

CS-218 DATA STRUCTURES AND ALGORITHMS

LECTURE 3

BY UROOJ AINUDDIN



IN THE LAST LECTURE...

- We differentiated between the algorithm and the pseudocode.
- We learned how to do empirical analysis of algorithms.
- We discussed the disadvantages of empirical analysis.
- We learned how to do theoretical analysis of algorithms.
- We were introduced to primitive operations.
- We discussed best, average and worst cases of algorithms.
- We analyzed algorithms for the first time!



ANALYSIS OF ALGORITHMS

BOOK 1 CHAPTER 4

BOOK 2 CHAPTER 2



//Computes sum of an $n \times n$ matrix A , and sum of each of its rows

Algorithm matSum(A , n)

totalsum $\leftarrow 0$

for $i \leftarrow 0$ to $n - 1$ do

 rowsum[i] $\leftarrow 0$

 for $j \leftarrow 0$ to $n - 1$ do

 rowsum[i] \leftarrow rowsum[i] + $A[i, j]$

 totalsum \leftarrow totalsum + $A[i, j]$

rowsum[n] \leftarrow totalsum

return rowsum

ANALYZE THE ALGORITHM FOR ITS
BEST AND WORST CASES.



//Computes sum of an $n \times n$ matrix A, and sum of each of its rows

Algorithm matSum(A, n)

No of operations

1. totalsum $\leftarrow 0$

1

2. for i $\leftarrow 0$ to $n - 1$ do

$1 + n + 1 + n = 2n + 2$

3. rowsum[i] $\leftarrow 0$

$2n$

4. for j $\leftarrow 0$ to $n - 1$ do

$(1 + n + 1 + n)n = 2n^2 + 2n$

5. rowsum[i] \leftarrow rowsum[i] + A[i, j]

$5n^2$

6. totalsum \leftarrow totalsum + A[i, j]

$3n^2$

7. rowsum[n] \leftarrow totalsum

2

8. return rowsum

1

Total primitive operations

$10n^2 + 6n + 6$

$$T(n) = 10n^2 + 6n + 6$$

ANALYSIS



//Computes sum of an $n \times n$ matrix A , and sum of each of its rows

Algorithm matSum(A , n)

totalsum $\leftarrow 0$

for $i \leftarrow 0$ to $n - 1$ do

 rowsum[i] $\leftarrow 0$

 for $j \leftarrow 0$ to $n - 1$ do

 rowsum[i] \leftarrow rowsum[i] + $A[i, j]$

 totalsum \leftarrow totalsum + rowsum[i]

rowsum[n] \leftarrow totalsum

return rowsum

ANALYZE THE ALGORITHM FOR ITS
BEST AND WORST CASES.



//Computes sum of an $n \times n$ matrix A, and sum of each of its rows

Algorithm matSum(A, n)

No of operations

1. totalsum $\leftarrow 0$

1

2. for i $\leftarrow 0$ to $n - 1$ do

$1 + n + 1 + n = 2n + 2$

3. rowsum[i] $\leftarrow 0$

$2n$

4. for j $\leftarrow 0$ to $n - 1$ do

$(1 + n + 1 + n)n = 2n^2 + 2n$

5. rowsum[i] \leftarrow rowsum[i] + A[i, j]

$5n^2$

6. totalsum \leftarrow totalsum + rowsum[i]

$3n$

7. rowsum[n] \leftarrow totalsum

2

8. return rowsum

1

Total primitive operations

$7n^2 + 9n + 6$

$$T(n) = 7n^2 + 9n + 6$$

ANALYSIS



Algorithm matSum(A, n)

totalsum \leftarrow 0

for i \leftarrow 0 to n - 1 do

 rowsum[i] \leftarrow 0

 for j \leftarrow 0 to n - 1 do

 rowsum[i] \leftarrow rowsum[i] + A[i, j]

 totalsum \leftarrow totalsum + A[i, j]

rowsum[n] \leftarrow totalsum

return rowsum

$$T(n) = 10n^2 + 6n + 6$$

Algorithm matSum(A, n)

totalsum \leftarrow 0

for i \leftarrow 0 to n - 1 do

 rowsum[i] \leftarrow 0

 for j \leftarrow 0 to n - 1 do

 rowsum[i] \leftarrow rowsum[i] + A[i, j]

 totalsum \leftarrow totalsum + rowsum[i]

rowsum[n] \leftarrow totalsum

return rowsum

$$T(n) = 7n^2 + 9n + 6$$

WHICH IS THE BETTER ALGORITHM?



- ▶ By now, we can say that our algorithm runs in $T(n)$ time when presented with n inputs.
- ▶ However, the running time of an algorithm does not only depend on the input size. It depends on other factors too, like:
 - ▶ The hardware,
 - ▶ The programming language,
 - ▶ The programmer's skill.

