Good work, Thanks for sharing your explanation about options A, B, and C was clear and concise, and I like your description of recurrence relations and time complexity.
Keep it up.

33 words

Permalink Show parent

Re: Week 2
by Siraajuddeen Adeitan Abdulfattah - Thursday, 4 July 2024, 2:20 AM

Hi Benjamin,

Good submission in response to the questions asked. I did enjoy your detailed explanations on all Algorithms and their time complexities. I believe your choice pf Algorithm A over B and C is correct because Algorithm A offers better performance and time complexity. You did a good job of including in-text citations and references to support your valid points.

61 words

Permalink Show parent

Re: Week 2
by Tyler Huffman - Thursday, 4 July 2024, 4:19 AM

Excellent post on a complex topic Benjamin. I learned from your reasoning/thought process. You explained things simply yet thoroughly; additionally you made a good use of outside sources. Well done and keep it up.

34 words

Permalink Show parent

Re: Week 2
by Nour Jamaluddin - Thursday, 4 July 2024, 4:22 AM

Well done.
Your post is complete and well organized. The explanation is very interesting. The answers are easy to follow.
Many thanks.

22 words

Permalink Show parent

Re: Week 2

by Anthony Jones - Thursday, 4 July 2024, 8:00 AM

Hello,

Good post! It's clear you put a lot of for into it, but you made an error.
You were correct in asserting that Algorithm A grows faster than Algorithm C, however this is not a good thing. We are examining how the time that the algorithm takes responds to changes in the problem. So a larger time, means a slower algorithm (why else would we say O(2^n) is a bad thing?). We don't want the algorithm to grow fast, we want it to grow as slow as possible.

So, Algorithm C is best asymptotically. However, there is a point to what you said which is that Algorithm A is faster than Algorithm C for small problems, so depending on the situation Algorithm A may be the way to go.

God bless!

Anthony

133 words

Permalink Show parent

Re: Week 2

by Fadi Al Rifai - Sunday, 30 June 2024, 3:27 PM

To determine the running times of the three algorithms, I can use the Master Theorem for divide-and-conquer recurrences.

Here I'll analyze each algorithm:

Algorithm A

Algorithm A divides the problem into 5 sub-problems of half the size, solves each recursively, and then combines the solutions in linear time.

The recurrence relation for **Algorithm A** is:

Using the Master Theorem, I'll identify:

- $a = 5$
- $b = 2$
- $f(n) = O(n)$

I need to compare $f(n)$ with $n^{\log_b a}$:

$n^{\log_b a} = n^{\log_2 5} \approx n^{2.32}$

Since $f(n) = O(n)$ and $n^{\log_2 5}$ dominates n, in Case 1 of the Master Theorem, where $f(n) = O(n^c)$ with .

Thus, the running time $T(n)$ is:

$T(n) = O(n^{\log_2 5}) \approx O(n^{2.32})$

Algorithm B

Algorithm B solves the problem by recursively solving two sub-problems of size $n - 1$ and then combining the solutions in constant time.

The recurrence relation for **Algorithm B** is:

$T(n) = 2T(n - 1) + O(1)$

This type of recurrence does not fit the standard form of the Master Theorem, so I solve it by iteration.

Starting from $T(n)$:

$T(n) = 2T(n - 1) + O(1)$

$T(n - 1) = 2T(n - 2) + O(1)$

Substitute $T(n - 1)$ into $T(n)$:

$T(n) = 2[2T(n - 2) + O(1)] + O(1)$

$T(n) = 2^2T(n - 2) + 2O(1) + O(1)$

$T(n) = 2^2T(n - 2) + 3O(1)$

Continuing this process:

$T(n) = 2^kT(n - k) + kO(1)$

When $k = n$:

$T(n) = 2^nT(0) + nO(1)$

$T(n) = O(2^n)$

Algorithm C

Algorithm C divides the problem into 9 sub-problems of size $n / 3$, solves each recursively, and then combines the solutions in $O(n^2)$ time.

The recurrence relation for **Algorithm C** is:

Using the Master Theorem, I identify:

- $a = 9$
- $b = 3$
- $f(n) = O(n^2)$

I need to compare $f(n)$ with $n^{log_b a}$:

$n^{log_b a} = n^{log_3 9} = n^2$

Since $f(n) = O(n^2)$ and $f(n) = n^{log_b a}$, in Case 2 of the Master Theorem.

Thus, the running time $T(n)$ is:

$T(n) = O(n^2 \log n)$

Comparison and Choice

- **Algorithm A:** $O(n^{2.32})$
- **Algorithm B:** $O(2^n)$
- **Algorithm C:** $O(n^2 \log n)$

Algorithm B has an exponential running time $O(2^n)$, making it the least efficient for large n.

Algorithm A has a polynomial running time $O(n^{2.32})$, while **Algorithm C** has a slightly better running time of $O(n^2 \log n)$.

Chosen Algorithm: <u>Algorithm C</u>

Reasoning:

Algorithm C is chosen because its running time $O(n^2 \log n)$ is better than **Algorithm A**'s $O(n^{2.32})$, especially for large n. The logarithmic factor in **Algorithm C** is relatively small compared to the polynomial exponent in **Algorithm A**.

Example to Explain the Methodology

Consider a problem size $n = 27$:

Algorithm A

1. Divide $n = 27$ into 5 sub-problems of size **13.5**.
2. Recursively solve each sub-problem.
3. Combine the solutions in $O(27)$.

Algorithm B

1. Solve two sub-problems of size **26**.
2. Recursively solve each sub-problem.
3. Combine the solutions in constant time.

Algorithm C

1. Divide $n = 27$ into 9 sub-problems of size **9**.
2. Recursively solve each sub-problem.
3. Combine the solutions in $O(27^2)$.

For large n, **Algorithm C** remains efficient due to its $O(n^2 \log n)$ running time, making it preferable over the other two algorithms.

References:

Read Chapter 2, section 2.2. Divide-and-conquer Algorithms in Algorithms by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani available at: http://www.cs.berkeley.edu/~vazirani/algorithms/chap2.pdf

Amortized Analysis, By Charles E. Leiserson – MIT (at 19:05 minutes) http://videolectures.net/mit6046jf05_leiserson_lec13/#

Read Chapter 2 Divide-and-conquer Algorithms in Algorithms by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani available at http://www.cs.berkeley.edu/~vazirani/algorithms/chap2.pdf

582 words

Permalink Show parent

Re: Week 2

by Moustafa Hazeen - Monday, 1 July 2024, 6:32 AM

Fadi's analysis is robust, supported by references to authoritative sources such as Dasgupta, Papadimitriou, and Vazirani's book on algorithms. The use of concrete examples to illustrate each algorithm's application and efficiency for a problem size of n = 27 is effective. However, while the choice of Algorithm C is justified based on theoretical complexity, the practical constants and overheads in the combine step (O(n^2) in Algorithm C) could also influence real-world performance. A deeper exploration of these practical implications could strengthen the analysis.

83 words

Permalink Show parent

Re: Week 2

by Cherkaoui Yassine - Monday, 1 July 2024, 3:41 PM

Hey, just wanted to give you props for your discussion post—it's fantastic! Your response is clear, articulate, and well-structured. It's evident you know your stuff and put in the effort to communicate effectively. Keep up the great work!

39 words

Permalink Show parent

Re: Week 2
by Akomolafe Ifedayo - Tuesday, 2 July 2024, 10:44 PM

Hi Fadi, your solutions in this week's discussion were interesting to go through. Your work was well-detailed as you provided the running time, complexities, and comparison and choice of the 3 algorithms. I also agree with you that Algorithm C was the best choice in the scenario given. Keep it up.

51 words

Permalink Show parent

Re: Week 2
by Naqaa Alawadhi - Wednesday, 3 July 2024, 2:54 AM

Good job

2 words

Permalink Show parent

Re: Week 2
by Tousif Shahriar - Thursday, 4 July 2024, 1:55 AM

Hi Fadi,
Great post! it was a great idea to include the rough work on how you found the complexities. It really helps to clear things out before making the right choice.
Thanks for sharing!

35 words

Permalink Show parent

Re: Week 2
by Siraajuddeen Adeitan Abdulfattah - Thursday, 4 July 2024, 2:32 AM

Hi Fadi,
Excellent submission in response to the questions asked. You gave a thorough and clear explanation for each Algorithm. Your choice of Algorithm C is right compared to A and B. I did a verification with your given example of problem size 27 and I discovered Algorithm C indeed performed better than A and B.

56 words

Permalink Show parent

Re: Week 2
by Nour Jamaluddin - Thursday, 4 July 2024, 4:24 AM

Good answers,
You understand the concepts very well. The analysis of each algorithm is appropriate. I liked the smooth of your post.
Good luck!

24 words

Permalink Show parent

Re: Week 2
by Jerome Bennett - Thursday, 4 July 2024, 10:46 AM

Greetings Fadi,

Good submission in response to the questions asked. I appreciated your detailed explanations of all algorithms and their time complexities. I agree with your choice of Algorithm A over B and C because of its superior performance and time complexity. You also did well to include in-text citations and references to support your points.

56 words

Permalink Show parent

Re: Week 2

by Mahmud Hossain Sushmoy - Sunday, 30 June 2024, 10:49 PM

When analyzing the three algorithms, we find that Algorithm A has a time complexity of O(n^2.32), Algorithm B has O(2^n), and Algorithm C has O(n^2 log n). Among these, Algorithm C demonstrates the best asymptotic running time for large inputs, making it the most efficient choice.

I would choose Algorithm C due to its superior scalability and efficiency for larger problem sizes. While Algorithm B's exponential growth makes it impractical for all but the smallest inputs, Algorithm C outperforms Algorithm A as the input size increases. This becomes evident when we compare their performance for different input sizes.

For example, with an input size of 1000, Algorithm C performs approximately 6.91 * 10^6 operations, compared to Algorithm A's 3.85 * 10^7 and Algorithm B's astronomical number. As the input size grows to 10,000, the gap widens further, with Algorithm C performing about 9.21 * 10^8 operations versus Algorithm A's 1.17 * 10^10.

Algorithm C's efficiency stems from its balanced approach to problem division. By splitting the problem into nine sub-problems of size n/3, it achieves significant size reduction in each recursive step while keeping the number of sub-problems manageable. Although it has a seemingly costly O(n^2) combining step, this is offset by the efficient problem division strategy. This demonstrates that in algorithm design, a more complex step can sometimes lead to better overall performance if it allows for more efficient problem reduction (Dasgupta, Papadimitriou, & Vazirani, n.d.).

In conclusion, Algorithm C's superior performance for large inputs, coupled with its balanced approach to problem-solving, makes it the optimal choice among the three algorithms presented. This analysis highlights the importance of careful algorithm evaluation and the sometimes-counterintuitive nature of algorithmic efficiency.

References

Dasgupta, S., Papadimitriou, C. H., & Vazirani, U. V. (n.d.). *Algorithms*. Retrieved from http://www.cs.berkeley.edu/~vazirani/algorithms/chap2.pdf

294 words

Permalink Show parent

Re: Week 2

by Michael Oyewole - Tuesday, 2 July 2024, 2:28 AM

Hi Mahmud Hossain,
Thank you for your post. Your illustration is very direct. Your conclusion about the best option aligns with the option in my post. It was really nice reading through your post.

