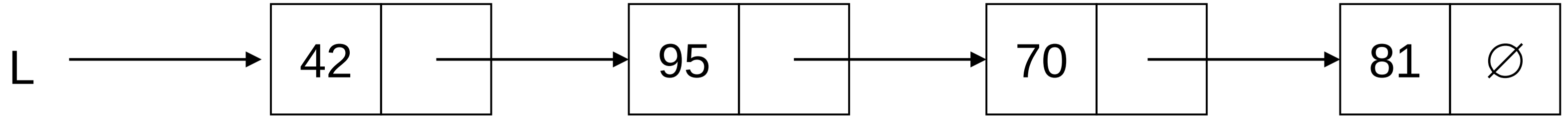
Discussion Week 2

Linked List

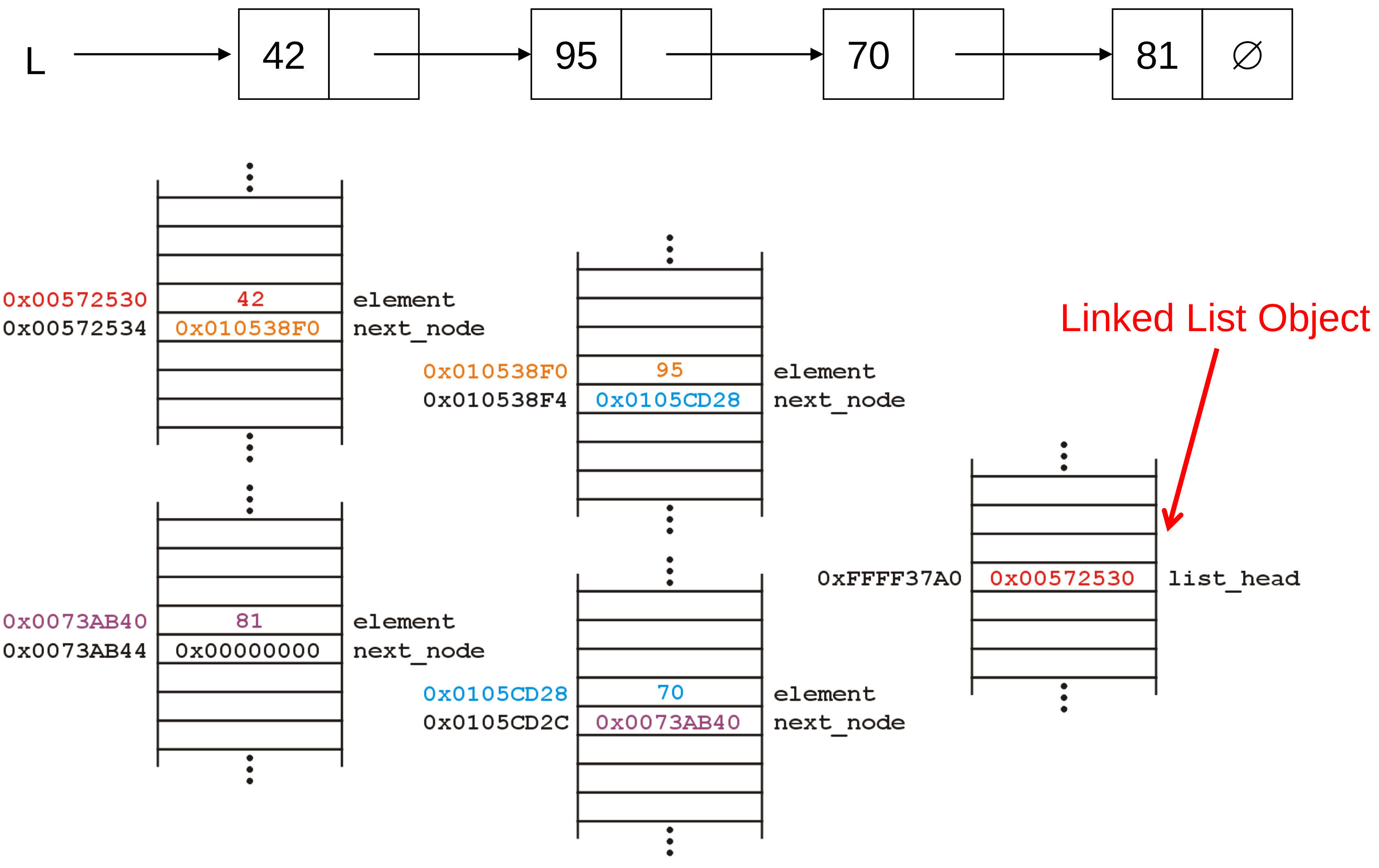
Definition

A linked list is a data structure where each object is stored in a *node*

As well as storing data, the node must also contain a reference/pointer to the node containing the next item of data



