2.9.3

Name	Meaning	Example	Result
+	Addition	34+1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

2.11

Vocabulary

Operator Precedence Rule: Multiplication, division and remainder operators are applies first.
left to right
Addition, Subtraction.

2.13

Operator	Name	Example	Equivalent
+=	Addition assignment	i += 8	i = i+8
-=	Subtraction assignment	i -= 8	i = i-8
*=	Multiplication assignment	i *= 8	i = i * 8
/=	Division assignment	i /= 8	i = i / 8
%=	Remainder assignment	i %= 8	i = i % 8

2.14

Operator	Name	Description	Example
++var	preincrement	Increment var by 1, and use the var in the statement.	int j = ++i; // j is 2, i is 2 Before.
var++	postincrement	Increment var by 1, but use the original var value in the statement.	int j = i++; // j is 1 i is 2 After
--var	predecrement		
var--	postdecrement		

Vocabulary

Increment operator: shorthand operators. → handy.

Decrement operator: increase by 1 / dec by 1

Post-increment: postfix increment

Pre-increment: prefix increment

~~Implicit~~

2.15

Vocabulary

Casting: an operation that converts a value of one data type into a value of another data type.

Widening of a type: Casting a type with a small range to a type with large range.

explicit casting

Narrowing of a type: large range to small range

implicit casting.

2.18

Error	Description
1 Undeclared / Uninitialized Vars and unused vars	
2 Integer Overflow	int value = 2147483647 + 1; // value will actually be -2147483648
3 Round-off Errors	1.0 - 0.9 = 0.0999999...98 , not 1 x 10 x 10/9 ÷ 10 = 0.1
4 Unintended Integer Division	int num1 = 1 num2 = 2; double x = (1+2)/2 = 1 double x = (1+2)/2.0 = 1.5;
5 Redundant Input Objects.	use only one input object. ~~Scanner input = new Scanner (System.in);~~ { Scanner input2 = ... }

redundant only need one.

aveat = warming

Name: Jim Li
Date: Sep 15th Block:

Identifying A Computer

Due date: September 5th, 2017 @ 23:59 on Schoology

Now that you know about the main types of hardware categories related to computing, you are going to disassemble a computer and label each part.

This will also end up as a studying material for your tests/exams.

You will be creating a document with a table as shown:

Part Name	Hardware Component Category	Picture	Description

You write down name parts, the category they belong to, and a description of that part and why/how it belongs to the category. You will also be taking pictures of the components and putting them in the document.

You will hand in your document on Schoology.

Marking Scheme

Correct Part Identification: /10

-all parts named correctly (abbreviations and long-form names)

Correct Component Category Identification: /10

-hardware which falls into the categories discussed are put in correctly

Component Description: /10

- purpose; accurate descriptions of what this particular part does in the role of the category

Pictures: /2

- clear, can identify which part is being shown

Formatting: /2

-table, good visual text formatting

English: /6

-proper grammar, good spelling, full sentences and ideas

Bonus – Labelling other hardware + 4

Total: /40 + 4

Date: 8.5 Block: ____

Binary

Relation to Computer Memory

- Recall that memory is represented by switches
- Either on (1) or off (0)
- We can represent each switch (bit) with a number, either 0 or 1
- We can calculate like the CPU does (by bytes)

Understanding Decimal

3 6 1

$3\times100 + 6\times10 + 1$

$3\times10^2 + 6\times10^1 + 1\times10^0$

Understanding Binary

2^2 2^1 2^0

1 0 1

Conversion to Decimal

2^2 2^1 2^0

1 0 1

$1\times2^2 \quad 0\times2^1 + 1\times2^0$

$= 5$

Conversion to Binary

3 6 1

2 | 361 ... 1

2 | 180 ... 0

2 | 90 ... 0

2 | 45 ... R1

2 | 22 ... 0

2 | 11 ... R1

2 | 5 ... R1

2 | 2 ... 0

1 R1.

101101001

Name: JIMLI
Date: 9.6.

Languages

Evolution of Language

- Computer language has developed over the years

1. Machine Language
   - Oldest computer language
   - 0s and 1s
   - Language that the computer can understand without help
   - Since computers are different, machine language is specific to the machine
   - Examples: 0 or 1
2. Assembly language (Assembly Language)
   - Older language
   - Short commands
   - Acts as direct translation to commands in machine language
   - Since translation of machine language, is specific to the machine
   - Examples: Spare, RAM
3. High level languages
   - ☆English-like
   - Instructions called statements
   - Interpreters or compilers take the code and translate it to machine language
   - Typically can be used in different computers with different compilers
   - Examples: Java, Python

Compiled Vs. Interpreted

	Compiled	Interpreted
Definition	When language is compiled, is expressed in language of the machine	Language is read by another program to be run
Diagram	language → compiled → Machine code → Ready to Run	language → Ready to Run → interpreted → virtual Machine → Machine Code (IML)
Pros	-Faster running by using native language of the machine -opportunity to apply optimizations while compiling 最佳化	-easier to run (can run on any machine) -can execute code on the fly -easier to make (compilers are hard to make)

Operating Systems

- Manages and controls computer's activities
- 3 main tasks:
  - Control and monitor Computer system (监控)
  - ~~Manage resources~~ (RAM, HD)
  - Scheduling operations (when to send commands to CPU) (调度 Scheduling)

How Java Works

- Both compiled and interpreted
  - Compiles into bytecode
    - Is interpreted by the Java Virtual Machine (JVM) into machine language
- Java Development Kit (JDK) has the necessary programs to compile and interpret your code for your computer
- Can use integrated development environment (IDE) to quickly make java programs
  - Ie. Netbeans, IntelliJ, Eclipse, Android Studio, BlueJ
- Has library/ Application program interface (API) of ready-made parts to help make programming easier

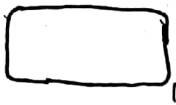
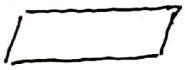
Key Vocabulary	Other New Words

Date: 8/13 Block: 7

Flowcharts

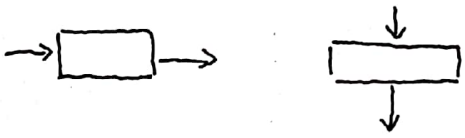
What are flowcharts?

- Graphical representation of an algorithm
- Play a vital role in programming
  - Understanding the logic of problems
- Easy to translate to high-level languages
- Expressing a program to others

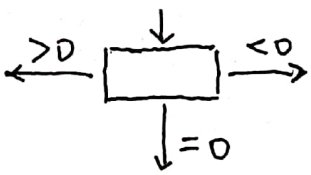
Symbol	Description
 rounded	start or end of the program
 rectangular shape.	Computational steps/ processing
 rhombus rhombus	Input/ output operation
 diamond ?	Decision making: branching
	Connector; page break

Guidelines for Flowcharting

- Necessary requirements should be listed in logical order
- Clear, neat, easy to follow (left to right, top to bottom)
- One line going into a procedure, one line going out



- One line going into a decision, many lines out of the block



- Avoid intersections of lines
- Ensure they have a logical start and finish
- Test its validity using simple data test

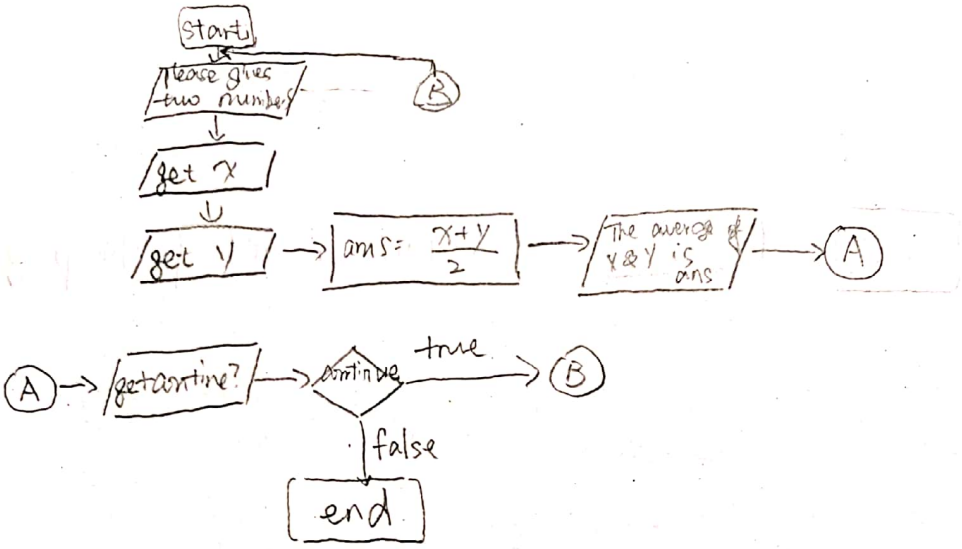
correct.