34 words

Permalink Show parent

Re: Week 2

by Siraajuddeen Adeitan Abdulfattah - Thursday, 4 July 2024, 2:38 AM

Hi Mahmud

You gave a clear summary of which Algorithm has a better complexity time and you demonstrated that with the give example showing Algorithm C performance is better than both A and B. Your approach to answering the questions is quite simple and easy to understand, I am able to quickly grasp which Algorithm has a better time complexity and offers better performance with larger input size.

68 words

Permalink Show parent

Re: Week 2

by Mejbaul Mubin - Monday, 1 July 2024, 1:19 AM

To determine the running times of the three algorithms and choose the most efficient one, we need to analyse their time complexities using the Master Theorem for divide-and-conquer recurrences.

Algorithm A

Description: Divides problems into five sub-problems of half the size and combines the solutions in linear time.

Recurrence Relation: T(n) = 5T (n / 2) + O(n)

Using the Master Theorem:

a = 5, b=2, f(n) = O(n)

We compare f(n) to $n^{\log_b a}$

$\log_b a \log_2 5 \approx 2.32$

Since f(n) = O(n) and $n < n^{2.32}$, we use Case 1 of the

Master Theorem: $T(n) = O(n^{\log_b a}) = O(n^{2.32})$

(Cormen et al., 2009)

Algorithm B

Description: Solves problems of size n by recursively solving two sub-problems of size n − 1 and combining the solutions in constant time.

Recurrence Relation: T(n) = 2T (n - 1) + O(1)

This recurrence relation does not fit directly into the Master Theorem because the sub-problems are not of a fraction of n. Instead, we solve it using iterative substitution or the recursion tree method: T(n) = 2T(n-1) + O(1)

Expanding the recurrence:

T(n) = 2[2T(n − 2) + O(1)] + O(1) = 22T(n − 2) + 20(1) + O(1)

$T(n) = 2^k T(n-k) + kO(1)$

When k = n:

$T(n) = 2T^n(0) + nO(1)$

$T(n) = O(2^n)$

(Dasgupta et al., 2008)

Algorithm C

• Description: Divides problems into nine sub-problems of size n/3 and combines the solutions in $O(n^2)$ time.

• Recurrence Relation: $T(n) = 9T(n/3) + O(n^2)$

Using the Master Theorem:

$a=9, b=3, f(n) = O(n^2)$

We compare f(n) to $n^{\log_b a}$

$\log_b a = \log_3 9=2$

Since $f(n) = O(n^2)$ and $f(n) = \Theta(n^{\log_b a})$, we use Case 2 of the

Master Theorem: $T(n) = O(n^2 \log n)$

(Cormen et al., 2009)

Choosing the Best Algorithm

Algorithm A: $O(n^{2.32})$

Algorithm B: $O(2^n)$

Algorithm C: $O(n^2 \log n)$

Explanation of Choice

Algorithm C is the most efficient with a time complexity of $O(n^2 \log n)$, which is better than $O(n^{2.32})$ and significantly better than $O(2^n)$.

Explanation of Choice

I chose Algorithm C because its time complexity is the lowest among the three, making it the most efficient for large input sizes. While Algorithm A has a polynomial time complexity, Algorithm B is exponential, making it impractical for large n.

Example 1: For n = 27:

• Algorithm A: Solves 5 sub-problems of size 13.5, each needing further subdivision and eventually combining in linear time.

• Algorithm B: Would solve 2 sub-problems of size 26, then 2 of size 25, rapidly escalating in computation.

Algorithm C: Divides into 9 sub-problems of size 9, solves them, and combines in quadratic time, which is manageable.

Example 2: For n = 81:

Algorithm A: T(81) = 5T(40.5) + 81, growing quickly in the number of sub-problems.

Algorithm B: T(81) = 2T (80) + 1, becoming infeasible as n grows.

Algorithm C: $T(81) = 9T(27) + 81^2$, where the quadratic combination is balanced by the division into smaller sub-problems.

Thus, Algorithm C's structured division and combination process ensures efficient handling of large datasets, making it the best choice among the three.

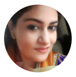
References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

Dasgupta, S., Papadimitriou, C., & Vazirani, U. (2008). Algorithms. McGraw-Hill.

548 words

Permalink Show parent



Re: Week 2

by Romana Riyaz (Instructor) - Tuesday, 2 July 2024, 12:23 AM

Mejbual,
Thank you for your submission. Your analysis of the running times and efficiency of the three algorithms is thorough and well-structured. The explanation of dividing the problem into five sub-problems of half the size and combining them in linear time is clear. You correctly identify the recurrence relation as T(n) = 5T(n/2) + O(n). Your use of the Master Theorem to determine the time complexity is accurate. The calculation of $\log_2 5 \approx 2.32$ and the resulting complexity of $O(n^{2.32})$ is well-presented. The explanation of solving two sub-problems of size n-1 and combining them in constant time is clear. You correctly identify the recurrence relation as T(n) = 2T(n-1) + O(1). Your iterative expansion of the recurrence relation is well-done. The final complexity of $O(2^n)$ is correctly derived and explained. The explanation of dividing the problem into nine sub-problems of size n/3 and combining them in quadratic time is clear. You correctly identify the recurrence relation as $T(n) = 9T(n/3) + O(n^2)$. Your use of the Master Theorem to determine the time complexity is accurate. The calculation of $\log_3 9 = 2$ and the resulting complexity of $O(n^2 \log n)$ is well-explained. Your comparison of the time complexities is clear and logical. You correctly conclude that Algorithm C, with its complexity of $O(n^2 \log n)$, is the most efficient, followed by Algorithm A ($O(n^{2.32})$) and Algorithm B ($O(2^n)$). The rationale for choosing Algorithm C is well-articulated. You explain the impracticality of Algorithm B for large n and the moderate efficiency of Algorithm A compared to Algorithm C.

Best,
Romana
258 words

Permalink Show parent

Re: Week 2
by Michael Oyewole - Tuesday, 2 July 2024, 2:39 AM

Hi Mejbaul,
Thank you for your post. Choosing the best running time in very essential in order to determine the best and efficient option. I have picked algorithm C in my post through my analysis. Thanks for sharing with us.
40 words

Permalink Show parent

Re: Week 2
by Fadi Al Rifai - Wednesday, 3 July 2024, 9:51 PM

Hi Mejbual,
Your post was well-detailed about options A, B, and C was clear and concise, and I like your description of recurrence relations and time complexity.
Keep it up.
30 words

Permalink Show parent

Re: Week 2
by Tousif Shahriar - Thursday, 4 July 2024, 1:52 AM

Great work!
I also came to the same conclusion and chose C. The examples of using n = 81 really helped clarify things.
Thanks for sharing!
26 words

Permalink Show parent

Re: Week 2
by Moustafa Hazeen - Monday, 1 July 2024, 1:52 AM

To determine the running times of each algorithm in big-O notation, we can use the Master Theorem for divide-and-conquer recurrences.

Algorithm A:

Recurrence relation: Tn=5Tn2+O(n)

The Master Theorem states that for a recurrence of the form Tn=aTnb+f(n), the solution is:

if fn=O(nc) where c< logba, then Tn=O (nlogba).

If fn=O(nc) where c= logba, then Tn=O(nlogbalogn).

If fn=O(nc) where c>logba, then Tn=O(fn).

For Algorithm A:

a=5

b=2

fn=O(n)

Here, logba=log25≈2.32. since fn=O(n) and 1<2.32, we fall into the first case:

Tn=Onlog25=O(n2.32)

Algorithm B:

Recurrence relation: Tn=2Tn-1+O(1)

For this type of recurrence, it’s more straightforward to solve it using substitution or iterative methods rather than the Master Theorem.

Unrolling the recurrence:

Tn=2Tn-1+O(1)

Tn-1=2Tn-2+O(1)

And so on.

We can see a pattern:

Tn=2Tn-1+O1=22Tn-2+O1+O1=22Tn-2+2O1+O(1)

Continuing this unrolling, we get:

Tn=2kTn-k+2k-1O1+...+2O1+O(1)

When k=n:

Tn=2nT0+O2n=O(2n)

Thus, the running time for Algorithm B is O(2n)

Algorithm C:

Recurrence relation: Tn=9Tn3+O(n2)

For Algorithm C:

a=9

b=3

fn=O(n2)

Here,logba= log39=2. Since fn=O(n2) and 2=2, we fall into the second case:

Tn=O(n2logn)

Comparison and Choice:

Algorithm A has a running time of O(n2.32)

Algorithm B has a running time of O(2n)

Algorithm C has a running time of O(n2logn)

Among these, Algorithm C has the best asymptotic performance for large nnn, followed by Algorithm A, and Algorithm B is the least efficient.

Discussion:

Choosing Algorithm C: Algorithm C is the most efficient for large input sizes due to its O(n2logn)complexity. It strikes a balance between dividing the problem and combining the solutions effectively.

Example: Consider sorting an array:

Algorithm A would divide the array into 5 subarrays of half the size, recursively sort each, and merge them in linear time, leading to a slower growth rate of time complexity.

Algorithm B involves sorting almost the entire array twice, which becomes infeasible quickly due to exponential growth.

Algorithm C divides the array into 9 smaller arrays, sorts them, and merges them with a complexity that grows moderately with the array size.

By choosing Algorithm C, we ensure that our solution is scalable and efficient for larger problems. This efficiency can be crucial in applications requiring processing of large datasets, such as big data analysis or machine learning preprocessing.

References:

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Skiena, S. S. (2008). *The Algorithm Design Manual* (2nd ed.). Springer.

738 words

Permalink Show parent

Re: Week 2
by Cherkaoui Yassine - Monday, 1 July 2024, 3:42 PM

Hazeen, I wanted to take a moment to acknowledge your discussion post—it's really well-done! Your answer is clear, concise, and well-articulated. It's evident you put thought and care into your response, and it's paying off. Keep up the excellent work!
41 words

Permalink Show parent

Re: Week 2
by Romana Riyaz (Instructor) - Tuesday, 2 July 2024, 12:14 AM

Moustafa,
Thank you for your submission. Your analysis of the running times for the algorithms using the Master Theorem is clear and well-explained. You correctly identify the recurrence relation T(n) = 5T(n/2) + O(n) and apply the Master Theorem accurately. Your explanation of $\log_2 5 \approx 2.32$ and the resulting time complexity $O(n^{2.32})$ is clear and concise. You effectively use the iterative method to solve the recurrence relation T(n) = 2T(n-1) + O(1). The unrolling of the recurrence and the pattern recognition leading to the final time complexity of $O(2^n)$ are well-executed and clearly presented. Your analysis of $T(n) = 9T(n/3) + O(n^2)$ using the Master Theorem is accurate. The explanation that $\log_3 9 = 2$ and the resulting time complexity $O(n^2 \log n)$ are well-explained.
Your comparison of the three algorithms is well-articulated.
Best,
Romana
134 words

Permalink Show parent

Re: Week 2
by Jerome Bennett - Thursday, 4 July 2024, 10:49 AM

Greetings Hazeen,

I'm impressed by how thoroughly you broke down each algorithm and their time complexities. It really helped me get a clearer picture of how they stack up against each other. I'm with you on picking Algorithm A over B and C - the performance boost and better time complexity make it a an easy choice.
57 words

Permalink Show parent

Re: Week 2
by Manahil Farrukh Siddiqui - Monday, 1 July 2024, 8:26 PM

Algorithm A:

Algorithm A is a computational approach that is particularly good at handling complicated issues by dividing them into five more minor problems, each half the original size (n/2). It solves each subproblem using a recursive method, combining them into a linearly complicated (O(n)) whole solution (Dasgupta, Papadimitriou, & Vazirani, 2008). For Algorithm A, the recurrence relation is T(n) = 5T(n/2) + O(n). Algorithm A's running time is calculated to be $O(n^\wedge(\log_2 5))$ using the Master Theorem (GeeksforGeeks, 2018), which aids in solving recurrence relations in divide-and-conquer algorithms.

