Data Structures and Algorithms

- Chapter 0 – Review of Java Programming
- Chapter 1 – Software Engineering, Algorithmic Complexity
- Chapter 2- List ADT – Array and Linked Lists
- Chapter 3 – Stack ADT and Queue ADT
- Chapter 4 - Recursion
- Chapter 5 – Binary Trees and Binary Search Trees
- Chapter 6 – Heaps and Priority Queues
- Chapter 7 – Comparison Based Sorting: MergeSort
- Chapter 8 - Hashing

Chapter 1

Software Development

Chapter Objectives

- Discuss the goals of software development
- Identify various aspects of software quality
- Examine several development life cycle models
- Explore the notation of the Unified Modeling Language (UML)
- Examine issues related to error handling
- Introduce the concept of algorithm analysis
- Introduce ADT and Array ADT

Software Development

- *Software Engineering* is the study of the techniques and theory that support the development of high-quality software
- The focus is on controlling the development process to achieve consistently good results
- We want to:
  - satisfy the *client* – the person or organization who sponsors the development
  - meet the needs of the *user* – the people using the software for its intended purpose

Goals of Software Engineering

- Solve the right problem
  - more difficult than it might seem
  - client interaction is key
- Deliver a solution on time and under budget
  - there are always trade-offs
- Deliver a high-quality solution
  - beauty is in the eye of the beholder
  - we must consider the needs of various *stakeholders*

FIGURE 1.1
Aspects of software quality

Quality Characteristic	Description
Correctness	The degree to which software adheres to its specific requirements.
Reliability	The frequency and criticality of software failure.
Robustness	The degree to which erroneous situations are handled gracefully.
Usability	The ease with which users can learn and execute tasks within the software.
Maintainability	The ease with which changes can be made to the software.
Reusability	The ease with which software components can be reused in the development of other software systems.
Portability	The ease with which software components can be used in multiple computer environments.
Efficiency	The degree to which the software fulfills its purpose without wasting resources.

FIGURE 1.1 Aspects of software quality

Development Models

- A development life cycle defines a process to be followed during product development
- Many software development models have been introduced
- All of them address the following fundamental issues in one way or another:
  - problem analysis
  - design
  - implementation
  - evaluation

Problem Analysis

- We must specify the requirements of the software system
- Must be based on accurate information
- Various techniques:
  - discussions and negotiations with the client
  - modeling the problem structure and data flow
  - observation of client activities
  - analysis of existing solutions and systems

Design

- We must specify the solution
- You would not consider building a bridge without a design
- Design involves determining:
  - the overall software structure (architecture)
  - the key objects, classes, and their relationships
- Alternatives should be considered

Implementation

- We must turn the design into functional software
- Too many people consider this the primary act of software development
- May involve the reuse of existing software components

Evaluation

- We must verify that the system conforms to the requirements
- This includes, but goes way beyond, testing code with test cases
- It is possible to build a system that has no bugs and yet is completely wrong

Maintenance

- After a system is initially developed, it must be maintained
- This includes:
  - fixing errors
  - making enhancements to meet the changing needs of users
- The better the development effort, the easier the maintenance tasks will be

The Waterfall Model

- One of the earliest development models
- Each stage flows into the next
- Driven by documentation
- Advantages:
  - Lays out clear milestones and deliverables
  - Has high visibility – managers and clients can see the status
- Disadvantages:
  - late evaluation
  - not realistic in many situations

FIGURE 1.2 The waterfall software development model

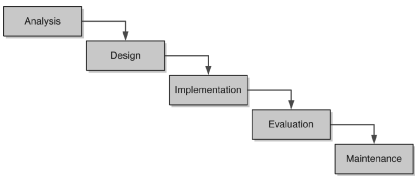


FIGURE 1.2 The waterfall software development model

The Unified Modelling Language

- The Unified Modelling Language (UML) has become a standard notation for software design
- It is unified in the sense that it is the synthesis of three separate notations
- It is language independent
- It includes various types of diagrams which use specific icons and notations
- However, it is flexible – the details you include in a given diagram depend on what you are trying to capture and communicate