Name: _____

Date: _____ Block: ____

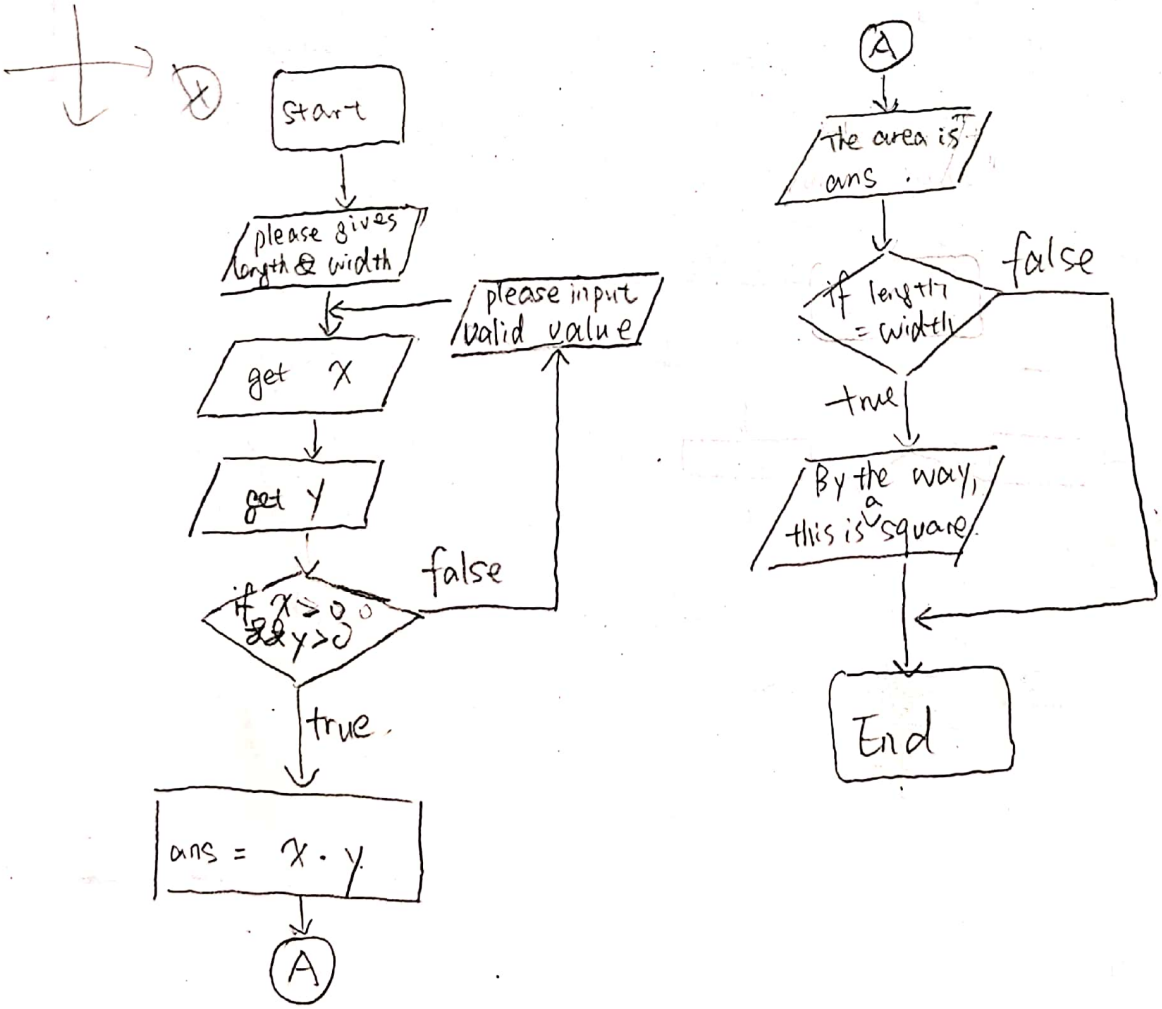
Example of a Flowchart

- Problem: Finding the average of two numbers
- Input: two numbers, x and y
- Output: average of x & y



Write an algorithm for finding the area of a rectangle. (Challenge: write it such that if it is a square the program says it is a square) (Challenge 2: Write it such that the input will be asked until the input is valid)

- Input: length, width
- Output: ans



Program Design

Read: 1.1 – 1.3.7

Code: P.54 1.3, 1.4; P.59 1.16, 1.17, 1.21

What is an Object?

character to control something.

UML Class Diagram: Virtualization of your object

Object ClassName
Data, attribute
Action/Function

1.2 Designing Good Programs

Software life cycle	different phases involved in the design and development of a computer program.
Specification phase	provide a statement of the problem and a detailed description of what the program will do
Design phase	describe the details of various classes, methods, and data that we will use
Implementation phase	actual coding of the coding of the program into Java
Testing phase	test the performance to make sure it is correct

program's

KEY: The sooner you begin to type code, the longer the program will take to finish

Careful design of the program must ~~precede~~ precede coding.

muse

欲先

- Software Engineer Life Cycle
  - Specification phase:
  - Design phase:
  - Implementation phase:
  - Testing phase:
- Main principle: divide-and-conquer principle
  - Repeatedly dividing problems until you have a collection of relatively easy-to-solve subproblems
  - In this way, the program is divided into a collection of interacting objects

1.3.1 Understanding the Problem

1. What exactly is the problem to be solved?
2. How will the program be used?
3. How should the program behave?

Name:_____
Date:_____ Block: ____

1.3.2 Design

1. What role will the object perform in the program?
2. What data or information will it need?
3. What actions will it take?
4. What interface will it present to other objects?
5. What information will it hide from other objects?

1.3.3 Data, Methods, and Algorithms

Algorithm	step-by-step description of the solution to a problem
Pseudocode	a hybrid language that combines English and programming language
Tracing	step through it line by line with some sample data.

1. What type of data will be used to represent the information needed by the program?
2. What specific task will the method perform?
3. What information will it need to perform its task?
4. What algorithm will the method use?
5. What result will the method produce?

- Pseudocode is a hybrid language that combines English and programming language structure without being too fussy about programming syntax
- Designing algorithms also include _____: _____

without being too fussy about programming syntax.

1.3.4 Coding

Syntax	set of rules that determine wether a particular statement is correctly formulated
Semantics	the semantic meaning of each statement

- The right way to code is use the principle of stepwise refinement.
  - Program is coded in small stages
  - After each stage the code is ~~code~~ compiled and tested
  - In this way, small errors are caught before moving on to the next stage
- The difference between syntax and semantics:

1.3.7 Writing Readable Programs

- Readability: Programs should be easy to read and understand. Comments should be used to document and explain the program's code.
- Clarity: Programs should employ well-known constructs and standard conventions
- Flexibility: and should avoid programming tricks and unnecessarily obscure or complex code.

Programs should be designed and written so that they are easy to modify.

Intro Input and Variables

Variables

- A kind of storage for information you want to use in your program
  - temperary
- 2 kinds of data
  - Primitive data type variables – make space directly for what is being stored
  - Complex data type variables – make space for the location of where the data will be
- Many types (of primitive)
  - int/short/long – integer values
  - double/float – decimal values
  - char – characters
  - boolean – true/false values
  - byte – single piece of information

Making Variables

- Declaring a Variable
  - In order for the computer to remember something, we need to tell the computer (the compiler) to make space for it
  - This is called Declaring a variable
  - What type of data we are storing
  - What we call it in our program (Variable name)
  - Sample: <variable type> <variable name>
- Initializing a Variable
  - Initializing a variable is to give it a starting value
  - Beforehand, it is "empty" or NULL; cannot be used
  - On the left hand side: ~~the name of the variable to~~ be changed
  - On the right hand side: what is to be stored
  - Order of operation: calculate the right side, then store them (into) in the left.
  - Also called assignment statement
  - Sample: <variable name> = <expression>

Scope: Part of the program where the variable can be referenced

KEY RULE (for now): Variable must be declared and initialized before it can be used

Constants

- Represents data that never changes
- Initialized and declared in the same way as variables
- Benefits
  1. Don't have to type same number again and again
  2. Only have to change one value
  3. Easier to understand / read.

Naming Variables

- For programs to be read more easily.
- Avoid errors

1. Variables and methods: lowercase
   - Several words = 1st word lowercase, 2nd + words capitalized
2. Class name: capitalized
3. Constants: all capitalized.

Input

- Getting other people's code to use
  1. Import code
  2. Create a scanner object
  3. Use the object's methods to get data
- Important Commands
  - nextInt () – gets next integer
  - nextDouble () – gets next decimal number
  - nextLine () – gets until enter
  - next () – gets until the next space

Key Vocabulary

Vocabulary	Definition

Date: _____

Classes and Objects

Objects

- An **object** can be considered a "thing" that can:
  - perform a set of related actions (called methods)
  - has specific attributes related to the object.
- an object is an instance of a class.

Class

- A ***class*** is the blueprint, or plan, or template that describes the details of an *object*.
- A class can also be a collection of like methods / values
- A class is the blueprint from which the individual objects are created
- *Class* is composed of three things:
  - a name
  - attributes, and
  - Behaviour (methods)
- Relationship
  - Object *is* an example of a class
  - An object is defined as a class

Object sam → is a → Class Student

Using Objects

- Create an object
- The object calls a method / attribute

Using Classes

- Since no object is created, we are using the blueprint directly
- Use the class name to call the method/attribute.
- To find the methods, we can look at the java api documentation

Primitive Data Types	Complex/ Reference
• Known size • Stores directly into memory • No other actions, just data	• Unknown size (can be as large/small as programmer decides it to be) • Stores an address (reference) to the actual data • Has the ability to call methods.

Complicating Primitive Data

- There are actually classes for our primitive data types
  - Called Wrapper classes
- Has a set of relevant functions to be used
- Cannot be as easily cast from one type to another (why?)

Methods and Attributes to Know:

Math	String	Integer	Scanner

Getting to Know Classes

- What is the difference between how data is managed for primitive data types and complex data types?
  - Primitive data types store values directly
  - Complex data types store addresses also known as references
- What we know as complex data types are called Classes
- Parts of a class:
  - Attributes
  - Constructors
  - methods

Creating a Class

- Why do we name classes in uppercase? Ie. Student, Book
  - Since we want to distinguish between:
    - Keywords (all lower case: public, int, import)
    - Variable identifiers (all lower case: myDogAge, studentNumber)
- Attributes
  - Modifier, Variable type, and identifier (similar to creating a variable in the main method)
  - Attributes generally should be accessible
    - means to be accessible by only the object, to be changed by methods that control how it can be changed
- Constructive
  - Meant to initialize attributes inside an object
  - Can have 0-infinite parameters
    - Some input to be processed/to use for processing
- Methods (Basic)
  - Getter Methods
    - A method designed to retrieve information ("get" an attribute).
    - It will always return the type of information you want
    - It will "usually" have no parameters
    - Most getter methods have just 1-line of code:
      return ..... ;
  - Setter Methods
    - Similar to a getter, except its meant to CHANGE an attribute ("set" an attribute)
    - Will always return NOTHING, but will alter an attribute in some way.
    - Most have a parameter, because you need to tell it what to change to.

