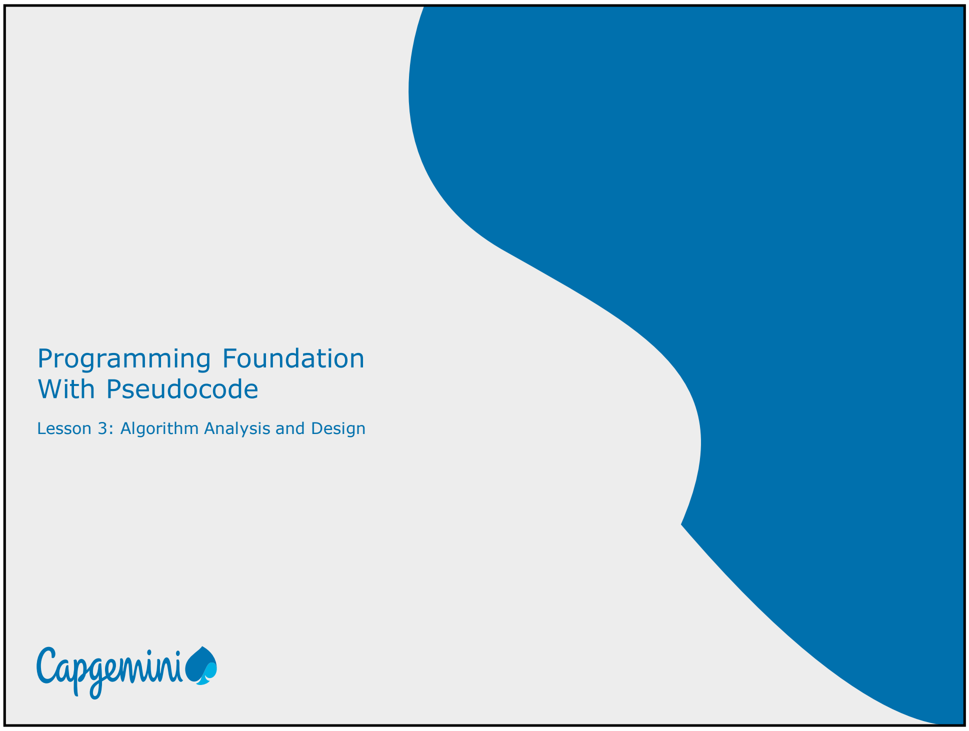
Instructor Notes:



Programming Foundation With Pseudocode

Lesson 3: Algorithm Analysis and Design

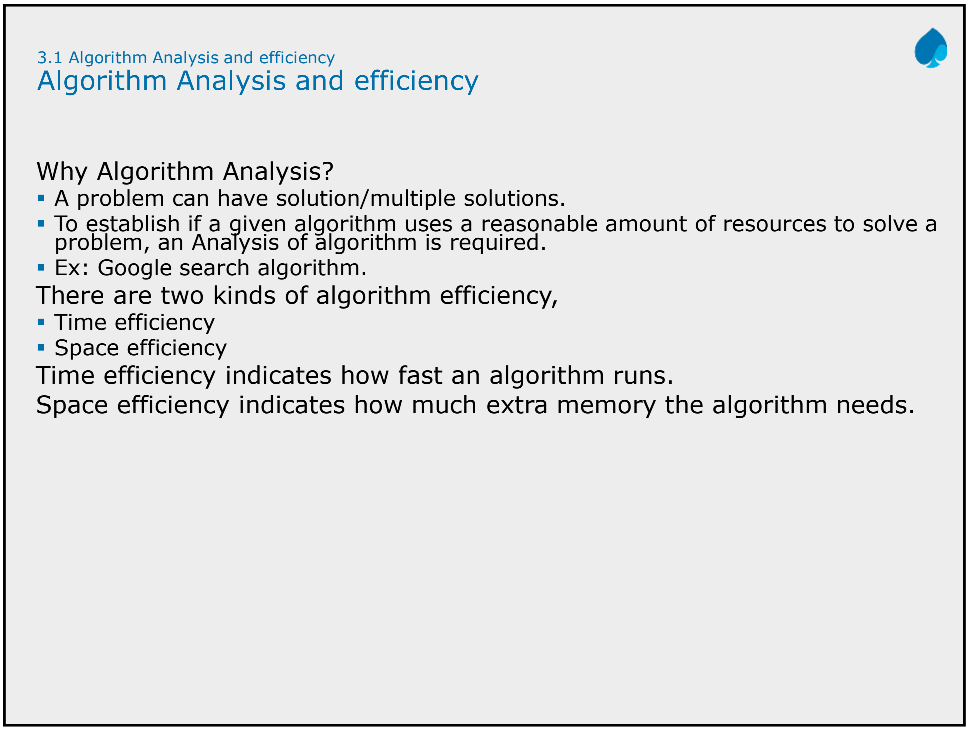
Instructor Notes:

Lesson Objectives

To understand the following concepts

- Algorithm Analysis and efficiency
- Measuring Unit for Algorithm
- Order of Growth
  - Asymptotic notations
- Best/Worst/Average case
- Example

Instructor Notes:



3.1 Algorithm Analysis and efficiency

Algorithm Analysis and efficiency

Why Algorithm Analysis?

- A problem can have solution/multiple solutions.
- To establish if a given algorithm uses a reasonable amount of resources to solve a problem, an Analysis of algorithm is required.
- Ex: Google search algorithm.

There are two kinds of algorithm efficiency,

- Time efficiency
- Space efficiency

Time efficiency indicates how fast an algorithm runs.

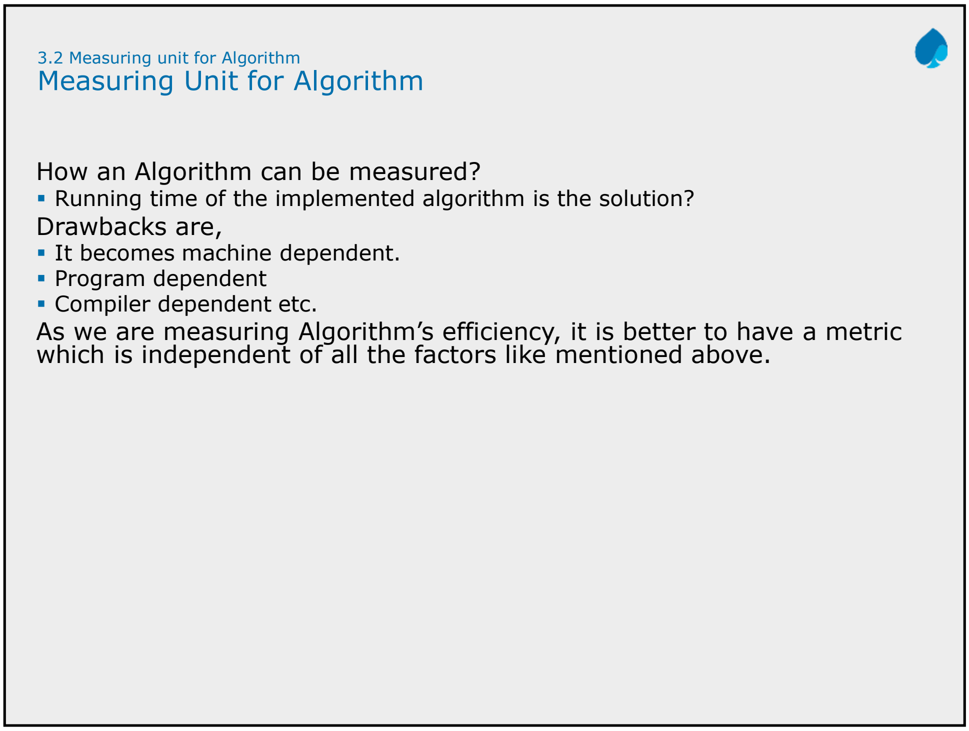
Space efficiency indicates how much extra memory the algorithm needs.

Algorithm is a step by step instruction to solve a given problem. Algorithm can have multiple solutions.
For Ex: We have multiple solutions to find GCD of two numbers.
Hence we need a method to compare/analyze between multiple solutions/algorithms and find the efficient solution. This chapter explains how to analyze and find efficiency of computer algorithms.

Why Algorithm Analysis?

An analysis is useful to establish if a given algorithm uses a reasonable amount of resources to solve a problem. If the algorithm needs too many resources then, even if it is correct, it cannot be used in practice. Such an algorithm is not an efficient one. When we are talking about efficiency we can refer to space efficiency (it deals with the space required by the algorithm) or to time efficiency (it indicates how fast an algorithm runs).
One must usually make compromise between time and space efficiency. Compromise usually depends on the situation or application demands/requirements. One can be achieved with the trade off of another. No doubt, space was very much an issue in earlier days but even today for mobile and embedded applications, we will be working with limited memory.

Instructor Notes:



3.2 Measuring unit for Algorithm

Measuring Unit for Algorithm

How an Algorithm can be measured?

- Running time of the implemented algorithm is the solution?