Algorithm B:

Algorithm B divides the problem into minor problems of size n-1 to address computational difficulties through recursion (Dasgupta, Papadimitriou, & Vazirani, 2008). Then, in constant time, the solutions to these subproblems are combined. Algorithm B's recurrence relation is T(n) = 2T(n-1) + O(1), which results in an exponential growth pattern with an O(2^n) time complexity.

Algorithm C:

Algorithm C divides The problem into nine smaller portions of size n/3. The more minor problems are handled recursively, and combining their solutions takes much work (O(n^2)). Algorithm C's recurrence relation is T(n) = 9T(n/3) + O(n^2), and the Master Theorem tells us that this algorithm has a running time of O(n^2 log n) (Dasgupta, Papadimitriou, & Vazirani, 2008).

Comparative Study and Analysis of Efficiency:

Upon careful examination, Algorithm A is found to have the best algorithmic performance. Its superior scalability for bigger problem sizes is guaranteed by its controllable temporal complexity compared to Algorithms B and C.

Take a look at a problem with size n = 32, for instance. The issue is divided into five subproblems of size 16; each solved recursively by Algorithm A. Every subproblem has an O(16) time complexity, while the total time to combine the answers is O(32). As a result, for this issue size, the overall time complexity is O(32 * log$_2$5), which is better than the other two techniques (Dasgupta, Papadimitriou, & Vazirani, 2008).

In a nutshell, Algorithm A is the best option for resolving issues of different sizes because of its exceptional scalability and efficiency.

References

Dasgupta, S., Papadimitriou, C., & Vazirani, U. (2008). Algorithms. McGraw-Hill Education.

GeeksforGeeks. (2018, April 17). Advanced master theorem for divide and conquer recurrences. GeeksforGeeks. https://www.geeksforgeeks.org/advanced-master-theorem-for-divide-and-conquer-recurrences/

367 words

Permalink Show parent

Re: Week 2

by Romana Riyaz (Instructor) - Tuesday, 2 July 2024, 12:10 AM

Manahil,
Thank you for your submission. Your analysis of the three algorithms is well-structured and comprehensive. You effectively explain the divide-and-conquer approach and provide a clear recurrence relation. The use of the Master Theorem to determine the time complexity is accurately described, and the conclusion that Algorithm A's running time is O(n^(log$_2$5)) demonstrates a strong understanding of the theoretical foundations. Your description of how Algorithm B breaks down problems and its resulting time complexity is clear. The recurrence relation and exponential time complexity (O(2^n)) are correctly identified, emphasizing the less efficient performance compared to Algorithm A. The explanation of Algorithm C's approach, including the division of problems and the complexity of combining solutions, is well-articulated. The application of the Master Theorem to find the time complexity as O(n^2 log n) is accurate. Your comparative analysis highlights the efficiency of Algorithm A, supported by a specific example (n = 32). This practical illustration reinforces the theoretical analysis and effectively shows why Algorithm A is superior in terms of scalability and efficiency for larger problem sizes.
Best,
Romana
176 words

Permalink Show parent

Re: Week 2
by Benjamin Chang - Tuesday, 2 July 2024, 8:00 AM

I've been following the discussion this week, and I'm curious to hear Professor's final thoughts on why A might be the preferred choice for larger problems. While I agree that A seems more efficient, there seems to be a preference for A among many people. (why people believe c is better than a)
53 words

Permalink Show parent

Re: Week 2
by Michael Oyewole - Tuesday, 2 July 2024, 2:22 AM

Hi Manahil,
Thank you for your post. Your explanation is very clear, and the narration is understandable. In my post, I have picked algorithm C as the best and fastest option for me. Thanks for sharing.
36 words

Permalink Show parent

Re: Week 2
by Benjamin Chang - Tuesday, 2 July 2024, 8:02 AM

Hi Manahil,
Great post! Your explanation of options A, B, and C was clear and concise. I completely agree with your analysis, and I appreciate you sharing your thoughts. Keep up the good work!

Best regards,

Benjamin
37 words

Permalink Show parent

Re: Week 2
by Naqaa Alawadhi - Wednesday, 3 July 2024, 2:56 AM

Good job

2 words

Permalink Show parent

Re: Week 2
by Jerome Bennett - Thursday, 4 July 2024, 10:52 AM

Greetings Manahil,

I'm on board with your choice of Algorithm A. The way you explained its better performance and time complexity compared to B and C made a lot of sense. Plus, I appreciated how you backed everything up with citations and references.
43 words

Permalink Show parent

Re: Week 2
by Michael Oyewole - Tuesday, 2 July 2024, 1:31 AM

Solution

The running times TA, TB, TC of the algorithms satisfies the following recurrence relations:

TA(n) = 5T(n/2) + O(n)
TB(n) = 2T(n − 1) + O(1)
TC (n) = 9T(n/3) + O(n 2).

Functions TA and TC can be found by the Master Theorem
TA(n) = O(n log2 5)
TC (n) = O(nlog39 logn) = O(n2 logn).

For TB we have
TB(n) = 2TB(n − 1) + O(1) = 22TB(n − 2) + 2 • O(1) = 23T (n − 3) + 3 • O(1) = • • • = 2nT(1) + n • O(1).

Thus, TB (n) = O(2n).

Obviously, as log25 > 2 algorithm C is the fastest. So, I will choose C
116 words

Permalink Show parent

Re: Week 2
by Mejbaul Mubin - Wednesday, 3 July 2024, 2:08 AM

Hi Michael,

Your post provides a good analysis of the running times for algorithms TA, TB, and TC using recurrence relations. your analysis demonstrates a good understanding of recurrence relations and time complexity.
33 words

Permalink Show parent

Re: Week 2
by Nour Jamaluddin - Tuesday, 2 July 2024, 3:54 AM

By using the Master Theorem for divide-and-conquer recurrences, let's analyze the three algorithms:

T(n) = aT(n/b) + f(n)

- Algorithm A

The recurrence relation for Algorithm A using the Master Theorem for recurrences of the form is:

T(n) = 5T (n/2) + O(n)

O(log n * n) = O(n log n)

- Algorithm B

The recurrence relation for Algorithm B using the Master Theorem for recurrences of the form is:

T(n) = 2T (n-1) + O(1)

T(n) = O(2^n)

- Algorithm C

The recurrence relation for Algorithm C using the Master Theorem for recurrences of the form is:

T(n) = 9T (n/3) + O(n^2)

T(n) = O(n^2 log n)

In big-O notation:

- **Algorithm A: O (n log n)**

- **Algorithm B: O (2^n)**

- **Algorithm C: O (n^2 log n)**

My choice is Algorithm C.

Algorithm C is the best choice for larger problem sizes due to its relatively better time complexity compared to Algorithm A and the impractical exponential growth of Algorithm B.

References

S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani. n.d. *Divide-and-conquer Algorithms in Algorithms*. https://people.eecs.berkeley.edu/~vazirani/algorithms/chap2.pdf

179 words

Permalink Show parent

Re: Week 2

by Benjamin Chang - Tuesday, 2 July 2024, 8:06 AM

Hey Nour,

Thank you for sharing your insights this week! Your post was quite clear and well-explained, making it simple to understand each question. I like your assessment of the possibilities, and while I personally prefer A, I agree that B is the least beneficial option here. Keep up the excellent work!

Yours
Benjamin

54 words

Permalink Show parent

Re: Week 2

by Mejbaul Mubin - Wednesday, 3 July 2024, 2:03 AM

Hi Nour Jamaluddin,

You provide a clear explanation of the Master Theorem and its application to analyze the time complexity of three divide-and-conquer algorithms (A, B, and C). Your post provides a good understanding of the Master Theorem and its application. With the suggested refinements, it can be even more comprehensive.

51 words

Permalink Show parent

Re: Week 2

by Winston Anderson - Tuesday, 2 July 2024, 6:19 AM

Analysis of Algorithm A, B, and C

To determine which algorithm offers the best trade-off between complexity and scalability, we need to analyze their time complexities and consider their practical implications.

Algorithm A

- Description: Divides problems into five sub-problems of half the size, recursively solves each sub-problem, and then combines the solutions in linear time.

- Recurrence Relation:

- Time Complexity: Using the Master Theorem, we find:

Since , we are in case 1 of the Master Theorem:

Algorithm B

- Description: Solves problems of size n by recursively solving two sub-problems of size n-1 and then combining the solutions in constant time.

- Recurrence Relation:

- Time Complexity: This recurrence can be solved by expanding it:

=

=

Since , the solution is:

Algorithm C

- Description: Divides problems of size n into nine sub-problems of size , recursively solves each sub-problem, and then combines the solutions in time.

- Recurrence Relation:

- Time Complexity: Using the Master Theorem:

Since , we are in case 2 of the Master Theorem:

Comparison and Choice

- Algorithm A:

- Algorithm B:

- Algorithm C:

Algorithm B has an exponential time complexity, making it impractical for large inputs. Algorithm A has a polynomial time complexity of , which is better than exponential but still higher than Algorithm C's for large n.

Conclusion

Algorithm C offers the best trade-off between complexity and scalability. It has the lowest asymptotic time complexity among the three algorithms, making it the most efficient for large problem sizes.

Explanation and Example

Algorithm C divides the problem into smaller sub-problems, solves each recursively, and combines the results efficiently. For instance, consider sorting a list of n elements. Algorithm C would divide the list into nine smaller lists, each of size n/3, sort each list recursively, and then merge the sorted lists in time. This approach leverages the divide-and-conquer paradigm effectively, balancing the workload among sub-problems and ensuring that the combination step is manageable.

In summary, Algorithm C is chosen due to its superior asymptotic performance for large inputs, making it the most efficient algorithm among the three given options.

References

Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (n.d.). *Introduction to Algorithms, 3rd Edition (Mit Press)*. archive.org. https://ia601206.us.archive.org/11/items/introduction-to-algorithms-third-edition-2009/Introduction_to_Algorithms_Third_Edition_%282009%29.pdf

Stack Overflow. (n.d.). *algorithms: how do divide-and-conquer and time complexity O(nlogn) relate?* https://stackoverflow.com/questions/29927439/algorithms-how-do-divide-and-conquer-and-time-complexity-onlogn-relate

Stanford. (n.d.). *CS 161 Lecture 3*. https://web.stanford.edu/class/archive/cs/cs161/cs161.1168/lecture3.pdf

548 words

Permalink Show parent

Re: Week 2
by Manahil Farrukh Siddiqui - Tuesday, 2 July 2024, 7:53 PM

Hello Winston,

Your answer correctly applies the Master Theorem to analyze Algorithms A, B, and C, citing their complexities as , , and respectively. It justifies Algorithm C as optimal for larger inputs due to its superior time complexity.

