CSEN 703 - Analysis and Design of Algorithms

Lecture 4 - Divide and Conquer II

Dr. Nourhan Ehab
nourhan.ehab@guc.edu.eg

Department of Computer Science and Engineering
Faculty of Media Engineering and Technology

In the Previous Lecture

- We looked into designing problems using the D&C strategy.
- We learned how to write recurrences to represent the running time of D&C algorithms.
- We learned about solving recurrences using the recursion tree method.

Outline

1. The Master Method
2. Quick Sort
3. Recap

Solving Recurrences

GUC German University in Cairo

The Master Method

The Master Theorem

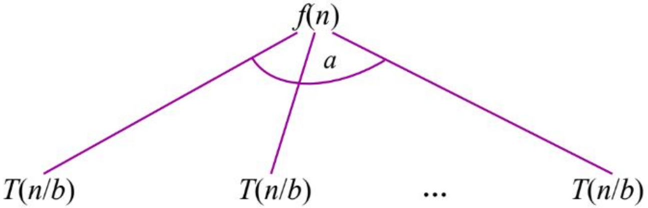
A cookbook for solving recurrences of the form

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

where $a \geq 1$, $b > 1$, and $f(n)$ is an asymptotically positive function.

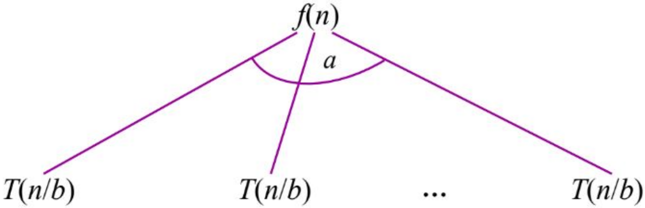
Intuition

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



Intuition

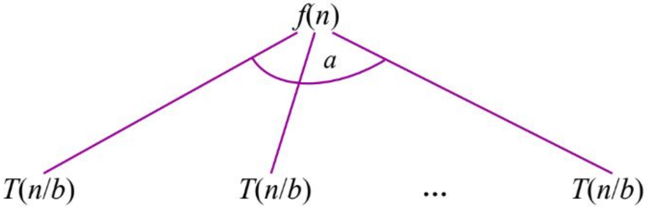
$$T(n) = aT(\frac{n}{b}) + f(n)$$



- To open up the next round of recurrences, we substitute $\frac{n}{b}$ in our recurrence.

Intuition

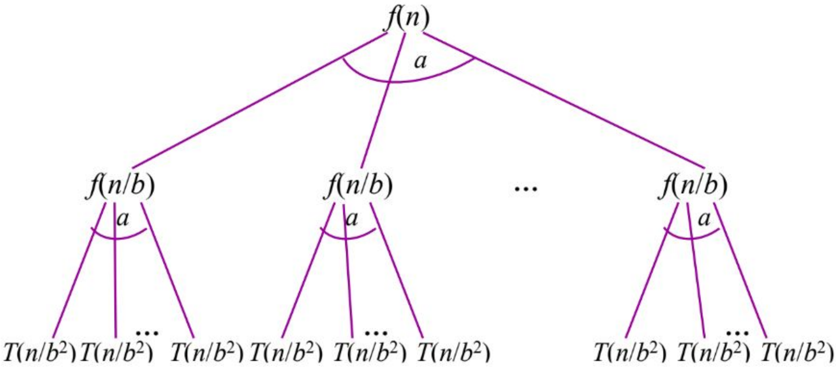
$$T(n) = aT(\frac{n}{b}) + f(n)$$



- To open up the next round of recurrences, we substitute $\frac{n}{b}$ in our recurrence.
- We get $T(\frac{n}{b}) = aT(\frac{\frac{n}{b}}{b}) + f(\frac{n}{b}) = aT(\frac{n}{b^2}) + f(\frac{n}{b})$