Definition



L
42
95
70
81
∅
0x00572530
42
element
0x00572534
0x010538F0
next_node
0x010538F0
95
element
0x010538F4
0x0105CD28
next_node
Linked List Object
0xFFFF37A0
0x00572530
list_head
0x0073AB40
81
element
0x0073AB44
0x00000000
next_node
0x0105CD28
70
element
0x0105CD2C
0x0073AB40
next_node

Node Class

The node must store data and a pointer:

```
class Node {
    private:
        int element;
        Node *next_node;
    public:
      … … …
};
```

Linked List Class

Because each node in a linked lists refers to the next, the linked list class need only link to the first node in the list

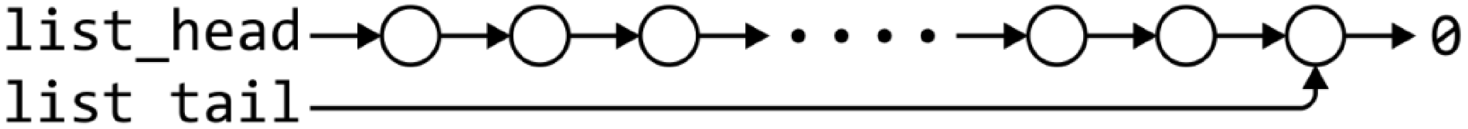
The linked list class requires member variable: a pointer to a node

```
class List {
    private:
        Node *list_head;
    // ...
};
```

Linked list

	Front/1st node	kth node	Back/nth node
Find	$\Theta(1)$	$O(n)$	$\Theta(1)$
Insert Before	$\Theta(1)$	$O(n)$	$\Theta(n)$
Insert After	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$
Replace	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$
Erase	$\Theta(1)$	$O(n)$	$\Theta(n)$
Next	$\Theta(1)$	$\Theta(1)^*$	n/a
Previous	n/a	$O(n)$	$\Theta(n)$

* These assume we have already accessed the kth entry—an $O(n)$ operation

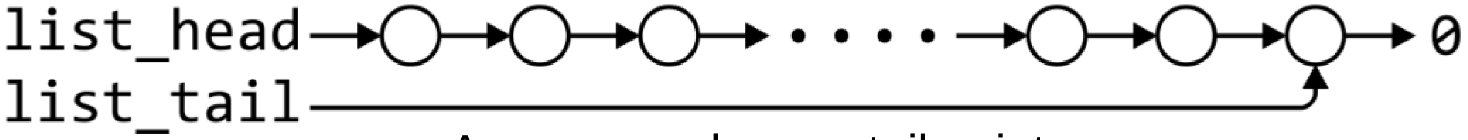


Assume we have a tail pointer

Linked list

	Front/1st node	kth node	Back/nth node
Find	$\Theta(1)$	$\mathrm{O}(n)$	$\Theta(1)$
Insert Before	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$
Insert After	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$
Replace	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$
Erase	$\Theta(1)$	$\Theta(1)^*$	$\Theta(n)$
Next	$\Theta(1)$	$\Theta(1)^*$	n/a
Previous	n/a	$\mathrm{O}(n)$	$\Theta(n)$