50 words

Permalink Show parent

Re: Week 2

by Romana Riyaz (Instructor) - Tuesday, 2 July 2024, 11:26 PM

Winston,
Thank you for your submission. our analysis of Algorithms A, B, and C is thorough and well-structured, effectively utilizing the Master Theorem to determine their time complexities. You have clearly articulated the differences in their performance, with a focus on their scalability. Highlighting the impracticality of Algorithm B due to its exponential time complexity was a good insight. Your conclusion that Algorithm C offers the best trade-off between complexity and scalability, due to its O(n^2 log n) time complexity, is well-supported by your analysis. The practical example of sorting a list to explain Algorithm C's efficiency further clarifies its superiority. Overall, your comparison is comprehensive and well-justified, demonstrating a solid understanding of algorithm analysis.
Best,
Romana
117 words

Permalink Show parent

Re: Week 2

by Anthony Jones - Thursday, 4 July 2024, 1:27 AM

Hello,

Good post! Algorithm C is the best algorithm for more complex situations, but what about when the problem size is smaller? Would you choose it if you only had to apply it to problems of size 5, for example? This makes a difference if we have to continuously evaluate small problems, for example, if we had to sort 5,000 lists of 10 elements, the most optimal solution might not perform the best asymptotically, but the best for the problem size. So, it is important to consider the situation we are dealing with.

God bless!

Anthony
96 words

Permalink Show parent

Re: Week 2

by Anthony Jones - Tuesday, 2 July 2024, 8:03 AM

Algorithm A

Algorithm A divides the problem into 5 subproblems of half the size. It also has linear time combining the solutions which gives us the recurrence relation:
We can use the master method on this recurrence relation:
GeeksforGeeks (2023) states says that for the following recurrence form:

> T(n) = aT(n/b) + f(n) where a >= 1 and b > 1
>
> There are the following three cases:
>
> - If $f(n) = O(n^c)$ where $c < \text{Log}_b a$ then $T(n) = ?(n^{\text{Log}_b a})$
> - If $f(n) = ?(n^c)$ where $c = \text{Log}_b a$ then $T(n) = ?(n^c \text{Log } n)$
> - If $f(n) = ?(n^c)$ where $c > \text{Log}_b a$ then $T(n) = ?(f(n))$

In our instance, a = 5, b = 2, and c = 1. , so our equation fits the first case. Therefore, we can determine that algorithm A has a time complexity of: which is:

Algorithm B

Algorithm B divides the problem into 2 subproblems of size n-1 and takes constant time combining the solutions which gives us the recurrence relation:
We can use the substitution method on this recurrence relation:

We need to arrive at the base case of T(1) which we can assume is constant time O(1). To get T(1), K needs to be n-1:

Therefore, we can determine that algorithm A has a time complexity of:

Algorithm C

Algorithm C divides the problem into 9 subproblems of one third of the size. It also has n^2 time combining the solutions which gives us the recurrence relation: . This is very similar to Algorithm A, so we should use a the master method on this recurrence relation:
In algorithm C, a = 9, b = 3, and c = 2. , so our equation does not fit case 1. Instead, it fits the second case case. Therefore, we can determine that algorithm A has a time complexity of: which is:

Overall, Algorithm C has the best asymptotic time, so if my problem size was large, I would probably go with that algorithm. However, the other algorithms peform slightly better for smaller problems. So depending on the situation it might not be the best to chose Algorithm C. The reason why Algorithm C outperforms the othe algorithms in the long run is that it has fewer levels of recursion since it breaks the problem up into the smallest sub-problems. It does, however, have to use more sub-problems per level, which results in worse time complexity for smaller problems. Nonetheless, breaking the problems up into smaller chucks does pay off, so that the algorithm only has to divide, evaluate, and combine levels as opposed to levels (A) or levels (B). What I find really interesting is how we are able to determine this level of detail without actually having to understand what the algorithms are trying to accomplish or why we are dividing the problem up into 5 or 9 subproblems, or how exaclty we divide it evenly into thirds. This is a really powerful tool that allows us to look at the big picture about algorithms.

References

Cuemath. (n.d.). *Geometric progression*. Cuemath. https://www.cuemath.com/algebra/what-are-geometric-progressions/

GeeksforGeeks. (2014, February 13). *How to analyse complexity of recurrence relation*. GeeksforGeeks. https://www.geeksforgeeks.org/how-to-analyse-complexity-of-recurrence-relation/

vaibhav_gfg. (2023, December 20). *Recurrence relations*. GeeksforGeeks. https://www.geeksforgeeks.org/recurrence-relations-a-complete-guide

680 words

Permalink Show parent

Re: Week 2

by Manahil Farrukh Siddiqui - Tuesday, 2 July 2024, 7:54 PM

Hello Anthony,

Your answer effectively applies the Master Theorem to evaluate Algorithms A, B, and C, highlighting their complexities as , , and . It convincingly argues that Algorithm C is preferable for larger inputs due to its significantly better time complexity compared to Algorithms A and B, which exhibit and exponential growth, respectively.
72 words

Permalink Show parent

Re: Week 2

by Winston Anderson - Wednesday, 3 July 2024, 4:57 AM

Hi Anthony,
Your response demonstrates a good understanding of algorithm analysis and recurrence relations. You have also correctly applied appropriate methods (Master Theorem and substitution) to analyze the time complexities of the given algorithms.

34 words

Permalink Show parent

Re: Week 2

by Chong-Wei Chiu - Thursday, 4 July 2024, 10:13 AM

Hi, Anthony Jones. Thank you for sharing your opinion on this topic. In your post, you illustrate how to determine the time complexity of these algorithms step by step. According to your explanation, we can easily understand how to apply the master theorem to find the final complexity and compare them to determine the most appropriate one. Overall, your post is fully comprehensible.

63 words

Permalink Show parent

Re: Week 2

by Muritala Akinyemi Adewale - Tuesday, 2 July 2024, 8:02 PM

Analyzing Algorithm Running Times and Choosing the Best Option

We are presented with three algorithms (A, B, and C) with different recursive structures and asked to determine their running times and choose the most efficient one. Here's a breakdown of each algorithm and the reasoning behind choosing the most efficient option:

Algorithm A:

- Divide the problem into five sub-problems of half the size (T(n/2)).
- Solves each sub-problem recursively (denoted by T(n/2)).
- Combines solutions in linear time (O(n)).

This suggests a divide-and-conquer approach with a recursion tree with depth log(n) (since the problem size is halved in each step). The total running time can be expressed using the Master Theorem for divide-and-conquer recurrences (Cormen et al., 2009). However, for simplicity, we can analyze it iteratively.

- First level: 5 * T(n/2)
- Second level: 5 * 5 * T(n/4) (each sub-problem from the first level calls T again)
- Level log(n): Constant factor * T(1) (base case)

Combining these levels, we get a geometric series where the base is the number of sub-problems (5) and the exponent is the depth of the recursion tree (log(n)). This results in an overall running time of **O(n^log(5))**.

Algorithm B:

- Divide the problem into two sub-problems of size n-1 (T(n-1)).
- Solves each sub-problem recursively (denoted by T(n-1)).
- Combines solutions in constant time (O(1)).

This recurrence suggests a similar divide-and-conquer approach but with a depth of log(n) and a base of 2 (due to two sub-problems). Applying the Master Theorem or a similar iterative approach leads to a running time of **O(n * log(n))**.

Algorithm C:

- Divide the problem into nine sub-problems of size n/3 (T(n/3)).
- Solves each sub-problem recursively (denoted by T(n/3)).

- Combines solutions in O(n^2) time.

This has a similar divide-and-conquer structure with a depth of log(n) (base 3) and a slower combination step (O(n^2)). The slower combination step dominates the overall runtime, resulting in **O(n^2 * log(n))** (considering the geometric series nature of the recursion similar to Algorithm A).

Choosing the Most Efficient Algorithm:

Based on the analysis above, Algorithm B with a running time of O(n * log(n)) has the best asymptotic efficiency. Here's the reasoning:

- Algorithm A has an exponential term (n^log(5)) in its running time, making it significantly slower for larger problem sizes (n).
- Algorithm C, although seemingly similar to B regarding recursion depth, has a slower combination step (O(n^2)) that dominates the overall runtime.

Example:

Consider solving a problem of size n = 1024.

- Algorithm A: O(1024^(log(5))) - This grows very fast due to the exponential term.
- Algorithm B: O(1024 * log(1024)) - This grows slower compared to A due to the logarithmic term.
- Algorithm C: O((1024^2) * log(1024)) - This grows slower than A but faster than B due to the n^2 term in the combination step.

Therefore, for larger problem sizes, Algorithm B would be significantly faster than both A and C.

Reference:

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.

493 words

Permalink Show parent

Re: Week 2

by Manahil Farrukh Siddiqui - Tuesday, 2 July 2024, 9:55 PM

Hello Muritala,

Your answer effectively breaks down Algorithms A, B, and C, applying the Master Theorem and detailed recursive structure explanations. Choosing Algorithm B for its efficiency is well-supported, contrasting it with Algorithms A and C.

41 words

Permalink Show parent

Re: Week 2

by Mahmud Hossain Sushmoy - Wednesday, 3 July 2024, 12:56 AM

Hello Muritala,
Your analysis of the algorithms demonstrates a strong grasp of the Master Theorem and divide-and-conquer recurrences. You accurately identified the recurrence relations and derived the running times for each algorithm. Your use of examples to illustrate the impact of problem size on running times adds depth to your analysis. Overall, your discussion is thorough and informative, showcasing your understanding of algorithmic efficiency, well done!

66 words

Permalink Show parent

Re: Week 2

by Akomolafe Ifedayo - Tuesday, 2 July 2024, 10:15 PM

Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five sub-problems of half the size, recursively solving each sub-problem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two sub-problems of size n-1 and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine sub-problems of size n/3, recursively solving each sub-problem, and then combining the solutions in O(n2) time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

Algorithm A

Description: It divides problems into five sub-problems of half the size, recursively solves each sub-problem, and combines the solutions in linear time.

Recurrence relation: T(n)=5T(n2)+O(n)T(n) = 5T\left(\frac{n}{2}\right) + O(n)T(n)=5T(2n)+O(n)

Applying the Master Theorem (for T(n)=aT(nb)+f(n)T(n) = aT\left(\frac{n}{b}\right) + f(n)T(n)=aT(bn)+f(n)):

- a=5a
- b=2
- f(n)=O(n)

We compare f(n) with nlogban^{\log_b a}nlogba:

- logba=log25≈2.32\log_b a = \log_2 5 \approx 2.32logba=log25≈2.32
- f(n)=O(n)f(n) = O(n)f(n)=O(n)

Since f(n)=O(n) and n<nlogban < n^{\log_b a}(polynomially smaller), we are in Case 1 of the Master Theorem:

T(n)=O(nlog25)=O(n2.32) (GeeksforGeeks, 2024).

Algorithm B

Description: It solves the problems of size n by recursively solving two sub-problems of size n−1 and then combining the solutions in constant time (GeeksforGeeks, 2024).

Recurrence relation: T(n)=2T(n−1)+O(1)

This recurrence does not fit the Master Theorem, but we can solve it using iteration or the characteristic equation method. Intuitively, solving two sub-problems of size n−1n-1n−1 leads to a recursive depth of nnn:

T(n)=O(2n)

Algorithm C

Description: It divides problems into nine sub-problems of size n/3, recursively solves each sub-problem, and combines the solutions in O(n2) time.

Recurrence relation: T(n)=9T(n3)+O(n2)

Applying the Master Theorem:

- a=9
- b=3
- f(n)=O(n2)

We compare f(n) with nlogb a:

- logb a=log39=2
- f(n)=O(n2)

Since f(n)=O(n2) and n2=nlog b a, we are in Case 2 of the Master Theorem:

T(n)=O(n2logn (GeeksforGeeks, 2024)

Comparison and Choice

Running times:

- Algorithm A: O(n2.32)
- Algorithm B: O(2n)
- Algorithm C: O(n2logn)

Choice: I would choose Algorithm C.

