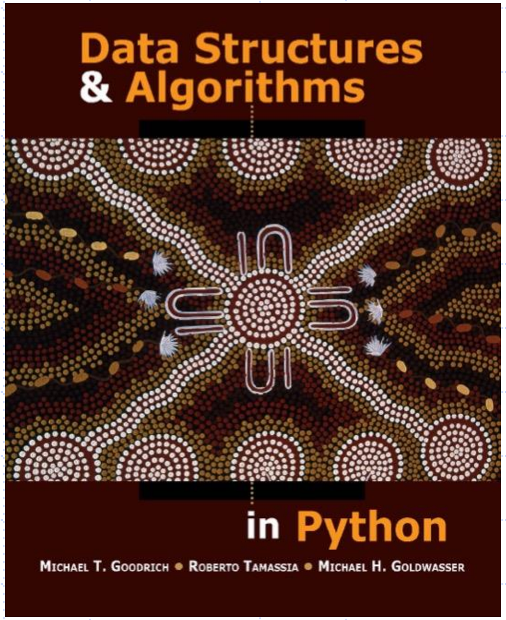
Welcome to DS2030 – Data Structures and Algorithms for Data Science

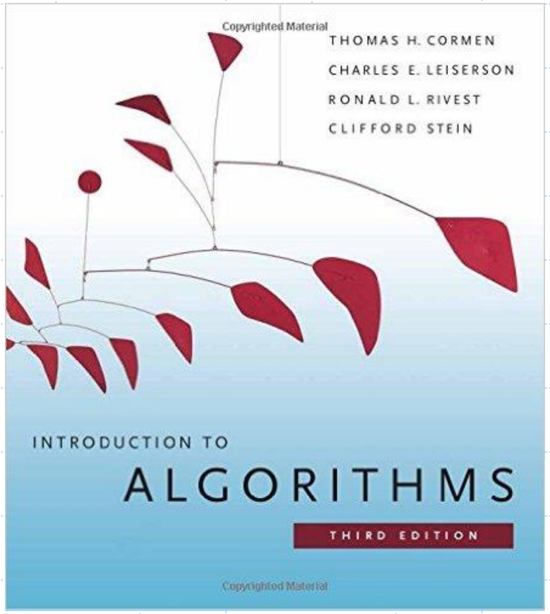
Narayanan (CK) C Krishnan
ckn@iitpkd.ac.in

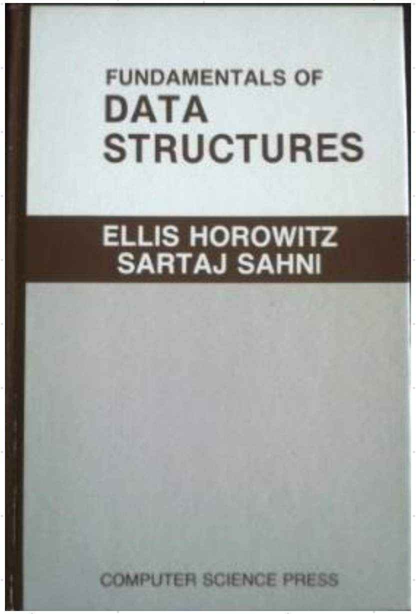
Administrative Trivia

- Course Structure
  - 3-0-3 (5 credits)
- Lecture Timings
  - Monday 10.00-10.50am
  - Wednesday 8.00-8.50am
  - Friday 11.00-11.50am
- Lab hours – Dr. Sahasranand
  - Tuesday 2.00-5.00pm
- Teaching Assistant
  - Head TA for Theory – Vidhya S
  - Head TA for Lab – Bijin Elsa Baby
  - TAs
    - Sudhin S
    - Yedavali Siva Kumar
- Office Hours
  - Instructor – M,W,F immediately after class or appointment over email
- Pre-registered students have already been added
  - other students – send email to the instructor

Reference Material







Data Structures and Algorithms in Python
Goodrich, Tamasia and Goldwasser
1st edition

Introduction to Algorithms
Cormen, Leiserson, Rivest, Stein
3rd edition

Fundamentals of Data Structures
Horowitz Sahni

Tentative Course Schedule

Sl Num	Week	Topic	Quiz	Compensatory Class
1	Jul 28 – Aug 1	Introduction, OOP, Arrays, and Linked Lists		Aug 2 9.00am
2	~~Aug 4 – Aug 8~~	Analysis Tools and Recursion		Aug 9 9.00am
3	Aug 11 – Aug 16	Stacks and Queues		
4	Aug 18 – Aug 22	Tree Structures and Heaps		
5	Aug 25 – Aug 29	Sorting and Priority Queues		
6	Sep 1 – Sep 5	Maps, Hash Tables		
7	Sep 8 – Sep 12	Search Tree Structures I	Q1 (Test 1) Sep 12	
8	Sep 15– Sep 19	Search Tree Structures II		
9	Sep 22 – Sep 26	Search Tree Structures III		
10	Sep 29 – Oct 3	Buffer Week		
11	Oct 6 – Oct 10	String Processing		
12	Oct 13 – Oct 17	Dynamic Programming	Q2 (Test 2) Oct 17	
13	Oct 20 – Oct 24	Graphs I		
14	Oct 27 – Oct 31	Graphs II		
15	Nov 3 – Nov 7	Graphs III		
16	Nov 10 – Nov 14	Buffer Week		

Quizzes – 20%

- Institute mandated Test 1 and Test 2
- Cover material discussed from the previous quiz till the current week
- Duration – 50 minutes
- Makeup quizzes will be allowed Test 1 and Test 2 according to the institute mandated guidelines

Quiz	Week
Q1	7
Q2	12

Exams – 30%

- End-semester Exam – 30%
  - As per the institute exam schedule
  - Covers the entire syllabus

Grading Scheme

- Tentative Breakup for the Theory
  - Quizzes (2) - 20%
  - End-Semester Exam - 30%
- Lab will comprise of the remaining 50%

Attendance – Bonus 1%

- Encourage you to attend classes
- Attendance will be taken during every class
- 1% will count towards the final score
  - Borderline grades will have an edge

Honor Code

- Encourage to form study groups to review class lectures/textbook material
  - Group discussions are encouraged
- Unless explicitly stated all quizzes and labs
  - are to be done individually.
  - web trawled code or code copied from classmates is strictly forbidden.
- Any infraction will be dealt in severest terms
  - Direct Fail grade in the course.

