**Java 2 Student Manual**

**Education Unlimited Summer 2017**

**Command Line Basics:**

Open your terminal. Here you can type commands. Commands can have parameters, which are optional letters you type after the command after a dash (-). Press “Tab” to complete the time of what you’re typing. “.” is the current folder, “..” is the parent folder, “\*” is all files in your location. Common commands:

* ls: list all files in the current location
* cd: change directories.
  + Example: if you have a folder called “stuff” in your location, typing “cd stuff” will move you into the folder “stuff.”
  + Typing “cd ..” will take you up to the parent folder
* mkdir folderName: makes a new folder with the name “folderName”
* rmdir folderName: removes the folder called “folderName”
* rm fileName: removes the file called “fileName”
  + Typing “rm \*” will remove all the files in your current directory
* nano fileName: very simple file creator/editor. Only supports typing. When done, type “control-x”, then “y”.
* clear: clears the screen
* javac fileName.java: compiles the Java file
* java fileName.java: runs Java file if it has a name

**Git Basics:**

* Create a github account here: <https://github.com/>, download git to your computer.
* Creating a local repo: When creating a new project on your local machine using git, you'll first create a new repo. To use git you'll be using the terminal. To begin, open up a terminal and move to where you want to place the project on your local machine using the cd command. Then use the mkdir command to make a folder for your project. Move into the folder. To initialize a git repository in the folder, run “git init”
* Creating and committing a file: Go ahead and add a new file to the project, using any text editor you like. After creating the new file, you can use the “git status” command to see which files git knows exist. The response basically says is, "Hey, we noticed you created a new file, but unless you use the 'git add' command we aren't going to do anything with it." Add the file to the staging environment using the “git add” command. Run the command “git commit -m "Your message about the commit."” The message at the end of the commit should be something related to what the commit contains - maybe it's a new feature, maybe it's a bug fix, maybe it's just fixing a typo.
* Creating a github repo and connecting it to your local: To create a new repo on GitHub, log in and go to the GitHub home page. You should see a green '+ New repository' button. After clicking the button, GitHub will ask you to name your repo and provide a brief description. When you're done filling out the information, press the 'Create repository' button to make your new repo. GitHub will ask if you want to create a new repo from scratch or if you want to add a repo you have created locally. In this case, since we've already created a new repo locally, we want to push that onto GitHub so follow the **'....or push an existing repository from the command line'** section. Now we'll **push** the commit in your branch to your new GitHub repo. To push changes onto a new branch on GitHub, you'll want to run “git push origin yourbranchname”. If you don’t know your branch name, run the command “git branch” and the name with an asterisk (\*) next to it is the branch you are on.

**NetBeans**



* The white rectangle with green plus is add new file, the yellow one next to it is add new project. The Green arrow is run, the smaller green arrow to the right of it is Step Through, or debug: goes through your code step-by-step.

**Arrays**

Arrays in Java: An indexed list of elements, all of which have the same type. Indices begin at 0 (ie. 0, 1, 2, 3, …)

* + To create a new array: dataType[] arrayName = new dataType[arraySize];
    - Example: String[] classNames = new String[20];
  + To access an indexed element of an array: arrayName[index];
    - Example: classNames[5]= “Jonah”
    - Example: firstStudent = classNames[0]

**ArrayLists**

Java’s most commonly used List implementation.

Construction syntax for an ArrayList of Strings called obj:

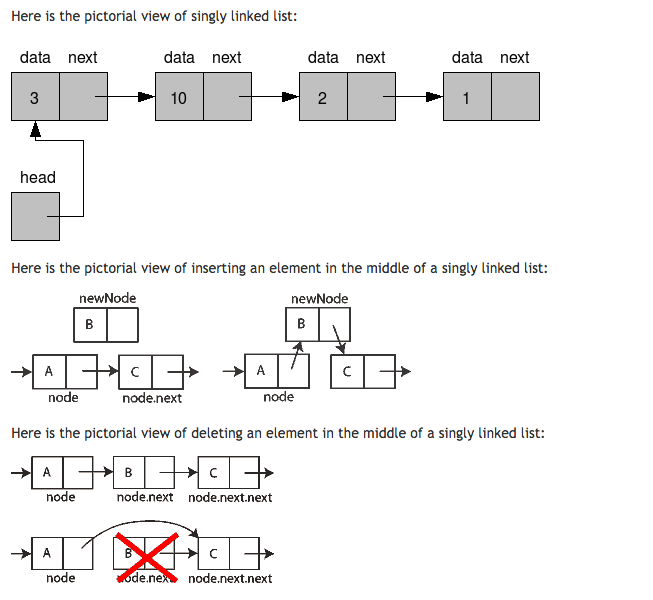
ArrayList<String> obj = new ArrayList<String>();

Important methods:

* add(String s) adds String s at the end of the list
* remove(String s) removes String s from the list
* add(int index, String s) adds String s at that index in the list
* remove (int index) removes the String at that index from the list

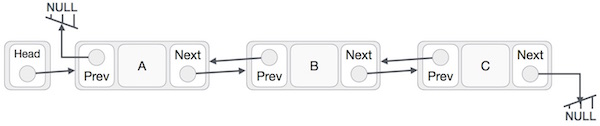
**Singly Linked List**

In a singly linked list, each node in the list stores the contents of the node and a pointer to the next node in the list. It is called a singly linked list because each node only has a single link to another node. To store a single linked list, you only need to store a pointer to the first node in that list. The last node has a pointer to nothingness to indicate that it is the last node.



**Doubly Linked List**

In a doubly linked list each node in the list stores the contents of the node, a pointer to the previous node in the list, and a pointer to the next node in the list. It is called a doubly linked list because each node has two links to other nodes. To store a doubly linked list, you only need to store a pointer to the first node in that list. The last node has a pointer to nothingness to indicate that it is the last node.

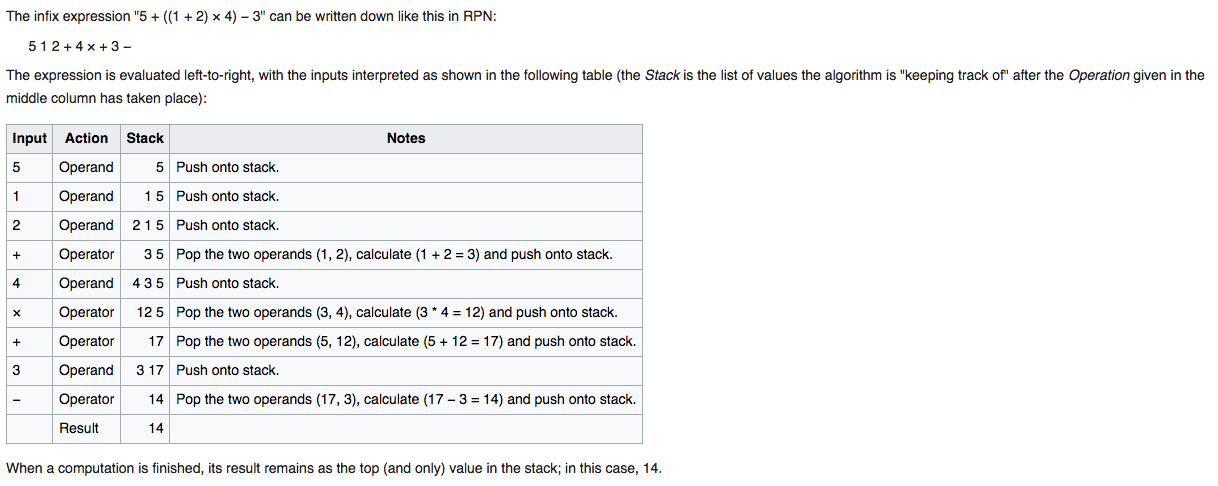


**Reverse Polish Notation - Calculators**

Reverse Polish notation is a mathematical notation in which every operator follows all of its operands, in contrast to Polish notation (PN), which puts the operator before its operands. It is also known as postfix notation. Examples:

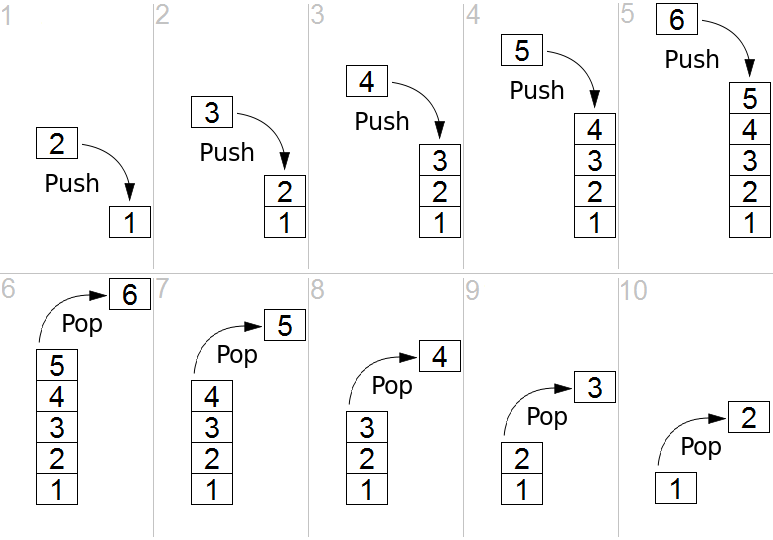
* + 3+4 is written as 3 4 +
  + 5-2 is written as 5 2 -
  + (3 - 4) x 5 is written as 3 4 - 5 x or 5 3 4 - x

Step-by-step longer example:



**Stacks**

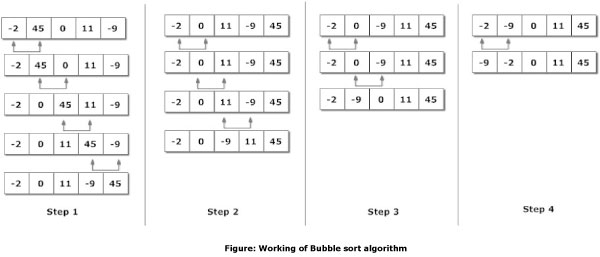
A stack is a last in, first out data structure. Stacks have two principal operations: *push*, which adds an element to the collection, and *pop*, which removes the most recently added element that was not yet removed. Additionally, apeek operation may give access to the top without modifying the stack.



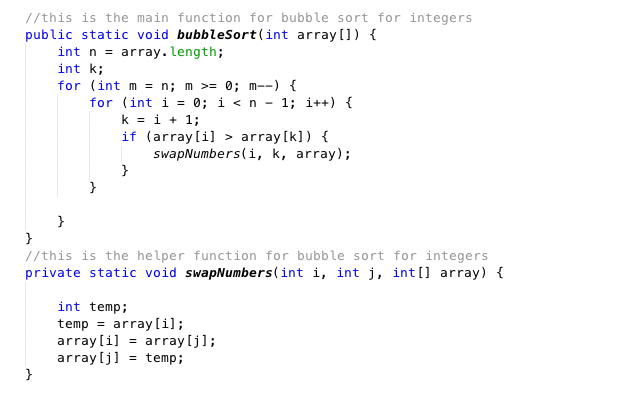
**Sorting Algorithms**

**Bubble Sort**

Bubble sort is a simple sorting algorithm. This algorithm is not suitable for large data sets. Bubble sort compares each pair of adjacent values and swaps them if they are not in the correct order. It continues until it does not have to perform any more swaps, at which point the data is sorted.