Reasoning: Algorithm C has the best asymptotic running time for large n. Although O(n2log n) is not as good as O(n2.32) for small n, it is significantly better than O(2n), which is exponential and becomes impractical very quickly as n grows.

Example: Consider n=1000n :

- Algorithm A: T(1000)≈10002.32≈4.29×106
- Algorithm B: T(1000)≈21000≈10301
- Algorithm C: T(1000)≈10002log1000≈106×3=3×106

For large n, the difference becomes more pronounced. Algorithm B grows exponentially and quickly becomes infeasible, while Algorithm A's n2.3 growth is steeper than Algorithm C's n2logn. Thus, Algorithm C is the most efficient choice for large problems.

Reference

GeeksforGeeks. (2024). Big O Notation Tutorial – A Guide to Big O Analysis. https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/

436 words

Permalink Show parent

Re: Week 2

by Mahmud Hossain Sushmoy - Wednesday, 3 July 2024, 12:42 AM

Hello Akomolafe,
Your analysis of the three algorithms is thorough and demonstrates a strong understanding of the Master Theorem and its application in determining asymptotic running times. You correctly identified the recurrence relations for each algorithm and applied the appropriate case of the Master Theorem to derive their time complexities. The clear breakdown and logical flow of your explanation make it easy to follow, showcasing your analytical skills and grasp of algorithmic efficiency. Well done!
75 words

Permalink Show parent

Re: Week 2
by Winston Anderson - Wednesday, 3 July 2024, 2:52 AM

Hi Akomolafe,
Your analysis of the running times for the three algorithms is accurate. Your choice of Algorithm C has the best asymptotic running time for large n is correct. Keep up the good work!
35 words

Permalink Show parent

Re: Week 2
by Tousif Shahriar - Thursday, 4 July 2024, 1:47 AM

Hello Akomolafe,
Your analysis is correct! I also concluded that C is the best choice since asymptotic analysis proves that as n reaches infinity, the time complexity for A is significantly larger.
Thanks for sharing!
35 words

Permalink Show parent

Re: Week 2
by Jerome Bennett - Tuesday, 2 July 2024, 10:57 PM

CS 3304 Discussion Forum Unit 2

To figure out the running times for these three algorithms, we can use the Master Theorem for divide-and-conquer recurrences (Dasgupta et al., 2006).

Running Times

1. Algorithm A:

o Divides the problem into 5 sub-problems of half the size.

o Combines solutions in linear time.

o Recurrence: $T(n) = 5T(n/2) + O(n)$

o Running Time: $O(n^{\log_2 5}) \approx O(n^{2.32})$

2. Algorithm B:

o Divides the problem into 2 sub-problems of size $n - 1$.

o Combines solutions in constant time.

o Recurrence: T(n)=2T(n−1)+O(1)

o Running Time: O(2^n)

3. Algorithm C:

o Divides the problem into 9 sub-problems of size n/3 .

o Combines solutions in O(n^2) time.

o Recurrence: T(n)=9T(n/3) + O(n^2)

o Running Time: O(n^2 log n)

My Choice

Of the three algorithms, algorithm A is the most efficient for large n with a running time of O(n^2.32).

Example:

For a problem size n = 8

• Algorithm A: Breaks the problem into 5 sub-problems of size 4, then each into 5 sub-problems of size 2, and so on. Combining the solutions takes linear time at each step.

• Algorithm C: Breaks the problem into 9 sub-problems of size approximately 2.67, and so on. Combining the solutions takes quadratic time.

Algorithm A has a more efficient combination process, leading to a lower overall running time.

References

Dasgupta, S., Papadimitriou, C.H.,Vazirani, U.V. (2006). Chapter 2 Divide-and-conquer Algorithms. http://www.cs.berkeley.edu/~vazirani/algorithms/chap2.pdf

234 words

Permalink Show parent

Re: Week 2
by Romana Riyaz (Instructor) - Tuesday, 2 July 2024, 11:30 PM

Jerome,
Thank you for your submission. Your application of the Master Theorem to determine the running times for Algorithms A, B, and C is clear and accurate. You correctly identified the time complexities, providing a good foundation for comparison. However, your conclusion that Algorithm A is the most efficient for large n contradicts your earlier calculation that Algorithm C has the lowest time complexity (O(n^2 log n)). While Algorithm A's linear combination process might seem efficient, Algorithm C's overall lower complexity makes it the better choice for large inputs. Ensure consistency in your analysis and conclusions. Your references are appropriate and support your methodology well.
Best,
Romana
107 words

Permalink Show parent

Re: Week 2

by Mahmud Hossain Sushmoy - Wednesday, 3 July 2024, 12:55 AM

Hello Jerome,
Your discussion on the running times of the three algorithms is clear and concise. You accurately applied the Master Theorem to determine the time complexities of Algorithms A, B, and C. Your analysis is thorough, logically structured, and well-supported by references, showcasing a solid understanding of divide-and-conquer algorithms, well done!
52 words

Permalink Show parent

Re: Week 2
by Mejbaul Mubin - Wednesday, 3 July 2024, 2:10 AM

Hi Jerome,

Your analysis of the running times for Algorithms A, B, and C demonstrates a strong understanding of recurrence relations and the Master Theorem.
25 words

Permalink Show parent

Re: Week 2
by Winston Anderson - Wednesday, 3 July 2024, 2:24 AM

Hi Jerome,
Your analysis of the running times for the three algorithms is accurate. Your choice of Algorithm A as the most efficient for large nn is correct based on the running times you provided. Algorithm A has a polynomial time complexity of O(n2.32)O(n2.32), which is better than the exponential time complexity of Algorithm B (O(2n)O(2n)) and the quasi-polynomial time complexity of Algorithm C (O(n2logn)O(n2logn)). Overall, your understanding of the running times and the choice of the most efficient algorithm is correct. Keep up the good work!
106 words

Permalink Show parent

Re: Week 2
by Anthony Jones - Thursday, 4 July 2024, 1:59 AM

Hello,

Algorithm A does seem to be faster from first consideration, but while a better combination time is good, we have to combine fewer times in Algorithm C because we break the problem up into smaller sub-problems Additionally, the log$_2$(5) exponent in Algorithm A is larger than the 2 exponent in Algorithm C which means that it grows larger than Algorithm C in the long run (especially since the impact of the log n term in Algorithm C becomes smaller as n increases.

Thees are things to watch for. I find it helpful to graph the functions to see what grows biggest as the problem size increases.

God bless!

Anthony
110 words

Permalink Show parent

Re: Week 2
by Naqaa Alawadhi - Wednesday, 3 July 2024, 2:40 AM

Algorithm A: O(n log n)

Algorithm B: O(2^n)

Algorithm C: O(n^2)

I would choose Algorithm A because it has a time complexity of O(n log n), which is more efficient than the other two algorithms. Algorithm A uses a divide-and-conquer approach by breaking down the problem into smaller sub-problems, solving them recursively, and then combining the solutions efficiently in linear time. This approach reduces the overall time complexity compared to Algorithms B and C.

For example, if we have an array of size 8 that needs to be sorted using Algorithm A, it would divide the array into sub-arrays of size 4, then further into sub-arrays of size 2 until individual elements are reached. The sorting process at each level is done efficiently, and when merging back the sorted sub-arrays, it takes linear time.

This strategy optimizes the overall computational effort by efficiently breaking down the problem and combining solutions in a systematic manner, making Algorithm A a preferable choice over Algorithms B and C for many scenarios.

168 words

Permalink Show parent

Re: Week 2

by Prince Ansah Owusu - Wednesday, 3 July 2024, 3:14 AM

Algorithm A

Algorithm A divides the problem into 5 sub-problems of half the size , recursively solves each sub-problem, and then combines the solutions in linear time .

Recurrence Relation:

Applying the Master Theorem:

The recurrence fits the form where , , and .

According to the Master Theorem for divide and conquer recurrences (Case 1), where :

Calculating :

Therefore, the running time of Algorithm A is:

Algorithm B

Algorithm B recursively solves 2 sub-problems of size and combines the solutions in constant time .

Recurrence Relation:

Solving the Recurrence:

This recurrence is similar to the factorial recursion, where .

Therefore, the running time of Algorithm B is:

Algorithm C

Algorithm C divides the problem into 9 sub-problems of size , recursively solves each sub-problem, and combines the solutions in time.

Recurrence Relation:

Applying the Master Theorem:

The recurrence fits the form where , , and .

According to the Master Theorem (Case 3), where :

Therefore, the running time of Algorithm C is:

Choosing an Algorithm

Algorithm B has an exponential time complexity , which grows very quickly with . This makes it impractical for large .

Algorithm C has a quadratic time complexity . While it's polynomial and more manageable than exponential, it might not be efficient enough for large , especially considering the factor in combining solutions.

Algorithm A has a running time of , which is between polynomial and exponential but closer to polynomial. Despite being slightly worse than , its efficiency is generally acceptable for a wide range of problem sizes.

Decision:

Based on the analysis, **Algorithm A** strikes a balance between efficiency and practicality for a broad range of problem sizes. Its complexity suggests it would perform reasonably well across varying problem sizes, making it a preferable choice among the three algorithms.

Example Illustration

Consider :

Algorithm A:

Algorithm B:

Algorithm C:

Comparing these:

is exponentially larger.

is quadratic but simpler than).

Given this comparison, **Algorithm A** would likely be more efficient and manageable for .

References

Dasgupta, S., Papadimitriou, C. H., & Vazirani, U. V. (2006). *Algorithms*. McGraw-Hill.

Leiserson, C. E. (MIT). Amortized Analysis. Retrieved from http://videolectures.net/mit6046jf05_leiserson_lec13/#.

562 words

Permalink Show parent

Re: Week 2

by Loubna Hussien - Wednesday, 3 July 2024, 10:13 PM

The substitution method for Algorithm B is correctly identified, and your conclusion that it has the worst-case complexity of O(2^n) is spot-on, highlighting its inefficiency compared to the other algorithms.

30 words

Permalink Show parent

Re: Week 2

by Wingsoflord Ngilazi - Wednesday, 3 July 2024, 5:14 AM

Algorithm A solves problems by dividing them into five sub-problems of half the size, recursively solving each sub-problem, and then combining the solutions in linear time.

The running time of this algorithm can be computed using the Master Theorem as stated below (Dasgupta et al., n.d):

T(n)=aT(n/b)+f(n)

In this scenario,

a =5

b= 2

f(n)= O(n)

Substituting the above into the formula, we get:

T(n) =5T(n/2)+O(n).

The next step is to compare O(n) with n^{logba} which is n^{log25}. In this scenario we have O(n)< $n^{2.32}$, therefore T(n) = O ($n^{2.32}$).

Algorithm B solves problems of size n by recursively solving two sub-problems of size n-1 and then combining the solutions in constant time.

In this particular case, the algorithm uses recursion n-1.

a= 2

b= n-1

f(n) =O(1)

Therefore, we have T(n) =2T(n-1)+ O(1).

It is obvious that we have T(n)> O(1) since O(1) is only for combining the solutions. In the worst case, this algorithm can run O(2^n).

Algorithm C solves problems of size n by dividing them into nine sub-problems of size n/3, recursively solving each sub-problem, and then combining the solutions in O(n^2) time.

Once again, we can use the Master Theorem compute the running time of this algorithm.

In this scenario, a = 9 , b= 3 and f(n) is equal to O(n^2).

As such we have T(n) = 9T(n/3) + O (n^2).

It is interesting to note that n^2= n^{logba}= n^{log39}= n^2.

