

A collection of historical artifacts is arranged on a light-colored, textured surface. In the top left, a portion of a wooden chessboard with a checkered pattern and several chess pieces is visible. Below the chessboard, there are two medals. The top medal features a red ribbon with a circular rosette and a silver star-shaped pendant with a central emblem. The bottom medal has a blue ribbon with a circular rosette and a silver star-shaped pendant with a central emblem. To the right of the medals, a pair of thin-framed, round-rimmed glasses with a coiled spring bridge lies diagonally. In the bottom left corner, a circular compass with a white face and black markings is partially visible. The text "Midterm Review -1" is overlaid on the right side of the image in a large, black, serif font. The name "Yan Yan" is printed in a smaller, black, serif font in the bottom right corner.

Midterm Review -1

Yan Yan



Contents

1. C review -- pointers
2. Complexity
 1. Calculations of Big-O, Big Omega, & Theta
 2. Sorting algorithms
3. Linked list



Pointers: what for?

- ◆ A pointer is defined by its type and holds a value
- ◆ value: indicates where in memory the pointer refers to
 - always a memory address
- ◆ type: indicates what in memory the pointer refers to
 - almost always indicates the object's size – there are two exceptions:
 - Pointer to void
 - Pointer to function

Pointers Examples

int x = 4;

x: 4

int *a = &x;

a: &x → x: 4

int **b = &a;

b: &a → a: &x → x: 4

x	4	a	addr(x)	b	addr(a)
&x	addr(x)	&a	addr(a)	&b	addr(b)
*x	illegal	*a	4	*b	addr(x)
*(&x)	4	**a	illegal	**b	4
				***b	illegal

*b == a == &x
 **b == *a == x



Pointers and Arrays

- ◆ Array name is a pointer as well.
- ◆ In an array X , element index i
 - Address: $\&X[i]$ or $(X+i)$
 - Value: $X[i]$ or $*(X+i)$
- ◆ First element address is the base address of the array
 - E.g. X or $\&X[0]$
- ◆ Increment of Array name is illegal
 - E.g. $X++$ // illegal

```
int *pX = X;  
pX++; // legal
```



Analyzing Algorithm -- Operations

- ◆ **Best Case $B(n)$:**
 - constraints on the input, other than size, resulting in the fastest possible running time.
- ◆ **Worst Case $W(n)$:**
 - constraints on the input, other than size, resulting in the slowest possible running time.
- ◆ **Average Case $A(n)$:**
 - average running time over every possible type of input (usually involves the probabilities of different types of input).



Analyzing Algorithm -- Operations

Some examples – what's the total operations?

► $x = x + 1;$

Constant

► for ($i = 1; i \leq n; i++$)

$x = x + 1;$

Linear Loop

► for ($i = 1; i \leq n; i++$)

for ($j = 1; j \leq n; j++$)

$x = x + 1;$

Nested Loop (Quadratic)



Order of Growth

- ◆ Sub-linear, Linear, Polynomial and Exponential

$$1 < \log n < n < n \log n < n^2 < n^3 < 2^n < n!$$



Big-O

◆ Definition

Let f and g be two functions $f, g : N \rightarrow R^+$.

We say that $f(n) \in O(g(n))$

if $\exists c \in R^+$ and $n_0 \in N$ such that for every integer $n \geq n_0$, $f(n) \leq cg(n)$.



Omega Notation (Ω)

◆ Definition

Let f and g be two functions $f, g : N \rightarrow R^+$.

We say that $f(n) \in \Omega(g(n))$

if $\exists c \in R^+$ and $n_0 \in N$ such that for every integer $n \geq n_0$, $f(n) \geq cg(n)$.



Theta Notation (Θ)

◆ Definition

Let f and g be two functions $f, g : N \rightarrow R^+$.

We say that $f(n) \in \Theta(g(n))$

if $f \in \Omega(g)$ and $f \in O(g)$

(or $\exists c_1, c_2 \in R^+$ and $n_0 \in N$, $c_1g(n) \leq f(n) \leq c_2g(n)$, $\forall n \geq n_0$)



Little-o

◆ Definition

Let f and g be two functions $f, g : N \rightarrow R^+$.

We say that $f(n) \in o(g(n))$

if $\exists c \in R^+$ and $n_0 \in N$ such that **for any $c > 0$, $n_0 > 0$,**

$f(n) \leq cg(n)$, for every integer $n \geq n_0$



Little Omega Notation (ϖ)

◆ Definition

Let f and g be two functions $f, g : N \rightarrow R^+$.