Drawbacks are,

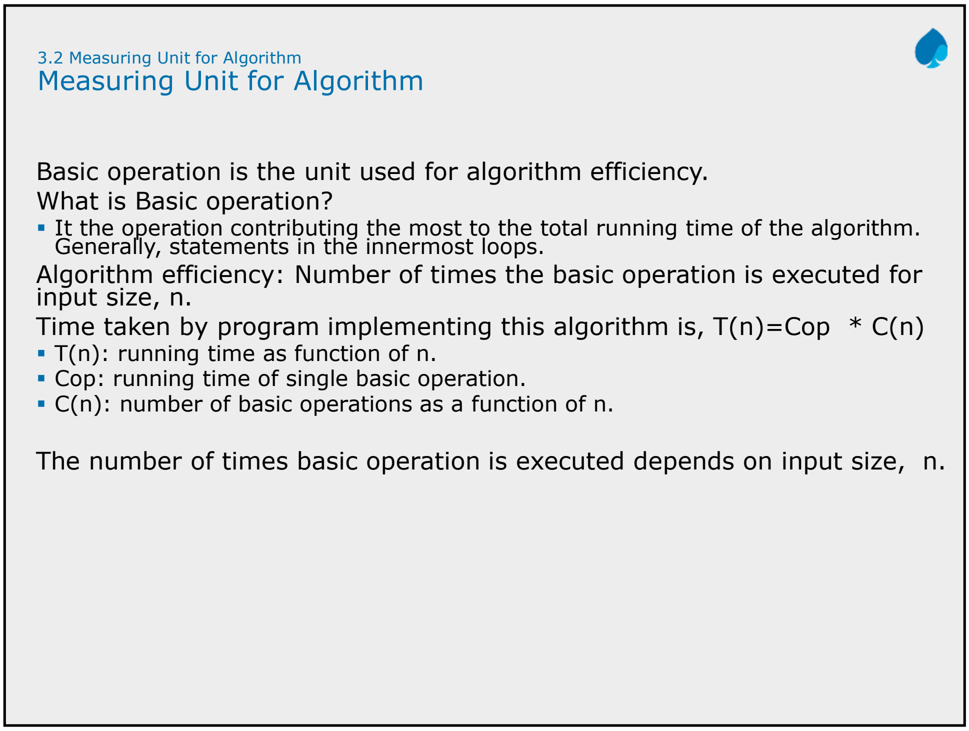
- It becomes machine dependent.
- Program dependent
- Compiler dependent etc.

As we are measuring Algorithm's efficiency, it is better to have a metric which is independent of all the factors like mentioned above.

When we talk about comparison, we need some unit. So when we talk about Algorithms measurement, time cannot be a good unit. Drawbacks are,

- ➢It becomes machine dependent.
- ➢Program dependent
- ➢Compiler dependent etc.

Instructor Notes:



3.2 Measuring Unit for Algorithm

Measuring Unit for Algorithm

Basic operation is the unit used for algorithm efficiency.

What is Basic operation?

- It the operation contributing the most to the total running time of the algorithm. Generally, statements in the innermost loops.

Algorithm efficiency: Number of times the basic operation is executed for input size, n.

Time taken by program implementing this algorithm is, T(n)=Cop * C(n)

- T(n): running time as function of n.
- Cop: running time of single basic operation.
- C(n): number of basic operations as a function of n.

The number of times basic operation is executed depends on input size, n.

Hence a proper measuring unit will be basic operation, the operation contributing the most to the total running time.

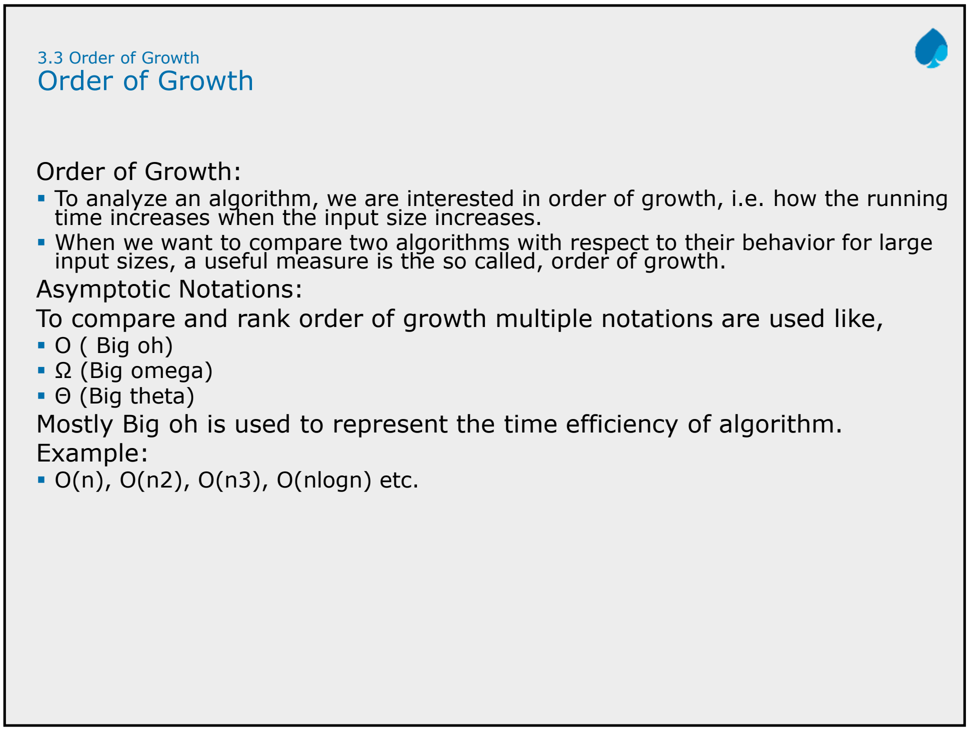
Basic operation: Maximum times executing statement in algorithm will be considered as basic operation. Usually statements in the inner loop are executed maximum number of times.

Other thing to note is that, there is no meaning in comparing the algorithms with lower input size. There is no much difference. Algorithms vary in efficiency only when their input size is very large. Hence while discussing further with the input size, n, in this chapter, we mean a large value for n.

To calculate efficiency of algorithm, compute the number of times the basic operation is executed on input size of n.

Algorithm efficiency is the number of times basic operation is executed for the given input size, n.

Instructor Notes:



3.3 Order of Growth

Order of Growth

Order of Growth:

- To analyze an algorithm, we are interested in order of growth, i.e. how the running time increases when the input size increases.
- When we want to compare two algorithms with respect to their behavior for large input sizes, a useful measure is the so called, order of growth.

Asymptotic Notations:

To compare and rank order of growth multiple notations are used like,

- O (Big oh)
- Ω (Big omega)
- Θ (Big theta)

Mostly Big oh is used to represent the time efficiency of algorithm.

Example:

- O(n), O(n2), O(n3), O(nlogn) etc.

Asymptotic Notations:

➢O : A function t(n) is said to be in O(g(n)), if t(n) is bounded above by some constant multiple of g(n) for all n, i.e. if there exist some positive constant c and some nonnegative integer n_0 such that

$$t(n)<=c(g(n)) \text{ for all } n>=n_0$$

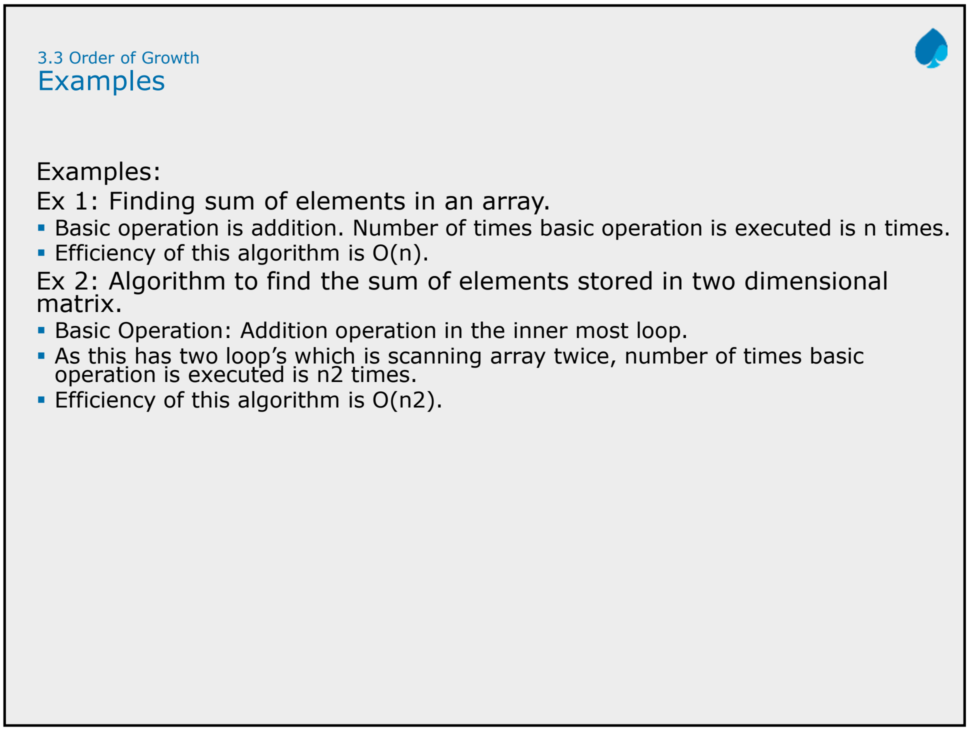
➢Ω : A function t(n) is said to be in Ω(g(n)), if t(n) is bounded below by some constant multiple of g(n) for all n, i.e. if there exist some positive constant c and some nonnegative integer n_0 such that

$$t(n)>=c(g(n)) \text{ for all } n>=n_0$$

➢Θ : A function t(n) is said to be in Θ(g(n)), if t(n) is bounded both above and below by some constant multiple of g(n) for all n, i.e. if there exist some positive constant c1 and c2 and some nonnegative integer n0 such that

$$c_2g(n)<=t(n)<=c_1(g(n)) \text{ for all } n>= n_0$$

Instructor Notes:



3.3 Order of Growth

Examples

Examples:

Ex 1: Finding sum of elements in an array.