By replacing the value in the node in question, we can speed things up

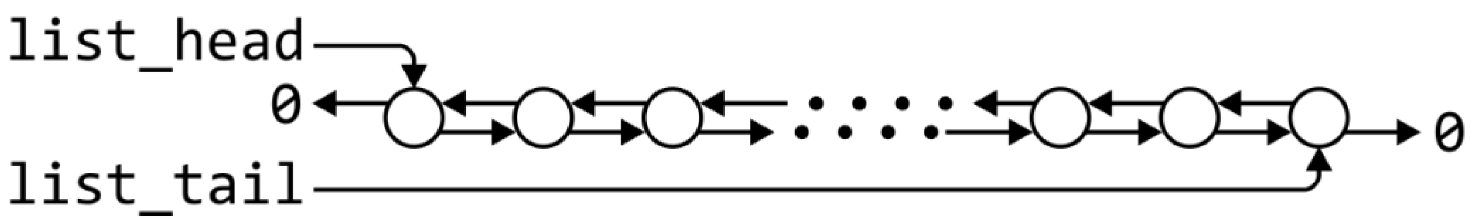


Assume we have a tail pointer

Doubly linked lists

	Front/1^{st} node	k^{th} node	Back/n^{th} node
Find	$\Theta(1)$	$O(n)$	$\Theta(1)$
Insert Before	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$
Insert After	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$
Replace	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$
Erase	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$
Next	$\Theta(1)$	$\Theta(1)^*$	n/a
Previous	n/a	$\Theta(1)^*$	$\Theta(1)$

* These assume we have already accessed the k^{th} entry—an $O(n)$ operation

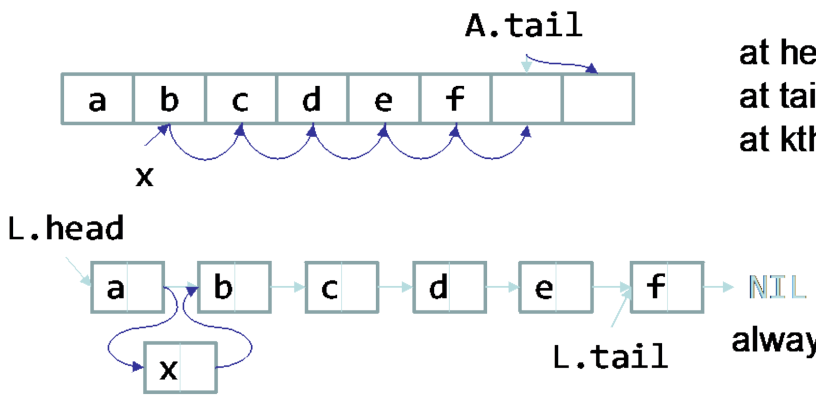


Linked List   v.s.   Array

Array

	Accessing the k^{th} entry	Insert or erase at the Front	Insert or erase at the k^{th} entry	Insert or erase at the Back
Array	$\Theta(1)$	$\Theta(n)$	$\mathrm{O}(n)$	$\Theta(1)$

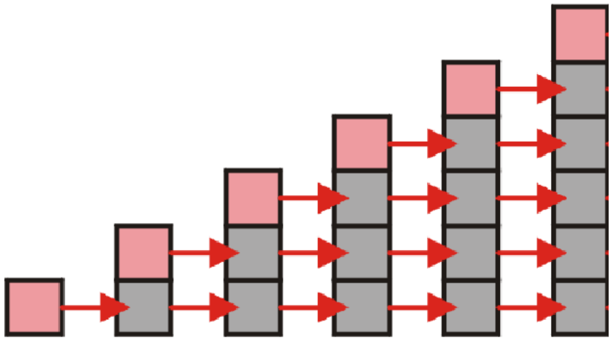
Insertion



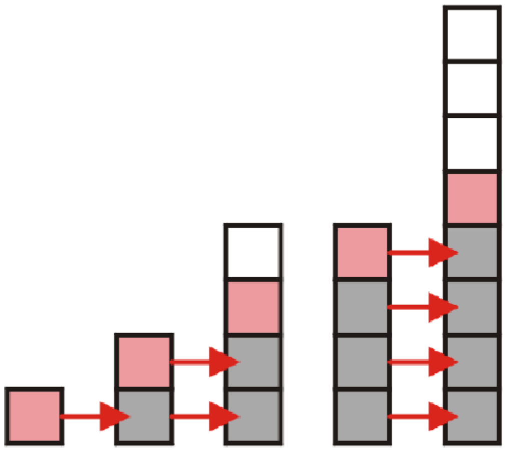
at head: O(n)
at tail: O(1)
at kth: O(n-k)