We say that $f(n) \in \varpi(g(n))$

if $\exists c \in R^+$ and $n_0 \in N$ such that **for any $c > 0$, $n_0 > 0$,**

$f(n) \geq cg(n)$, for every integer $n \geq n_0$



Sample Long Answer Questions

- ◆ Show that $2n^2 + n = \Theta(n^2)$
- ◆ Show that $2n = \Omega(n)$
- ◆ Questions and examples in the course slides and A1



Sample Long Answer Questions

- ◆ Describe the difference between big-O and little-o



Sorting

- ◆ **Sorting**: Rearranging the values in an array or collection into a specific order (usually into their "natural ordering").
 - one of the fundamental problems in computer science
- ◆ Input: A sequence of n objects
 - ◆ $s = \langle a_1, a_2, \dots, a_n \rangle$
- ◆ Output: A permutation (reordering) $\langle a'_1, a'_2, \dots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.



Sorting Algorithms

- Bubble sort
- Insertion sort
- Selection sort

} Implement and write
the pseudocode/steps

- Merger sort
- Quick sort

} Describe the process



Sorting Algorithms

- ◆ **bubble sort:** swap adjacent pairs that are out of order
- ◆ **selection sort:** look for the smallest element, move to front
- ◆ **insertion sort:** build an increasingly large sorted front portion
- ◆ **merge sort:** recursively divide the array in half and sort it
- ◆ **quick sort:** recursively partition array based on a middle value



Insertion Sort Algorithm

- ◆ Uses only a fixed amount of storage that needed for the data:
- ◆ Pseudocode:

```
Algorithm: Insertion-Sort(A)
for j = 2 to A.length
    key = A[j]
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```



Bubble Sort Algorithm

- ◆ Uses only a fixed amount of storage that needed for the data:
- ◆ Pseudocode:

```
Algorithm: Sequential-Bubble-Sort (A)
for i ← 1 to length [A] do
  for j ← length [A] down-to i +1 do
    if A[j] < A[j-1] then
      Exchange A[j] ↔ A[j-1]
```




Selection Sort Algorithm

- ◆ Uses only a fixed amount of storage that needed for the data:
- ◆ Pseudocode:

```
Algorithm: Selection-Sort (A)
for i ← 1 to n-1 do
    min j ← i;
    min x ← A[i]
    for j ← i + 1 to n do
        if A[j] < min x then
            min j ← j
            min x ← A[j]
    A[min j] ← A [i]
    A[i] ← min x
```



Sorting Algorithms Complexity Summary

Algorithm	Time Complexity		Space Complexity
	Best	Worst	Worst
Bubble Sort	$\Omega(n)$	$O(n^2)$	$O(1)$
Selection Sort	$\Omega(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$\Omega(n)$	$O(n^2)$	$O(1)$
Merge Sort	$\Omega(n \log(n))$	$O(n \log(n))$	$O(n)$
Quick Sort	$\Omega(n \log(n))$	$O(n^2)$	$O(n)$

Sample Short Answer Questions

- ◆ Some sorting algorithms are applied to the following data [25, 23, 17, 5, 13]. After 2 passes, the rearrangement of the data are shown below. Match the data with the sorting algorithms used.

[17, 5, 13, 23, 25]

[17, 23, 25, 5, 13]

[5, 13, 17, 25, 23]

Insertion Sort

Selection Sort

Bubble Sort



Sample Short Answer Questions

- ◆ Some sorting algorithms are applied to the following data [25, 23, 17, 5, 13]. After 2 passes, the rearrangement of the data are shown below. Match the data with the sorting algorithms used.

[17, 5, 13, 23, 25]

Bubble Sort

[17, 23, 25, 5, 13]

Insertion Sort

[5, 13, 17, 25, 23]

24 Selection Sort



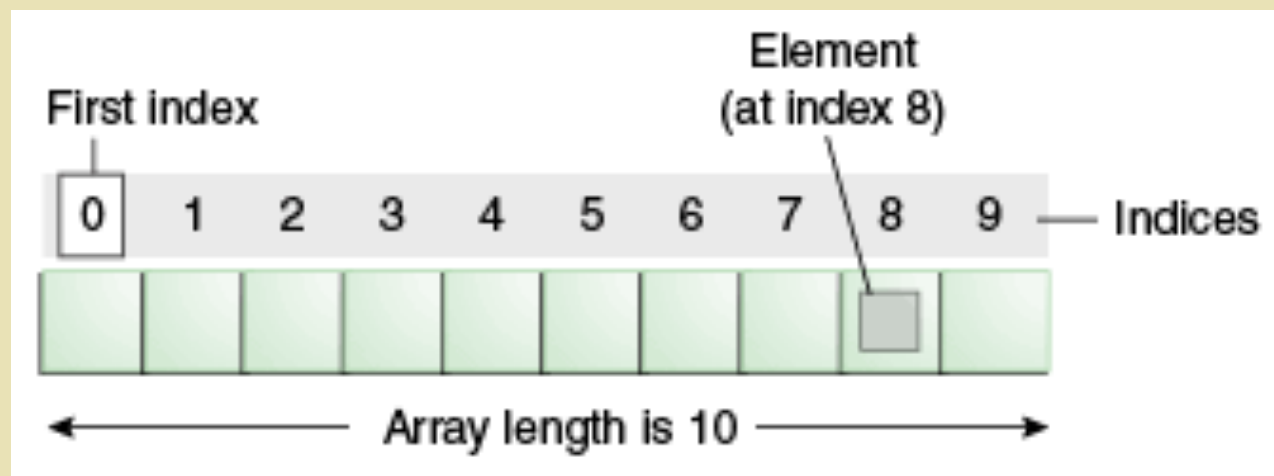
Sample Short Answer Questions

- ◆ In the **worst** case, **selection** sort has a time complexity of n^2 .