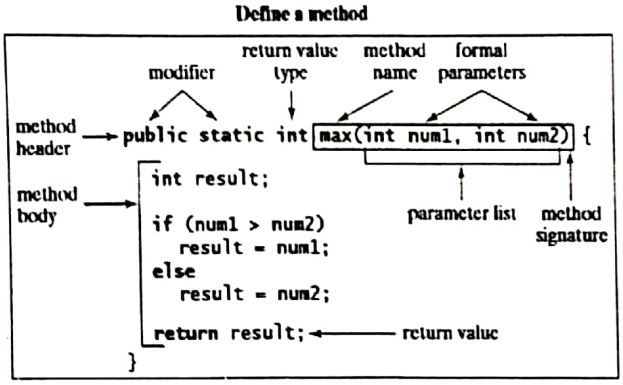
Key Vocabulary	Definitions

Date: ~~Sep~~ 10.9

Methods

Methods

- To Simplify code, to make code easier to look at and read
- To perform a repeated type of action in modified ways
- To have an Object do an action



Parts of A Methods

- Access modifier
  - All lower-case
  - public: anyone can call it (even from another program)
  - private: only the object/class can call it from within itself
  - protected: in between public and private; can only be called by itself or its subtypes (sb-datatypes.) (don't really deal with this)

Who can use it.

- Modifier
  - Either Static or Non-Static
  - Non-static, we do not write anything and just skip this
  - Static: actions are done independent of an object
  - Non-Static: actions are to be done by/to an object

object-related or just use

- return type
  - What should it give me after I have done the actions?
  - Can also be Void, meaning No return value
- Name → Identifier
  - Naming convention is exactly same as variables: ~~first letter of every~~ other word capital
  - Should be very descriptive of what the method does
- Parameters
  - Also known as formal parameters
  - Values we get (input) to have our method work
  - Values passed in are called actual parameters or arguments
  - Changing the actual parameters inside the method ~~will not change~~ the value outside (unless it is complex)

Javadoc Comments for Methods

Scope

- The "area" in which a variable exists
- Can be determined by where the variable is initialized in relation to the closest set of { }, also known as the coding block
- Outside of where the variable is initialized, "it does not exist"

Passing of Values

- The variables write in the definition of our methods are called formal parameters - generic, and receives values from whatever calls the method
- The values that we pass in when we call the method are actual parameters - the actual values used when we call a method
- Complex data types are stored as references / arguments address
- Primitive data types are stored values
- When we call a method and pass parameters, we can either pass a value or pass a reference
- Significance?

① Passing of reference → store as addes → it could chage
② Passing of value → copy value, it not chage.

toString Method

- Overwrite what happens when we print an object

```
public String toString() {
    return whatever you want it to, String value
}
```

Keyword: this

- "this" is a keyword specifically for classes
- Used to call a method/variable that is found within the class
- Used to differentiate between formal parameters and attributes, mainly
- Not necessary, but helpful

Object S = new

x (S)

address

public void x (Object a){ chage S

a.change (2) ;

Keywords	Definitions

]

Name: JimLi
Date: 2017/10/19

Complex Conditionals

Nested Conditionals

- Conditionals are for doing different actions based on a prior calculation
- Sometimes, we might say that a question must be asked if another question is answered in a certain way
- Key Idea: putting a conditional ___inside___ another conditional

Visualisation:

```
if (boolean-expression) {
    if (b-e) {
        statements();
    } else {
        s();
    }
} else {

}.
```

Compound Conditionals

- ___Combining___ conditionals
- ___&&___ means AND
  - ___All___ conditions must be true in order for the expression to be true
- ___||___ means OR
  - ___The___ condition must be true in order for the expression to be true
- ___!___ means NOT
  - ___One and only one___ conditions must be true in order for the expression to be true
- ___^___ means XOR (_____)

- When doing Boolean expressions, as with mathematical expressions, _____ **matter**
  - _____ work to your advantage

Basic Logic Symbols

Meaning	Symbol	Java Equivalent
And	∧	&&
OR	∨	||
XOR	⊕	^
Not	~	!
implies	⊃	
is equivalent	≡	

Date: 10.17

3.1 Boolean and If Statements

What are Conditionals

- Making decisions, based on whether something is true or false
  - Also known as selection statements
  - Is a process to apply Logic

3.2 Boolean

- Datatype that holds either true or false
- **Boolean** expressions
  - Calculations /comparisons resulting in a true or false value
  - Key: we DO NOT USE = because it is meant for assigning values to variables
  - Example: boolean isOld = age<32;

boolean isOld = age <32

Java Operator	Math Symbol	Name	Example(radius=5)	Result
<	<	Less than	radius < 0	false
<=	≤		radius <= 0	false
>	>	Greater than	radius > 0	true
>=	≥		radius >= 0	true
==	=	Equals	radius == 0	false
!=	≠	Unequals	radius != 0	true

3.3 If Statements

- Making decisions
- Looks like:

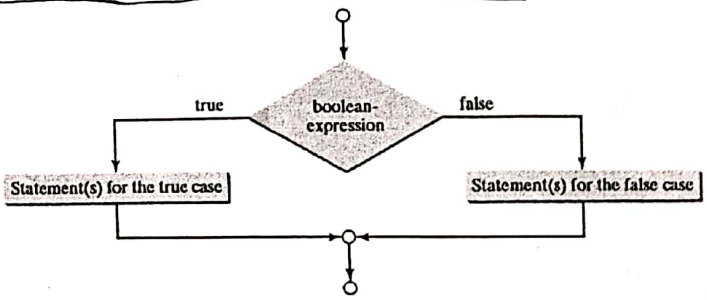
```
if ( < boolean-expression > ){
    statement(s);
}
```

- Each new "if" means a new question is asked, if at the same level
  - This is why Indenting is important!

3.4 If-else Statements

- Part of the if statement; catches all other scenarios in which the conditions are not met
- Looks like:

```
if ( < boolean-expression > ){
    statement(s);
} else {
    statement(s);
}
```



3.5 Combining If-else Statements

- Multi-way if statements – having more than two possible outcomes (you may hear other say "else-if statements")
- Since code runs from top to bottom, the only the code below the first fulfilled condition is run
- Note: Indentation and curly brackets!

Name: Jim L
Date: 2021/10/18

Random and Characters

Random Class

- Random class allows us to create an object that will help create random numbers
- You need to:
  - import java.util. Random
  - Create a Random object
  - Method: nextInt(int upper), where upper is the upper limit of a random number (chooses between 1 and upper, inclusive)

Characters

- Characters are a primitive data type:

  char
- Represented by numbers called the ASCII system
  - (*American Standard Code for Information Interchange*)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	⌂

Special Characters Input

Character	How to Type	Decimal Number
Tab	\t	9
NewLine	\n	10
Backslash	\\	92
Double Quote	\"	34

Comparing Complex Data

- Some objects can be compared
- Has an interface of Comparable

Example: String

- *string1*.equals(*string2*)
  - Returns boolean: true if the strings contain the same value
- *string1*.compareTo(*string2*)
- VERY DIFFERENT! Returns an integer
- if *string1* is alphabetically later than *string2*, **Returns** +ve **numbers**
- if *string1* is alphabetically earlier than *string2*, **Returns** -ve **numbers**
- if they are equal, **Returns** 0

Name: Jim Li
Date: 17.10.24

Loops

Loops

- Control statement that allows the repeating of a certain set of actions
- Is the first step away from analog to virtualization/computation
- A variant of a conditional

Three Kinds of Loops

- Do-While Loops
  - Does the action once, then checks the condition at the end
  - Repeat until the condition is No Longer met
- While Loops
  - Looks very similar, but instead checks the conditional before running
- For Loops
  - Used when we are repeating a certain number of times
  - A loop with a counter
    - Counter: a variable to keep count of how many times it has been run
  - 3 parts
    - for (int i = 0; i <5; i ++)
    - int = 0 -> initializing a counter : INITIAL ACTION
    - i <5 -> boolean expression for exiting : LOOP CONDITION
    - i++ -> increment : ACTION AFTER EACH ITERATION (LOOP)

Solving Loop Problems

- ALWAYS go to the base case
  - What is the most basic situation? What are the actions I need to do?
- Find out what the ending conditional is
  - What is my limit? When should I stop?
- THEN put it together

Nested Loops

- Same with nested conditionals in scope
- Requires THINKING and READING
- What is the base of what you want to repeat?
  - What kind of action do you want to do again and again?
  - Treat the inner loop as a "certain action"

Name: Jim Li
Date: 10.23

Recursion

递归

What is Recursion?

- Having a method call itself in order to repeat a certain action
- In some cases, recursive thinking can reduce a complex problem into a simpler one

Parts of a Recursive Method

- Base case or *stopping condition* – the solution to the simplest case
- Recursive call to sub-problem – reducing the current problem to a simpler case that is identical to the original problem

回文结构

Palindrome Breakdown 分解,

- Base Case(s):

- Sub-problem:

Real-Life Applications

- Mitosis is a process in which a cell divides itself into half to make two identical copies. Mitosis takes place in all types of cells in the human body except few. More importantly, the human body builds itself using recursion algorithm.
- While browsing, following a bunch of links and then hitting the 'back' button makes us follow another series of links are all example of recursion.