I reserve the right to question you, if I suspect any misconduct in any submission.

Course Website

- All class related material will be accessible from the course module on LMS.

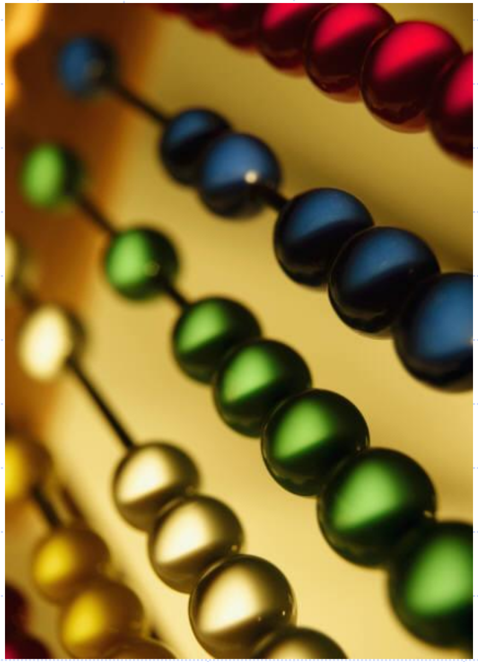
Data Structures

- What are data structures?

Data Structures

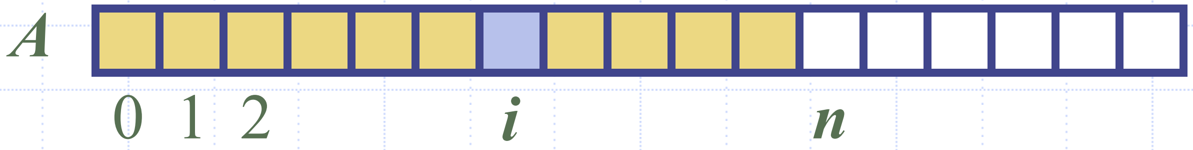
- What are data structures?
  - a way to organize data
  - easier access/manipulate to data
- Topics for the next couple of weeks
  - Concrete Data Structures
    - Arrays and Linked Lists
  - Algorithmic Analysis
  - Recursion

Arrays



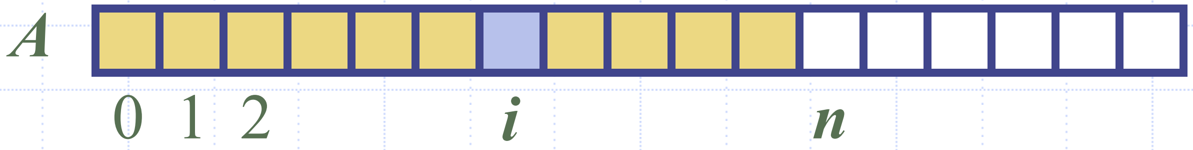
Python Sequence Classes

- Python has built-in types, **list**, **tuple**, and **str**.
- Each of these **sequence** types supports indexing to access an individual element of a sequence, using a syntax such as A[i]
- Each of these types uses an **array** to represent the sequence.
  - An array is a set of memory locations that can be addressed using consecutive indices, which, in Python, start with index 0.



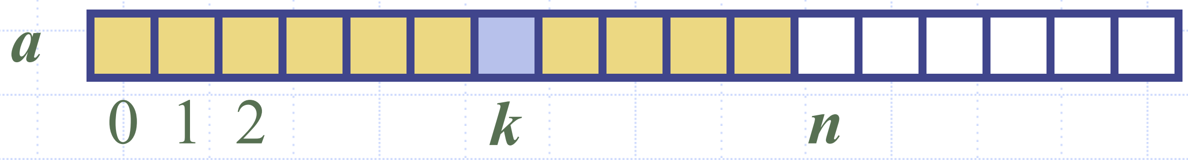
Array Definition

- An ***array*** is a sequenced collection of variables all of the same type. Each variable, or ***cell***, in an array has an ***index***, which uniquely refers to the value stored in that cell. The cells of an array, A, are numbered 0, 1, 2, and so on.
- Each value stored in an array is often called an ***element*** of that array.



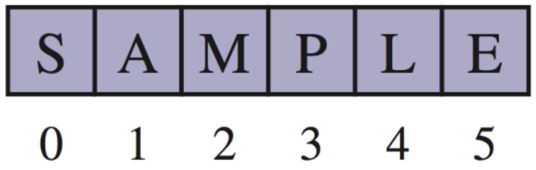
Array Length and Capacity

- Since the length of an array determines the maximum number of things that can be stored in the array, we will sometimes refer to the length of an array as its ***capacity***.
- In Python, the length of an array named *a* can be accessed using the syntax *len()*. Thus, the cells of an array, *a*, are numbered 0, 1, 2, and so on, up through *len(a)*–1, and the cell with index *k* can be accessed with syntax *a*[*k*].