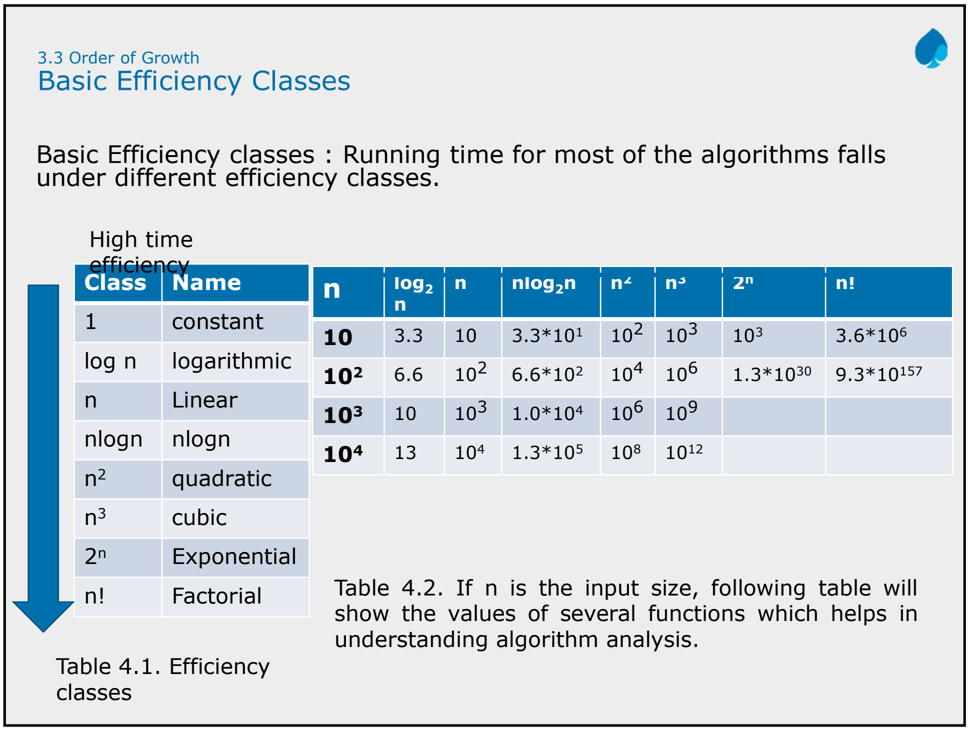
- Basic operation is addition. Number of times basic operation is executed is n times.
- Efficiency of this algorithm is O(n).

Ex 2: Algorithm to find the sum of elements stored in two dimensional matrix.

- Basic Operation: Addition operation in the inner most loop.
- As this has two loop’s which is scanning array twice, number of times basic operation is executed is n2 times.
- Efficiency of this algorithm is O(n2).

Write an algorithm for the above examples before finding its efficiency.

Instructor Notes:



3.3 Order of Growth

Basic Efficiency Classes

Basic Efficiency classes : Running time for most of the algorithms falls under different efficiency classes.

High time efficiency

Class	Name
1	constant
log n	logarithmic
n	Linear
nlogn	nlogn
n^2	quadratic
n^3	cubic
2^n	Exponential
n!	Factorial

Table 4.1. Efficiency classes

n	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n	n!
10	3.3	10	$3.3*10^1$	10^2	10^3	10^3	$3.6*10^6$
10^2	6.6	10^2	$6.6*10^2$	10^4	10^6	$1.3*10^{30}$	$9.3*10^{157}$
10^3	10	10^3	$1.0*10^4$	10^6	10^9		
10^4	13	10^4	$1.3*10^5$	10^8	10^{12}		

Table 4.2. If n is the input size, following table will show the values of several functions which helps in understanding algorithm analysis.

Constant Time: When instructions of program are executed once or at most only a few times , then the running time complexity of such algorithm is know as constant time. it is independent of the problem's size. It is represented as O(1). For example, linear search best case complexity is O(1)

Logarithmic: The running time of the algorithm in which large problem is solved by transforming into smaller sizes sub problems is said to be Logarithmic in nature. In this algorithm becomes slightly slower as n grows. It does not process all the data element of input size n. The running time does not double until n increases to n2 . It is represented as O(log n). For example binary search algorithm running time complexity is O(log n).

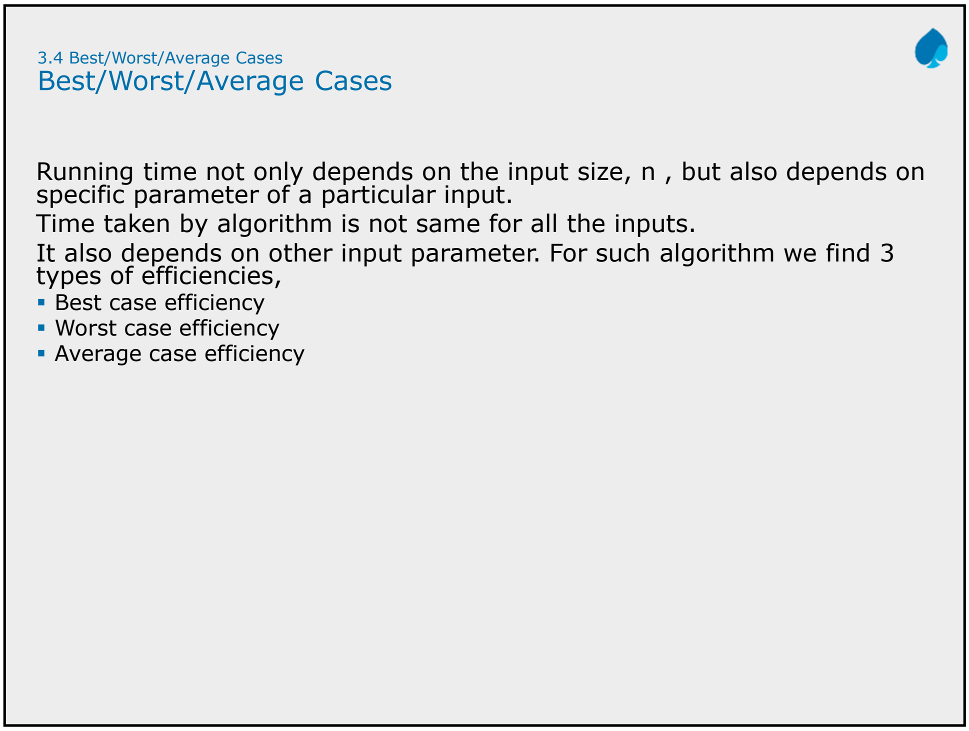
Linear: In this the complete set of instruction is executed once for each input i.e input of size n is processed. It is represented as O(n). This is the best option to be used when the whole input has to be processed. In this situation time requirement increases directly with the size of the problem. For example linear search Worst case complexity is O(n).

Quadratic : Running time of an algorithm is quadratic in nature when it process all pairs of data items. Such algorithm will have two nested loops. For input size n, running time will be O(n2). Practically this is useful for problem with small input size or elementary sorting problems. In this situation time requirement increases fast with the size of the problem. For example insertion sort running time complexity is O(n2).

Exponential: Running time of an algorithm is exponential in nature if brute force solution is applied to solve a problem. In such algorithm all subset of an n element set is generated. In this situation time requirement increases very fast with the size of the problem. For input size n, running time complexity expression will be O(2n).For example Boolean variable equivalence of n variables running time complexity is O(2n). Another familiar example is Tower of Hanoi problem where running time complexity is O(2n).

Factorial: Typical for algorithms that generate all permutations of n-element set.

Instructor Notes:



3.4 Best/Worst/Average Cases

Best/Worst/Average Cases

Running time not only depends on the input size, n , but also depends on specific parameter of a particular input.

Time taken by algorithm is not same for all the inputs.

It also depends on other input parameter. For such algorithm we find 3 types of efficiencies,

- Best case efficiency
- Worst case efficiency
- Average case efficiency

For few algorithms, running time not only depends on the input size, n , but also depends on specific parameter of a particular input.

Time taken by algorithm is not same for all the inputs. Hence time taken by algorithm may be different for the same size of input, n. Because It also depends on other input parameter.