UML Class Diagrams

- UML class diagrams may include:
  - The classes used in the system
  - The static relationships among classes
  - The attributes and operations of each class
  - The constraints on the connections among objects
- An *attribute* is class level data, including variables and constants
- An *operation* is essentially equivalent to a method
- May include visibility details

FIGURE 1.5 LibraryItem class diagram

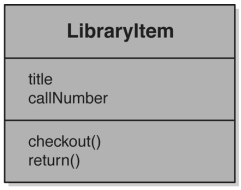


FIGURE 1.5 LibraryItem class diagram

FIGURE 1.6 A UML class diagram showing inheritance relationships

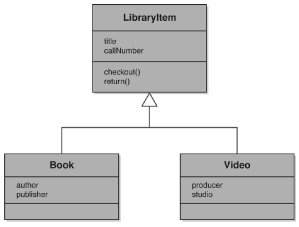


FIGURE 1.6 A UML class diagram showing inheritance relationships

FIGURE 1.7 A UML class diagram showing an association

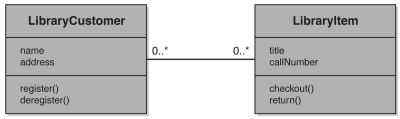


FIGURE 1.7 A UML class diagram showing an association

FIGURE 1.8 One class shown as an aggregate of other classes

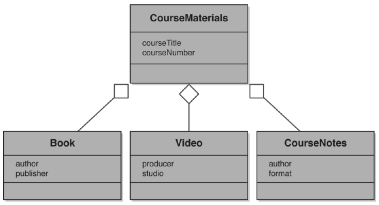


FIGURE 1.8 One class shown as an aggregate of other classes

FIGURE 1.9 A UML diagram showing a class implementing an interface

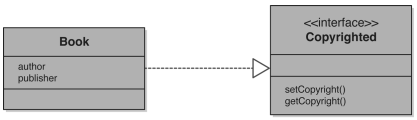


FIGURE 1.9 A UML diagram showing a class implementing an interface

FIGURE 1.10 One class indicating its use of another

FIGURE 1.10 One class indicating its use of another

Error Handling

- How problems are handled in a software system is a key design element
- In Java, an *error* generally represents an unrecoverable situation
- An *exception* is an unusual situation that might be handled in various ways
- Design questions include:
  - how are problems identified?
  - where are exceptions thrown and caught?

Data Structures

Data structure is a data type whose values can be decomposed into a set of component elements each of which is either atomic or another data structure.

Arrays – could be decomposed into whatever the arrays contain.

Vector – could be decomposed into elements.

Strings – could be decomposed into characters

Sets – could be decomposed into elements of the set.

Lists – could be decomposed into elements the list contains.

Data Structure Specification

Data structures are specified by:

a) Component data in the data structure with domain of each component

b) the operations that can be done on the data structure.

Data Structure

Data structures supports:

a) information hiding (similar to encapsulation)

b) abstraction

Algorithms

- Algorithm is a set of steps to solve a problem.
- Algorithms are independent of the programming language.

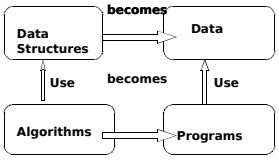
Here is an algorithm to find largest data element in an Array:

Step 1: Assume that the first item is the largest

Step 2: Look at the all the elements from the second element on, and then change the value of largest as needed

Step 3: Return the largest

Algorithm and Data Structures



Example of Algorithmic- Code

```
class ArrayFunctions{
 private int[] arr;

 public int findLargest(){

   int largeIndex;
   largeIndex = 0;

   for (int i =1; i< arr.length; i++){
        if (arr[largeIndex] < arr[i]) then begin
           largeIndex = i;
   }

   return(largeIndex);
 }
}}}e
d;
```


Execution of Algorithm

45	22	84	41	19	37	largeIndex = 0
45	22	84	41	19	37	
	i = 1					largeIndex = 0
45	22	84	41	19	37	
		i = 2				largeIndex = 2
45	22	84	41	19	37	
			i = 3			largeIndex = 2

Execution of Algorithm

45	22	84	41	19	37	
				i = 4		largeIndex = 2
45	22	84	41	19	37	
					i = 5	largeIndex = 2

findLargest returns 2

Time Complexity of Algorithms

- Time Complexity of an Algorithm is the time the algorithm takes to run.
- Since the algorithm consists of statements, we will compute the running time of an algorithm by computing the running time of each statement.