always O(1)* *assume that we have already access the kth element

Augmentation

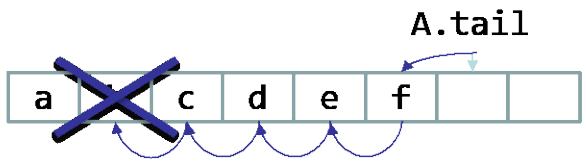


Linked List

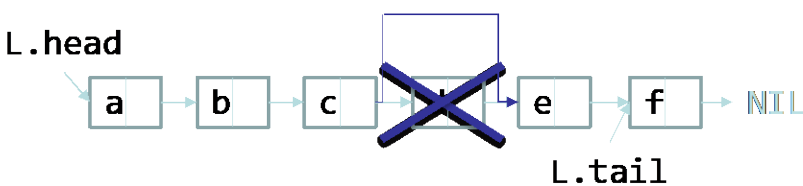


Array

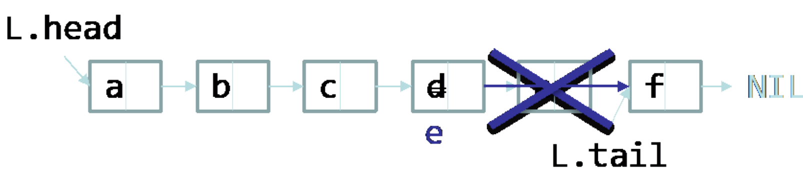
Deletion



O(n-k)

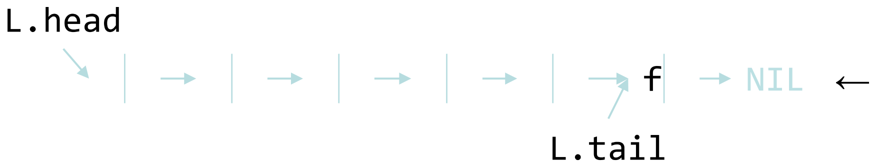


How to find the previous pointer?



a better solution

Search

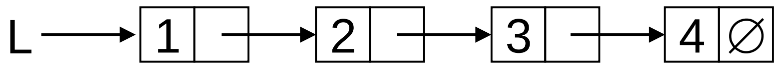


Reverse Linked List

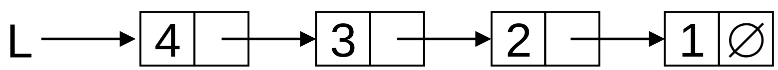
Reverse Linked List

Reference: https://www.geeksforgeeks.org/reverse-a-linked-list/

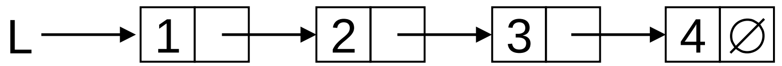
Input:



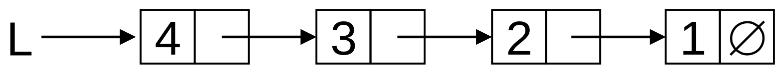
Output:



Input:



Output:



Tasks:

1. Reverse next pointer

2. Reset head

3. Last node point to NULL

Iterative Method

1. 3 Pointers:

```
prev = NULL;
curr = head;
next = NULL;
```

Tasks:

1. Reverse next pointer
2. Reset head
3. Last node point to NULL

Iterative Method

2. Iterate through the linked list:

```
// set next node
next = curr -> next;
```

Tasks:

1. Reverse next pointer

2. Reset head

3. Last node point to NULL

Iterative Method

2. Iterate through the linked list:

```
// set next node
next = curr -> next;

// reverse next pointer
curr -> next = prev;
```

Tasks:

1. Reverse next pointer
2. Reset head
3. Last node point to NULL

Iterative Method

2. Iterate through the linked list:

```
// set next node
next = curr -> next;

// reverse next pointer
curr -> next = prev;

// move prev &
// next one step forward
prev = curr;
curr = next;
```

Tasks:

1. Reverse next pointer
2. Reset head
3. Last node point to NULL

Iterative Method

3. Out of loop:

```
while (curr != NULL)
{
    …    …    …
}

// curr = NULL
// prev = original last node
head = prev;
```

Tasks:

1. Reverse next pointer
2. Reset head
3. Last node point to NULL

Iterative Method

2. Iterate through the linked list:

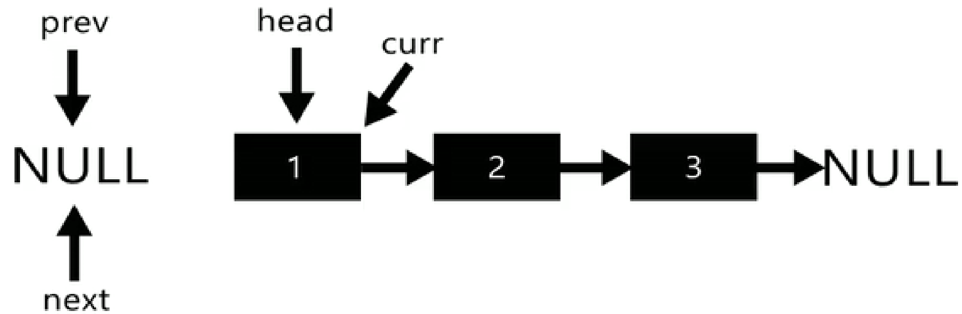
```
// the first iteration:
// curr: original first node
     last node after reversal
// prev: NULL
          // set next node
          next = curr -> next;

          // reverse next pointer
          curr -> next = prev;
```

Tasks:

1. Reverse next pointer
2. Reset head
3. Last node point to NULL

Iterative Method




```
while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
```

Iterative Method

Time Complexity: $O(n)$

Space Complexity: $O(1)$

[Optional] Recursive Method

1. Divide the linked list into 2 parts:

   the first node & the reset of the linked list

2. Call reverse for the reset of the linked list.

Recursive Method

```
Node* reverse(Node* head)
{
  if (head == NULL || head->next == NULL)
    return head;

  // reverse the rest list and put
  // the first element at the end
  Node* rest = reverse(head->next);
  head->next->next = head;

  // why do we need this line ?
  head->next = NULL;

  // fix the head pointer
  return rest;
}
```

Tasks:

1. Reverse next pointer
2. Reset head
3. Last node point to NULL

Recursive Method

```
Node* reverse(Node* head)
{
    if (head == NULL || head->next == NULL)
        return head;

    // reverse the rest list and put
    // the first element at the end
    Node* rest = reverse(head->next);
    head->next->next = head;

    // why do we need this line ?
    head->next = NULL;

    // fix the head pointer
    return rest;
}
```

Tasks:

1. Reverse next pointer
2. Reset head
3. Last node point to NULL

Recursive Method

```
Node* reverse(Node* head)
{
    if (head == NULL || head->next == NULL)
        return head;

    // reverse the rest list and put
    // the first element at the end
    Node* rest = reverse(head->next);
    head->next->next = head;

    // why do we need this line ?
    head->next = NULL;

    // fix the head pointer
    return rest;
}
```

Tasks:

1. Reverse next pointer
2. Reset head
3. Last node point to NULL

Recursive Method

```
Node* reverse(Node* head)
{
    if (head == NULL || head->next == NULL)
        return head;

    // reverse the rest list and put
    // the first element at the end
    Node* rest = reverse(head->next);
    head->next->next = head;

    // why do we need this line ?
    head->next = NULL;

    // fix the head pointer
    return rest;
}
```

Tasks:

1. Reverse next pointer
2. Reset head
3. Last node point to NULL

Course Info

Course Schedule

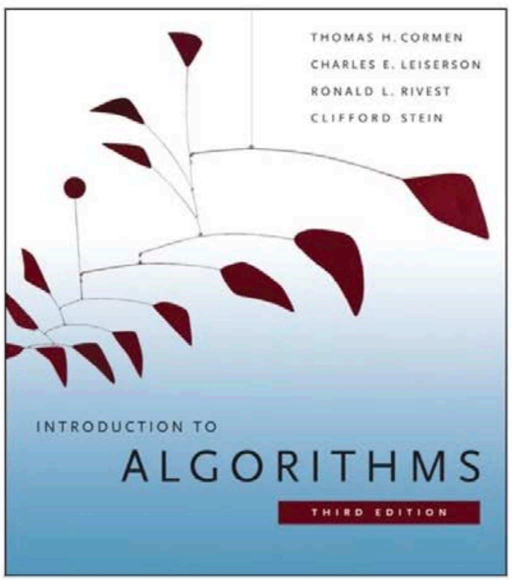
Week	Date	Content
1	Tue	Introduction
	Thu	Array and Lists
2	Tue	Stack and Queue
	Thu	Sorting: Insertion, Bubble
3	Tue	Big O/Theta/Omega
	Thu	Trees: Introduction, DFS, BFS
4	Tue	
	Thu	
5	Tue	Binary Trees
	Thu	Heap and Heap Sort
6	Tue	Binary Search Trees
	Thu	Balanced Binary Search Trees: AVL
7	Tue	Disjoint Sets
	Thu	Graphs: Intro, Traversal
8	Tue	Minimum Spanning Trees
	Thu	Topological Sorts
9	Tue	Shortest Path Alg: Dijkstra/A*
	Thu	Middle Term Exam