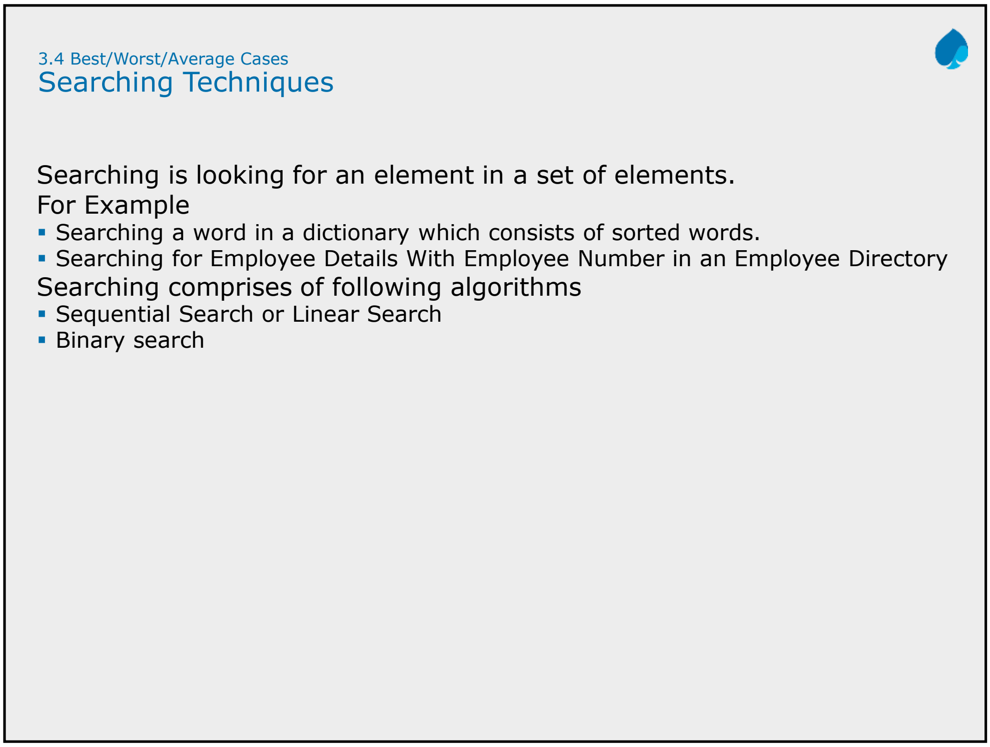
For example, Consider that one is looking for a certain card in a deck of unsorted cards. One card is opened at a time from the deck. Now there are multiple cases,

- We can find the card for the first time.
- We can never find the card.
- We can find the card at the last.
- Or we can find the card somewhere in between.

So here, efficiency differs for different scenarios. This is because, it doesn’t matter, how many total cards were found, but it depends on how soon the card is found. Hence for such algorithms we find,

- Best case
- Worst case
- Average case

Instructor Notes:



3.4 Best/Worst/Average Cases

Searching Techniques

Searching is looking for an element in a set of elements.

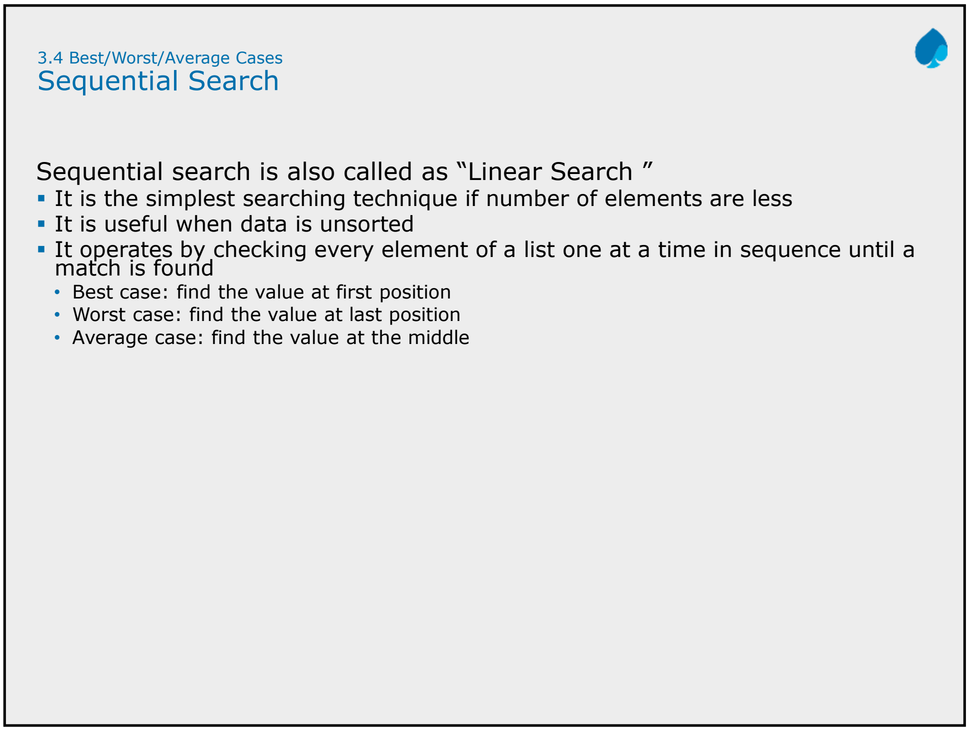
For Example

- Searching a word in a dictionary which consists of sorted words.
- Searching for Employee Details With Employee Number in an Employee Directory

Searching comprises of following algorithms

- Sequential Search or Linear Search
- Binary search

Instructor Notes:



3.4 Best/Worst/Average Cases

Sequential Search

Sequential search is also called as "Linear Search "

- It is the simplest searching technique if number of elements are less
- It is useful when data is unsorted
- It operates by checking every element of a list one at a time in sequence until a match is found
  - Best case: find the value at first position
  - Worst case: find the value at last position
  - Average case: find the value at the middle

Linear search:

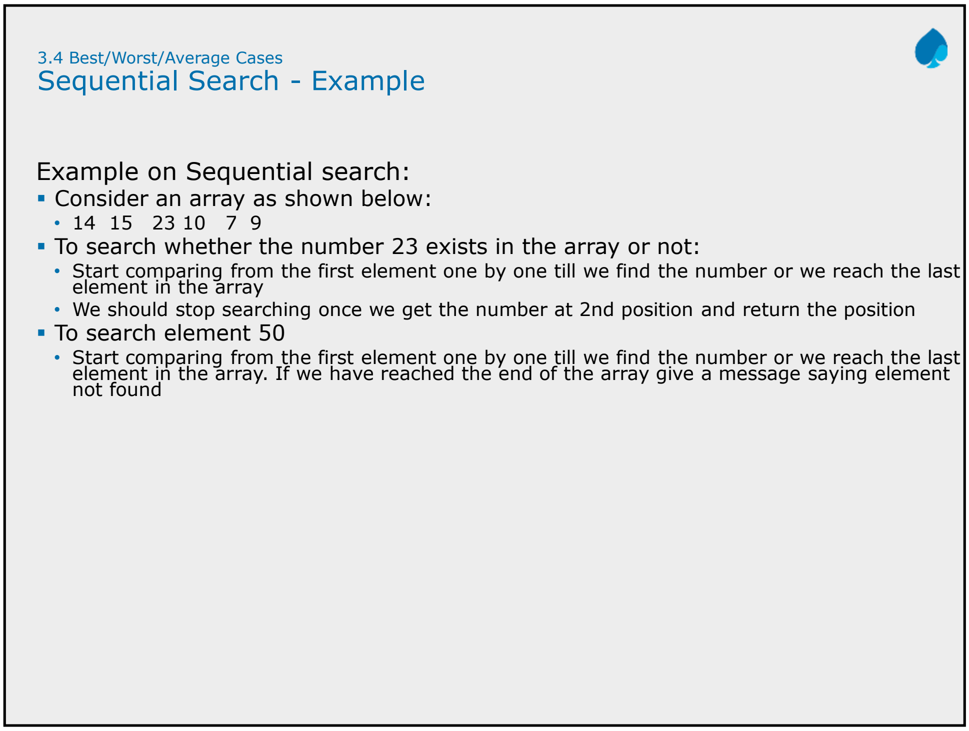
The Linear search technique is normally used when data is not in the sorted order. Linear search is another name for "sequential search".

If we find the value at first position, then it is "best case" because search requires to do only one comparison.

However, if we find the value at the end, then this is "worst scenario" because we have to search for all the n values sequentially till last value

If we find data at the middle then we require to do only n/2 comparisons.