- ◆ In the **worst** case, **insertion** sort has a time complexity of n^2 . ^{best + n}
- ◆ In the **worst** case, **merge** sort has a time complexity of $n \log n$.
- ◆ In the **best** case, **insertion** sort has a time complexity of n .

List Implementations

◆ Array:

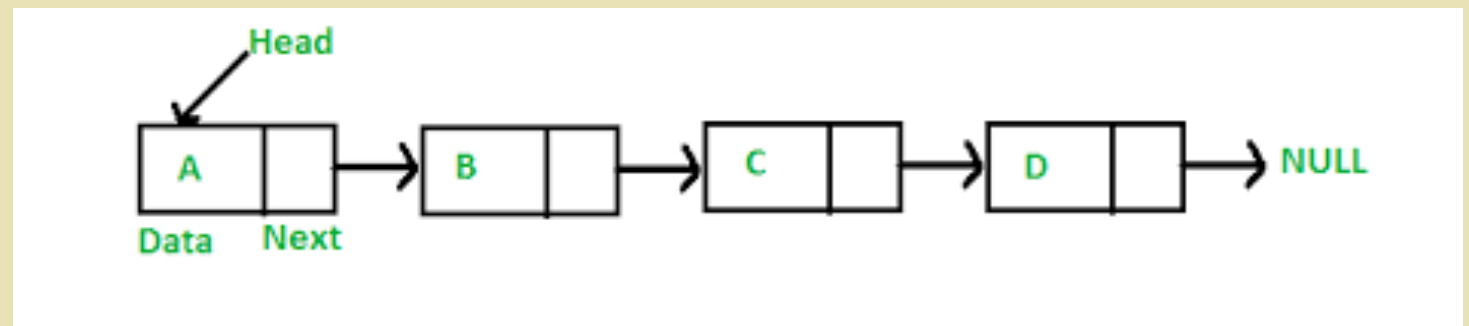
- a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created.



List Implementations

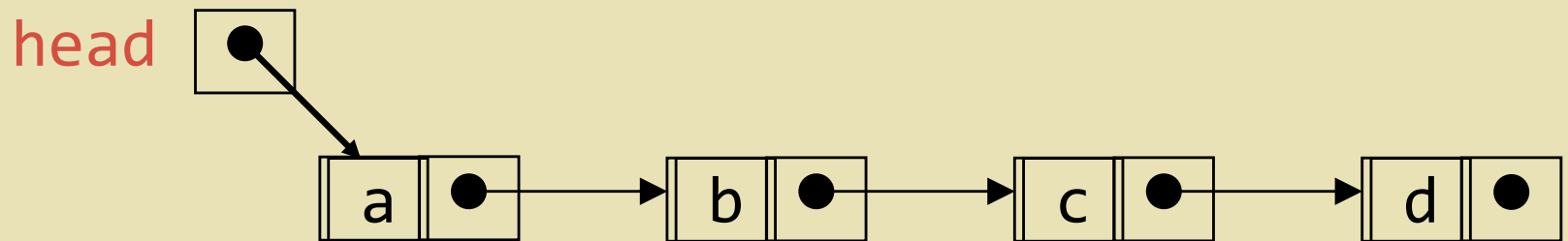
◆ Linked Lists

- data structure which can change during execution. Successive elements are connected by pointers. Last element points to NULL.



Anatomy of a linked list

- ◆ A linked list consists of:
 - A sequence of **nodes**



Each node contains a **value**
and a **link** (pointer or reference) to some other node
The last node contains a **null link**



More terminology

- ◆ A node's **successor** is the next node in the sequence
 - The last node has no successor
- ◆ A node's **predecessor** is the previous node in the sequence
 - The first node has no predecessor
- ◆ A list's **length** is the number of elements in it
 - A list may be **empty** (contain no elements)

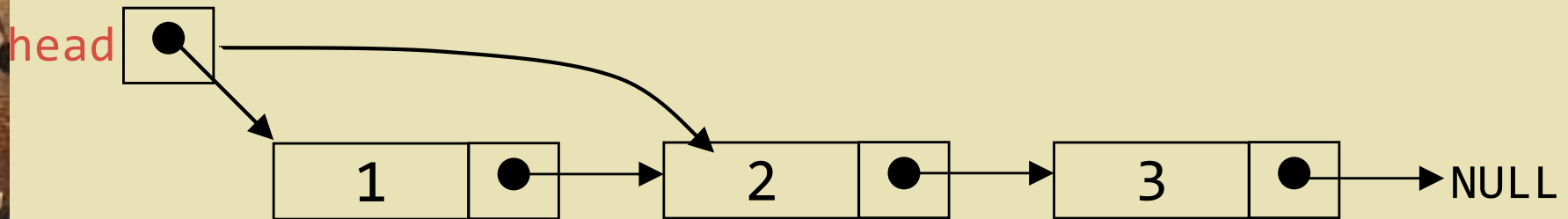


Inserting a node into a SLL

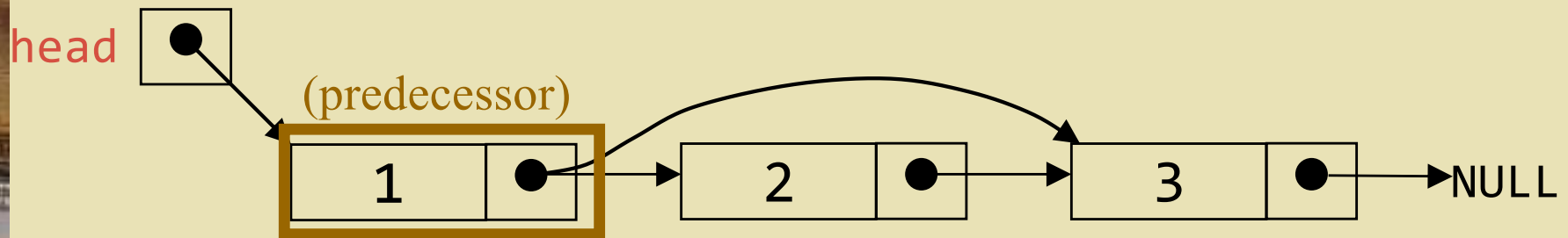
- ◆ Insert a new node into a list:
 - As the new first element
 - As the new last element
 - After a given node