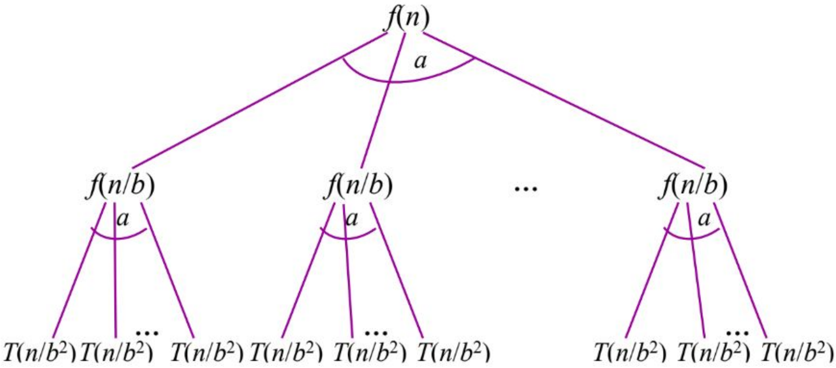
Intuition

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



Intuition

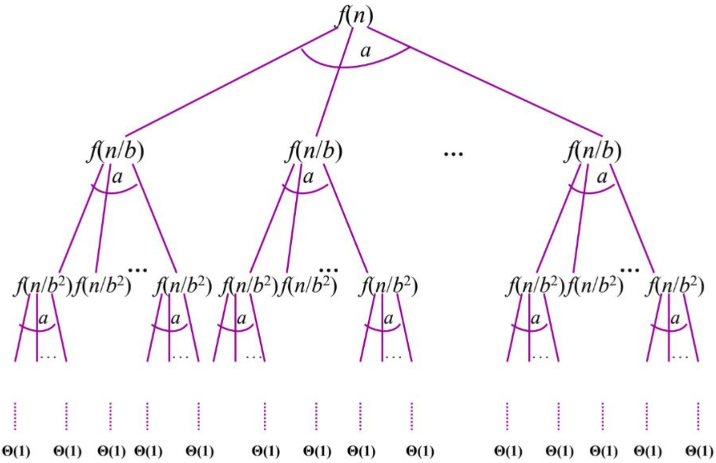
$$T(n) = aT(\frac{n}{b}) + f(n)$$



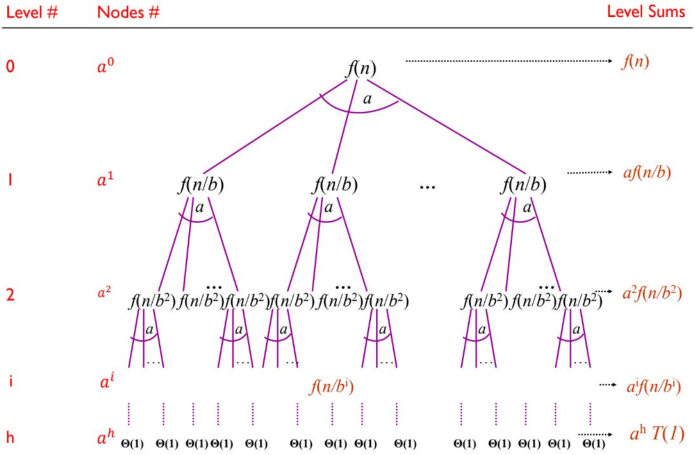
We can continue expanding the recursion tree until the sub-problems bottom out, that is, they reduce to a size of 1.

Intuition

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



Intuition



$$T(n) = \Sigma_{i=0}^{h-1} a^i f(\frac{n}{b^i}) + a^h \text{ with } h = log_b(n)$$

Intuition

$$T(n) = \Sigma_{i=0}^{log_b(n)-1} a^i f(\frac{n}{b^i}) + n^{log_b(a)}$$

Intuition

$$T(n) = \Sigma_{i=0}^{log_b(n)-1} a^i f(\frac{n}{b^i}) + n^{log_b(a)}$$

We have three cases:

1. The summation is an increasing geometric series.
   - $T(n) = 3T(\frac{n}{2}) + n$

Intuition

$$T(n) = \Sigma_{i=0}^{log_b(n)-1} a^i f(\frac{n}{b^i}) + n^{log_b(a)}$$

We have three cases:

1. The summation is an increasing geometric series.
   - $T(n) = 3T(\frac{n}{2}) + n \Rightarrow$ Cost is dominated by the leaves.

Intuition

GUC German University in Cairo

$$T(n) = \Sigma_{i=0}^{log_b(n)-1} a^i f(\frac{n}{b^i}) + n^{log_b(a)}$$

We have three cases:

1. The summation is an increasing geometric series.
   - $T(n) = 3T(\frac{n}{2}) + n \Rightarrow$ Cost is dominated by the leaves.
2. The summation is a constant series.
   - $T(n) = 4T(\frac{n}{2}) + n^2$

Intuition

$$T(n) = \Sigma_{i=0}^{log_b(n)-1} a^i f(\frac{n}{b^i}) + n^{log_b(a)}$$

We have three cases:

1. The summation is an increasing geometric series.
   - $T(n) = 3T(\frac{n}{2}) + n \Rightarrow$ Cost is dominated by the leaves.
2. The summation is a constant series.
   - $T(n) = 4T(\frac{n}{2}) + n^2 \Rightarrow$ Cost is distributed on the tree.