Instructor Notes:



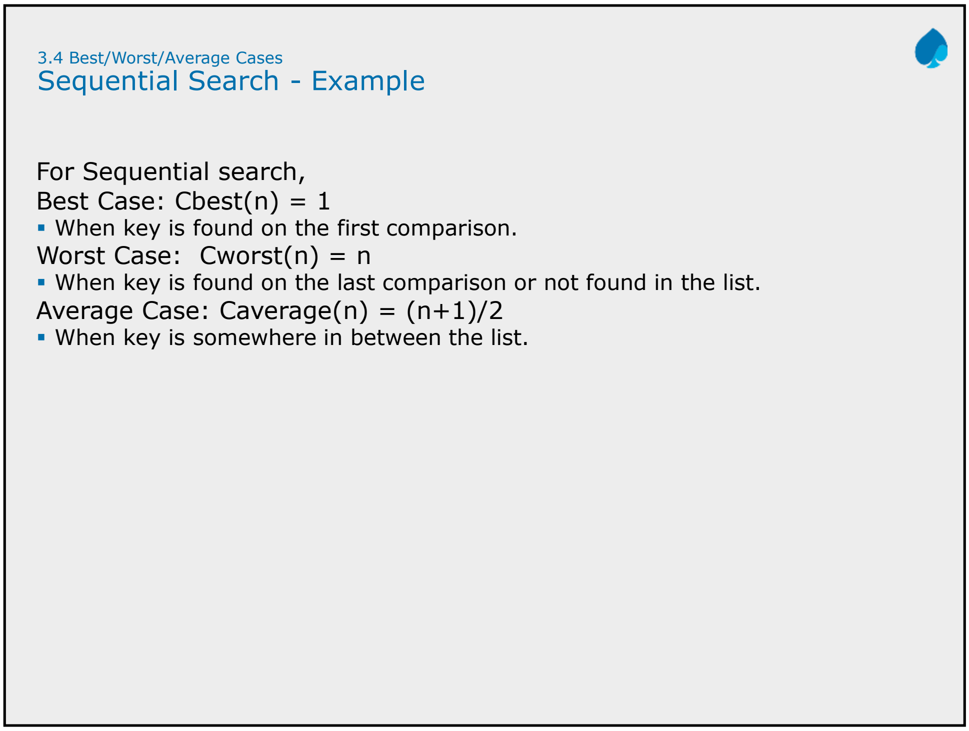
3.4 Best/Worst/Average Cases

Sequential Search - Example

Example on Sequential search:

- Consider an array as shown below:
  - 14 15 23 10 7 9
- To search whether the number 23 exists in the array or not:
  - Start comparing from the first element one by one till we find the number or we reach the last element in the array
  - We should stop searching once we get the number at 2nd position and return the position
- To search element 50
  - Start comparing from the first element one by one till we find the number or we reach the last element in the array. If we have reached the end of the array give a message saying element not found

Instructor Notes:



3.4 Best/Worst/Average Cases

Sequential Search - Example

For Sequential search,

Best Case: Cbest(n) = 1

- When key is found on the first comparison.

Worst Case: Cworst(n) = n

- When key is found on the last comparison or not found in the list.

Average Case: Caverage(n) = (n+1)/2

- When key is somewhere in between the list.

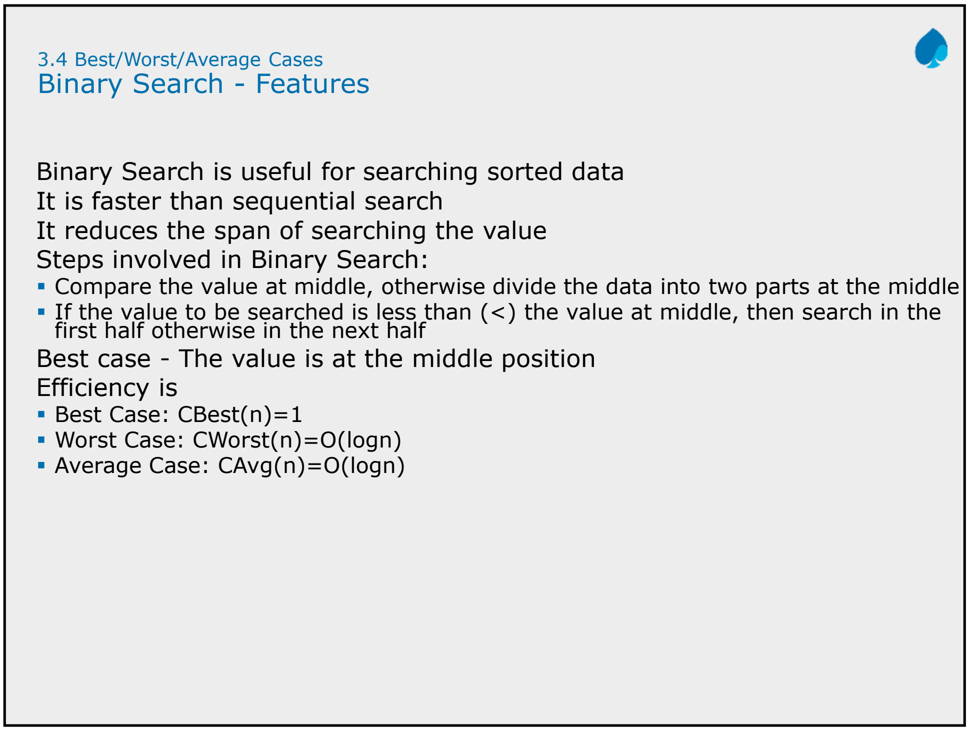
➢For Sequential seacrch,

➢Best Case: $C_{best}(n) = 1$: Here key is found in the fist place of the list. Hence only one comparison is done.

➢Worst Case: $C_{worst}(n) = n$: Worst case scenario for sequential search is, when key is found on the last comparison or not found in the list. So if the list contains n elements the n number of comparisons is done.

➢Average Case: $C_{average}(n) = (n+1)/2$: When key is somewhere in between the list. (n+1)/2 is derived from the mathematical computation with probability concepts.

Instructor Notes:



3.4 Best/Worst/Average Cases

Binary Search - Features

Binary Search is useful for searching sorted data

It is faster than sequential search

It reduces the span of searching the value

Steps involved in Binary Search:

- Compare the value at middle, otherwise divide the data into two parts at the middle
- If the value to be searched is less than (<) the value at middle, then search in the first half otherwise in the next half

Best case - The value is at the middle position

Efficiency is

- Best Case: CBest(n)=1
- Worst Case: CWorst(n)=O(logn)
- Average Case: CAvg(n)=O(logn)

Binary search:

It is useful to search information from the sorted data. It is faster than “sequential search”.

Steps involved in Binary search:

Compares the value at mid position, otherwise divides the data into two parts at the middle.

If the value to be searched is less than (<) the value at middle, then search in the fist half otherwise in the next half.

Repeat the steps 1 and 2 for the selected half till we get the value to be searched.

Example:

Suppose you are searching for the word ‘psychology’ in the dictionary, while searching if you find the word “Quoits” then we will definitely search in previous pages. In this example too we are reducing span of searching because dictionary is sorted alphabetically. Similar technique is used in binary search.

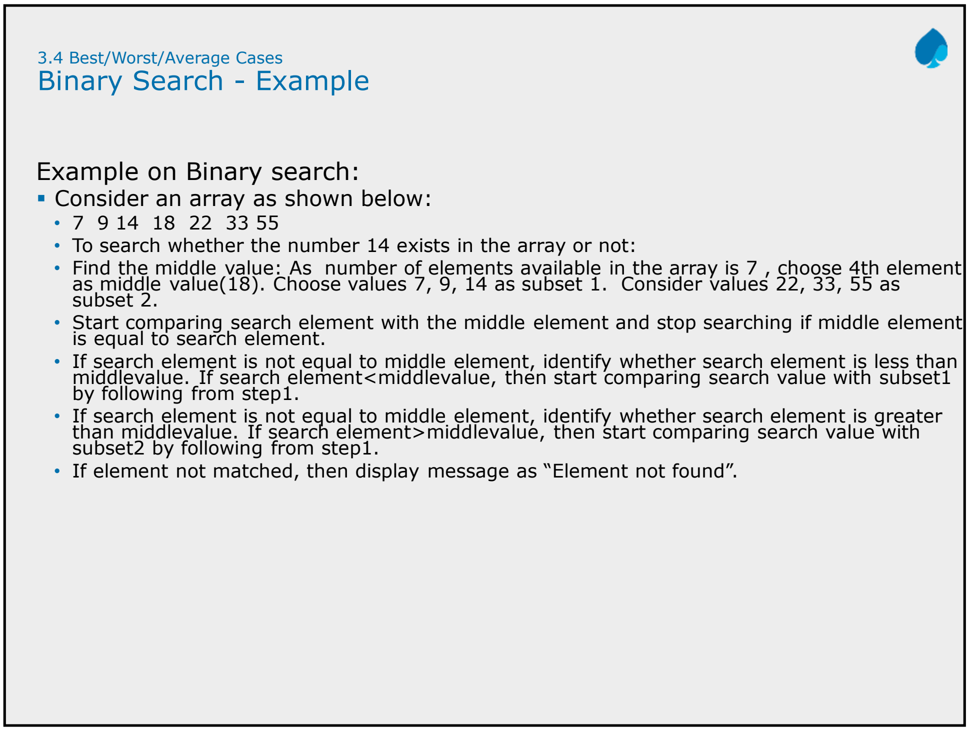
Consider an array of numbers:

14 15 18 24 25 78 82 85 89 92 100

Search whether number 15 is in the array or not.

Since the array is in the sorted order, we can use binary search algorithm.