FACTORS AFFECTING RUNNING TIME OF ALGORITHMS





- ▶ The answer is **NO**.
- ▶ When comparing two algorithms that perform the same function, a **relative estimate** is sufficient.
- ▶ If we know that algorithm A does more operations for the same input size than algorithm B, it can be easily deduced that **B is faster**, without knowing the actual running times of both A and B.

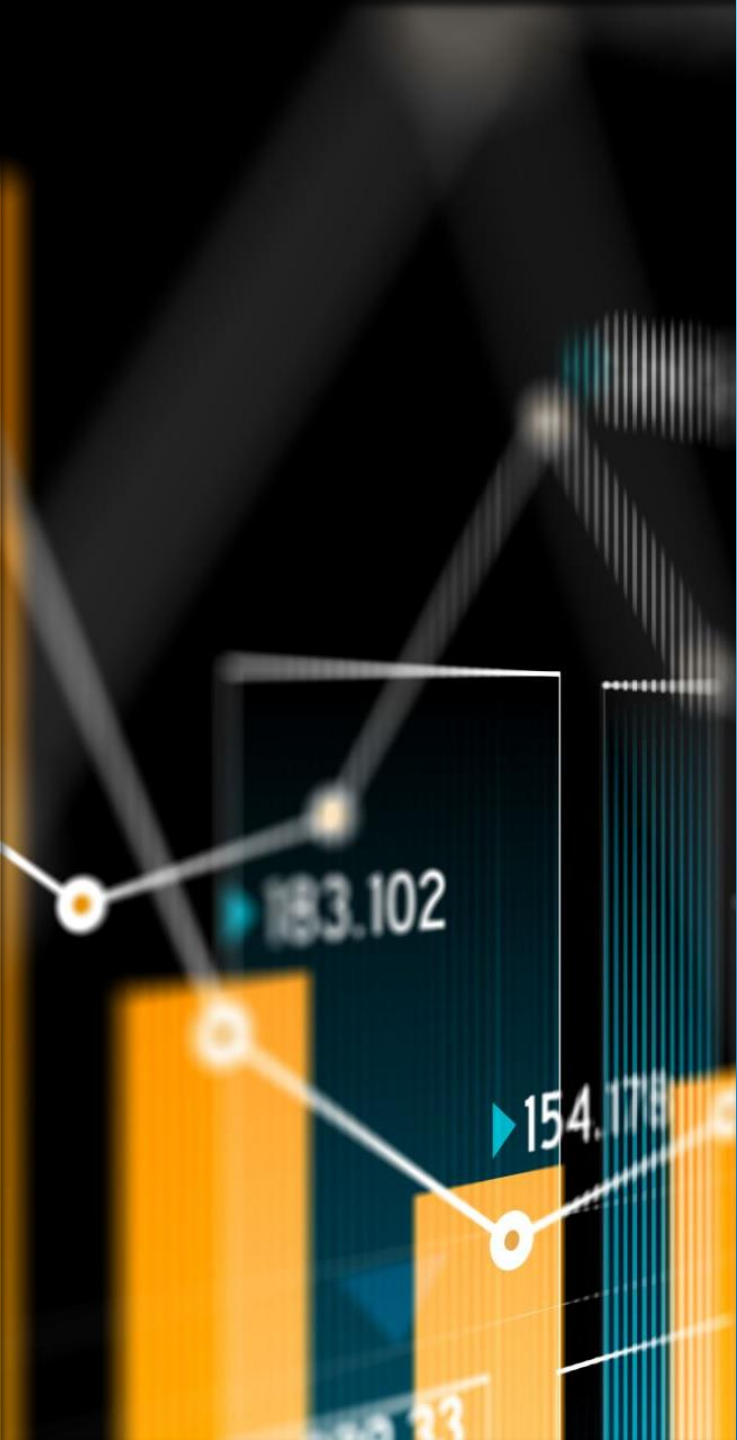
DO WE REALLY NEED
ABSOLUTE RUNNING TIME?



- ▶ The analysis of algorithms is mostly done for large input sizes.
- ▶ We are interested in how the performance of an algorithm degrades as the input size increases.
- ▶ In evolving systems like data mining, data warehousing, big data, and cloud computing, the data size is enormous.
- ▶ We need algorithms that compute fast and use less memory in doing so.