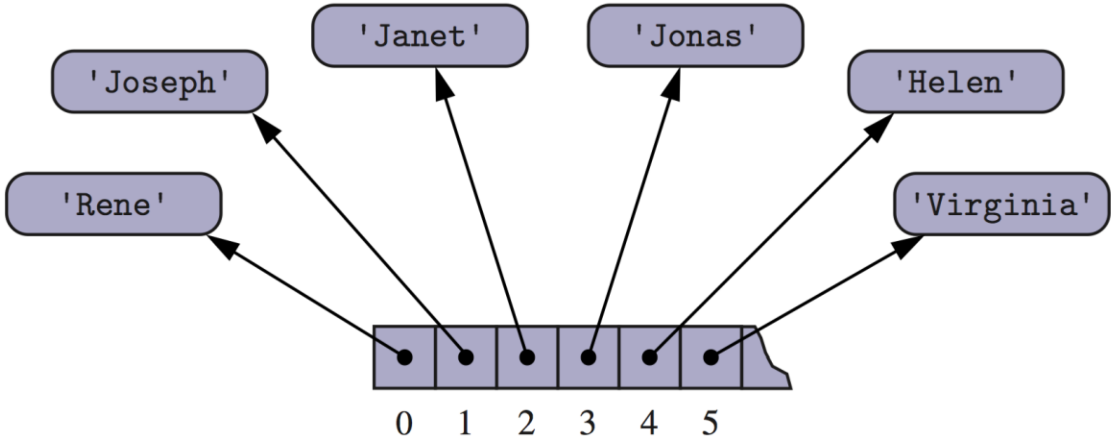


Arrays of Characters or Object References

- An array can store primitive elements, such as characters, giving us a **compact array.**



- An array can also store references to objects.



Compact Arrays

- Primary support for compact arrays is in a module named **array**.
  - That module defines a class, also named array, providing compact storage for arrays of primitive data types.
- The constructor for the array class requires a type code as a first parameter, which is a character that designates the type of data that will be stored in the array.

```
primes = array('i', [2, 3, 5, 7, 11, 13, 17, 19])
```

Type Codes in the array Class

- Python's array class has the following type codes:

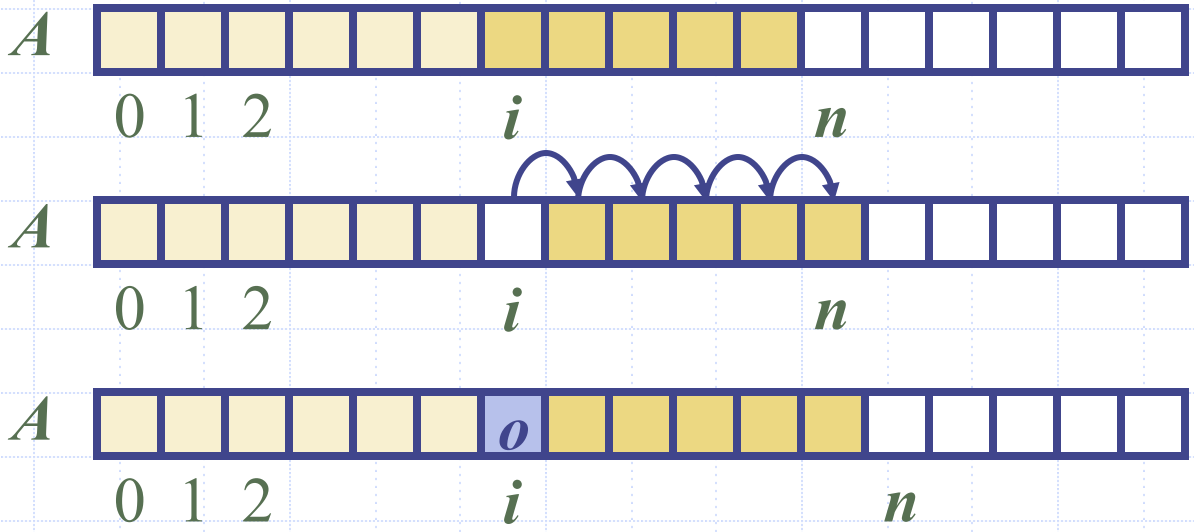
Code	C Data Type	Typical Number of Bytes
'b'	signed char	1
'B'	unsigned char	1
'u'	Unicode char	2 or 4
'h'	signed short int	2
'H'	unsigned short int	2
'i'	signed int	2 or 4
'I'	unsigned int	2 or 4
'l'	signed long int	4
'L'	unsigned long int	4
'f'	float	4
'd'	float	8

Insertion

- Array - A, Length - N
- Insert object ‘o’ at index k
  - A[0], A[1],…A[N]
- Solution - modifies the array
  - A[k] = o
- Solution
  - Create a new array B - size N+1
    - Copy all the elements in A to B until the index k
    - B[k] = o
    - Copy the remaining elements in A starting from k to N into B from k+1 through N+1

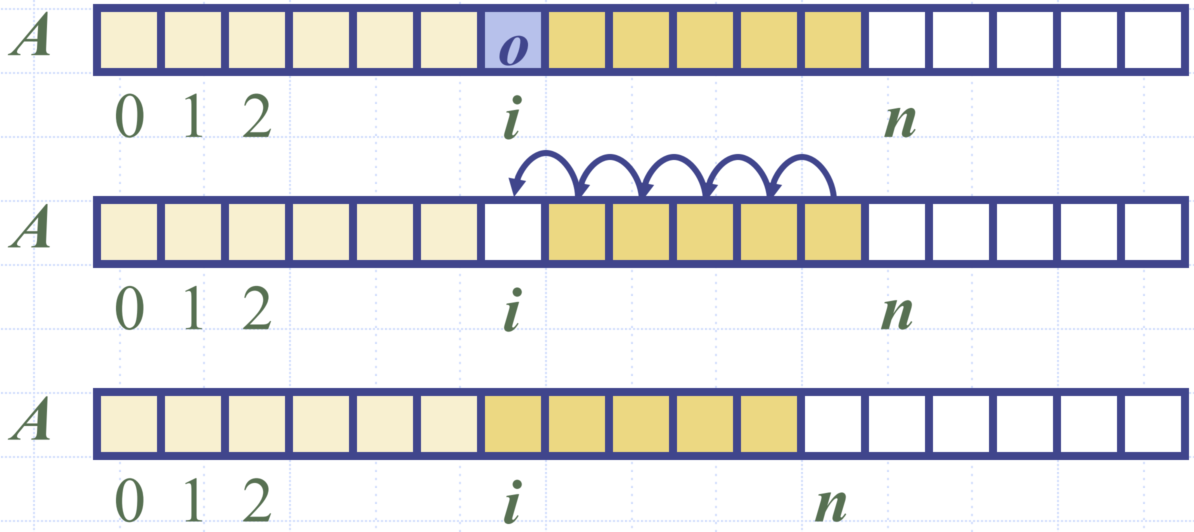
Insertion

- In an operation $add(i, o)$, we need to make room for the new element by shifting forward the $n - i$ elements $A[i], \ldots, A[n-1]$
- In the worst case ($i = 0$), this takes $O(n)$ time