Intuition

$$T(n) = \Sigma_{i=0}^{log_b(n)-1} a^i f(\frac{n}{b^i}) + n^{log_b(a)}$$

We have three cases:

1. The summation is an increasing geometric series.
   - $T(n) = 3T(\frac{n}{2}) + n \Rightarrow$ Cost is dominated by the leaves.
2. The summation is a constant series.
   - $T(n) = 4T(\frac{n}{2}) + n^2 \Rightarrow$ Cost is distributed on the tree.
3. The summation is a decreasing geometric series.
   - $T(n) = 2T(\frac{n}{2}) + n^2$

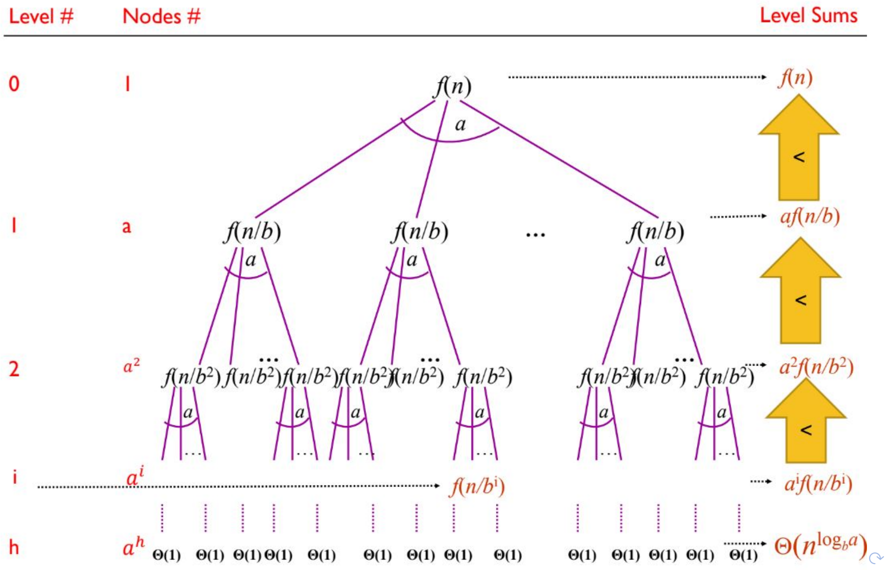
Intuition

$$T(n) = \Sigma_{i=0}^{log_b(n)-1} a^i f(\frac{n}{b^i}) + n^{log_b(a)}$$

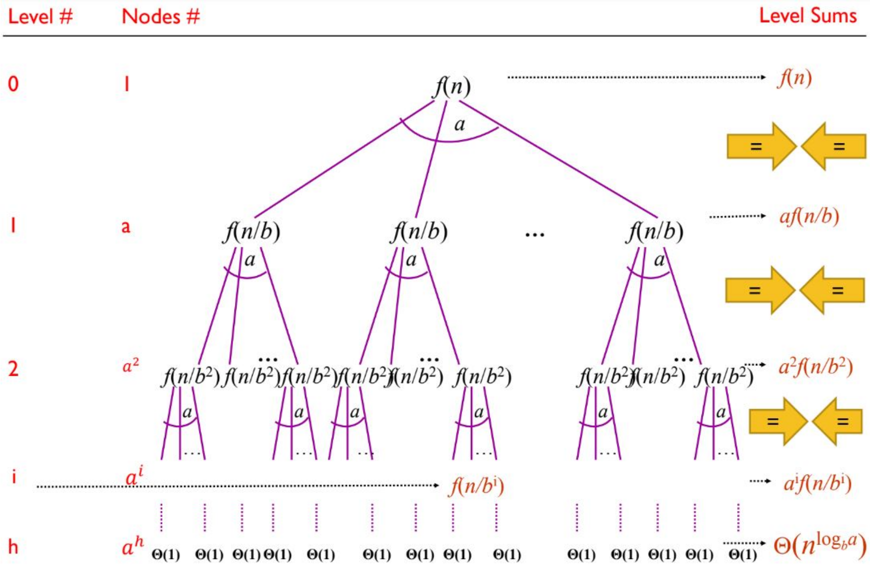
We have three cases:

1. The summation is an increasing geometric series.
   - $T(n) = 3T(\frac{n}{2}) + n \Rightarrow$ Cost is dominated by the leaves.
2. The summation is a constant series.
   - $T(n) = 4T(\frac{n}{2}) + n^2 \Rightarrow$ Cost is distributed on the tree.
3. The summation is a decreasing geometric series.
   - $T(n) = 2T(\frac{n}{2}) + n^2 \Rightarrow$ Cost is dominated by the root.

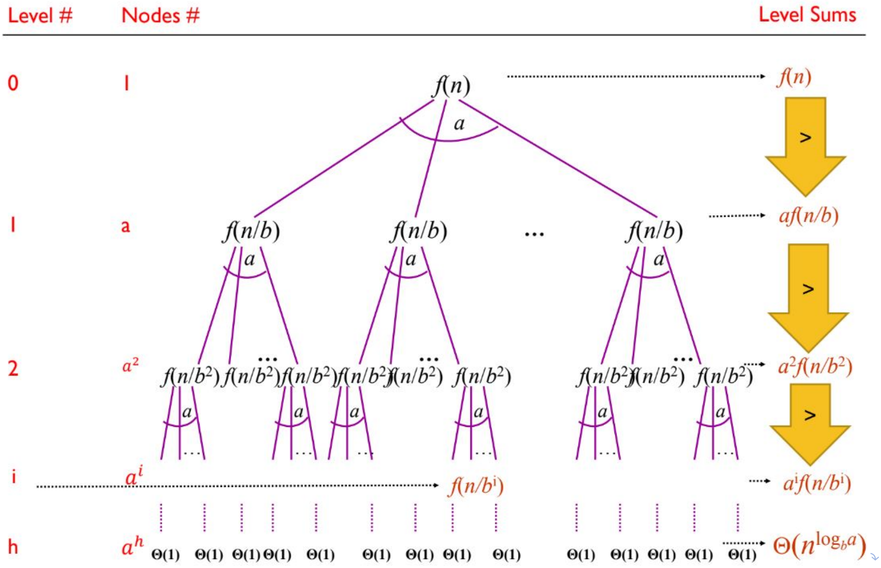
Case 1 - Cost Dominated by Leaves



Case 2 - Cost Distributed along the Tree



Case 3 - Cost Dominated by Root



The Master Method

The Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, $f(n)$ be an asymptotically positive function, and $T(n) = aT(\frac{n}{b}) + f(n)$.
$T(n)$ can be asymptotically bounded in 3 cases:

The Master Method

The Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, $f(n)$ be an asymptotically positive function, and $T(n) = aT(\frac{n}{b}) + f(n)$.
$T(n)$ can be asymptotically bounded in 3 cases:

1. If $f(n) = O(n^{log_b(a)-\varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^{log_b(a)})$.

The Master Method

The Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, $f(n)$ be an asymptotically positive function, and $T(n) = aT(\frac{n}{b}) + f(n)$.
$T(n)$ can be asymptotically bounded in 3 cases:

1. If $f(n) = O(n^{log_b(a)-\varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^{log_b(a)})$.
2. If $f(n) = \Theta(n^{log_b(a)})$, then $T(n) = \Theta(n^{log_b(a)} log(n))$.

The Master Method

The Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, $f(n)$ be an asymptotically positive function, and $T(n) = aT(\frac{n}{b}) + f(n)$.
$T(n)$ can be asymptotically bounded in 3 cases:

1. If $f(n) = O(n^{log_b(a)-\varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^{log_b(a)})$.
2. If $f(n) = \Theta(n^{log_b(a)})$, then $T(n) = \Theta(n^{log_b(a)}log(n))$.
3. If $f(n) = \Omega(n^{log_b(a)+\varepsilon})$ for some $\varepsilon > 0$ and $af(\frac{n}{b}) \leq cf(n)$ for $0 < c < 1$, then $T(n) = \Theta(f(n))$.

The Master Theorem - Examples

Example

Use the master theorem to obtain a bound on $T(n)$.

1. $T(n) = T(\frac{n}{2}) + \Theta(1)$.