Instructor Notes:



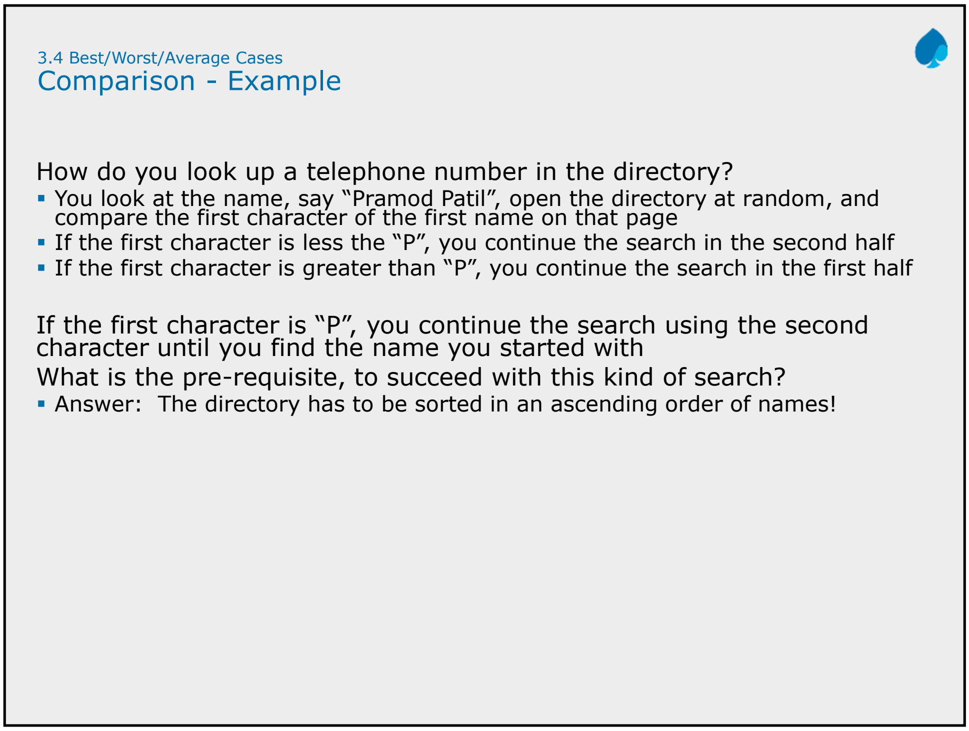
3.4 Best/Worst/Average Cases

Binary Search - Example

Example on Binary search:

- Consider an array as shown below:
  - 7 9 14 18 22 33 55
  - To search whether the number 14 exists in the array or not:
  - Find the middle value: As number of elements available in the array is 7 , choose 4th element as middle value(18). Choose values 7, 9, 14 as subset 1. Consider values 22, 33, 55 as subset 2.
  - Start comparing search element with the middle element and stop searching if middle element is equal to search element.
  - If search element is not equal to middle element, identify whether search element is less than middlevalue. If search element<middlevalue, then start comparing search value with subset1 by following from step1.
  - If search element is not equal to middle element, identify whether search element is greater than middlevalue. If search element>middlevalue, then start comparing search value with subset2 by following from step1.
  - If element not matched, then display message as “Element not found”.

Instructor Notes:



3.4 Best/Worst/Average Cases

Comparison - Example

How do you look up a telephone number in the directory?

- You look at the name, say “Pramod Patil”, open the directory at random, and compare the first character of the first name on that page
- If the first character is less the “P”, you continue the search in the second half
- If the first character is greater than “P”, you continue the search in the first half

If the first character is “P”, you continue the search using the second character until you find the name you started with

What is the pre-requisite, to succeed with this kind of search?

- Answer: The directory has to be sorted in an ascending order of names!

Steps:

The number at the middle is 78 != 15 but 15 < 78, hence search in the first half.

The value at the middle is 18 !=15 but 15 < 18, hence search it in the first half.

The value at the middle is 15, which is equal to the number we are searching for.

Display message, Number found at 2nd position or return the position.

Search whether number 10 is in the array or not.

Steps:

The number at the middle is 78 != 10 but 10 < 78, hence search in the first half.

The value at the middle is 18 !=10 but 10 < 18, hence search it in the first half.

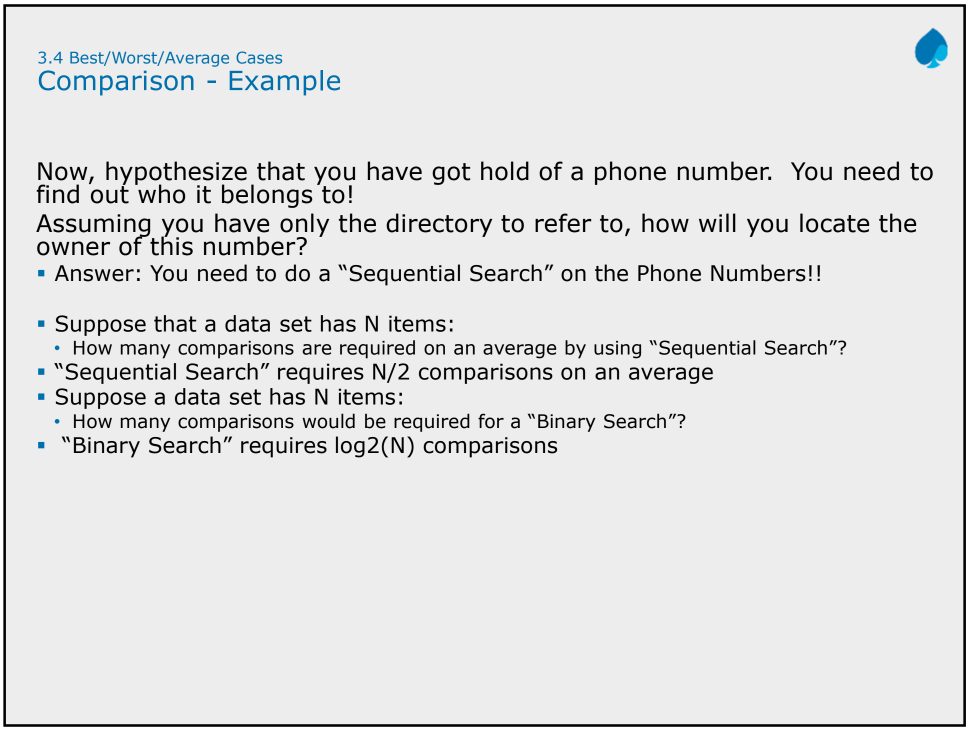
The value at the middle is 15 !=10, hence search it in the first half.

The value at the middle is 14 !=10, At this point we have reached the first element’s

position beyond which we cannot move.

Display message, Number not found

Instructor Notes:



3.4 Best/Worst/Average Cases

Comparison - Example

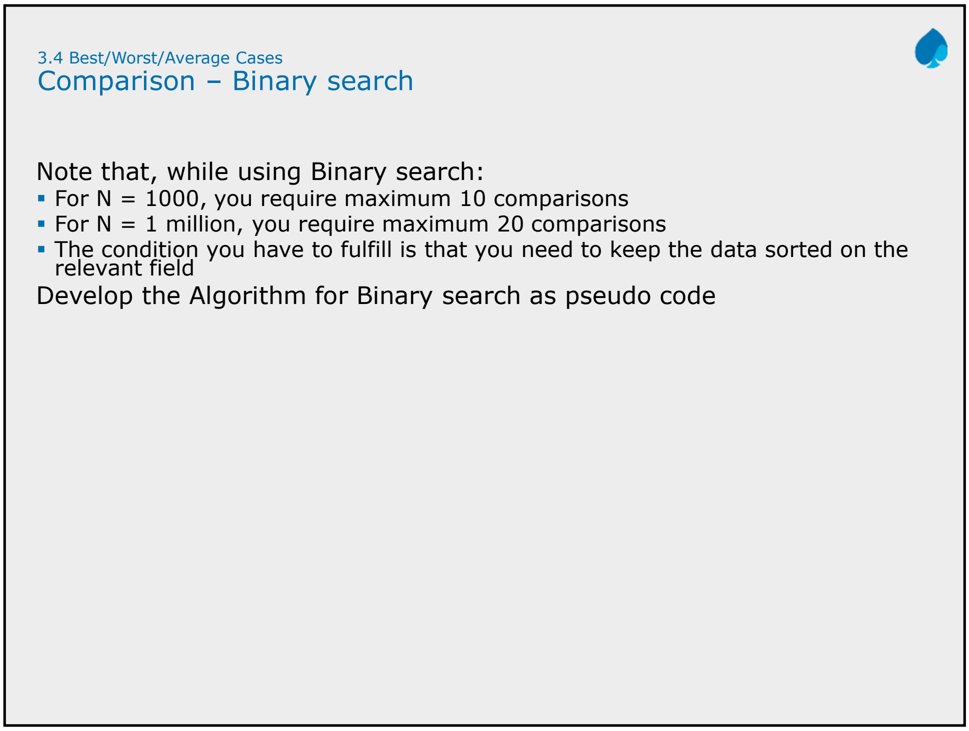
Now, hypothesize that you have got hold of a phone number. You need to find out who it belongs to!

Assuming you have only the directory to refer to, how will you locate the owner of this number?

- Answer: You need to do a “Sequential Search” on the Phone Numbers!!