Element Removal

- In an operation *remove*(i), we need to fill the hole left by the removed element by shifting backward the $n - i - 1$ elements $A[i + 1], \ldots, A[n - 1]$
- In the worst case ($i = 0$), this takes $O(n)$ time



Performance

- In an array-based implementation of a dynamic list:
  - The space used by the data structure is $O(n)$
  - Indexing the element at I takes $O(1)$ time
  - ***add*** and ***remove*** run in $O(n)$ time in worst case
- In an ***add*** operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one…

Growable Array-based Array List

- In an add(o) operation (without an index), we could always add at the end
- When the array is full, we replace the array with a larger one
- How large should the new array be?
  - Incremental strategy: increase the size by a constant c
  - Doubling strategy: double the size

Algorithm $add(o)$
  if $t = \text{len}(S) - 1$ **then**
    $A \leftarrow$ **new array of size …**
    for $i \leftarrow 0$ **to** $n-1$ **do**
      $A[i] \leftarrow S[i]$
    $S \leftarrow A$
  $n \leftarrow n + 1$
  $S[n-1] \leftarrow o$

Comparison of the Strategies

- We compare the incremental strategy and the doubling strategy by analyzing the total time $T(n)$ needed to perform a series of n add(o) operations
- We assume that we start with an empty array represented by an array of size 1
- We call amortized time of an add operation the average time taken by an add over the series of operations, i.e., $T(n)/n$

Incremental Strategy Analysis

- We replace the array $k = n/c$ times
- The total time $T(n)$ of a series of n add operations is proportional to

$$n + c + 2c + 3c + 4c + \ldots + kc =$$
$$n + c(1 + 2 + 3 + \ldots + k) =$$
$$n + ck(k + 1)/2$$

- Since c is a constant, $T(n)$ is $O(n + k^2)$, i.e., $O(n^2)$
- The amortized time of an add operation is $O(n)$

Doubling Strategy Analysis

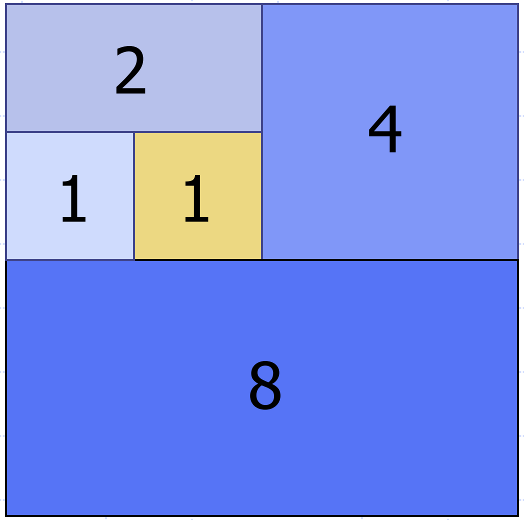
- We replace the array $k = \log_2 n$ times
- The total time $T(n)$ of a series of n add operations is proportional to