Advantages

- Reduce unnecessary function call
- Can make a complex problem elegant and clean
- ~~All algorithms~~ algrithms can be defined recursively

Disadvantages

- Requires patience to debug and understand recursive solution
- Lots of memory
- Requires more ~~processor~~ time

Name: Jim Li
Date:

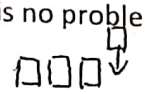
Array Usage

Incomplete Arrays

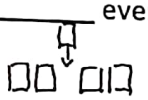
- Dealing with incomplete arrays is different as you no longer have the ~~length of the array~~ to determine how many elements there are in the array
- Therefore, you must create another variable to keep track of the elements

Inserting Elements

- If you are inserting an element at the end, there is no problem; increase number of elements by 1, and add to the end



- If inserting in the middle, you must shift every element between the inserting place and the end



Deleting Elements

- If you are deleting an element at the end, there is no problem; decrease number of elements by 1
- If deleting in the middle, you must shift every element between the deleting place and the end as well

"Expanding" an Array

- In order to add more space for elements to an array, you must
  - Create a new, large array double length
  - Copy elements to the new array
  - Replace the old array

object complex
↑
– hello.length()
– nums.length;
↓
ray simple

Name: _____
Date: _____

Arrays

An array is an indexed sequence, all the same type.

Real-life examples of arrays include the following: post office boxes; book pages; egg cartons; chess/checkerboards

Why Arrays?

- Easier to manage names of variables
- Grouping like variables together
- Organized structure

NullPointerException
↳ No value/space created.

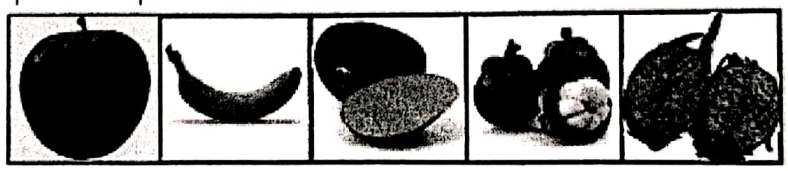
Array Use

When it is Good to Use	Not Good
• Same thing that is being done again and again (ie. 20 names, 20 marks) • Need to do same action on a list of things (change the marks of a certain grade 12 class) • sets	• Same variable type, but different purpose (ie. age and studentID are int, but are different)

Parts of an Array

Element(s) – individual parts. Length – the size of the array

sets



"Apple" [0] | "Banana" [1] | "Mango" [2] | "Mangosteen" [3] | "Dragonfruit" [4]

→ index
plu ↳ indices

length – the size of the array.

Declaring an Array

We've seen it before: public static void main (String [] args)

- [] after the variable type tells us it is an array
  - Ie. int [] myInts;
  - Ie. String [] args;
  - Ie. double [] grade12Marks;

Initializing an Array

- Manual Input
  - Elements in an array all inside ~~set~~ { }. Brace.
  - Elements separated by ,
  - String [] fruits = {" ", " ", " "};
- Empty Array
  - What if we didn't know our values from the beginning?
  - int [] people = new int [5];
  - Creates 5 ~~empty~~ int variables
  - Then we can fill it!

Accessing Elements in an Array

- Initializing Elements in an Array
  - Each individual element must be initialized before it can be used
    - When you initialize an array, you initialize length
  - Must access each individual element by its index
  - fruits[0] = "Strawberry";
- Accessing Elements of an Array
  - We can now use a structure called loops to go through an entire array
    - System.*out.println(fruits[0]);*

Therefore, elements of an array are basically variables

Date: _____

File IO

Possible Errors

- File is not there
- Don't have permission to go there
- There is a problem getting the information

Try- Catch

- Tries to do one thing, and in case of an error, does something else

- catch(IOException e)
  - Catches any errors with opening/reading/writing/closing a file

Reading a File

- File - a file
- FileReader an object that takes your file and reads it
- BufferedReader - an object that takes your file information and makes it useable

Writing to a File

- File - a file
- FileWriter - an object that takes your file and readies it for writing
- BufferedWriter - an object that takes your data and makes it able to be written in a file

import java.io.*; util.Scanner

Key Input Commands	Important Output Commands

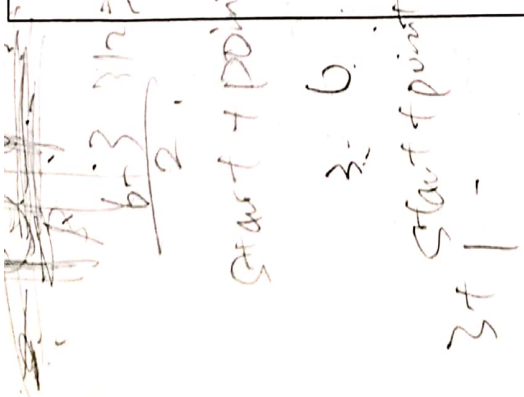
Class Relationships

Class Diagrams

- Describe the structure of your system
  - Not objects, but _____
  - _____
  - Describe relationships of classes and dependencies
  - Visibilities
    - Public = +
    - Private = −
    - Protected = #

+ name − String .

Relationship & Arrow Shape	Description
Association Class A —Associated with.→ Class B	• If they are somehow related to each other • Ie. Students and teacher are associated
Aggregation 聚合 child class ◇— parent class.	• Special case of association • When an object "has a" another object, then it is aggregation • Ie. Student has a textbook (have own)
Composition child class ◆— Parent class.	• A special case of aggregation. • When an object contains another object, and exists solely by other said object • Ie. class (must) contains student
Generalization . 泛化. Derived Class —▷ Base class	• "is a" relationship • Superclass subclass • Ie. student is a person.



Multiplicity

- Number of instances of one class linked to one instance of the other class
- Ie. a student may go to one school, but a school may have 1..3000 students

Indicator	Meaning
0 .. 1	Zero or 1
1	One only
0 .. *	0 or more
1 .. *	1 or more
0 .. n	0 to n (n>1)
1 .. n	1 to n (n>1)

n.

only n (where n > 1)

Homework

1. Create a student with name and student number and an array of courses and a count of credits earned
2. Create a teacher with a name and array of courses
3. Create a course with course name, an array of students, and teacher
4. Create an administrator with a name and array of courses, and an array of responsibilities
5. On this sheet, draw the UML diagram for these programs.

ASCII

char

16-bit

Name: Jim Li Date: 2021/12/18

Abstract Classes

Vocabulary	Translation
Abstract	

Review

- Superclass and subclass relationships are from generalized to specified
- Class design should ensure that a superclass contains common features of its subclasses

Abstract Classes

- Sometimes, a superclass is so generalized that creating instances of itself would be useless
- These classes can be defined as abstract classes
  - Ie. Shape:
  - Can you just create a shape? No
  - What methods do you think a shape should have?
  - _____

Abstract Methods

- Have the keyword abstract as a modifier
- Implementation of methods is different per subclass, so only defined
- That means subclasses Must have such a method

Important Details

- Abstract methods can only be defined in abstract classes
  - Subclasses must implement such a method
- Abstract classes cannot be instantiated (not create instance) instantiated.
- Abstract classes can have non-abstract methods
- Subclass can be abstract even when the superclass is concrete
- Abstract classes cannot be instantiated, but can be used as a datatype

When to Use Abstract Classes

- when you have a requirement where your base class should provide default implementation of certain methods whereas other methods should be open to being overridden by child classes use abstract classes.
- The purpose of an abstract class is to provide a common definition of a base class that multiple derived classes can share.

Abstract Classes

Vocabulary	Translation
Interface	

Review

- When we generalize so much that creating an instance of a class is not realistic;
- A plan of what other classes will look like
- Has abstract methods → only header

Interfaces

- Similar to abstract classes
- Purpose is to specify common behaviour for objects of related or unrelated classes
- Many similar key words
  - abstract
  - instanceof

Using an Interface

- To have a class use an interface, it is called interface inheritance (interface inheritance)
- You can only extend one class (one superclass to one subclass)
- You can have multiple different interfaces for one class

Interface: Comparable

definition

```
public interface Comparable<E> {
    public int compareTo(E o);
}
```

Using words, describe this code:

_____

_____

- compareTo method
  - Compares it with a _____
  - Returns _____, _____, or _____, if it is this object is less, equal, or greater than the one in the parameter
- Example classes with comparable: _____

Name: _____ Date: _____

	Variables	Constructors	Methods
Abstract Class	No Restrictions	Constructors are invoked by subclasses through constructor chaining. Cannot be instantiated	No restrictions
Interface	All variables must be public static final	No Constructors. Cannot be instantiated	All methods must be public abstract, instance methods

Abstract Class	Interface
• Can only inherit one • When creating a class library which will be widely distributed or reused • to define a common base class for a ~~for a~~ family of types • to provide defaults behavior • a base class in a hierarchy to which the class logically belongs	• Can inherit more than one • creating a standalone project which can be changed at will (more design flexibility • allowing a specific type to support numerous behaviors • to design a polymorphic hierarchy ~~for value~~ for value types • defines a very specific range of functionality

Name: Jim Li Date: 27.12.13

Inheritance

Vocabulary	Translation
Inheritance	继承
Superclass	
Subclass	
Overloading	
Overriding	

Attributes of a:

Teacher	Admin
Student	**Course**
_____	

Inheritance

- Having the properties of the parent
- Generalized -> Specific
- Ie. Vehicle : Car, Boat, Plane
- In java, we can make classes based on other classes by using the keyword extends
- Superclass vs. Subclass
  - Superclass refers to the more generalized class
    - ie. Vehicle
  - Subclass refers to the more Specific class
    - Ie. Car, Boat, Plane.
  - In java, subclasses can refer to its superclass by the keyword Super
  - What gets inherited?
  - Method, Attributes, Constructor (sort of) only the generic one

Method Overloading

- When, within one class, we create two methods of the same name
- How is this possible? _____

Method Overriding

- How the superclass acts may not be exactly how we want the subclass to act
- The subclass can override the superclass's method
- Ie. work in Human, Teacher, Administrator

Name: Jim Li Date: 2017/10/14

Polymorphism

form

Vocabulary	Translation
Polymorphism	many forms.
Dynamic Binding	
Casting	
Instance	

If CUP is a superclass, POP is a subclass, and MAM is also a subclass

Then:

ALL POP are CUP

All MAM are CUP

Some CUP are MAM

Some CUP are POP

Not all CUP may be POP

Not all CUP may be MAM

Polymorphism

- A variable of a superclass can hold an address to a subclass object
  - Human xr = new Student();

Dynamic Binding

- Finding out which Methods to use at runtime
  - Because there is a possibility for a superclass to be any one of the subclasses, deciding which method to call must happen at runtime
- Instance
  - How do we know if it is one subclass or another?
  - instanceof -> a boolean operator
  - Returns boolean
  - If something is an instance of a class, returns true
    - Instance: an actual thing ' an object of that type.
- Casting
  - What about using Subclass attributes/methods?
    - We need to cast first!
    - Implicit casting: casting a subclass as a superclass
      - ie. int to a double
    - say it. Explicit casting: casting a superclass as a subclass
      - Ie. double to a int

workinStudent

Human a = new Stu()

(a. work() implicit.