- Suppose that a data set has N items:
  - How many comparisons are required on an average by using “Sequential Search”?
- “Sequential Search” requires N/2 comparisons on an average
- Suppose a data set has N items:
  - How many comparisons would be required for a “Binary Search”?
- “Binary Search” requires log2(N) comparisons

Instructor Notes:



3.4 Best/Worst/Average Cases

Comparison – Binary search

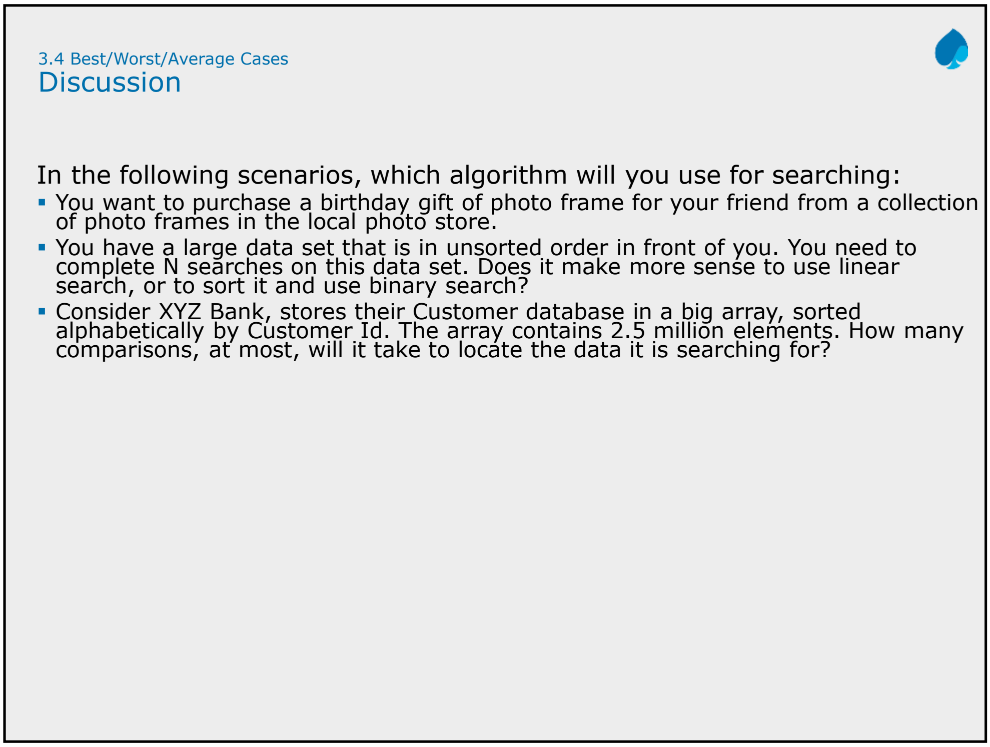
Note that, while using Binary search:

- For N = 1000, you require maximum 10 comparisons
- For N = 1 million, you require maximum 20 comparisons
- The condition you have to fulfill is that you need to keep the data sorted on the relevant field

Develop the Algorithm for Binary search as pseudo code

Instructor Notes:

Discuss scenarios with Participants to better understand the usage of searching algorithms.



3.4 Best/Worst/Average Cases

Discussion

In the following scenarios, which algorithm will you use for searching:

- You want to purchase a birthday gift of photo frame for your friend from a collection of photo frames in the local photo store.
- You have a large data set that is in unsorted order in front of you. You need to complete N searches on this data set. Does it make more sense to use linear search, or to sort it and use binary search?
- Consider XYZ Bank, stores their Customer database in a big array, sorted alphabetically by Customer Id. The array contains 2.5 million elements. How many comparisons, at most, will it take to locate the data it is searching for?

Answer for Scenario 1:

- Sequential Search need to be used, Since the frames are not located in any order.
- Starting at the first frame, examine each frame (Without skipping) until you find the frame you want.

Answer for Scenario 2:

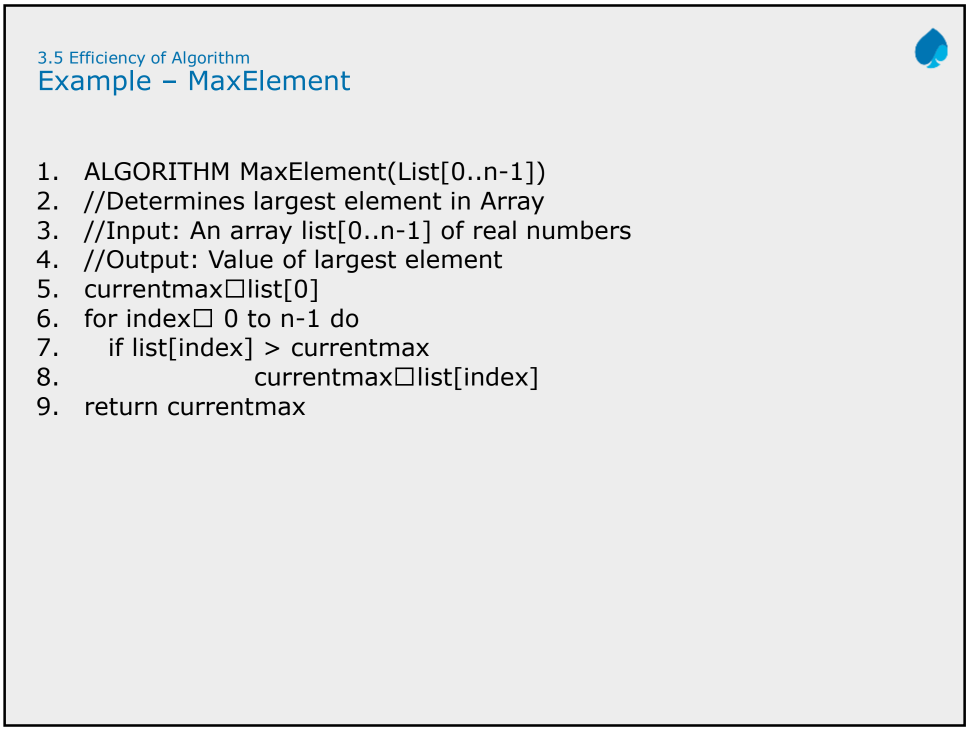
- It makes more sense to sort it and use binary search.
- Sorting of a data set can be done in ***O(nlogn)*** time.
- To do n linear searches will take n*O(n) = = O(n 2) time.
- To sort it and do n binary searches will take O(nlogn) + n*O(logn) = = O(nlogn) time

Answer for Scenario 3:

- Since the array is sorted, Binary search algorithm needs to be used.
- It will take at most 22 comparisons; ***ceil*** **(*log*(2, 500, 000)) = = 22** .

Instructor Notes:

More focus is given on how to find efficiency class for a given algorithm. We are not expecting one to study selection sort technique in depth.



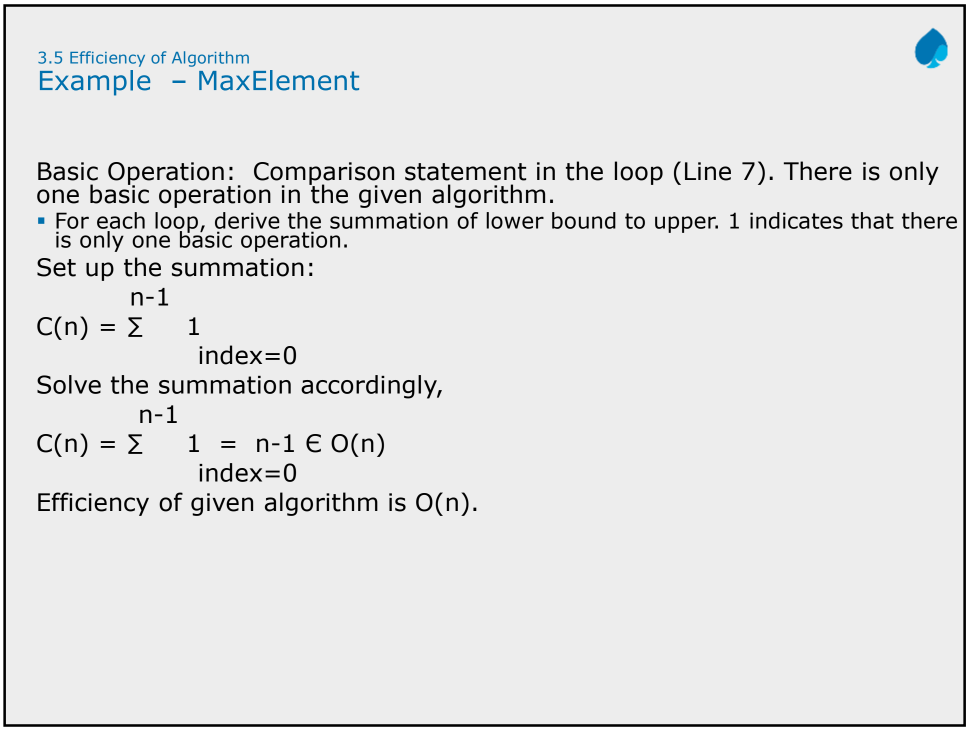
3.5 Efficiency of Algorithm

Example – MaxElement

```
ALGORITHM MaxElement(List[0..n-1])
//Determines largest element in Array
//Input: An array list[0..n-1] of real numbers
//Output: Value of largest element
currentmax□list[0]
for index□ 0 to n-1 do
   if list[index] > currentmax
                  currentmax□list[index]
return currentmax
```

Consider the above MaxElement algorithm. It also depicts proper way of writing algorithm. It should always start with a proper name and input as a parameter. First comment describes the algorithm. Second comment explains the input and third comment explains the output. Proper indentation must be maintained in the algorithm.