  Time for assignment statement is 1.

  Time for branching statement is time of the longest branch.

  Time for looping is (# of times the loop is executed times the time for the body of the loop).

  Time for comparison is 1.

  Time for function call is whatever time the function takes.
- Let us compute the time complexity of the algorithm for finding the larges element in array.

Time Complexity of Algorithms

```
class ArrayFunctions{
int[] arr;

public int findLargest(){

   int largeIndex;        --------------- 0 time
   largeIndex = 0;        --------------- 1 time

   for (int i =1; i< arr.length; i++){  ---- 2  time
        if (arr[largeIndex] < arr[i])   ----- 1 time
           largeIndex := i;             ------ 1 time
   }

   return(largeIndex);                  ---- 1 time
}}}end;
```


Time Complexity of Algorithms

Total Time = 1 + (n-1) *((body of the loop) +2)

= 1 + (n-1)* ((time for if statement) +2)

= 1 + (n-1) *((time for longest branch in the if statement) +2)

= 1 + (n-1) *(2 +2)

= 1 + (n-1)*(2 +2)

= 1+4n-4

= 4n-3

Time Complexity of the Algorithm = 4n-3

Dropping the constants and multiples

When computing time complexities we are only interested how the growth of the running time is, not the exact running time itself. This is called the asymptotic time complexity.

The functions n, n+1, 2n+1, etc grow at at the same rate. But n^2 grows much faster than 2n etc.

So, we drop the constants and multiples when talking about running times.

n, n+1, 2n+1, 3n+5 --- are same as n,

n^2, $3n^2+6$, $3n^2+6n + 5$ --- are all same as n^2

Time Complexity of the above Algorithm = O(n)

FIGURE 1.11 Some growth functions and their asymptotic complexity

Growth Function	Order
$t(n)=17$	$O(1)$
$t(n)=20n - 4$	$O(n)$
$t(n)=12n \log n + 100n$	$O(n \log n)$
$t(n)=3n^2 + 5n - 2$	$O(n^2)$
$t(n)=2^n + 18n^2 + 3n$	$O(2^n)$

FIGURE 1.11 Some growth functions and their asymptotic complexity

Growth Rate Table

Time	33n	46nlogn	$13n^2$	$3.4n^3$	2^n
n=10	.00033 sec	.0015sec	.0013sec	.0034sec	.001 sec
n=100	.003sec	.03sec	.13sec	3.4sec	$2*10^{24}$ hours
n=1000	.033sec	.45sec	13sec	.94hours	10^{282} years
n=10,000	.33sec	6.1sec	22min	39days	10^{3000} years
n=100,000	3.3sec	1.3sec	1.5days	108 years	$4*10^4$ centuries
Max size for 1min	1,800,000	82,000	2200	260	26

FIGURE 1.13 Comparison of typical growth functions

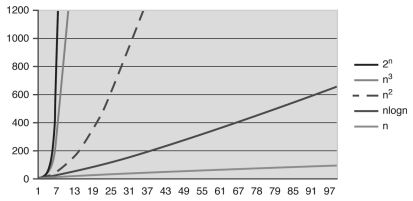


FIGURE 1.13 Comparison of typical growth functions

Algorithm to Search in an array

Given an array and a key we want to find the key in the array.