Course Schedule

10	Tue	Hash Table
	Thu	Divide and Conquer
11	Tue	Sorting: Merge
	Thu	Sorting: Quick
12	Tue	Median of Medians
	Thu	Greedy Algorithms I
13	Tue	Greedy Algorithms II
	Thu	Dynamic Programming I
14	Tue	Dynamic Programming II
	Thu	NP Completeness I
15	Tue	NP Completeness II
	Thu	Applications I
16	Tue	Applications II
	Thu	Review

Reference Book

- Introduction to Algorithms (3rd ed.). Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. MIT Press. ISBN 9780262033848.



Introduction to Algorithms, Third Edition

By Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein

The latest edition of the essential text and professional reference, with substantial new material on such topics as vEB trees, multithreaded algorithms, dynamic programming, and edge-based flow.

Reference Book

- Algorithm Design (1st ed.). Jon Kleinberg and Éva Tardos. Pearson Education. ISBN 0-321-29535-8.

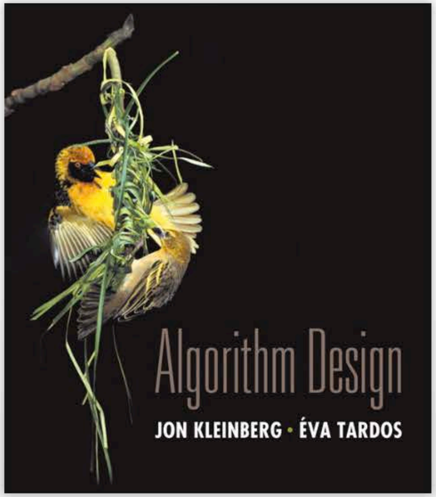
Algorithm Design

Jon Kleinberg, Cornell University

Éva Tardos, Cornell University

©2006 | Pearson | Available

Share this page



View larger

Grading

- Exams (45%): middle term: 25%; final: 20%
- Weekly Homework (20%): non-programming questions
- Programming Tasks (20%): 4-5 programming tasks (each lasts 3 weeks)
- In-Class Quizzes (15%): in lectures and discussions

Plagiarism

- All assignments must be done individually
  - You cannot copy directly from any other source
  - You cannot share solutions with any other students
  - Plagiarism detection software will be used on all the assignments

Plagiarism

- Punishment
  - When one student copied from another student, both students are responsible
  - Zero point on the assignment or exam in question
  - Disqualified from receiving any awards recommended by the school and from any competitive studying opportunities (e.g., international exchange)
  - Repeated violation will result in a F grade for this course as well as further punishment at the school/university level

Plagiarism Example

- Alex and Bob were roommates
- Bob let Alex use his laptop to complete an assignment
- Alex copied Bob’s solution for the assignment

Plagiarism Example

- Leslie asked if Morgan could send her his code so that she could look at it (promising, of course, not to copy it)
- Morgan sent the code
- Leslie copied it and handed it in

Plagiarism Example

- Garry and Harry worked together on a single source file initially and then worked separately to finish off the details
- The result was still noticeably similar with finger-print-like characteristics which left no doubt that some of the code had a common source

Plagiarism Example

- Jordan uploaded the projects to GITHUB.com without setting appropriate permissions. Kasey found this site, downloaded the projects and submitted them. Both are guilty.
  - This applies to any public forum, news group, etc., not just gitub.com…

Plagiarism Example

- Copied a peace of the codes from others or online repositories
- Copied someone’s solution from his/her USB drive
- Copied a peace of others’ codes and change all the variable/function names
- Unusual solutions appeared in different submissions

Quiz