(Student)a). study()

explicit

Name: _____ Date: _____

Generics

Vocabulary	Translation
Generic	

What are generics?

- generics enable types (classes or interfaces) to be parameters when defining classes
- like formal parameters used in method declarations, type parameters provide a way for you to re-use the same code with different inputs
- This time, not values, but data type !

Why generics?

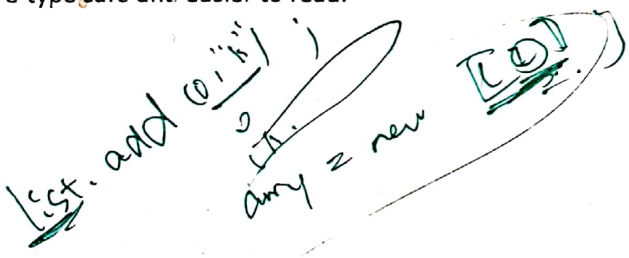
- Elimination of casts
- Enabling programmers to implement generic algorithms
  - can implement generic algorithms that work on collections of different types
  - can be customized, and are type safe and easier to read.

ArrayList<T> extends List<T>

- Like an array, but Class
- T represents the type of objects – the generic
- Can use Wrapper Class

Initializing

- ArrayList<Class> list = new ArrayList<Class>();



ArrayList <T>

void add(int index, Object element)
boolean add(Object o)
void clear()
boolean contains(Object o) – find wether exsist
void ensureCapacity(int minCapacity)
Object get(int index)
int indexOf(object o) //first
int lastIndexOf(object o)
Object remove(int index);
Object set(int index)
int size() //number of elements
void trimToSize(); //trim capacity

Title: Bubble Sort

Try this Algorithm:

3 5 6 2 1

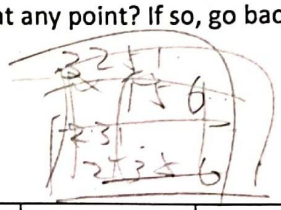
3 5 6 2 1

3 5 6 2 1

3 5 2 6 1

3 5 2 1 6

1. Go through all the cards, one at a time
   a. Put the current card in your left hand
   b. Put the next card in your right hand
   c. Is the current card is bigger than the next card?
      i. Put the current card to where the next card was
      ii. Put the "next card" where the current card use to be
   d. If not
      i. Put them back where they were originally
2. After going through all the cards this time, did you move the cards at any point? If so, go back to step 1



Sorting

Initially, my cards were:

6	2	10	8	3	7	4.

Now, my cards are:

2	3	4	6	7	8	10

Big-O Notation

In computer science, big O notation is used to classify algorithms according to how their running time or space requirements grow as the input size grows.

Notation	Name	Meaning
O(1)	Constant	Always x number of steps, regardless of n.
O(log n)	Logarithmic	Pass through a logarithmic function of n times
O(n)	Linear	Pass through n number of times
$O(n^2)$	Quadratic	Pass through n^2 amount of times.

Big(O) of _____

Best Case	Worst Case	Average Case
• Each element only 1 time. • O(n)	Completely reversed O(n^2)	O(n^2)

Name: Jimna

Searches

Purpose: Finding data in your collections

Linear Search

- Go through one by one
  - If it is what I want, return
- If not found, return not found

Big O

Best Case	Worst Case	Average Case
O(1)	O(n)	$\frac{(n+1)}{2} = O(n)$

Binary Search

- Search a range
  - Length = 0?
    - Not able to find
  - Find the midpoint
  - Is equal?
    - Return value
  - Is greater?
    - Search later half
  - Is smaller?
    - Search beginning half.

Key: must be sorted first

Big O

Best Case	Worst Case	Average Case
O(1)	$O(\log_2(n))$	$O(\log_2(n))$.

Jim Li

Heap sort Handout

Definition:

Heap sort is a comparison based sorting technique based on **Binary Heap data structure**. It is similar to selection sort where we **first find the maximum element** and place the maximum element at the end. We repeat the same process for remaining element.

A **Binary Heap** is a **Complete Binary Tree** where items are stored in a special order such that value in a **parent node** is greater(or smaller) than **the values in its two children nodes**.The heap can be represented by **binary tree** or **array**.

If the parent node is stored at index I

the left child can be calculated by: 2 * I + 1

right child by 2 * I + 2

Pseudocode:

1.to build a heap

- Put the the card which index is 0 to the top
- Put the index 2*index+1 in the left down side
- Put the index 2*index+2 in the right side

(if index < array.length cycle the second and third process to build a binary tree)

2.adjust the heap

- If left or right child is bigger than parent, switch the position

(until reaching the complete heap)

3.switch the position of top one with bottom one, AND put the bottom one into the array

Position is : array.length - cycle times between the process 2 and 3.

(go to the process 2 until the array is full and heap is empty)

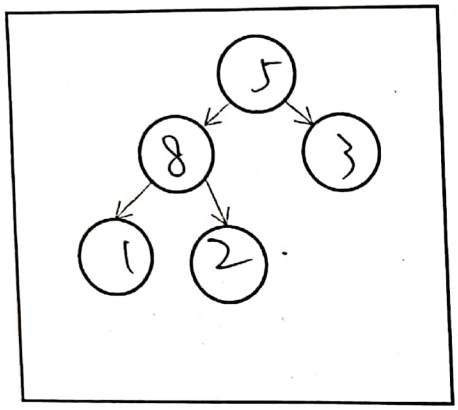
Practice

Choose 5 element(int),fill the blank

Array you choose:{ 5 8 3 1 2 }

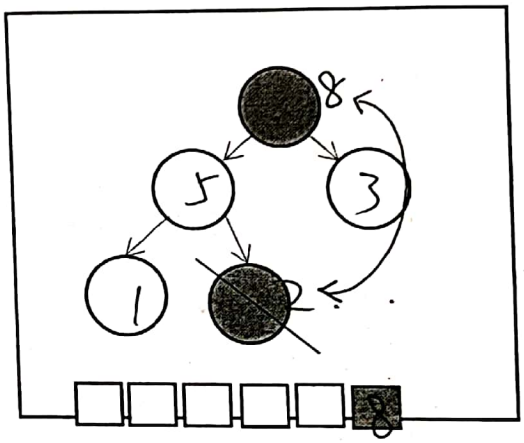
1.first time you build a heap

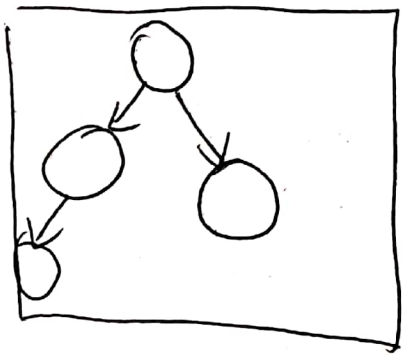
(parent bigger then children)