  $n + 1 + 2 + 4 + 8 + \ldots + 2^k =$

  $n + 2^{k+1} - 1 =$

  $3n - 1$
- $T(n)$ is $O(n)$
- The amortized time of an add operation is $O(1)$

geometric series



Insertion Sort - Example

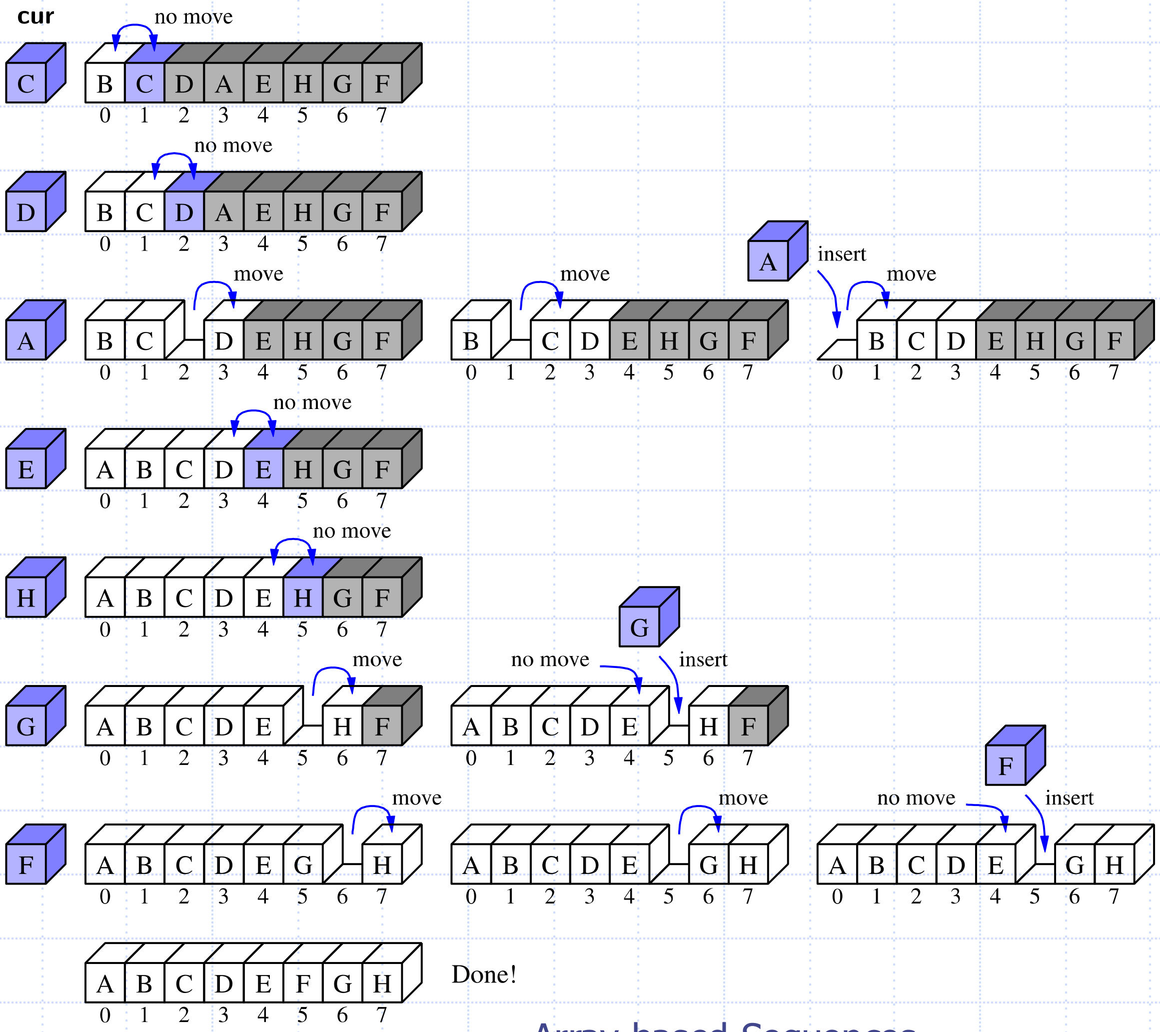


Figure 5.20

Sorting an Array – Insertion Sort

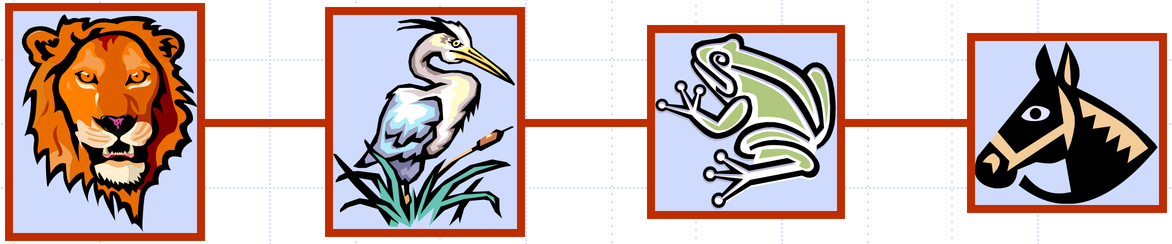
```python
def insertion_sort(A):
  """Sort list of comparable elements into nondecreasing order."""
  for k in range(1, len(A)):            # from 1 to n-1
    cur = A[k]                          # current element to be inserted
    j = k                               # find correct index j for current
    while j > 0 and A[j-1] > cur:       # element A[j-1] must be after current
      A[j] = A[j-1]
      j -= 1
    A[j] = cur                          # cur is now in the right place
```

Code Fragment 5.10

Drawbacks of Arrays

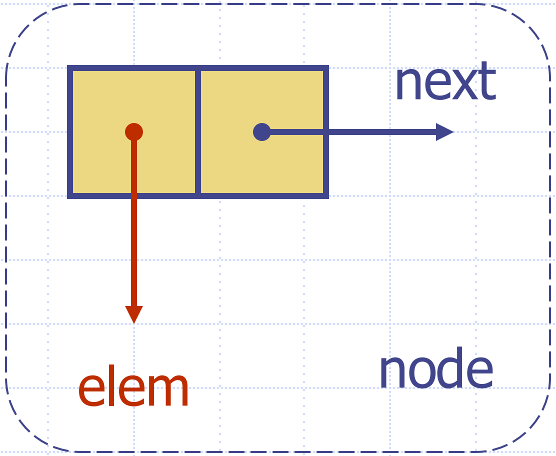
- Fixed capacity
- Insertions and Deletions involve many shifting operations