The Master Theorem - Examples

GUC German University in Cairo

Example

Use the master theorem to obtain a bound on $T(n)$.

1. $T(n) = T(\frac{n}{2}) + \Theta(1)$.
   $a = 1, b = 2, n^{log_b(a)} = n^{log_2(1)} = 1$
   Cost of root $= c$, Cost of leaves $= c$.
   This is Case 2 of the master theorem since $c = \Theta(1)$.
   Hence, $T(n) = \Theta(log(n))$.
2. $T(n) = 2T(\frac{n}{2}) + \Theta(n)$.

The Master Theorem - Examples

Example

Use the master theorem to obtain a bound on $T(n)$.

1. $T(n) = T(\frac{n}{2}) + \Theta(1)$.
   $a = 1, b = 2, n^{log_b(a)} = n^{log_2(1)} = 1$
   Cost of root $= c$, Cost of leaves $= c$.
   This is Case 2 of the master theorem since $c = \Theta(1)$.
   Hence, $T(n) = \Theta(log(n))$.
2. $T(n) = 2T(\frac{n}{2}) + \Theta(n)$.
   $a = 2, b = 2, n^{log_b(a) = n}$
   Cost of root $= n$, Cost of leaves $= n$
   This is Case 2 of the master theorem since $n = \Theta(n)$.
   Hence, $T(n) = \Theta(nlog(n))$.

The Master Theorem - Examples

Example

Use the master theorem to obtain a bound on $T(n)$.

3. $T(n) = 16T(\frac{n}{4}) + \Theta(n)$.

The Master Theorem - Examples

Example

Use the master theorem to obtain a bound on $T(n)$.

3. $T(n) = 16T(\frac{n}{4}) + \Theta(n)$.
   $a = 16, b = 4, n^{log_b(a)} = n^{log_4(16)} = n^2$
   Cost of root $= cn$, Cost of leaves $= cn^2$.
   Looks like Case 1 of the master theorem.
   $n \leq \frac{n^2}{n^{\varepsilon}}$ for $\varepsilon = 1$. Case 1 proved. Hence, $T(n) = \Theta(n^2)$.
4. $T(n) = 2T(\frac{n}{2}) + \Theta(n^4)$.

The Master Theorem - Examples

Example

Use the master theorem to obtain a bound on $T(n)$.

3. $T(n) = 16T(\frac{n}{4}) + \Theta(n)$.
   $a = 16, b = 4, n^{log_b(a)} = n^{log_4(16)} = n^2$
   Cost of root $= cn$, Cost of leaves $= cn^2$.
   Looks like Case 1 of the master theorem.
   $n \leq \frac{n^2}{n^\varepsilon}$ for $\varepsilon = 1$. Case 1 proved. Hence, $T(n) = \Theta(n^2)$.
4. $T(n) = 2T(\frac{n}{2}) + \Theta(n^4)$.
   $a = 2, b = 2, n^{log_b(a)} = n$
   Cost of root $= cn^4$, Cost of leaves $= cn$
   Looks like Case 3 of the master theorem.
   $n^4 \geq n.n^\varepsilon$ for $\varepsilon = 1$.
   Regularity condition: $2(\frac{n}{2})^4 \leq cn^4$, $\frac{n^4}{2^3} \leq cn^4$ for $c = \frac{1}{8}$.
   Case 3 proved. Hence, $T(n) = \Theta(n^4)$.

The Master Theorem - Examples

GUC German University in Cairo

Example

Use the master theorem to obtain a bound on $T(n)$.

5. $T(n) = 2T(\frac{n}{2}) + \Theta(\frac{n}{log(n)})$.

The Master Theorem - Examples

Example

Use the master theorem to obtain a bound on $T(n)$.

5. $T(n) = 2T(\frac{n}{2}) + \Theta(\frac{n}{log(n)})$.
   $a = 2, b = 2, n^{log_b(a)} = n$
   Cost of root $= c\frac{n}{log(n)}$, Cost of leaves $= cn$
   Looks like Case 1 of the master theorem.
   $\frac{n}{log(n)} \leq \frac{n}{n^{\varepsilon}}$, $log(n) \geq n^{\varepsilon} \Rightarrow$ does not hold!
   This is a gap between cases 1 and 2 and can not be solved using the master theorem.

The Master Theorem - Examples

GUC German University in Cairo

Example

Use the master theorem to obtain a bound on $T(n)$.

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

The Master Theorem - Examples

Example

Use the master theorem to obtain a bound on $T(n)$.