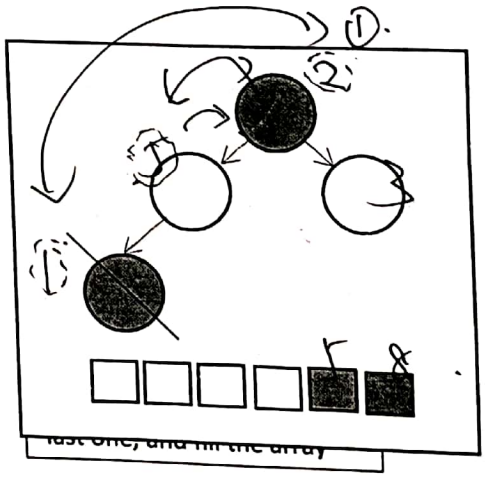
2. switch the top one with the last one

and store the last one

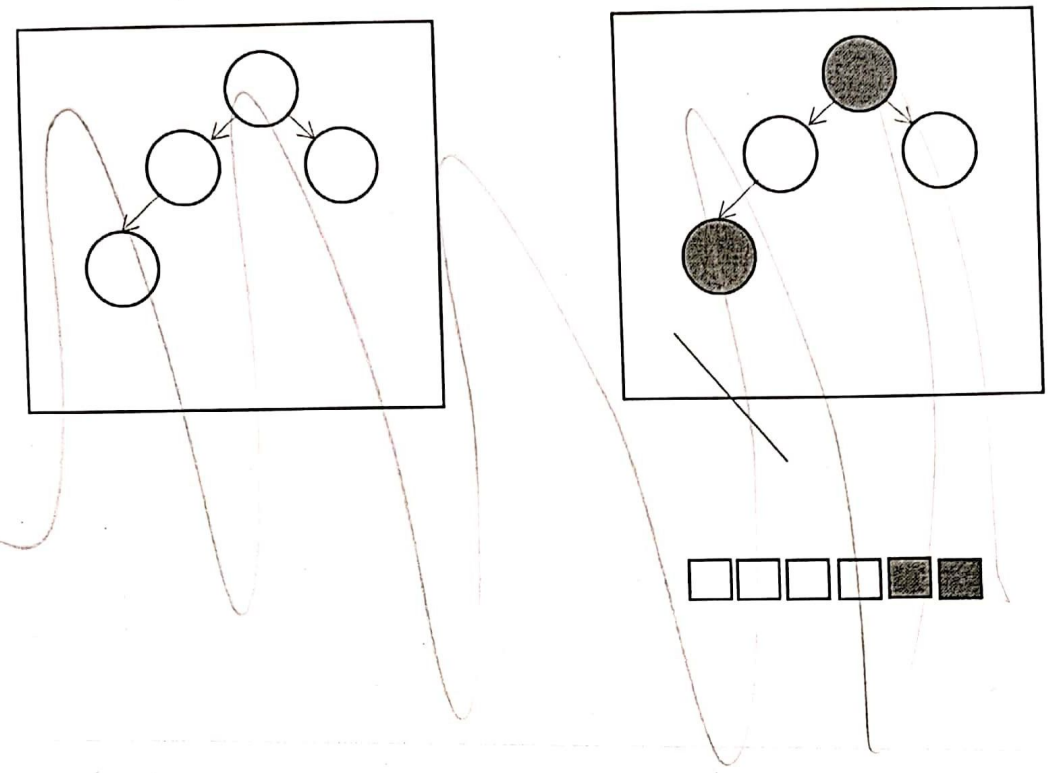




3.adjust the heap



Same as 2 process



Do last steps by yourselves!

Finally, the array is{ 1 2 3 5 8 **}**

Summary

Best case	Worst case	Average case
$O(n \times \log_2 n)$	$O(n \times \log_2 n)$	$O(n \log_2 n)$

Advantage:

- Efficiency
- Memory Usage
- Simplicity
- Consistency

Shell sort

Given an array of 7 numbers:

3	0	1	8	7	2	5

swap.

Shell Sort is mainly a variation of Insertion Sort.

In insertion sort, we move elements only one position ahead. When an element has to be moved far ahead, many movements are involved.

We calculate gap for each pass, and then select the elements according to gap. (gap is h)

The idea of shell sort is to allow exchange of far items. In shell sort, we make the array for a large value of h. We keep reducing the value of h until it becomes 1 .

Big O notation in best/worst/average case

Best case	Worst case	Average case
$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$

Pseudocode

Gap

n n/2 n/4

Name: Jim Li
Date: 2017.12.4

Merge Sort

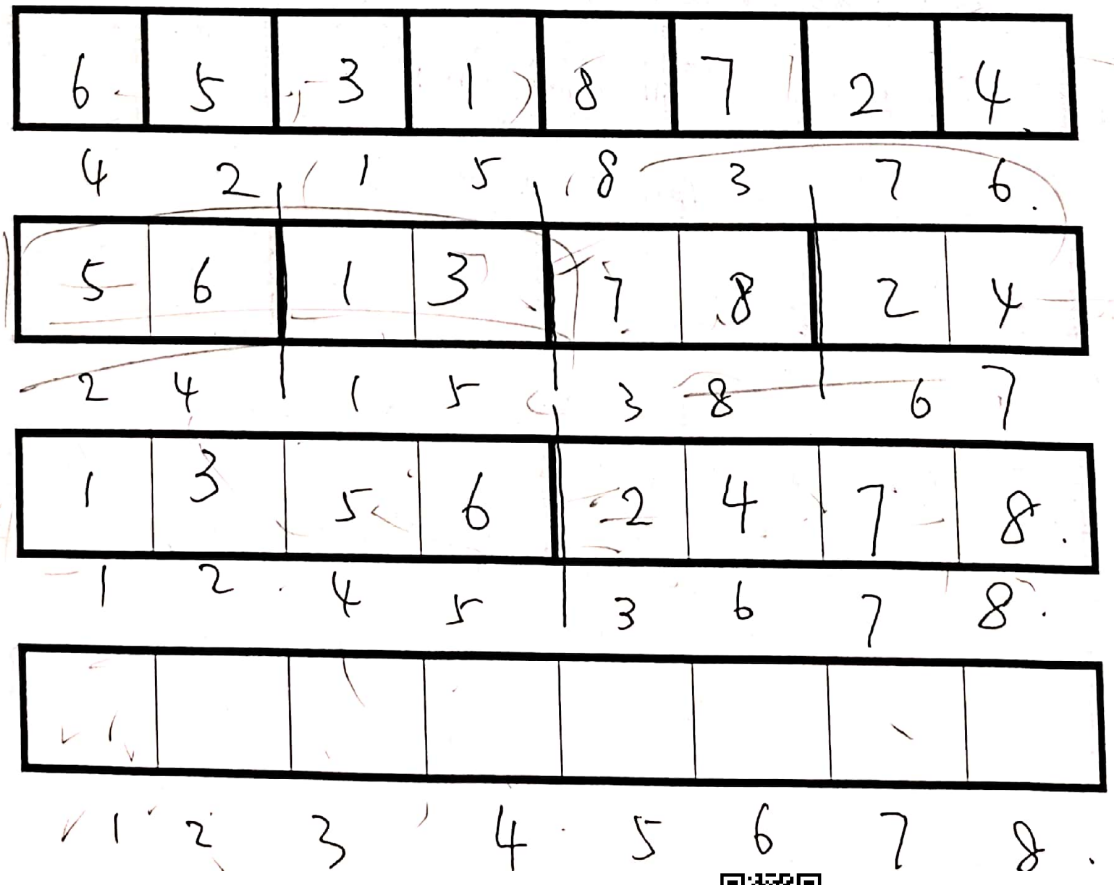
Characteristics

- Divide and conquer algorithm
- Efficient, general-purpose, Comparison-based sorting algorithm, and most implementations produces stable sort

Algorithm

1. Divide array into several sub-arrays, each containing one element.
2. Then merge sub-arrays repeatedly to produce new, until there is only sub-array left. This will be the sorted array.
3. In order to achieve, we need recursion

Demonstration



Name: _____

Date: _____

Big O of Merge Sort

Best Case	Worst Case	Average Case
O(n log n)	O(n log n)	O(n log n)

Implementation

- Divide input array in two halves, calls itself for the two halves and then the two sorted halves.

Pseudocode

right index

```
MergeSort [ arr ], l , r );
If r > l

    middle m = (l+r)/2;

    call mergeSort (arr, l , m )

                   (arr , m+1, r )

                   (arr, l, m, r)
```

Insertion sort

*What is the purpose?

Arranging a disorder sequence by a specific order.

Definition:

Insertion sort is a simple sorting algorithm that builds the final sorted array one item at a time.

Advantages:

Simple implementation.

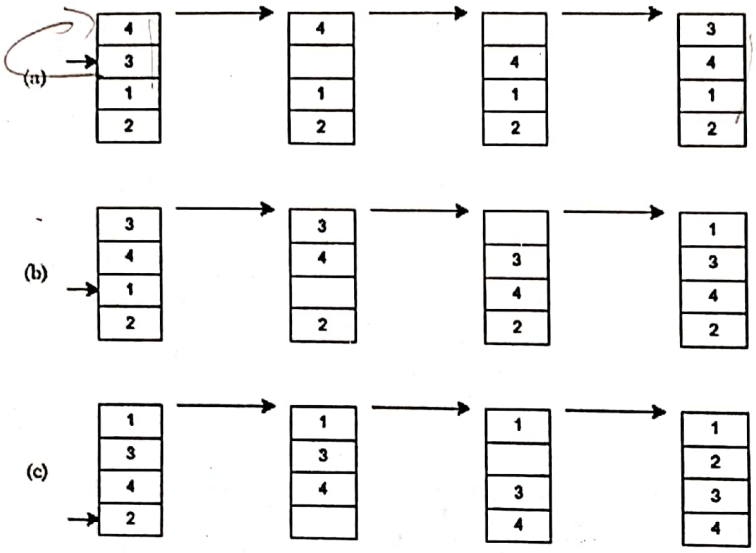
Efficient for small data sets.

Efficient for data sets that are already substantially sorted—— (adaptive.

does not change the relative order of elements with equal keys—— (stable)

only requires a constant amount O(1) of additional memory space

*example



Big O

Best case	Worst case	Average case
O(N)	O(N²)	O[(N+N²)/2]

tha reprai sigme guysa tyou I fe el.

tha guysa reprai sigme tyou I fe el.

thap tyou guysa reprai sigme I fe el

I tha tyou guysa reprai sigme I fe el

>

I fe el tha tyou guysa reprai sigme

Quicksort

Jim Li

1.Quicksort is a fast, sorting algorithm

2.Quicksort is suitable for sorting big data volumes

3.Quicksort is a comparison sort, meaning that it can sort items of any type for which a "less-than" relation (formally, a total order) is defined.

4.In efficient implementations it is not a stable sort, meaning that the relative order of equal sort items is not preserved.

5.Quicksort can operate in-place on an array, requiring small additional amounts of memory to perform the sorting.

Algorithm

The divide-and-conquer strategy is used in quicksort. Below the recursion step is described:

Choose a pivot value. We take the value of the middle element as pivot value, but it can be any value, which is in range of sorted values, even if it doesn't present in the array.

Partition. Rearrange elements in such a way, that all elements which are lesser than the pivot go to the left part of the array and all elements greater than the pivot, go to the right part of the array. Values equal to the pivot can stay in any part of the array. Notice, that array may be divided in non-equal parts.

Sort both parts. Apply quicksort algorithm recursively to the left and the right parts.