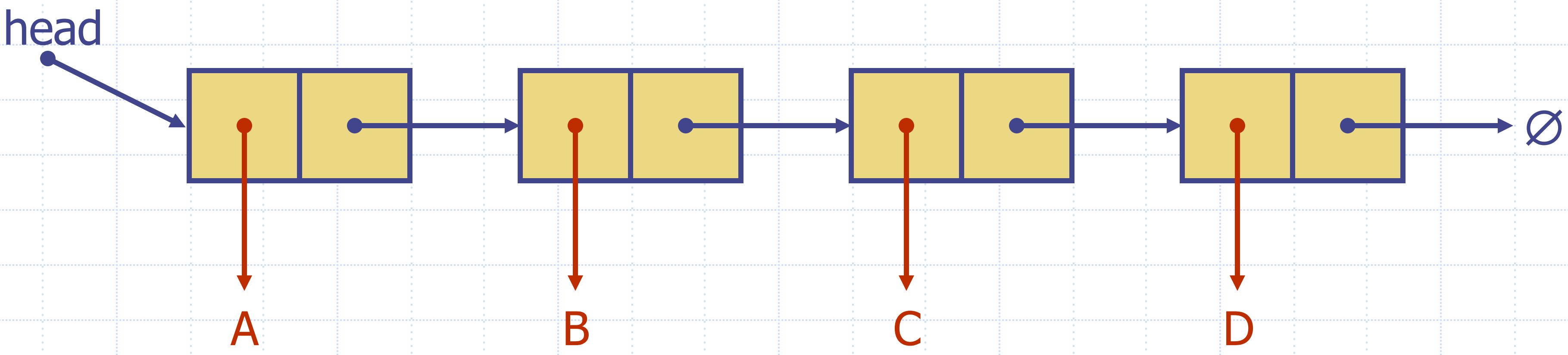
Singly Linked Lists



Singly Linked List

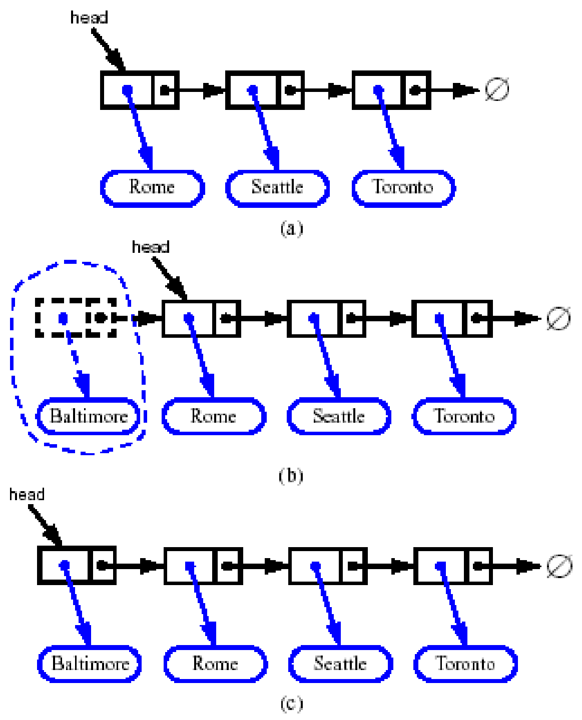
- A singly linked list is a concrete data structure consisting of a sequence of nodes, starting from a head pointer
- Each node stores
  - element
  - link to the next node





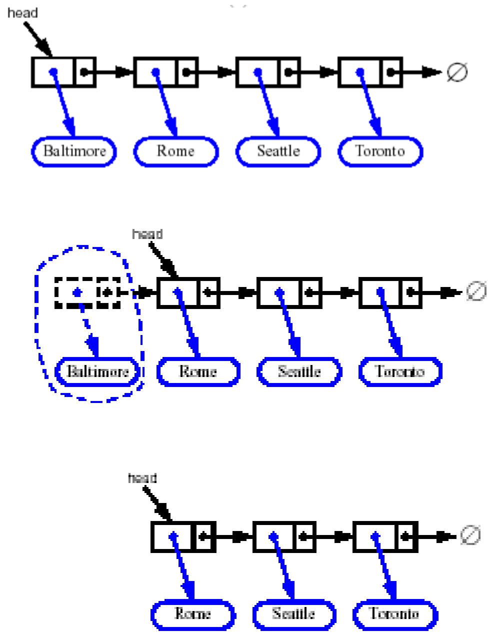
Inserting at the Head

1. Allocate a new node
2. Insert new element
3. Have new node point to old head
4. Update head to point to new node



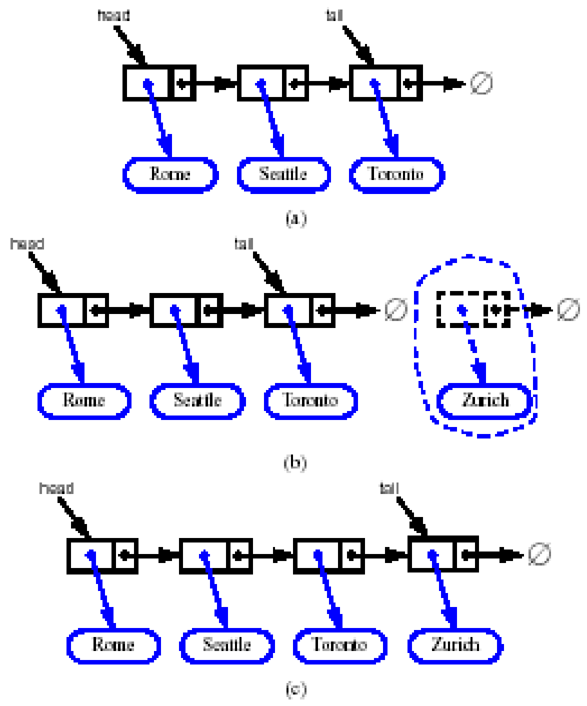
Removing at the Head

1. Update head to point to next node in the list
2. Allow garbage collector to reclaim the former first node



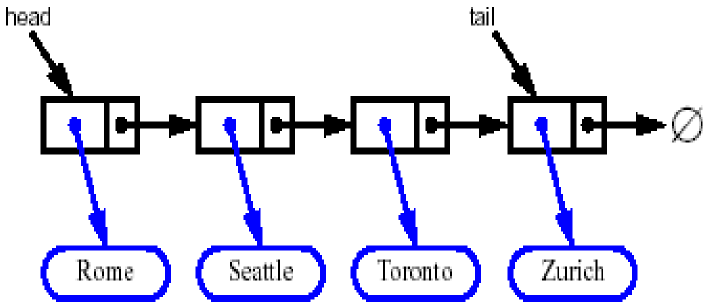
Inserting at the Tail

1. Allocate a new node
2. Insert new element
3. Have new node point to null
4. Have old last node point to new node
5. Update tail to point to new node

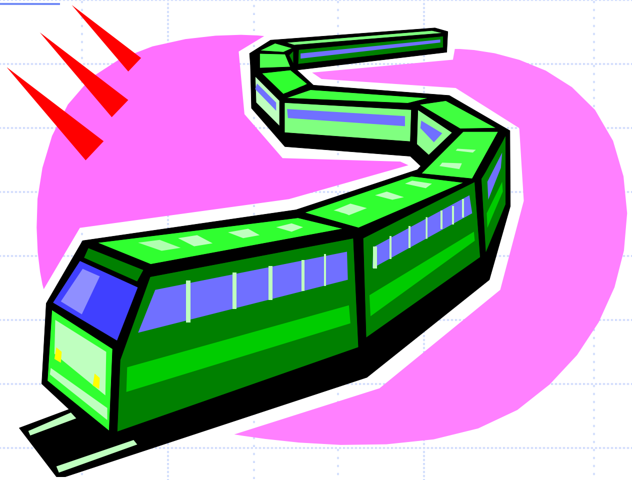


Removing at the Tail

- Removing at the tail of a singly linked list is not efficient!
- There is no constant-time way to update the tail to point to the previous node