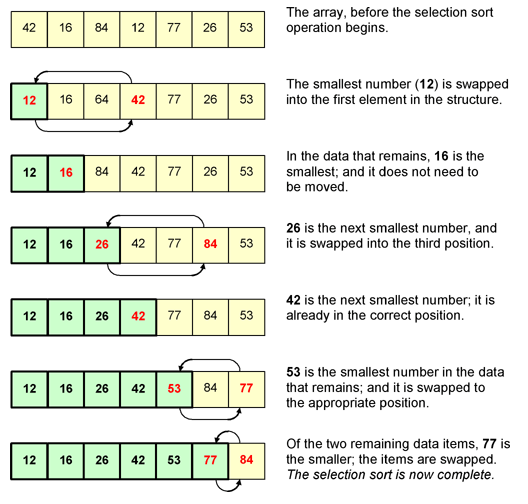


Java implementation:

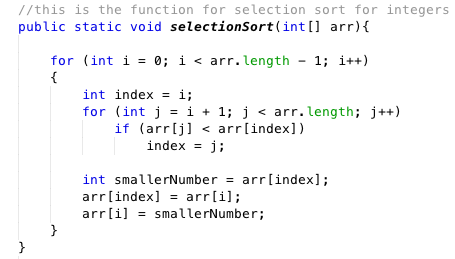


**Selection Sort**

Selection sort creates a sub-list which is always sorted, adding to it from smallest to largest element. It goes through the list and selects the smallest element, and puts it at the front of the list. Then, it continues to scan through the rest of the list, each time selecting the smallest element left and putting it behind the other small elements in the list.

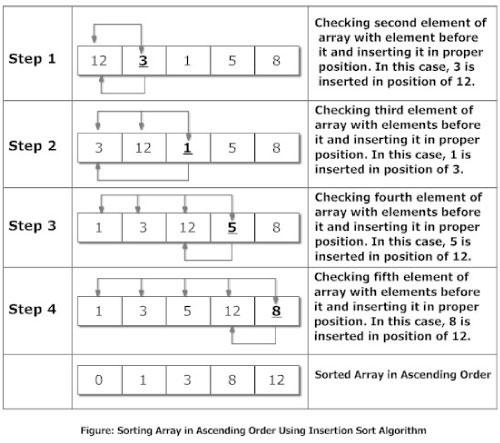


Java implementation:



**Insertion Sort**

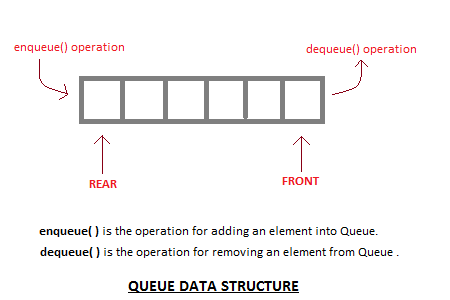
Insertion sort creates a sub-list which is always sorted. It first ensures the first 2 elements are in order. Then, for every other element, it scans through the sorted sublist and inserts the element into the correct sorted spot.



You will create your own Java implementation for Insertion Sort in class.

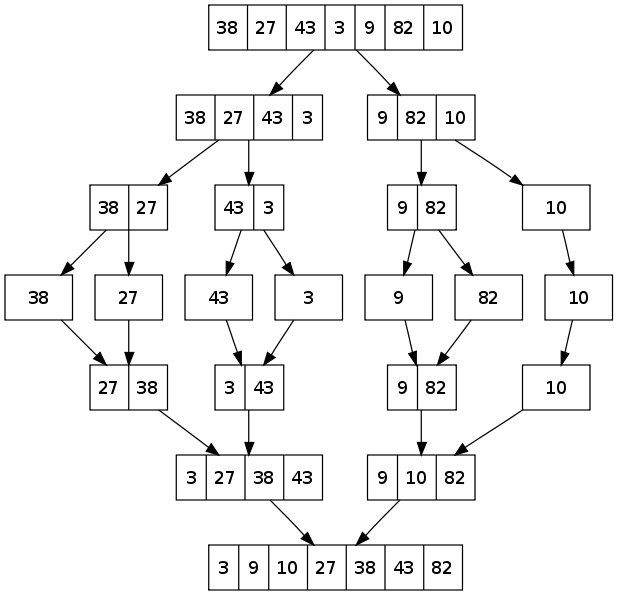
**Queues**

A queue is a first in, first out data structure. Queues have two principal operations: *enqueue*, which adds an element to the collection, and *dequeue*, which removes the least recently added element that was not yet removed. Additionally, apeek operation may give access to next element without modifying the queue.