Instructor Notes:



3.5 Efficiency of Algorithm

Example – MaxElement

Basic Operation: Comparison statement in the loop (Line 7). There is only one basic operation in the given algorithm.

- For each loop, derive the summation of lower bound to upper. 1 indicates that there is only one basic operation.

Set up the summation:

$$C(n) = \sum_{index=0}^{n-1} 1$$

Solve the summation accordingly,

$$C(n) = \sum_{index=0}^{n-1} 1 = n-1 \in O(n)$$

Efficiency of given algorithm is O(n).

➢How it works?

➢Basic Operation: Comparison statement in the loop.

➢Summation of lower bound to upper bound. For each loop derive the summation. Hence as given algorithm has only one loops, we end up with 1 summation. 1 indicates that there is only one basic operation.

➢Now solving the above expression with proper formula's, we end up with n-1. We are more interested in efficiency class. Hence constants can be ignored and so efficiency is O(n). i.e Linear.

Summation formula used:

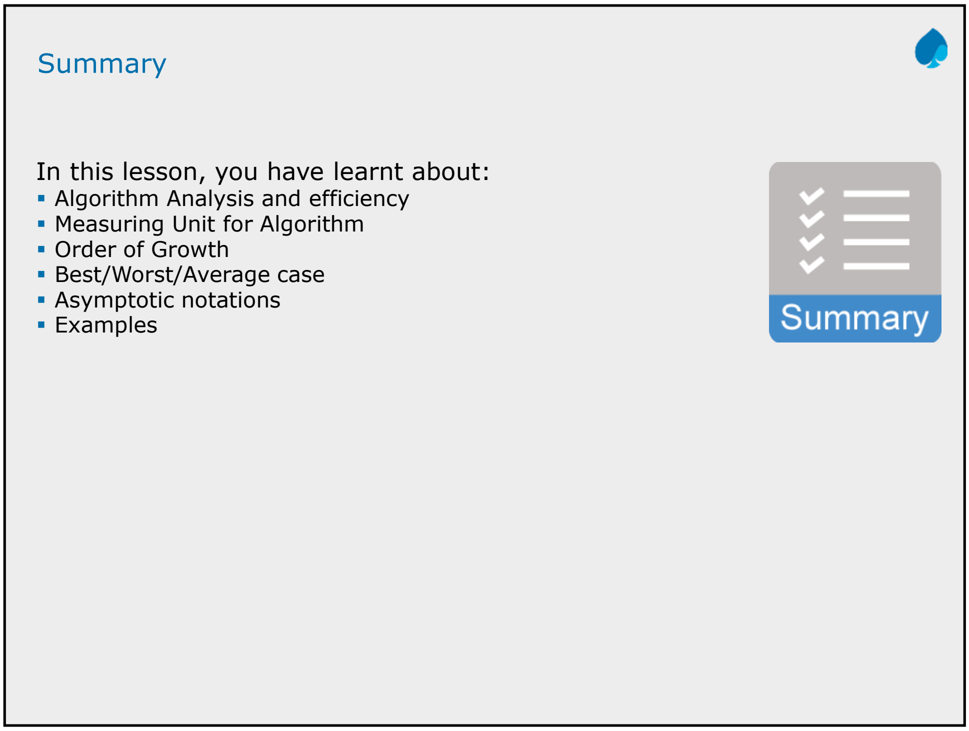
$$\sum_{i=l}^{u} 1 = u-l+1$$ where l<=u are lower and upper integer limits.

Instructor Notes:

Lab

Analyzing Algorithms Lab 3

Instructor Notes:



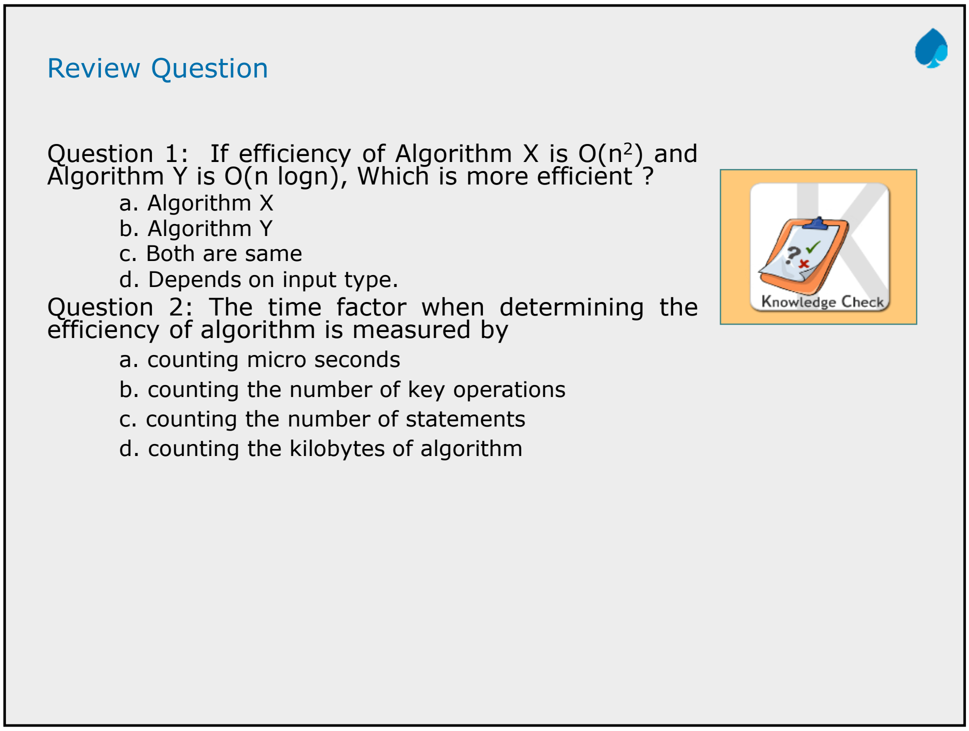
Summary

In this lesson, you have learnt about:

- Algorithm Analysis and efficiency
- Measuring Unit for Algorithm
- Order of Growth
- Best/Worst/Average case
- Asymptotic notations
- Examples

Instructor Notes:

Question 1: B
Question 2: B



Review Question

Question 1: If efficiency of Algorithm X is O(n^2) and Algorithm Y is O(n logn), Which is more efficient ?

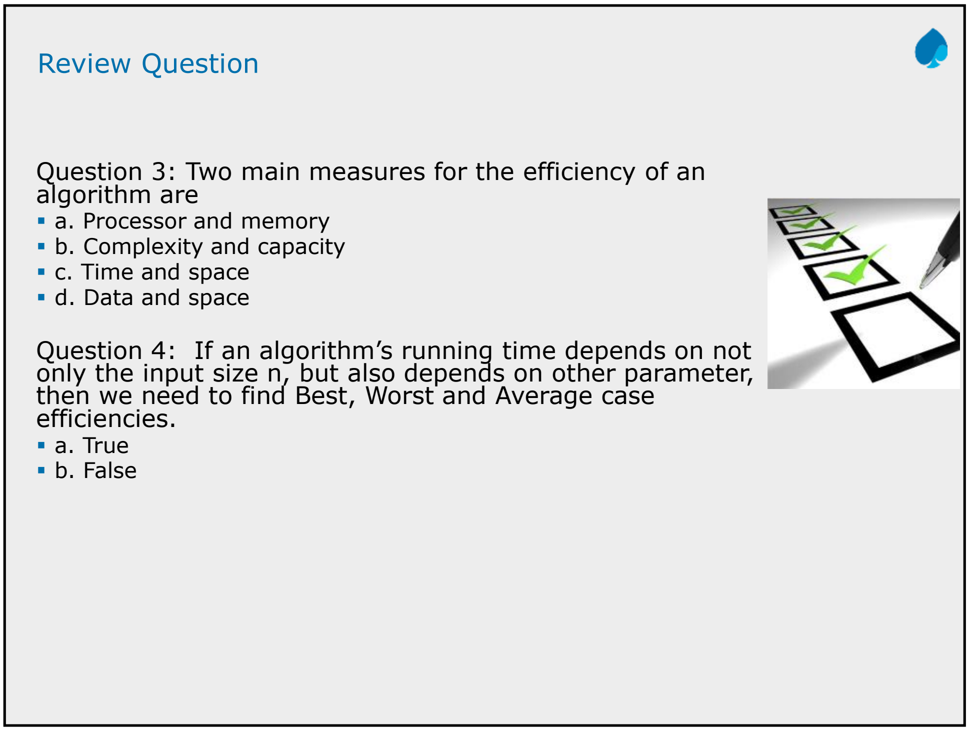
a. Algorithm X
b. Algorithm Y
c. Both are same
d. Depends on input type.

Question 2: The time factor when determining the efficiency of algorithm is measured by

a. counting micro seconds
b. counting the number of key operations
c. counting the number of statements
d. counting the kilobytes of algorithm

Instructor Notes:

Question 1: B
Question 2: B



Review Question

Question 3: Two main measures for the efficiency of an algorithm are

- a. Processor and memory
- b. Complexity and capacity
- c. Time and space
- d. Data and space

Question 4: If an algorithm’s running time depends on not only the input size n, but also depends on other parameter, then we need to find Best, Worst and Average case efficiencies.

- a. True
- b. False