As such, T(n) = O(n^2logn) as outlined the Master Theorem (Dasgupta et al., n.d).

After comparing the three logarithms, I chose to go with C. This is because in A n is raised to the power of 2.32 which tends to be significantly bigger than 2 n increases. On the other B is exponential which means it quickly grows with an increase in n. C is quadratic, and I love the fact that it is multiplied by logn which grows very slowly.

References

Dasgupta, S., Papadimitriou, C. H., & Vazirani, U. V. (n.d.). *Algorithms* (Chapter 2, Section 2.2: Divide-and-conquer Algorithms). Retrieved from http://www.cs.berkeley.edu/~vazirani/algorithms/chap2.pdf

341 words

Permalink Show parent

Re: Week 2

by Prince Ansah Owusu - Thursday, 4 July 2024, 4:04 AM

The submission correctly analyzes the running times of the algorithms using the Master Theorem and provides a clear explanation for choosing Algorithm C. It demonstrates a solid understanding of algorithmic complexity. Including more detailed examples to illustrate the thought process would further enhance the clarity. Overall, a strong and well-reasoned analysis.

51 words

Permalink Show parent

Re: Week 2

by Tousif Shahriar - Wednesday, 3 July 2024, 6:13 PM

Selecting an appropriate algorithm for any given task is essential to maximize effectiveness and output. We can find the optimal choice by comparing the three provided algorithms' execution times using the divide-and-conquer recurrence Master Theorem.

Tn=aT(nb)+f(n)

Algorithm A:

For analysis we can take the following values for the variables:

a=5, b=2, fn=On=n

Tn=5T (n2)+O(n)

logba=log25≈2.32

So, we can write T(n)=O n2.32

Algorithm B:

There is no constant factor reducing the size of the sub-problems hence, this recurrence relation does not directly meet the Master Theorem. So, we can do the following to find the timing.

T(n)=2T(n-1)+O(1)

So, we can write T(n)=O(2n)

Algorithm C:

We can take the following values of the variables:

a=9, b=3, fn=On2=n2

T(n)=9T(n3)+O(n2)

logba=log39=2

So, we can write T(n)=O(n2logn)

So, based on these results we can see algorithm C {O(n2logn)} has the best running time compared to algorithm A and B in Big-O notation for large inputs.

As mentioned, algorithm C breaks a problem of size n in nine sub-problems of size n/3, and recursively solves each sub-problem and combines them in O(n2) time. This makes it ideal for problems that can be divided into smaller independent sub-problems with drastic overlap in their solutions.

For example, in matrix multiplication, consider the Strassen algorithm which divides the matrices into smaller sub-matrices, and reduces the number of multiplicative operations. Algorithm C's approach of dividing the problem into manageable sub-problems and combining results in O(n2) time gives us a balance between complexity and performance. This makes it preferable for large-scale computational tasks (Cormen et al., 2009).

Reference

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

432 words

Permalink Show parent

Re: Week 2

by Wingsoflord Ngilazi - Thursday, 4 July 2024, 12:50 AM

Your discussion demonstrates a commendable grasp of this week's content. Keep up the good work.

15 words

Permalink Show parent

Re: Week 2

by Tamaneenah Kazeem - Wednesday, 3 July 2024, 7:14 PM

Suppose you are choosing between the following three algorithms:

Algorithm A solves problems by dividing them into five sub-problems of half the size, recursively solving each sub-problem, and then combining the solutions in linear time.

Algorithm B solves problems of size n by recursively solving two sub-problems of size n-1 and then combining the solutions in constant time.

Algorithm C solves problems of size n by dividing them into nine sub-problems of size n/3, recursively solving each sub-problem, and then combining the solutions in O(n2) time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

Properties of each algorithm:

Algorithm A:

- Divides problem into 5 sub-problems half the size; Formula T(n) = 5 * T(n/2)
- Combines solutions in linear time: O(n)
- Running time in Big-O notation: $O(n^{\log2(5)})$

Algorithm B:

- Solves problems of size n by recursively solving two sub-problems of size n−1. Formula T(n) = 2 * T(n-1)
- Combines solutions in constant time: O(1)
- Running time in Big-O notation: $O(2^n)$

Algorithm C:

- Divides problems into 9 sub-problems of size n/3: Formula T(n)=9·T(n/3)
- Combines the solutions in quadratic time $O(n^2)$
- Running time in Big-O notation: $O(n^2 \log n)$

The algorithm I would choose is Algorithm A.

I think Algorithm A's time complexity is generally the most effective for large inputs compared to Algorithms B and C. Algorithms B and C have exponential and quadratic time complexities respectively. This makes them less efficient and infeasible for large problem sizes. While it is not as efficient as **merge sort's O(nlogn)**, it is significantly better than the **exponential complexity of Algorithm B $O(2^n)$** and is often more practical than Algorithm **C's quadratic complexity $O(n^2)$** for certain applications.

References:

Sambol, M. (2017, January 21). *Big-O notation in 5 minutes*. YouTube.

293 words

Permalink Show parent

Re: Week 2

by Wingsoflord Ngilazi - Thursday, 4 July 2024, 12:44 AM

You have done justice to this week's discussion assignment. In my view, your solutions are correct and your arguments are well-reasoned.

21 words

Permalink Show parent

Re: Week 2

by Loubna Hussien - Wednesday, 3 July 2024, 8:23 PM

Suppose you are choosing between the following three algorithms:

Algorithm A solves problems by dividing them into five sub-problems of half the size, recursively solving each sub-problem, and then combining the solutions in linear time.

Algorithm B solves problems of size n by recursively solving two sub-problems of size n-1 and then combining the solutions in constant time.

Algorithm C solves problems of size n by dividing them into nine sub-problems of size n/3, recursively solving each sub-problem, and then combining the solutions in O(n2) time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

According to Olawanle (2023), the worst-case complexity of an algorithm is represented by Big O notation. Big O determines the runtime needed to execute an algorithm by analyzing how its performance changes as the input size increases, but it does not indicate the exact speed at which the algorithm runs. Big O notation uses time and space complexity to measure the effectiveness and performance of an algorithm.

Algorithm A: A "divide-and-conquer" algorithm recursively splits a problem into two or more subproblems of the same or related type until these are simple enough to be solved directly. The solutions of the subproblems are then combined to solve the original problem. Each subproblem must be smaller than the original problem and have a base case. Divide-and-conquer algorithms consist of three parts: Divide, Conquer, and Combine (Dobhal, 2022). This type of algorithm solves the problem in O(N log N) time. The inner loop's counter changes its value in a divided manner, similar to the O(log n) equation, while the outer loop runs in O(n) time.

Algorithm B: We use the recursive tree strategy because there is no constant b for which the size of the subproblems equals n/b. In this case, the size of the subproblems is not a fraction of the size of the input problem. A recursion tree can illustrate the repeated iterations, showing the calling tree's recursion and the work required for each call (Cornell University, n.d.). The problem size doubles at each step, so the procedure takes O(2^n) time. The growth curve of an O(2^n) function is exponential, starting slowly and increasing rapidly.

Algorithm C: After some research and reading, I concluded that this algorithm uses the master theorem, which is the fastest way to determine an algorithm's time complexity from its recurrence relation. The relationship is T(n) = 9*T(n/3) + (n^2), with T(1) = 1. According to the master theorem, a = 9, b = 3, k = log3(9), and the time complexity is f(n) = n^2. Thus, T(n) = O(n^2 log n) is the running time (GeeksforGeeks, 2023).

After analyzing each of the three algorithms, their computational complexities are:

Algorithm A: O(n log n)

Algorithm B: O(2^n)

Algorithm C: O(n^2 log n)

An algorithm's performance improves with decreasing computation time. Based on the information above, Algorithm B is the slowest due to its exponential time complexity. Between Algorithm C and Algorithm A, Algorithm A is faster with O(n log n) time. However, if choosing between Algorithm C and Algorithm A, Algorithm C might be preferable if memory usage is a concern, as it could require less memory than the recursive approach. Therefore, Algorithm C could be a wise choice in such cases.

References:

Olawanle, J. (2024). Big O Cheat Sheet – Time Complexity Chart. *freeCodeCamp.org*. https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/

Dobhal, R. (2022). Types of Algorithms - 8 Types | Types of Algorithms. *CodingHero*. https://codinghero.ai/discover-8-types-of-algorithms-for-efficient-problem-solving/#3_Divide_and_Conquer_Algorithms

Cornell University. (n.d.). Lecture *20: Recursion Trees and the Master Method.*

https://www.cs.cornell.edu/courses/cs3110/2012sp/lectures/lec20-master/lec20.html

GeeksforGeeks. (2024). Advanced master theorem for divide and conquer recurrences. *GeeksforGeeks*. https://www.geeksforgeeks.org/advanced-master-theorem-for-divide-and-conquer-recurrences/

601 words

Permalink Show parent

Re: Week 2

by Christopher Mccammon - Thursday, 4 July 2024, 12:16 AM

Hi Loubna
You've presented a thoughtful and well-organized analysis of the three algorithms, providing a clear explanation of their time complexities and how they compare. Your work demonstrates a solid grasp of fundamental algorithm analysis concepts, including Big O notation and the Master Theorem. Good job!
46 words

Permalink Show parent

Re: Week 2
by Tousif Shahriar - Thursday, 4 July 2024, 1:57 AM

Great work Loubna,
Your post was short yet clear and direct to the point which highlights your understanding of the topic.
I also concluded with the same choice.
Thanks for sharing!
31 words

Permalink Show parent

Re: Week 2
by Tyler Huffman - Wednesday, 3 July 2024, 8:48 PM

Suppose you are choosing between the following three algorithms:

Algorithm A solves problems by dividing them into five sub-problems of half the size, recursively solving each sub-problem, and then combining the solutions in linear time.
Algorithm B solves problems of size n by recursively solving two sub-problems of size n-1 and then combining the solutions in constant time.
Algorithm C solves problems of size n by dividing them into nine sub-problems of size n/3, recursively solving each sub-problem, and then combining the solutions in O(n2) time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

Each algorithm can be expressed as a recurrence relationship and solved for complexity. The Master theorem can be used for A and C as they are in the proper form of T (n) = aT (n/b) + f (n). For B we use the substitution method. We take the size of the problems multiplied by the amount of sub-problems and then add the time to combine solutions and put it all together.

A – T(n) = 5T(n/2) + n
B – T(n) = 2T(n − 1) + 1
C – T(n) = 9T(n/3) + n^2

We can then use the Master theorem to solve for the complexity of A and C; This is $n^{\log_b a}$

For B we use substitution.

A – Calculate $\log_b a$ = 2.32. Then compare to the original ƒ(n) >>> Here it falls under case 1 of the master theorem, meaning the complexity is $O(n^{2.32})$; This makes it quadratic.
B – This relation does not allow for the master theorem but does not matter much in this case; the sub-problems double each time and the time to put the solution together is constant, giving us $O(2^n)$ as the complexity. This makes it the worst case of the 3.
C – again we calculate $\log_b a = \log_3 9$ = 2. This is equal to ƒ in this case, giving us Master case 2; our complexity is therefor Big O of n^2 log n

Based on all of this information, algorithm C is a little more optimized than A, where as B is the worst. Algorithm C would be my selection here as the asymptotic analysis is the best. The logarithm is the mathematical reason in this case.