Persucode

partition(int arr[], int left, int right)

1. get pivot

   (left + right) / 2

2. go through elements that does not greater than pivot, or smaller than pivot

   while (arr[i] < pivot)

```
        i++;
    while (arr[j] > pivot)
        j--;
```

3.warp the elements that one is greater than pivot and one is smaller than pivot

wrapper element at i and element at j

i++

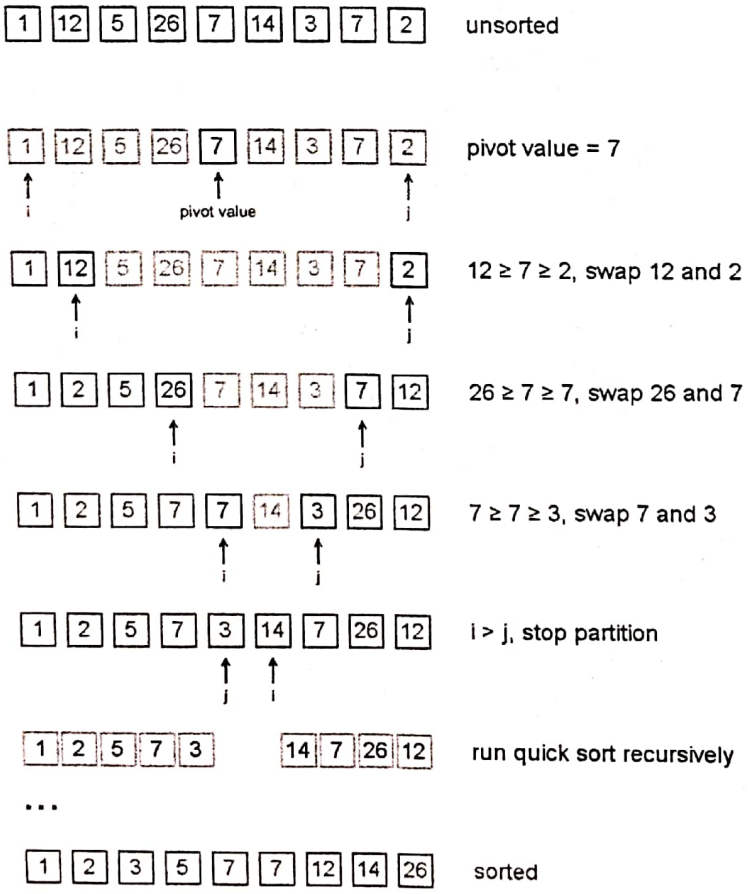
j++

4.if i & j does not check all the elements, go to step two

5.return the index where i and j are equal

quickSort(int arr[], int left, int right)

1. do the partition and find index
2. Sort both part



Name: Jimi Date:_____

Selection Sort

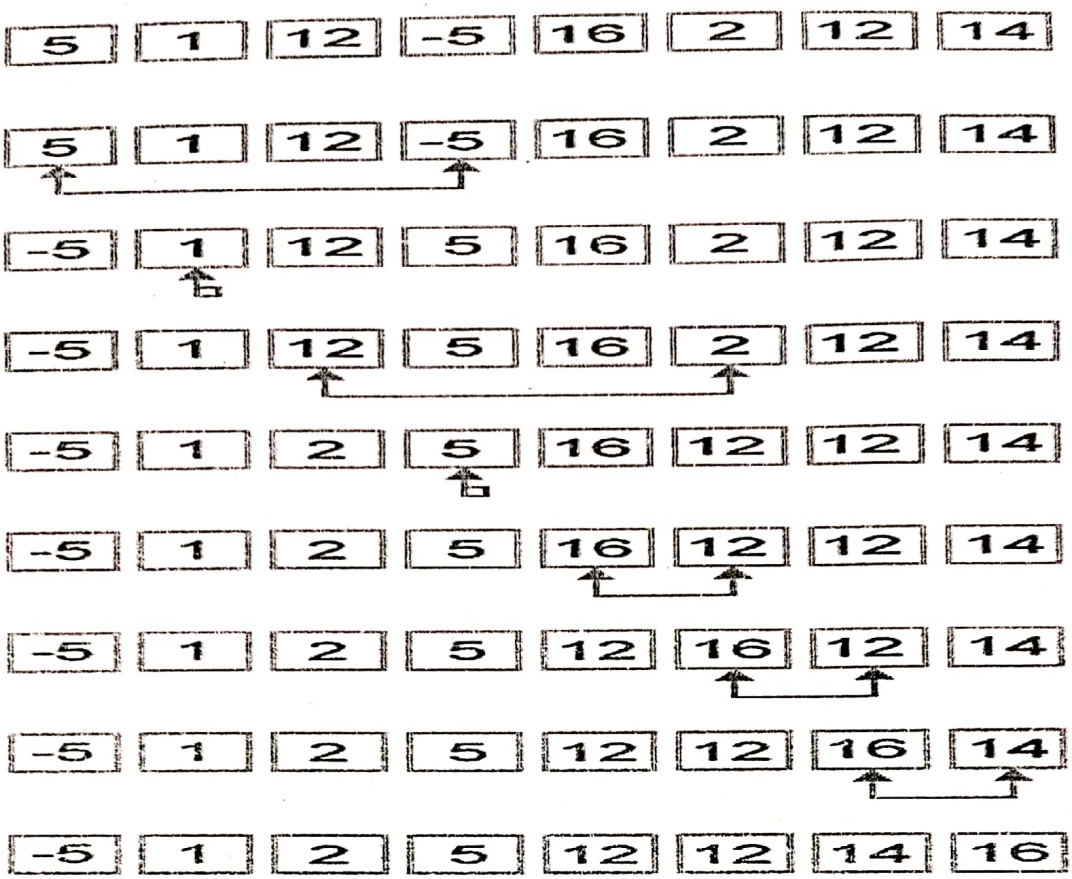
Definition:

In computer science, **selection sort** is a sorting algorithm, specifically an in-place comparison sort. It has $O(n^2)$ time complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and it has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.

Algorithm:

The idea of algorithm is quite simple. Array is imaginary divided into two parts - sorted one and unsorted one. At the beginning, sorted part is **empty**, while unsorted one contains **whole array**. *At every step*, algorithm finds **minimal element** in the unsorted part and adds it to the end of the sorted one. When unsorted part becomes **empty**, algorithm *stops*.

Demostration:



Pseudocode:

Void SelectionSort(){

//from smallest to largest

1. find out the smallest number in the array and put it at index[0].

2. find out the smallest number in the rest of the array(in this case need to except index[0]) and put it at index[0+1].

3. repeat the above step 2 until all the number is already sorted.

}

Jim Li

Title: ~~Bucket~~ Comb Sort

What is that?

The Comb Sort is another *comparison* and *exchange* sort which builds on the idea of the ~~bubble sort~~, and adds a potential optimization or two.

Common Terms

Turtle - a name given to small numbers that appear towards the end of a data set. When used in a bubble sort, small numbers at the end of the set take a very long time to get to the front of the list, and hence are called turtles because of the lack of speed.

The Algorithm

Each iteration of the algorithm consists of three stages:

1. Calculation of the gap value.
2. Iterating over the *data set* comparing each item with the item that is *"gap"* elements further down the list and swapping them if required.
3. Checking to see if the gap value has reached one and no swaps have occurred. If so, then the set has been sorted.

Calculating the "gap"

calculation of the *gap* is as simple as starting with the size of the *data set* and dividing by a *shrink factor* of 1.3, each iteration

Iterating and Swapping

same as the Bubble Sort. The only difference in the Comb Sort is that the items which are compared are i and i+gap as opposed to i and i+1.

Pseudocode

1. using the length/size of the set as the gap.
2. do the comb sort with the current gap
3. reset the gap value so that it shrinks(1.3)
4. in order the card

Big O notation in the best/worst/average case

Best case	Worse case	Average case
$O(n^2)$	$\Theta(nLogn)$	$\Omega(n^2/n^p)$

Bucket Sort

Name: _____

Definition

Bucket sort, or bin sort, is a sorting algorithm that works by ***distributing*** the elements of an array into ***a number of buckets***. Each bucket is then sorted ***individually***, either using a different sorting algorithm, or by recursively applying the bucket sorting algorithm.

Process

- **Set up** an array of **initially** empty "buckets".
- **Scatter**: Go over the original array, putting each object in its bucket.
- **Sort** each non-empty bucket.
- **Gather**: Visit the buckets in order and put all elements back into the original array.

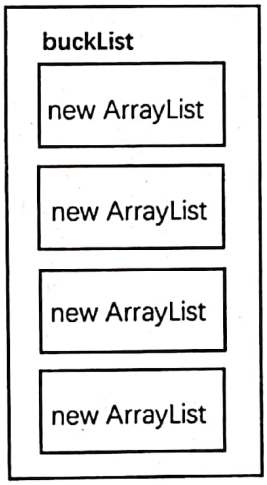
Pseudocode

1.Find the max and min in the array in order to get the bucket numbers

```
int bucketNum = xxxx;
```

2. Creat a array named buckList contains array:

```
List buckList=new ArrayList<List<Integer>>();
```



3. Creat buckets in buckList

```
for (int i=1;i<=bucketNum;i++){
buckList.add(new ArrayList<Integer>());
}
```

4.Put numbers in relative buckets

5.Sort numbers in each buckets (Usually use quick sort)

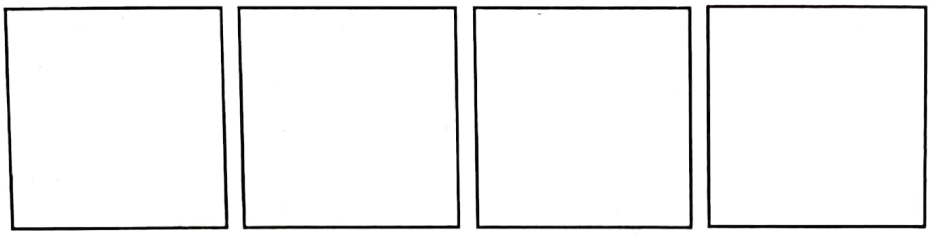
6.Combine all the non-empty buckets together