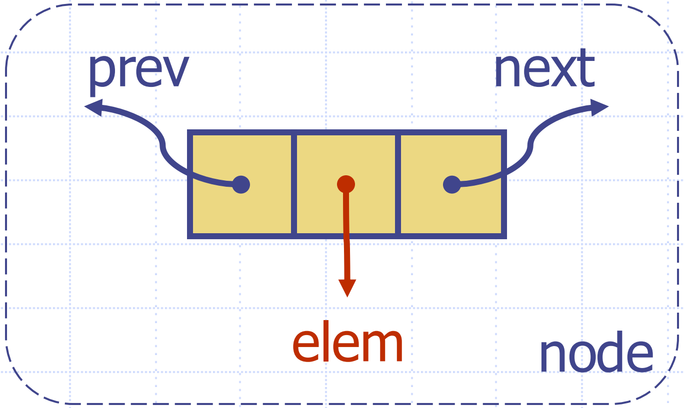


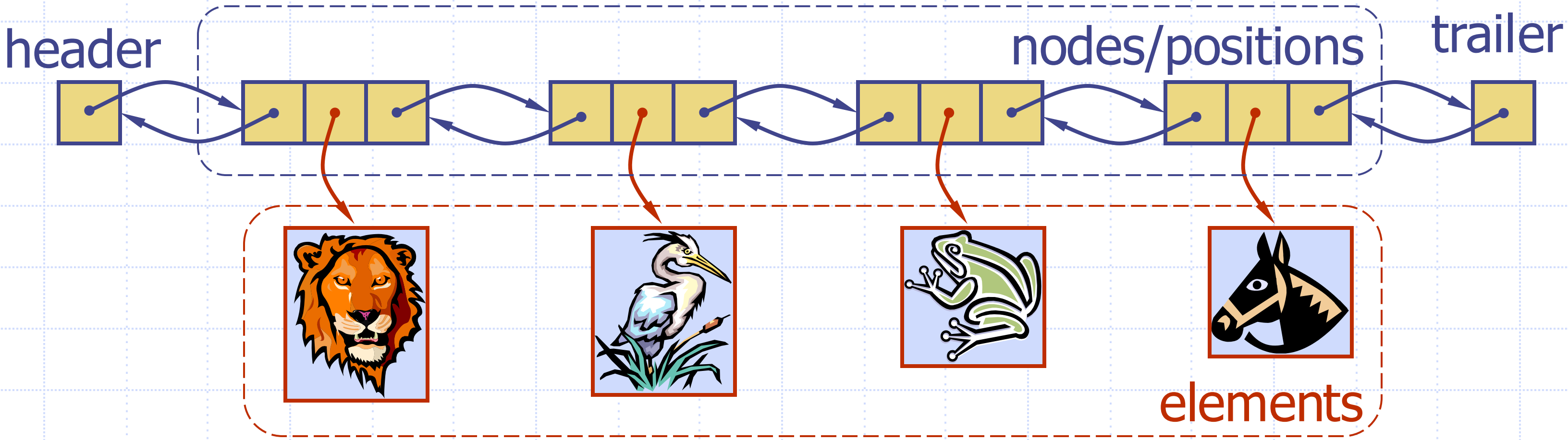
Doubly Linked Lists



Doubly Linked List

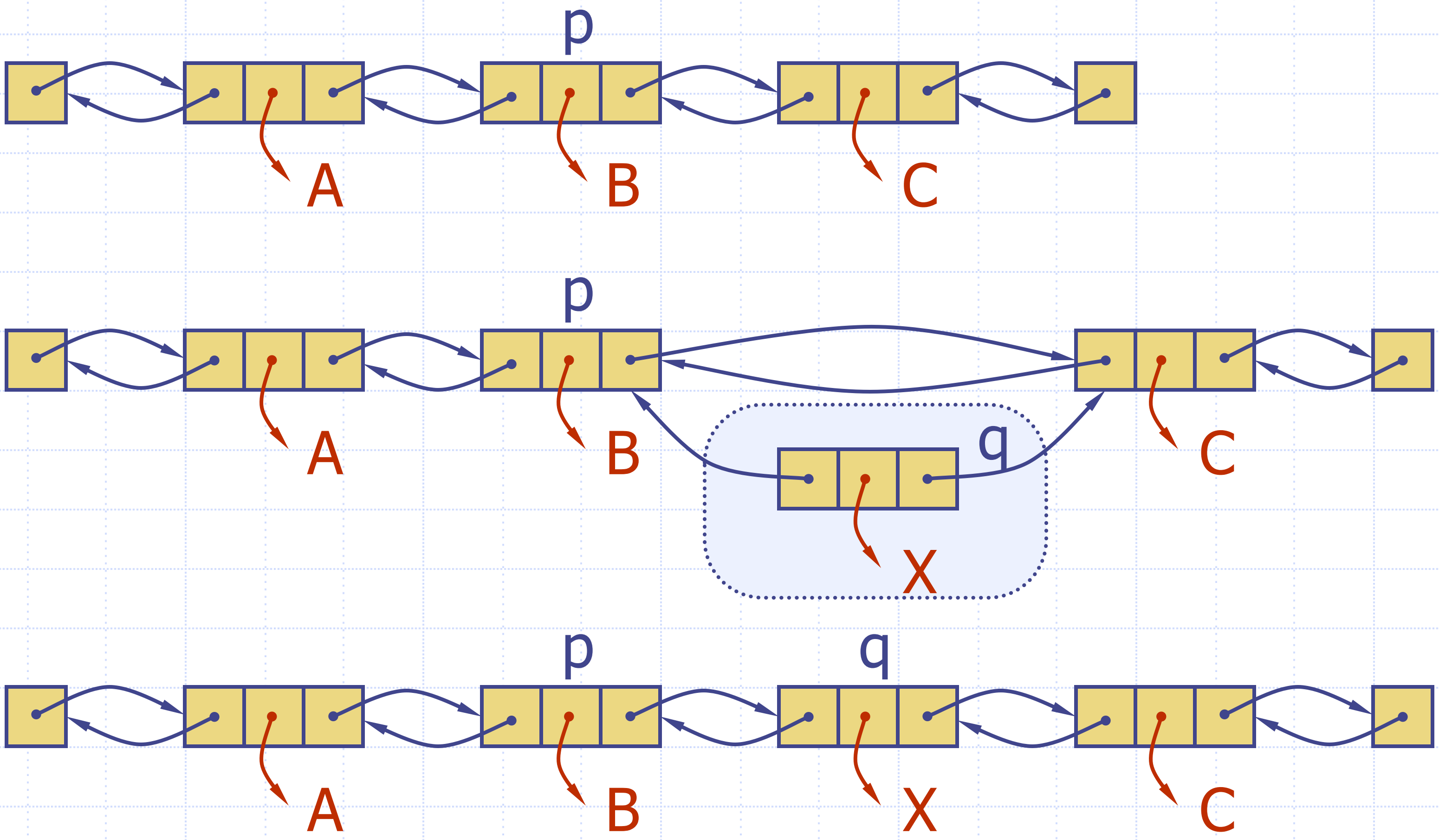
- A doubly linked list provides a natural implementation of the Node List ADT
- Nodes implement Position and store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes