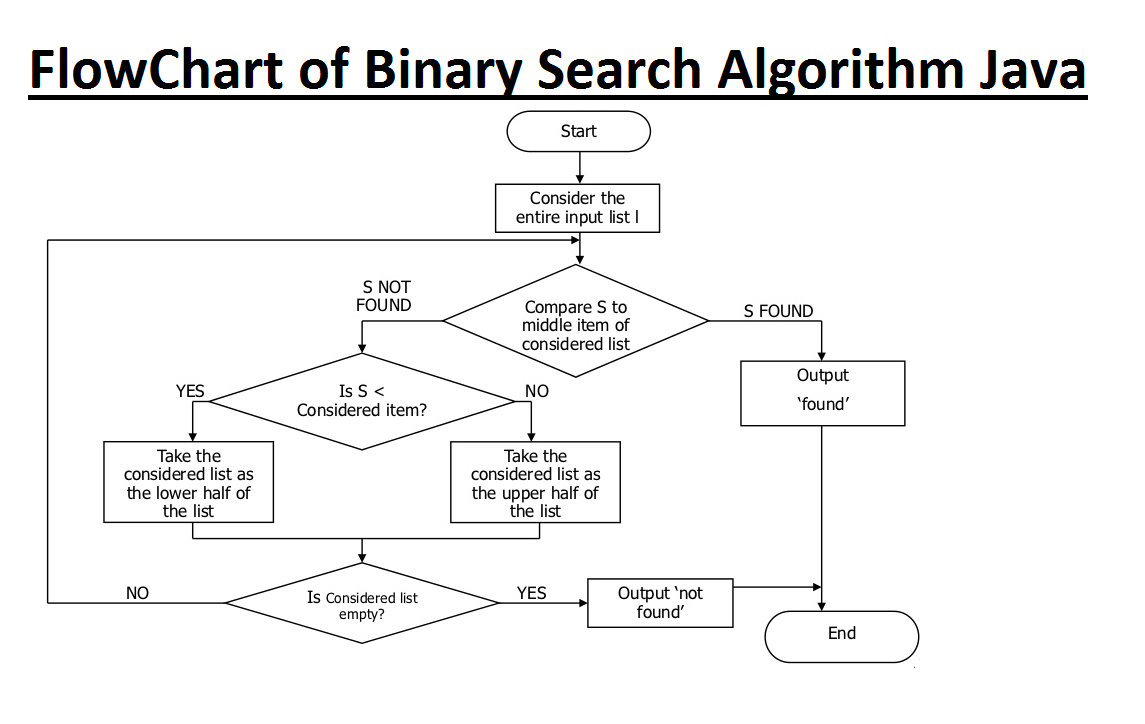
**Merge Sort**

This sorting algorithm starts by dividing the list in half repeatedly, until the sublists are only one element long. These lists are obviously sorted. Then, they are repeatedly merged together in sorted order until there is only 1 sublist left - the original list, now sorted. Merge sort often uses a “workspace”, a second list of the same size, to keep track of the sorted sublists while they are being merged. The sorted sublists are then copied back into the original list.