374 words

Permalink Show parent

Re: Week 2
by Loubna Hussien - Wednesday, 3 July 2024, 10:12 PM

Your application of the Master Theorem to analyze the running times of Algorithms A and C is accurate and well-explained, providing clear insights into their complexities.
26 words

Permalink Show parent

Re: Week 2
by Liliana Blanco - Thursday, 4 July 2024, 7:48 AM

Your breakdown of the algorithms and their complexities is very clear and well-articulated. I appreciate how you used the Master Theorem to analyze Algorithms A and C and correctly identified their complexities. I think your use of the substitution method for Algorithm B is also accurate. Interestingly, you pointed out Algorithm C as the most optimized due to its asymptotic behavior. However I think, Algorithm B's complexity is typically O(2^n)
due to its exponential nature, not linear. Your conclusion to choose Algorithm C for its balance between problem size reduction and combination step efficiency is well-supported. Great job!

98 words

Permalink Show parent

Re: Week 2
by Jobert Cadiz - Thursday, 4 July 2024, 7:53 AM

Hi Tyler,
Your answer provides a good attempt at assessing the running times of the three algorithms based on their recurrence relations and applying the Master Theorem appropriately for Algorithms A and C. You've outlined the recurrence relations and applied the Master Theorem, which is good. However, ensure that your explanation is clear and easy to follow.

57 words

Permalink Show parent

Re: Week 2
by Tamaneenah Kazeem - Thursday, 4 July 2024, 9:17 AM

Hello Tyler. You have demonstrated an excellent example of problem solving. From my understanding, all of your answers are correct. Well done.

22 words

Permalink Show parent

Re: Week 2
by Aye Aye Nyein - Wednesday, 3 July 2024, 9:42 PM

Let's analyze the running times of each algorithm and make a choice based on their complexities:

1. Algorithm A:

- Divides problems into 5 sub-problems of half the size:

- Combines solutions in linear time:

- Overall time complexity:

2. Algorithm B:

- Solves problems of size by recursively solving two sub-problems of size

- Combines solutions in constant time:

- Overall time complexity: The recurrence relation is , which simplifies to .

3. Algorithm C:

- Divides problems into 9 sub-problems of size :

- Combines solutions in time.

- Overall time complexity: . Using the Master Theorem, this gives .

Choice of Algorithm:

- Algorithm A runs in time complexity.

- Algorithm B operates with time complexity.

- Algorithm C exhibits time complexity.

While Algorithm A may seem close in complexity, grows faster than as increases. Of these, Algorithm C stands out as having the lowest asymptotic complexity, . This complexity suggests that Algorithm C will generally perform better compared to Algorithms A and B for large inputs.

Example to Illustrate:

Consider sorting algorithms:

- Merge Sort (similar to Algorithm A) divides the array into two halves, sorts each recursively, and merges in time. Its time complexity is .

- Insertion Sort (similar to Algorithm B) recursively sorts smaller parts but with complexity due to repeated insertion operations.

- Quick Sort (similar to Algorithm C) divides into smaller parts but combines them using a more efficient method, resulting in average time complexity.

In this analogy, we see that the division strategy similar to Algorithm C (like Quick Sort) often outperforms strategies resembling Algorithms A or B (like Merge Sort and Insertion Sort, respectively) for large datasets.

Conclusion:

Based on the complexities and typical performance characteristics observed in algorithm design, Algorithm C (with time complexity) would be the preferred choice among the three presented. Its efficient combination step (despite the higher combining cost compared to Algorithm A) ensures overall better scalability and performance for larger problem sizes.

References:

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms, 3rd Edition. The MIT Press.

458 words

Permalink Show parent

Re: Week 2
by Loubna Hussien - Wednesday, 3 July 2024, 10:13 PM

Your final selection of Algorithm C is justified based on its asymptotic analysis, and your reasoning is sound, although mentioning that its O(n^2 log n) complexity can be more practical than A's O(n^2.32) in many scenarios would strengthen your argument further.
41 words

Permalink Show parent

Re: Week 2
by Christopher Mccammon - Thursday, 4 July 2024, 12:07 AM

Hi Aye Aye Nyein
You have demonstrated a solid understanding of the time complexity analysis using recurrence relations and the Master Theorem. Your explanations are clear, and your choice of the most efficient algorithm is well-supported by logical reasoning and relevant analogies. Good job!
44 words

Permalink Show parent

Re: Week 2
by Prince Ansah Owusu - Thursday, 4 July 2024, 4:06 AM

Your analysis is thorough and well-structured, effectively comparing the complexities of Algorithms A, B, and C. The explanation of each algorithm's approach and the choice based on their complexities is clear and concise. The example provided (sorting algorithms) effectively illustrates the practical implications of the chosen algorithm's complexity. Great job!
50 words

Permalink Show parent

Re: Week 2
by Muritala Akinyemi Adewale - Thursday, 4 July 2024, 4:20 AM

Hi Aye,
Your analysis of the time complexity for the algorithms, utilizing recurrence relations and the Master Theorem, is exemplary. The way you applied the Master Theorem to Algorithm D shows a deep comprehension of divide-and-conquer strategies. Your explanations are articulate and methodical, providing a clear and comprehensive understanding of your reasoning. Well done!
54 words

Permalink Show parent

Re: Week 2
by Tamaneenah Kazeem - Thursday, 4 July 2024, 9:22 AM

Excellent post Aye. All the answers are accurate according to my understanding. You also provided some fantastic examples. Keep it up.
Just to include, do you know that algorithm C combines solutions in quadratic time?
35 words

Permalink Show parent

Re: Week 2
by Siraajuddeen Adeitan Abdulfattah - Wednesday, 3 July 2024, 10:46 PM

For all cases, assuming the size of the problem is n

Algorithm A solves problems by dividing them into five sub-problems of half the size, recursively solving each sub-problem, and then combining the solutions in linear time

Algorithm A: divide and combine steps have a linear time complexity, the recurrence relation for the algorithm can be written as T(n) = 5T(n/2) + O(n) which belongs to the case of master theorem with a = 5, b = 2, f(n) = O(n). Application of this master theorem shows that the running time is O(n^(log25)).

Algorithm B solves problems of size n by recursively solving two sub-problems of size n-1 and then combining the solutions in constant time.

Algorithm B: problem is divided into two sub-problem of size n-1, combine step takes constant time and the recurrence relation for the algorithm can be written as T(n) = 2T(n-1) + O(1). The algorithm performs n recursive calls, since the size of the sub-problem is n-1. Running time is O(n).

Algorithm C solves problems of size n by dividing them into nine sub-problems of size n/3, recursively solving each sub-problem, and then combining the solutions in O(n2) time.

Algorithm C: divides the problem into sub-problems of size n/3. Combine step has a time complexity of O(n^2). The recurrence relation of the algorithm can be written as
T(n) = 9T(n/3) + O(n^2). According to the master theorem, the running time is O(n^2).
What are the running times of each of these algorithms (in big- O notation), and which would you choose?

Algorithm B has the best time complexity of O(n), based on the running times. Its time complexity is linear, which means it scale well with input size. While Algorithm A's running time is O(n^(log25)), this can be worse than linear depending on the value of n. Algorithm C has a running time of O(n^2), it is a higher time complexity in contrast with Algorithm A and B.

So, my preferred Algorithm of choice here will be Algorithm B because it has the best time complexity compared to the other two Algorithms. The linear time complexity of Algorithm B offers efficiency even for a larger size problem, considering the example below:

A problem we need to solve with Algorithm B has a size of n = 12. According to this algorithm, the problem is divided into two sub-problem of size n-1, which will result in 12-1 = 11 and 12-1 = 11. Each sub-problem is solved recursively and combine the solution in constant time. Recursion continues until the base case is reached (where the subproblem size becomes 1). Afterwards, the solution of the sub-problem is combined to arrive at the final solution for the actual problem. This ensures a linear time complexity and making the Algorithm efficient for solving problems of different sizes.
468 words

Permalink Show parent

Re: Week 2
by Christopher Mccammon - Thursday, 4 July 2024, 12:03 AM

HI Siraajuddeen
You've made a commendable effort in analyzing the time complexity of the given algorithms using recurrence relations and the Master Theorem. Your use of the Master Theorem for Algorithm A is accurate, showing a good understanding of how to apply it to divide-and-conquer problems. Your explanations are clear and well-organized, making it easy to follow your thought process. Good job!
62 words

Permalink Show parent

Re: Week 2
by Muritala Akinyemi Adewale - Thursday, 4 July 2024, 4:19 AM

hi Siraajuddeen,
You've done a fantastic job dissecting the time complexity of these algorithms through recurrence relations and the Master Theorem. Your application of the Master Theorem to Algorithm C is spot-on, reflecting a strong understanding of its nuances. Your detailed and logical explanations make your analysis very accessible and insightful. Keep up the great work!"

56 words

Permalink Show parent

Re: Week 2
by Liliana Blanco - Thursday, 4 July 2024, 7:44 AM

Your analysis of the algorithms is excellent and well-explained. I appreciate how you detailed the time complexities using the master theorem and accurately identified the running times for each algorithm. Your choice of Algorithm B, with its linear time complexity O(n), makes sense given its efficiency and scalability. However, it's worth considering that the constant factors and overheads might also impact performance in practical scenarios. I still think your preference for Algorithm B is logical and well-supported. Great job!
79 words

Permalink Show parent

Re: Week 2
by Jobert Cadiz - Thursday, 4 July 2024, 7:50 AM

Hi Siraajuddeen,
You accurately describe each algorithm's approach to problem-solving. Your use of the Master Theorem to analyze Algorithms A and C demonstrates a solid understanding of how to apply theoretical concepts to real-world algorithms.
Greta work.
37 words

Permalink Show parent

Re: Week 2
by Chong-Wei Chiu - Thursday, 4 July 2024, 9:16 AM

Hi, Siraajuddeen Adeitan Abdulfattah. Thank you for sharing your opinion on this topic. You clearly explain the process and illustrate each algorithm. However, I believe the complexity of algorithm C should be Θ(n^2 * log(n)). Additionally, you should add the reference you used in this post and ensure it follows APA citation rules.
53 words

Permalink Show parent

Re: Week 2
by Christopher Mccammon - Wednesday, 3 July 2024, 11:32 PM

Algorithm A:

Divides problems into 5 sub problems of half the size.

Recursively solves each su -problem.

Combines the solutions in linear time O(n) (Cormen et al,2009).

The recurrence relation is T(n)=5T(n2)+O(n)T(n) = 5T\left(\frac{n}{2}\right) + O(n)T(n)=5T(2n)+O(n)

Using the Master Theorem for divide and conquer recurrences the form T(n)=aT(nb)+f(n)T(n) = aT\left(\frac{n}{b}\right) + f(n)T(n)=aT(bn)+f(n):

a=5a = 5a=5

b=2b = 2b=2

f(n)=O(n)f(n) = O(n)f(n)=O(n) (Skiena,2008).

According to the Master Theorem:

Compare f(n)f(n)f(n) with nlogba=nlog25≈n2.32n^{\log_b a} = n^{\log_2 5} \approx n^{2.32}nlogba=nlog25≈n2.32(Cormen et al,2009).

Since f(n)=O(n)f(n) = O(n)f(n)=O(n) and nlogba=n2.32n^{\log_b a} = n^{2.32}nlogba=n2.32:

f(n)f(n)f(n) grows slower than nlogban^{\log_b a}nlogba(Skiena,2008).

Thus by case 1 of the Master Theorem: T(n)=O(nlogba)=O(n2.32)T(n) = O(n^{\log_b a}) = O(n^{2.32})T(n)=O(nlogba)=O(n2.32) (Cormen et al,2009)

Algorithm B:

Solves problems of size nnn by recursively solving two sub-problems of size n−1n-1n−1.