If the key is in the array, we will return the index of the key element, otherwise we will return -1

Algorithm to Search An Array

```
class ArrayFunctions{
 private int[] arr;

 public int search(int key){

   boolean found;
   found = false;
   int i =0;
   while ( i < arr.length && !found){
        if (key  ==  arr[i])
           found = true;
        else
           i = i +1;
  }
   if (found)
      return(i);
   else
      return(-1);
}
```


Time Complexity of Search

We cannot find the running time exactly since, the loop may exit the first step of the iteration, or after examining every element o the array.

So, we in this class we will use the worst case time which is denoted by O(..).

Binary Search

For performing binary search, we assume that the array we are searching is already sorted.

We then split the array in half and look at the element at the midpoint of the array.

We then compare the *key* with element at midpoint,

If the key is equal to element at midpoint, we are done!

If the key is less than the element at midpoint, then we move the left half of the array and repeat the search process.

If the key is bigger than the element at midpoint, then we move the right half of the array and repeat the search process.

Binary Search Example-1

```
Key :    72
Array :  23  52   72   101   200   305   451   672   921
```

```
low=0                    mid=3                    high=7
 23     52      72      101      200      305      451      672
```

Since 72 < 101, set high = mid-1 **found=false**

```
low=0 mid=1    high=2
23     52      72      101      200      305      451      672
                                                  found=false
```

Since 72 > 52, set low = mid +1

Binary Search Example-1

```
                low=2
                mid=2                                  found=true
                high =2
 23     52      72      101      200      305      451      672
```

Since mid has item equal to the key (74) stop

Since 72 is found, we stop and return mid which is 2

Binary Search Example-2

```
Key : 24
Array: 23  52   72   101   200   305   451   672
```

```
low=1                    mid=3                    high=7
 23     52      72      101      200      305      451      672
```

Since 24 < 101, set high = mid -1 **found=false**

```
        mid=1    high=2
low=0
 23     52      72      101      200      305      451      672
```

Since 24 < 52, set high = mid -1 **found=false**

Binary Search Example-2

```
high=0
mid =0
low =0
23     52      72      101      200      305      451      672
                                                  found=false
```

Since 24 > 23, set low = mid+1

```
high =0 low = 1
 23     52      72      101      200      305      451      672
                                                  found=false
```

STOP HIGH < LOW !!!

Binary Search Code

```
class ArrayFunctions{
  int[] arr;

  public int binarySearch(int lowIndex, int highIndex){
  int low= lowIndex, mid, high= highIndex;
  boolean found =false;

  while(!found && low <= high){
      mid = (low+high)/2;
      if (arr[mid] == key)
         found = true;
      else
         if (key < arr[mid])
            high = mid -1;
          else
             low = mid+1;
  }
    if (found)
        return(mid);
    else
        return(lowIndex-1);
}
```


Time Complexity of Binary Search

Let us analyse the worst case time complexity of binary search.

For 8 elements, the array sizes of search are
8, 4, 2, 1--- 4 steps

For 16 elements the array sizes of search are
16, 8, 4, 2, 1 --- 5 steps.

For 32 elements, the array sizes of search are
32, 16, 8, 4, 2, 1 – 6 steps.

What is the time complexity ???

Exercises

1. Explain the following terms in your own words:
   domain, component data element
   specification of an abstract data type
   information hiding
   data structure

2. What are the order-of-magnitude estimates for the following expressions:
   a. $4 \times n^2 + 10 \times n + 20$
   b. $6.1 \times \log n + n^{1/2} + 5.5$
   c. $7 \times n \times ((\log n)/(5 \times n^2)) + 3 \times n^{1/2}$
   d. $(5/n) + 0.33 \times n^{1/4}$

Exercises

3. Write an algorithm for each of the tasks below and find the worst case, best case and average case times for the following algorithms.
   a. Find if an array has repeated elements.
   b. Find the smallest element in an array.

Exercises

4. Write an algorithm to find the second largest element in an array.
   What is the running time of the algorithm?

5. List the following functions from lowest to highest order.
   n, 2^n, nlgn, lnn, $n\text{-}n^3+7n^5$, lgn, n^{-5}, e^n n^2 +lgn, n^2, 2^{n-1}, lglgn, n^3, $(\lg n)^2$, n!