 - After a given value
- ◆ Review the course slides to understand the implementation.

Deleting an element from a SLL

- To delete the first element, change the head



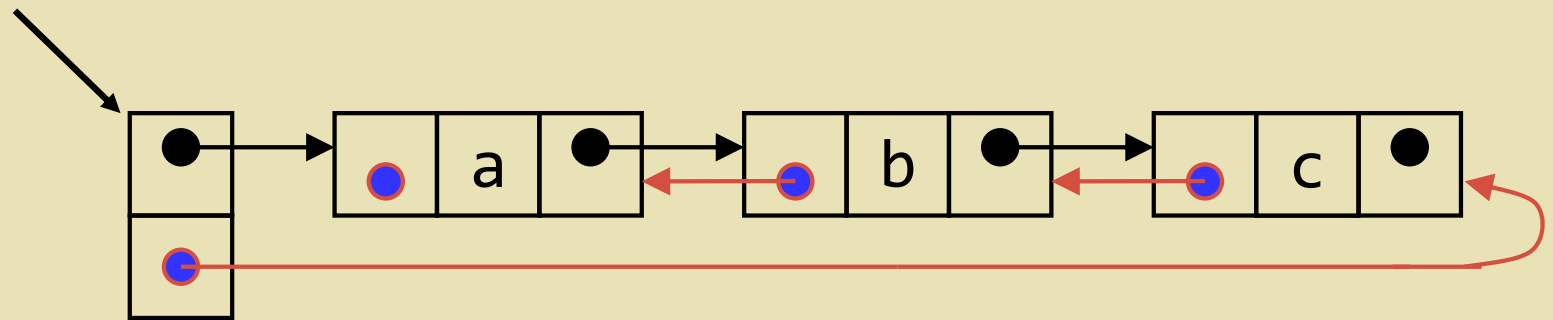
- To delete some other element, change the link in its predecessor



Doubly-linked lists

- ◆ Here is a doubly-linked list (DLL):

myDLL



- ◆ Each node contains a value, a link to its successor (if any), *and* a link to its predecessor (if any)
- ◆ The head points to the first node in the list *and* to the last node in the list (or contains null links if the list is empty). The two pointers are also called head and tail.
- ³²◆ DLL structure -- *two* pointers for each node



Doubly-linked lists

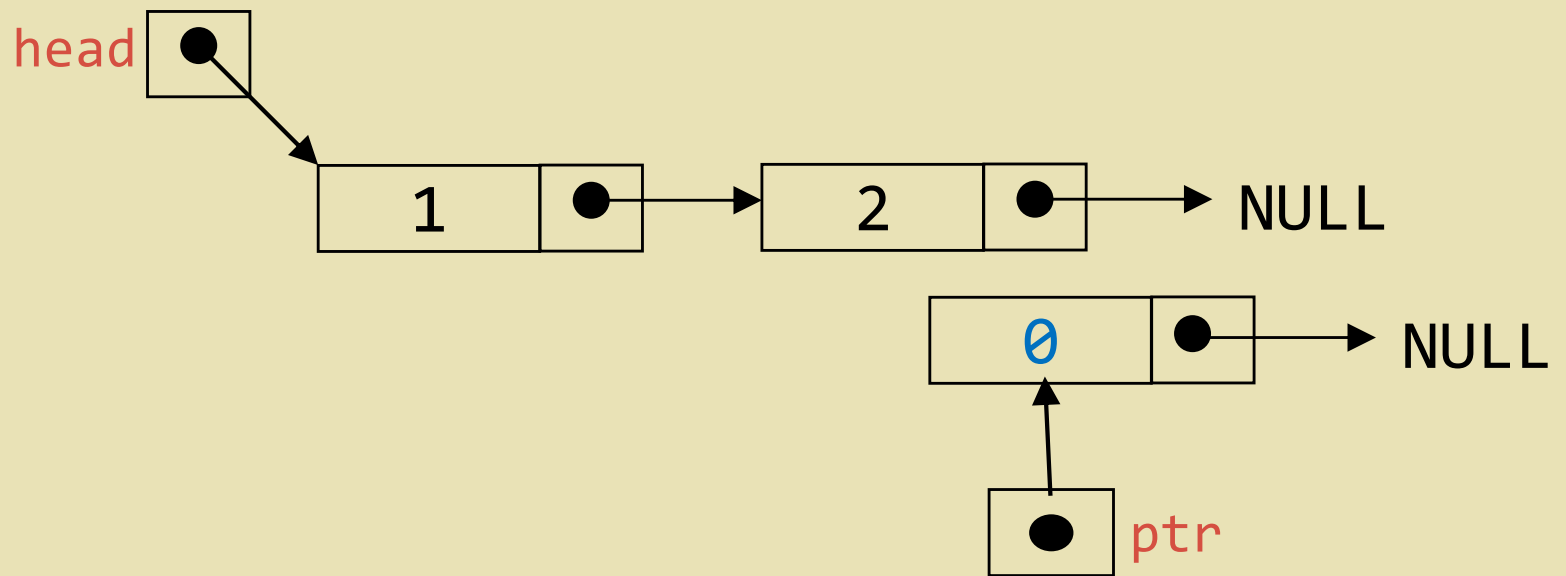
◆ Operations

- Append
- Prepend
- Traversal and Reverse Traversal

◆ Review course slides for examples

Sample Long Answer Questions

- ◆ Assume we have the following linked list created and a new node 0 created



- ◆ If you need to insert 0 before 1, how are the pointers will be updated?
- ◆ If you need to insert 0 after 2, how are the pointers will be updated?



Sample Long Answer Questions

- ◆ Write down one advantage and one disadvantage of doubly-linked list compared with singly-linked list.



More on Midterm Questions

- ◆ Please review ALL questions/examples/exercise in the lecture slides, labs, and A1.