LARGE INPUT SIZE



- 
- ▶ Consider an algorithm with many lines of code.
 - ▶ Counting primitive operations would be a very exhausting task for large algorithms.
 - ▶ We realize that we need a way to estimate the relative speed of algorithms in an effortless manner.
 - ▶ Let us look at the **rate of growth** or the **order of growth** of an algorithm to decide if it is suitable for our system.
 - ▶ The rate of growth of an algorithm is **the rate at which the running time of the algorithm grows as the size of its input grows.**

RATE OF GROWTH



ASYMPTOTIC ANALYSIS

- ▶ Asymptotic analysis refers to computing the limit of a function $f(n)$ as n tends to infinity.
- ▶ Assume that $f(n) = 3n^2 + 2n$.
- ▶ As n becomes very large, the contribution of the second term in the value of $f(n)$ becomes insignificant.
- ▶ As $f(n)$ is mostly dependent on $3n^2$, We say that $f(n)$ has a quadratic rate of growth.
- ▶ We say that $f(n) \sim n^2$ or **$f(n)$ is asymptotically equivalent to n^2** or **$f(n)$ is asymptotic to n^2** .
- ▶ Asymptotic analysis determines the rate of growth of the algorithm.



```
Algorithm arrayMax(A, n)
//Input list A of  $n$  integers
//Output maximum element of A
currentMax  $\leftarrow$  A[0]
for  $i \leftarrow 1$  to  $n - 1$  do
    if  $A[i] > \text{currentMax}$  then
        currentMax  $\leftarrow$  A[i]
return currentMax
```

ANALYZE THE ALGORITHM FOR ITS
BEST AND WORST CASES.




```
Algorithm arrayMax(A, n)
//Input list A of n integers
//Output maximum element of A
1.  currentMax ← A[0]
2.  for i ← 1 to n - 1 do
3.      if A[i] > currentMax then
4.          currentMax ← A[i]
5.  return currentMax
```

$$T(n) = c(n - 1)$$

BEST CASE ANALYSIS

The best case happens when the **if condition is always false**, i.e. when either ~~A is sorted in descending order~~ or all elements of A are equal.

or the first element of A is the largest element.



Algorithm arrayMax(A, n)

//Input list A of n integers

//Output maximum element of A

1. currentMax \leftarrow A[0]
2. for $i \leftarrow 1$ to $n - 1$ do
3. if $A[i] >$ currentMax then
4. currentMax \leftarrow A[i]
5. return currentMax

$$T(n) = c(n - 1)$$

The worst case happens when the **if condition is always true**, i.e. when A is sorted in ascending order.

WORST CASE ANALYSIS




```
Algorithm arrayMax(A, n)
//Input list A of n integers
//Output maximum element of A
currentMax ← A[0]
for i ← 1 to n - 1 do
    if A[i] > currentMax then
        currentMax ← A[i]
return currentMax
```

$$T(n) = c(n - 1) = cn - c$$

For large values of n , the second term can be ignored, as it does not contribute as much as the first term.

$$T(n) = cn$$

$$T(n) \sim n$$

We say that the algorithm has a linear rate of growth.

The running time of the algorithm grows linearly as the input size is increased.

SUMMING UP THE ANALYSIS...



Algorithm a1 (n)

sum \leftarrow 0

for i \leftarrow 1 to n do

 for j \leftarrow 1 to n do

 sum \leftarrow sum + 1

return sum

ANALYZE THE ALGORITHM FOR ITS
BEST AND WORST CASES.



Algorithm a1 (n)

1. $\text{sum} \leftarrow 0$
2. for $i \leftarrow 1$ to n do
3. for $j \leftarrow 1$ to n do
4. $\text{sum} \leftarrow \text{sum} + 1$
5. return sum

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

$$T(n) \sim n^2$$

We say that the algorithm has a quadratic rate of growth.

The running time of the algorithm grows quadratically as the input size is increased.

ANALYSIS



Consider two algorithms A and B with running time functions $100n$ and $2n^2$ respectively.