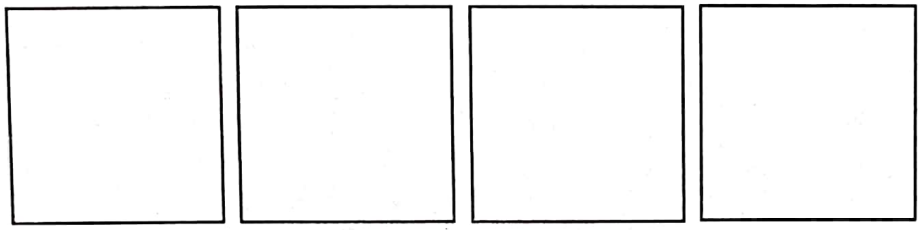
Big O Notation

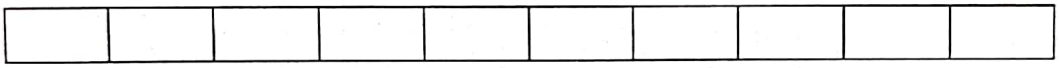
Best Case	Worst Case	Average Case
Ω (n+k)	$O(n^2)$	θ(n+k)

Demonstration

8	35	67	12	24	53	56	90	100	96







Features

- Stable
- Fast (faster than quick sort for most of the times)
- Need Large Memory

APCS Review by jimschenchen[]

Unit 1 - 4

Testing Range: Everything

Hardware

Hardware: can touch Software: cannot touch

Computer:

•Central Processing Unit (CPU) -> arithmetic unit / logic unit; transistors – emit pulses; clock; hertz(Hz); Core; simultaneous: at the same time

•Memory (RAM) -> min Storage: 8 bits = 1 byte; never empty; content lost

•Storage Devices -> volatile = unstable;

3 types : magnetic disk drives (HDD); optical disk drives (CD, DVD); flash drives (USB, SSD)

•Input / Output Devices -> I: mouse O: monitor

•Communication Devices -> Network Interface Card (NIC)

Subsystem: motherboard (BUS)

Warning: GM = gigabytes = 8 * gigabits

Binary – Not Testing

Not Testing

Language

1. Machine L
    a) computer can understand without help
    b) Ie. 0 / 1
2. Assembly L
    a) Direct translate to commands to Machine L
    b) Ie. RAM

3. High Level L

   a) English Like Interpreters || Compilers take code and translate to machine L

   b) Ie. Java, PY

Distinction: Compiled and Interpreted

Notes

Special Characters: name (){}[]//"";

Javadoc

Indentation rules -> tab && spacing rules -> space

Syntax errors && Runtime Errors && Logic Error

Bugs && Debugging

Operation

+-*/%

+= : addition assignment

-=: subtraction assignment

++Var: pre-increment

Var++: pos-increment

--Var: pre-decrement

Var--: post-decrement

Explicit Casting && Implicit Casting

Double -> int && int -> double

5 Types of errors

Flowcharts

Express algorithm

*rounded

*rectangular

*rhombus

*diamond

*circle

Warning: Write problems/input/output at the top of flowchart

Software Life Cycle

Specification phase

Design phase

Implementation Phase

Testing Phase

Pseudocode || Algorithm || Tracing

Program three advantages: readable, clarity, flexibility

Variables

Primitive data type

Complex data type

Vocab: declared and initialized

Constants: all capitalized

Input: Scanner

Class && Objects

Object: instance -> class: blueprint

Class: attribute, constructors and methods

Primitive Class

Complex Class / Reference – Address / ability to call methods

Wrapper Class

Methods

Access Modifier

Modifier

Return Type

Identifiers

kmeters:

- *Actual Parameter / Argument -> value pass in d
- *Formal Parameter

Complex data -> stored in reference / argument address -> changed

Primitive data -> stored in value -> not change

Complex Condition

Nested Condition

&&, ||, !, ^

Recursion

Good Luck!!!

Base case / stopping condition

sub – problems

Adv and Dis

Random and Characters

ASCII: American Standard Code for Information Interchange

Random: need to import -> java.util.Random;

Tab: \t NewLine: \n Backslash: \\ Double Quote: \"

String.equals(str2) -> return Boolean

Str.compareTo(str) -> return integer

Unit 5 Review

Parts of array

How arrays are used

how to manipulate an incomplete array

Arrays

Elements / length/ index (indices)

Unit 6 Review

How to sort

relationships of each sort

generally how they work

How to search

binary

BigO notation:

*purpose: It is to know the complexity of a sort / classify run time needed and space requirement

Tracing algorithms

Unit 7 Review

Class Relationship

Inheritance

Abstract Classes

Interfaces

Generics - <>

UML Class Diagram

How to use ArrayList<>

AP Definition

*Palindrome Breakdown 回文结构

*Base Case: the solution to the simple case

*sub-problem: reducing the current problem to a simple case that is identical to the original problem.

Binary Search, sort first! Find middle, compare ->go left or right -> find middle

Best Case: 1 Worse C: n Average C: n

BigO Notation: to classify algorithms according to the runtime and space needed.

1 快速排序（QuickSort）

快速排序是一个就地排序，分而治之，大规模递归的算法。从本质上来说，它是归并排序的就地版本。快速排序可以由下面四步组成。

（1） 如果不多于 1 个数据，直接返回。

（2） 一般选择序列最左边的值作为支点数据。

（3） 将序列分成 2 部分，一部分都大于支点数据，另外一部分都小于支点数据。

（4） 对两边利用递归排序数列。

快速排序比大部分排序算法都要快。尽管我们可以在某些特殊的情况下写出比快速排序快的算法，但是就通常情况而言，没有比它更快的了。快速排序是递归的，对于内存非常有限的机器来说，它不是一个好的选择。

2 归并排序（MergeSort）

归并排序先分解要排序的序列，从 1 分成 2，2 分成 4，依次分解，当分解到只有 1 个一组的时候，就可以排序这些分组，然后依次合并回原来的序列中，这样就可以排序所有数据。合并排序比堆排序稍微快一点，但是需要比堆排序多一倍的内存空间，因为它需要一个额外的数组。

3 堆排序（HeapSort）

堆排序适合于数据量非常大的场合（百万数据）。

堆排序不需要大量的递归或者多维的暂存数组。这对于数据量非常巨大的序列是合适的。比如超过数百万条记录，因为快速排序，归并排序都使用递归来设计算法，在数据量非常大的时候，可能会发生堆栈溢出错误。

堆排序会将所有的数据建成一个堆，最大的数据在堆顶，然后将堆顶数据和序列的最后一个数据交换。接下来再次重建堆，交换数据，依次下去，就可以排序所有的数据。

4 Shell 排序（ShellSort）

Shell 排序通过将数据分成不同的组，先对每一组进行排序，然后再对所有的元素进行一次插入排序，以减少数据交换和移动的次数。平均效率是 O(nlogn)。其中分组的合理性会对算法产生重要的影响。现在多用 D.E.Knuth 的分组方法。

Shell 排序比冒泡排序快 5 倍，比插入排序大致快 2 倍。Shell 排序比起 QuickSort，MergeSort，HeapSort 慢很多。但是它相对比较简单，它适合于数据量在 5000 以下并且速度并不是特别重要的场合。它对于数据量较小的数列重复排序是非常好的。

5 插入排序（InsertSort）

插入排序通过把序列中的值插入一个已经排序好的序列中，直到该序列的结束。插入排序是对冒泡排序的改进。它比冒泡排序快 2 倍。一般不用在数据大于 1000 的场合下使用插入排序，或者重复排序超过 200 数据项的序列。

6 冒泡排序（BubbleSort）

冒泡排序是最慢的排序算法。在实际运用中它是效率最低的算法。它通过一趟又一趟地比较数组中的每一个元素，使较大的数据下沉，较小的数据上升。它是 O(n^2)的算法。

7 交换排序（ExchangeSort）和选择排序（SelectSort）

这两种排序方法都是交换方法的排序算法，效率都是 $O(n^2)$。在实际应用中处于和冒泡排序基本相同的地位。它们只是排序算法发展的初级阶段，在实际中使用较少。

8 基数排序（RadixSort）

基数排序和通常的排序算法并不走同样的路线。它是一种比较新颖的算法，但是它只能用于整数的排序，如果我们要把同样的办法运用到浮点数上，我们必须了解浮

点数的存储格式，并通过特殊的方式将浮点数映射到整数上，然后再映射回去，这是非常麻烦的事情，因此，它的使用同样也不多。而且，最重要的是，这样算法也需要较多的存储空间。

9 总结

下面是一个总的表格，大致总结了我们常见的所有的排序算法的特点。

排序法	平均时间	最小时间	最大时间	稳定度	额外空间	备注
冒泡排序	$O(n^2)$	O(n)	$O(n^2)$	稳定	O(1)	n 小时较好
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	不稳定	O(1)	n 小时较好
插入排序	$O(n^2)$	O(n)	$O(n^2)$	稳定	O(1)	大部分已排序时较好
基数排序	$O(\log_R B)$	O(n)	$O(\log_R B)$	稳定	O(n)	B 是真数(0-9)，R 是基数(个十百)
Shell 排序	O(nlogn)	-	$O(n^s)$ $1<s<2$	不稳定	O(1)	s 是所选分组

快速排序	O(nlogn)	$O(n^2)$	$O(n^2)$	不稳定	O(logn)	n 大时较好
归并排序	O(nlogn)	O(nlogn)	O(nlogn)	稳定	O(n)	要求稳定性时较好
堆排序	O(nlogn)	O(nlogn)	O(nlogn)	不稳定	O(1)	n 大时较好