6. $T(n) = 2T(\frac{n}{2}) + \Theta(nlog(n))$.
   $a = 2, b = 2, n^{log_b(a)} = n^{log_2(2)} = n$
   Cost of root $= cnlog(n)$, Cost of leaves $= cn$.
   Looks like Case 3 of the master theorem.
   $nlog(n) \geq n.n^{\varepsilon}$, $log(n) \geq n^{\varepsilon} \Rightarrow$ does not hold!
   This is a gap between cases 2 and 3 and can not be solved using the master theorem.
7. $T(n) = T(n-1) + \Theta(n)$.

The Master Theorem - Examples

Example

Use the master theorem to obtain a bound on $T(n)$.

6. $T(n) = 2T(\frac{n}{2}) + \Theta(nlog(n))$.
   $a = 2, b = 2, n^{log_b(a)} = n^{log_2(2)} = n$
   Cost of root $= cnlog(n)$, Cost of leaves $= cn$.
   Looks like Case 3 of the master theorem.
   $nlog(n) \geq n.n^{\varepsilon}$, $log(n) \geq n^{\varepsilon} \Rightarrow$ does not hold!
   This is a gap between cases 2 and 3 and can not be solved using the master theorem.
7. $T(n) = T(n-1) + \Theta(n)$.
   Not in the form of the master theorem.

Outline

Quick Sort - D&C Approach

- Divide: Partition $A[p, \ldots, r]$ into two (possibly empty) arrays $A[p, \ldots, q-1]$ and $A[q+1, \ldots, r]$:
  - each element in $A[p, \ldots, q-1] \leq A[q]$
  - each element in $A[q+1, \ldots, r] \geq A[q]$
- Conquer: The subarrays by sorting them.
- Combine: No work needed! Array already sorted.

Quick sort - Pseudo Code

QuickSort(A, p, r)
if $p < r$ **then**
    $q =$ partition(A, p, r);
    QuickSort$(A, p, q - 1)$;
    QuickSort$(A, q + 1, r)$;
end

The complexity lies in the partition procedure which will rearrange the array such that all elements before the pivot has smaller values and all elements after the pivot has bigger values.

Ross was Probably Good at Quick Sort!

Quick sort - Partition

```
Partition(A, p, r)
pivot = A[r] ;
i = p - 1 ;
for j = p to r - 1 do
    if A[j] ≤ pivot then
        i++ ;
        Exchange A[i] and A[j] ;
    end
end
Exchange A[i + 1] and A[r] ;
return i+1;
```

Trace it on $[8, 1, 6, 4, 0, 3, 9, 5]$.

Quick Sort

Quick Sort - Analysis

- The runtime depends on the result of partition.

Quick Sort - Analysis

- The runtime depends on the result of partition.
  - If the resulting subarrays are balanced, then $T(n) = 2T(\frac{n}{2}) + \Theta(n)$.
    This is $\Theta(nlog(n))$ by Case 2 of the master theorem.
  - If the resulting subarrays are not balanced, then $T(n) = T(n-1) + \Theta(n)$. This is in $\Theta(n^2)$.

Quick Sort - Analysis

- The runtime depends on the result of partition.
  - If the resulting subarrays are balanced, then $T(n) = 2T(\frac{n}{2}) + \Theta(n)$.
    This is $\Theta(nlog(n))$ by Case 2 of the master theorem.
  - If the resulting subarrays are not balanced, then $T(n) = T(n-1) + \Theta(n)$. This is in $\Theta(n^2)$.
- The best case is as fast as merge sort. The worst case is as bad as insertion sort, but happens when the array is already sorted. The average case is as good as the best case.

Outline

1. The Master Method
2. Quick Sort
3. Recap

Points to Take Home

1. The Master Theorem.
2. Quick Sort.
3. Reading Material:
    - Introduction to Algorithms, Chapter 4: Section 4.5 and Chapter 7: Sections 7.1 and 7.2.

Next Lecture: Greedy Algorithms!

Due Credits

The presented material is based on:

1. Previous editions of the course at the GUC due to Dr. Wael Aboulsaadat, Dr. Haythem Ismail, Dr. Amr Desouky, and Dr. Carmen Gervet.
2. Stony Brook University's Analysis of Algorithms Course.
3. MIT's Introduction to Algorithms Course.
4. Stanford's Design and Analysis of Algorithms Course.