$$2n^2 > 100n \text{ for } n > 50.$$

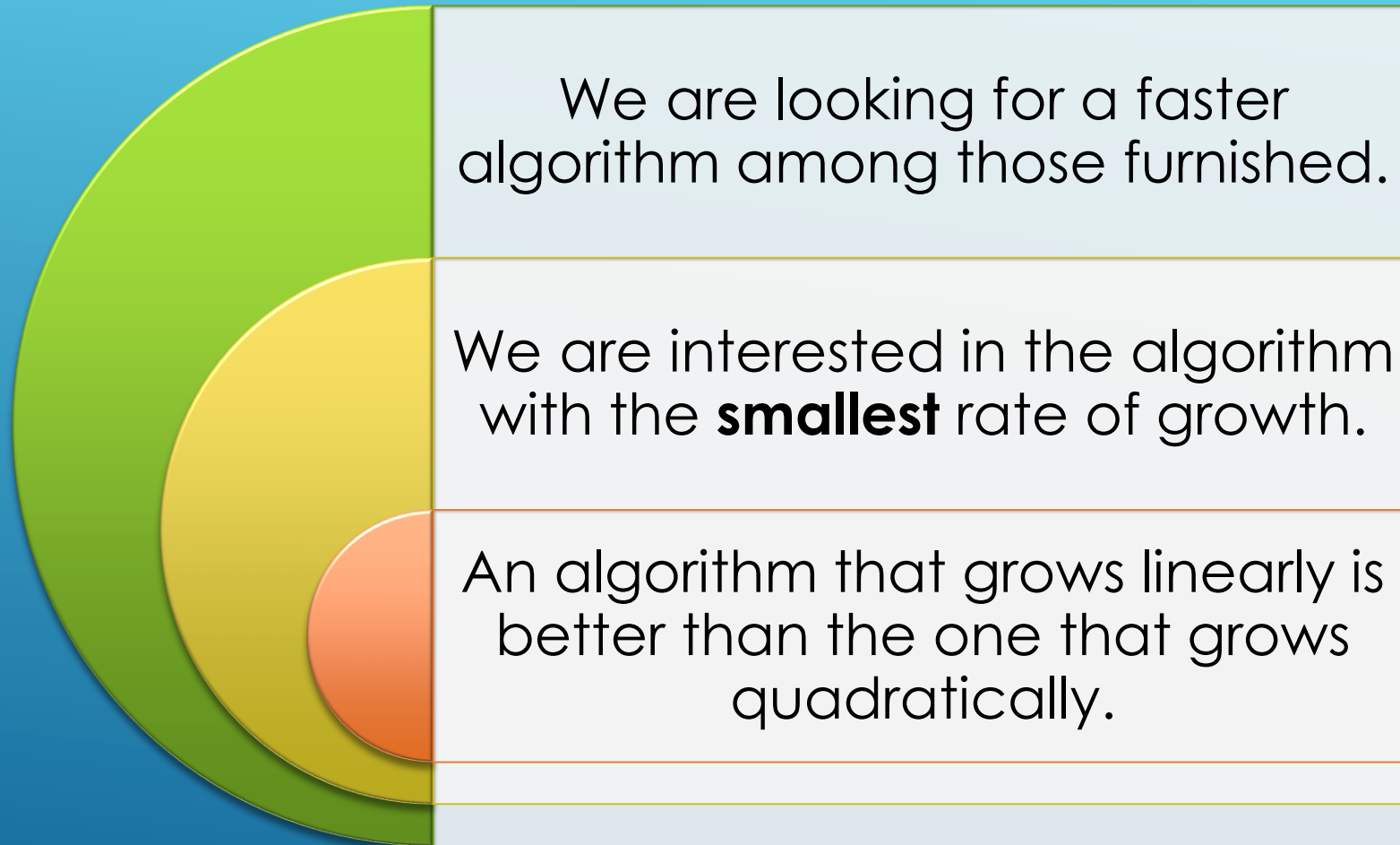
When $n > 50$, the running time of B is larger than that of A for the same input size.

For $n < 50$, B performs better than A.

For $n > 50$, A is obviously better than B.

ANOTHER
EXAMPLE





POINT TO NOTE



Input size	$\log n$	n	n^2	n^3
10	3.321928	10	100	1000
20	4.321928	20	400	8000
30	4.906891	30	900	27000
40	5.321928	40	1600	64000
50	5.643856	50	2500	125000
60	5.906891	60	3600	216000
70	6.129283	70	4900	343000
80	6.321928	80	6400	512000
90	6.491853	90	8100	729000
100	6.643856	100	10000	1000000

VALUES OF FUNCTIONS FOR LARGE
INPUT SIZES



Analyze the algorithm for its best and worst cases. Find **asymptotic relations** for $T(n)$ in both cases. You may do this on paper, or you may type your answer.

//Subtracts an $n \times n$ matrix B from $n \times n$ matrix A

//Stores result in A

Algorithm matSub(A, B, n)

for $i \leftarrow 0$ to $n - 1$ do

 for $j \leftarrow 0$ to $n - 1$ do

$A[i, j] \leftarrow A[i, j] - B[i, j]$

return

TASK

Save your snapshot, pdf or doc as <my_roll_no>_lec3. (If your roll no. is 500, the file should be named 500_lec3.)

Submit your response in the assignment titled "Lecture 3 tasks" in Google Classroom.



SO WHAT DID WE LEARN TODAY?

- We analyzed some more algorithms.
- We realized that there is no need to find absolute running time of an algorithm.
- We realized that we analyze algorithms typically for large input sizes.
- We were introduced to asymptotic analysis.
- We conducted asymptotic analysis of algorithms.



THINGS TO DO

Read

- the book!



Submit

- your answer to the task in this lecture.



Note

- your questions and put them up in the relevant online session.



Email

- suggestions on content or quality of this lecture at uroojain@neduet.edu.pk