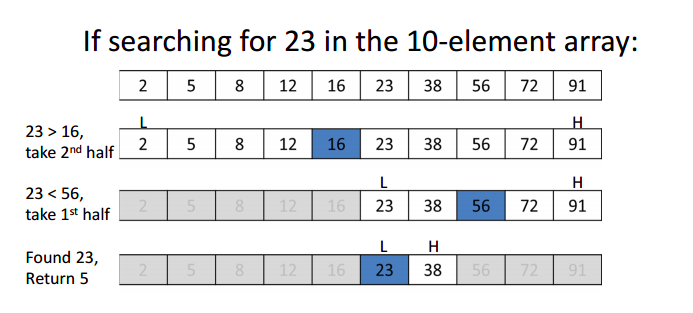


**Binary Search**

Binary search is an algorithm for searching for elements in an array that is assumed to be sorted in order from lowest to highest. Below is a flowchart of how the algorithm works:

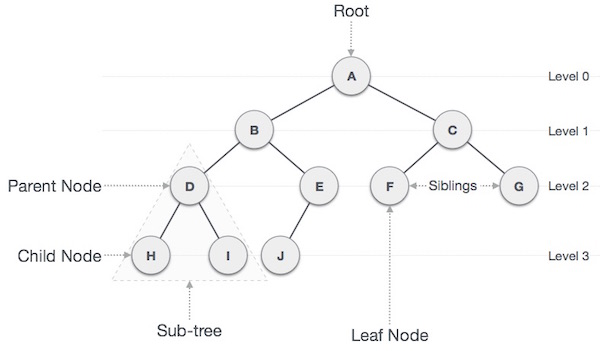


An example of binary search on an array of integers:



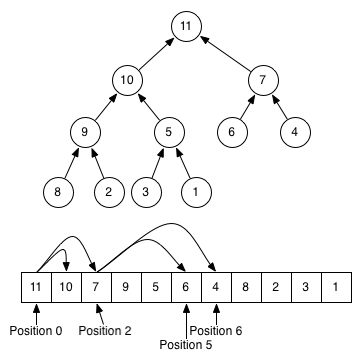
**Trees**

A tree is a data structure that simulates the structure of a tree, with a root value parent and children nodes. Each node consists of a value and references to other nodes, children, with no two parent nodes pointing to the same child and no parent node with the root as a child. A binary tree has each parent with at most 2 children.



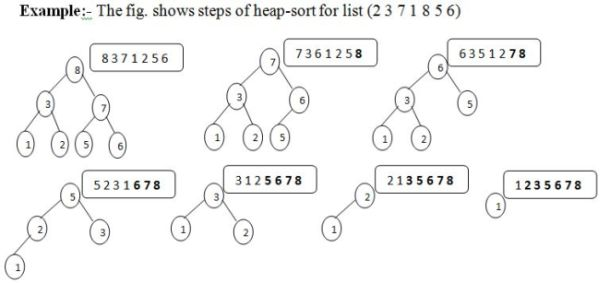
**Heaps**

A heap is a tree with special properties. All heaps are complete binary trees, which means every parent has exactly 2 children except the ones at the lowest depth, which are filled from left to right. A min-heap has the property that the value of all parents is less than or equal to their children, so the root will be the minimum. A max-heap has all parents with values greater than or equal to their children, with a maximum root.



**Heap Sort**

Heap sort is a sorting algorithm that uses a heap. It uses a process by which data is sorted by creating a heap and repeatedly picking off the top node (min or max) and re-heapifying the remaining data.



**Other Programming Languages**

**Python**

Python is an interpreted, high-level programming language. It used general-purpose, so it can be used for a lot of different types of projects. Python differs from many other languages in its use of whitespace instead of punctuation, and its extremely efficient and succinct syntax.

**C**

C is an imperative programming language with a static type system. C has a memory-use system that uses pointers, and is very fast. C is often used in operating systems, comilers, editers, databases, and database management systems.

**C++**

C++ is built off of C, so it is compatible with most C programs. It’s a compiled language, and can be one of the fastest languages when used correctly. It is an object-oriented language. Like C, C++ has low-level memory manipulation (pointers).

**Racket**

Racket is a multi-paradigm programming language. It is related to well-known languages Lisp and Scheme. It is used as a platform to develop other languages, for scripting, and for graphics. It is considered a great learning language.