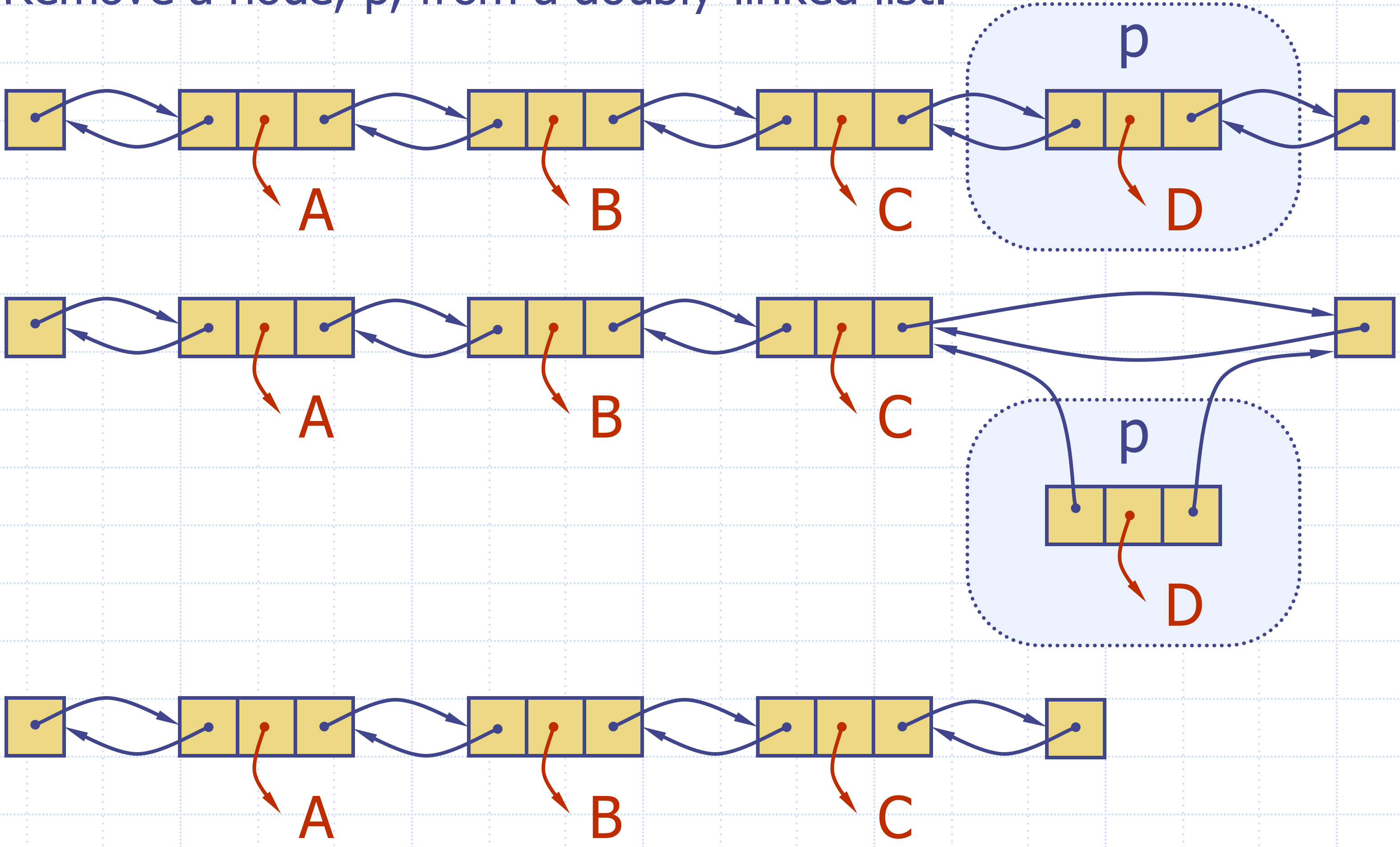
Insertion

- Insert a new node, q, between p and its successor.



Deletion

- Remove a node, p, from a doubly-linked list.



Performance

- In a doubly linked list
  - The space used by a list with n elements is $O(n)$
  - The space used by each position of the list is $O(1)$
  - All the standard operations of a list run in $O(1)$ time