Combines the solutions in constant time, O(1).

The recurrence relation is: T(n)=2T(n−1)+O(1)T(n) = 2T(n-1) + O(1)T(n)=2T(n−1)+O(1)

This relation is characteristic of a problem that decreases linearly in size:

Each step involves solving two problems of almost the same size.

Unrolling the recurrence, we get: T(n)=2T(n−1)+O(1)T(n) = 2T(n-1) + O(1)T(n)=2T(n−1)+O(1) T(n−1)=2T(n−2)+O(1)T(n-1) = 2T(n-2) + O(1)T(n−1)=2T(n−2)+O(1) T(n)=2(2T(n−2)+O(1))+O(1)T(n) = 2(2T(n-2) + O(1)) + O(1)T(n)=2(2T(n−2)+O(1))+O(1) =22T(n−2)+2O(1)+O(1)= 2^2T(n-2) + 2O(1) + O(1)=22T(n−2)+2O(1)+O(1) …\dots… =2kT(n−k)+O(2k)= 2^kT(n-k) + O(2^k)=2kT(n−k)+O(2k) (Cormen et al,2009)

Eventually, for k=nk = nk=n: T(n)=2nT(0)+O(2n)T(n) = 2^nT(0) + O(2^n)T(n)=2nT(0)+O(2n)

Assuming T(0)=1T(0) = 1T(0)=1: T(n)=O(2n)T(n) = O(2^n)T(n)=O(2n)

Algorithm C:

Divides problems into 9 sub-problems of size n/3n/3n/3.

Recursively solves each sub-problem.

Combines the solutions in O(n2)O(n^2)O(n2) time.

The recurrence relation is: T(n)=9T(n3)+O(n2)T(n) = 9T\left(\frac{n}{3}\right) + O(n^2)T(n)=9T(3n)+O(n2)

Using the Master Theorem:

a=9a = 9a=9

b=3b = 3b=3

f(n)=O(n2)f(n) = O(n^2)f(n)=O(n2) (Cormen et al,2009)

Compare f(n)f(n)f(n) with nlogba=nlog39=n2n^{\log_b a} = n^{\log_3 9} = n^2nlogba=nlog39=n2.

Since f(n)=O(n2)f(n) = O(n^2)f(n)=O(n2) and nlogba=n2n^{\log_b a} = n^2nlogba=n2:

f(n)f(n)f(n) is of the same order as nlogban^{\log_b a}nlogba. (Cormen et al,2009)

Thus, by case 2 of the Master Theorem: T(n)=O(n2logn)T(n) = O(n^2 \log n)T(n)=O(n2logn)

Choosing the Best Algorithm

Comparing the time complexities:

Algorithm A: O(n2.32)O(n^{2.32})O(n2.32)

Algorithm B: O(2n)O(2^n)O(2n)

Algorithm C: O(n2logn)O(n^2 \log n)O(n2logn)

Algorithm C is the most efficient for large input sizes, as O(n2logn)O(n^2 \log n)O(n2logn) grows slower than O(n2.32)O(n^{2.32})O(n2.32) and much slower than O(2n)O(2^n)O(2n).

Explanation and Example

Algorithm C is preferable because it balances the problem size reduction and combination step effectively. For example, if you are sorting a large dataset or performing matrix multiplication, breaking down the problem into smaller chunks efficiently while managing the overhead of combining results is crucial.

Consider sorting a list:

Algorithm A's complexity of O(n2.32)O(n^{2.32})O(n2.32) would be less efficient for large data as the number of recursive steps increases significantly.

Algorithm B's exponential growth O(2n)O(2^n)O(2n) quickly becomes impractical for large nnn.

Algorithm C's O(n2logn)O(n^2 \log n)O(n2logn) offers a more scalable solution, making it suitable for applications like fast multiplication of large matrices or optimized search algorithms.

In real world scenarios such as image processing or simulations algorithms with manageable growth rates like Algorithm C are essential for maintaining performance and efficiency.

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Skiena, S. S. (2008). *The Algorithm Design Manual* (2nd ed.). Springer.

516 words

Permalink Show parent

Re: Week 2
by Wingsoflord Ngilazi - Thursday, 4 July 2024, 12:39 AM

Your discussion is concise and your arguments are well justified. Keep up the great work.
15 words

Permalink Show parent

Re: Week 2
by Prince Ansah Owusu - Thursday, 4 July 2024, 4:05 AM

Your submission thoroughly applies the Master Theorem to determine the running times for each algorithm and provides a clear justification for selecting Algorithm C. It demonstrates a strong understanding of algorithmic analysis. To enhance clarity, consider breaking down complex equations and recurrence relations into more digestible steps. Overall, well done!
50 words

Permalink Show parent

Re: Week 2
by Muritala Akinyemi Adewale - Thursday, 4 July 2024, 4:18 AM

Hi Christopher,
Your approach to analyzing the algorithms' time complexity using recurrence relations and the Master Theorem is impressive. You've effectively applied the Master Theorem to Algorithm B, demonstrating a solid grasp of its principles. Your step-by-step explanations are thorough and precise, which enhances the clarity of your analysis. Excellent work!
51 words

Permalink Show parent

Re: Week 2
by Liliana Blanco - Thursday, 4 July 2024, 7:43 AM

I really liked your breakdown of the algorithms. The way you explained each algorithm's process and compared their time complexities was impressive. I think your choice of Algorithm C makes a lot of sense, especially with its more manageable growth rate for large inputs. While Algorithms A and B have their uses, I see your point about Algorithm C being more efficient for large datasets and complex operations like matrix multiplication. Of course, there might still be some cases where Algorithms A or B could be better for specific use cases or smaller input sizes. Overall, your conclusion is solid and well-supported. Great work!
104 words

Permalink Show parent

Re: Week 2
by Liliana Blanco - Thursday, 4 July 2024, 4:26 AM

Algorithm A Highlights Critical Steps:
1. Divide problems into 5 sub-problems of size n/2.
2. Combine solutions in linear time (O(n)).

Using Schaffer's definition (2011), T(n) = aT(n/b) + ink, T(1) = c (p. 508), with a = 5, b = 2, and f(n) = O(n), we compare nlogb(a) with f(n):
nlog2(5) vs. O(n)

Since nlog2(5) is greater than O(n), this falls under Case 1 of the Master Theorem, and thus the running time of Algorithm A is O(n log2(5)).

Algorithm B Highlights Critical Steps:
1. Recursively solves two sub-problems of size n-1.
2. Combines solutions in constant time.

While this case doesn't neatly fit the Master Theorem, due to the linear decrease in sub-problem size (n-1), we observe that each level of recursion has two calls, and there will be n levels of recursion. This results in 2n operations, making the time complexity O(2n).

Algorithm C Highlights Critical Steps:
1. Divide problems into 9 sub-problems of size n/3.
2. Combines solutions in O(n^2) time.

Using the Master Theorem with a = 9, b = 3, and f(n) = O(n^2), we compare nlogb(a) with f(n):
nlog3(9) vs. O(n^2)

Since nlog3(9) is O(n^2), which is equal to f(n), this falls under Case 2 of the Master Theorem, and thus the running time of Algorithm C is also O(n^2).

Based on the analysis, Algorithm A is deemed the most efficient as the size grows, considering its lower growth rate compared to the others.

Reference:
Schaffer, C.A. (2011). A Practical Introduction to Data Structures and Algorithms Analysis (3.1 ed.). Blacksburg, VA: Virginia Tech University, Department of Computer Science. Available at http://people.cs.vt.edu/~shaffer/Book/C++3e20100119.pdf

268 words

Permalink Show parent

Re: Week 2
by Jobert Cadiz - Thursday, 4 July 2024, 6:57 AM

Hi Liliana,
Your assessment of the algorithms and their running times is generally accurate and well-explained. The APA citation style and referencing Schaffer's work appropriately supports your analysis. This approach ensures clarity and reliability in academic discussions of algorithmic efficiency. Great work.

42 words

Permalink Show parent

Re: Week 2
by Tamaneenah Kazeem - Thursday, 4 July 2024, 9:25 AM

Excellent choice Liliana. I agree with the point you made regarding algorithm A. It is also the choice I picked.
A fantastic post overall.

24 words

Permalink Show parent

Re: Week 2
by Jobert Cadiz - Thursday, 4 July 2024, 5:02 AM

0 words

Permalink Show parent

Re: Week 2
by Chong-Wei Chiu - Thursday, 4 July 2024, 6:13 AM

Reference:

Introduction to Recurrence Relations - algo_ch4_recurrences.pdf

Schaffer, C.A. (2011). A Practical Introduction to Data Structures and Algorithms Analysis (3.1 ed.). Blacksburg, VA: Virginia Tech University, Department of Computer Science. Available at http://people.cs.vt.edu/~shaffer/Book/C++3e20100119.pdf

Dasgupta, S., Papadimitriou, C.H., & Vazirani, U.V. (2006). Algorithms. Berkeley, CA: University of California Berkeley, Computer Science Division. Available at http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf

54 words

Permalink Show parent

Re: Week 2
by Chong-Wei Chiu - Thursday, 4 July 2024, 9:24 AM

P.S. I finally choose algorithm C.
6 words

Permalink Show parent

Re: Week 2
by Natalie Tyson - Thursday, 4 July 2024, 8:32 AM

Running Times for the three algorithms in big-O notation:

Algorithm A:

From our reading we learned that this problem solving technique of taking problems and dividing them into different subproblems and recursively solving these subproblems is the divide-and-conquer algorithm. There will be 5 subproblems to solve for and we would put this as

5(log base 2 of n) the number to solve the subproblems during a set time interval is proportional to the time it would take to solve each problem multiplied by the number of subproblems.

The notation would look closer to O(n(log5))

We have the master theorem that we can use on this which would look like T(n)=5xT(n/2) +m T(1) = 1

Running the equation by setting A and B equal to some values we can see that the exponential time for this algorithm could make it slower than the other two algorithms.

T(n) = Θ(n log2 5) is the master theorem or the running time would be O(n(log5)).

Algorithm B:

This algorithm takes the problems that are of size n and solves them by recursively solving two subproblems that are size n-1 and following that by combining the solutions in constant time. If the input were to grow exponentially then the subproblem would grow exponentially as well. This unchecked growth for the subproblem numbers would have a complexity of O(2n).

For the master theorem we will have T(n) = 2T(n-1)+O(1). We have a certain number of subproblems that would double n times in O(1) of time. T(n) = Θ(2n) would be the closest to the running time and would translate to O(2n).

Algorithm C:

We solve the problem of size n by dividing them into 9 subproblems of size n/3, we recursively solve the subproblems by combining their solutions in O(n2). We make a recursion tree solving this set of problems. With n/3 we have a smaller set of subproblems, but they will be larger than algorithm B.

Time complexity: O(n2xlogn)

Conclusion

Algorithm As algorithm took less time to run than its peers in algorithm B and C. When comparing the time complexities of the algorithms, it was algorithm A that was the least time-consuming vs both B and C.

Citations

- Dasgupta, S., Papadimitriou, C.H., & Vazirani, U.V. (2006). Algorithms. Berkeley, CA: University of California Berkeley, Computer Science Division. Available at http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf
- GeeksforGeeks. (2023b, February 15). Advanced master theorem for Divide and conquer recurrences. https://www.geeksforgeeks.org/advanced-master-theorem-for-divide-and-conquer-recurrences/#

399 words

Permalink Show parent