505Sorting Algorithms

Terminology:

Stable - an algorithm that doesn’t change the relative ordering of elements with the same key

Inversions - amount of out of place values, which will determine how many swaps and comparisons will take place

Exchange - Swapping adjacent records

Insertion Sort - sort all numbers before n, use stacks

```
    static <E extends Comparable <? Super E>>
    void insert(E[] A)
    { for (int i=1; i<A.length; i++){
        for(int j=i;j>=;j--){
        DSutil.swap(A, j, j-1);
    }
```

Runtime = theta of(n^2)

Best case omega of (n-1)

Average case = theta (n^2)

Bubble Sort

```
    static<E extends Comparable<? Super E>>
    Void bubblesort(E[] Array){
    for(int i=0;i<i.length-1;i++){
        if(Array[i]>= Array[I+1]){
        Dsutil.swap(Array, i, i+1)
    }
    }
```

Theta of (n^2) comparison

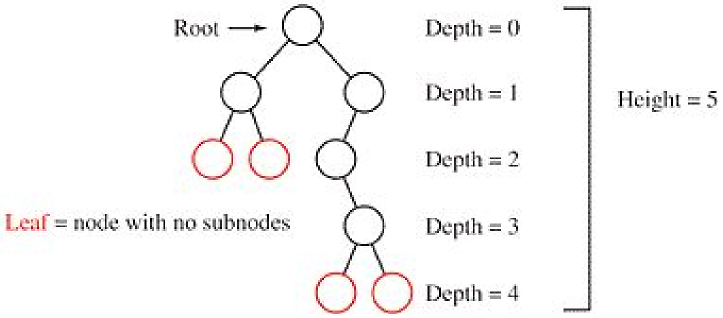
Selection Sort - finds the smallest, second smallest….so on.

```
    static<E extends Comparable<? Super E>>
    Void select Sort(E[] Array){
    for(int i=Array.length-1;i>0;i--){
    Int lowindex = il
        for(int j=0;j<i;j++){
            if(A[j].compareTo(A[lowindex] <0){
                low index = j;
                DSutil.swap(A, lowindex, j);
    }
    }
    }
```

	Insertion		
Comparisons	Θ(n)	Θ(n 2)	Θ(n 2)
Best Case	Θ(n^2)	Θ(n^2)	Θ(n^2)
Average Case	Θ(n^2)	Θ(n^2)	Θ(n^2)
Worst Case	Θ(n^2)	Θ(n^2)	Θ(n^2)
Swaps			
best	0	0	Θ(n)
Average	Θ(n^2)	Θ(n^2)	Θ(n)
Worst	Θ(n^2)	Θ(n^2)	Θ(n)

Shell Sort -
Uses disjointed increments which are equidistant away.

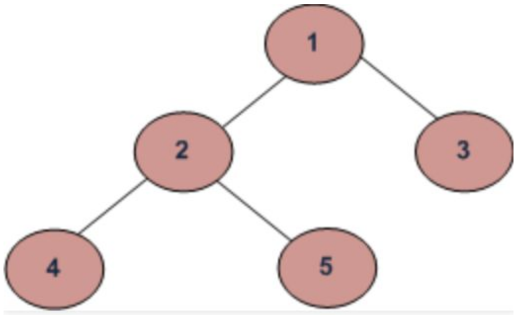
Binary Search Tree



A complete binary tree has restricted shape obtained through starting at the root and filling the tree from left to right, all levels except d-1 is completely filled
Each level d has 2^d nodes, and a complete tree has at most 2^n+1 -1 nodes
Full tree- where each node either has 0 or 2 children

Full Binary Tree Theorem: The number of leaves in a non-empty full binary tree is one more than the number of internal nodes.

Theorem 5.2 The number of empty subtrees in a non-empty binary tree is one more than the number of nodes in the tree.



Depth First Traversals:

(a) Inorder (Left, Root, Right) : 4 2 5 1 3
Fives nodes in non-decreasing order
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
Used in creating copy of the tree and getting the prefix expression
(c) Postorder (Left, Right, Root) : 4 5 2 3 1
Used in deleting the tree

```
void print Postorder(Node node)
   {
      if (node == null)
         return;

      // first recur on left subtree
      print Postorder(node.left);

      // then recur on right subtree
      print Postorder(node.right);

      // now deal with the node
      System.out.print(node.key + " ");
   }

   /* Given a binary tree, print its nodes in inorder*/
   void printInorder(Node node)
   {
      if (node == null)
         return;

      /* first recur on left child */
      printInorder(node.left);

      /* then print the data of node */
      System.out.print(node.key + " ");
```

```
        /* now recur on right child */
        printInorder(node.right);
    }

    /* Given a binary tree, print its nodes in preorder*/
    void printPreorder(Node node)
    {
        if (node == null)
            return;

        /* first print data of node */
        System.out.print(node.key + " ");

        /* then recur on left subtree */
        print Preorder(node.left);

        /* now recur on right subtree */
        print Preorder(node.right);
    }
```

Stack

Stacks are a data structure where elements can only be pushed and popped from one end, LIFO - last in first out

Push - insertion

Pop - removal (The pop function also returns a copy of the value it removes)

Top - the first empty array index, length == top

Linked Stack vs Array Stack
All operations for both take constant time.

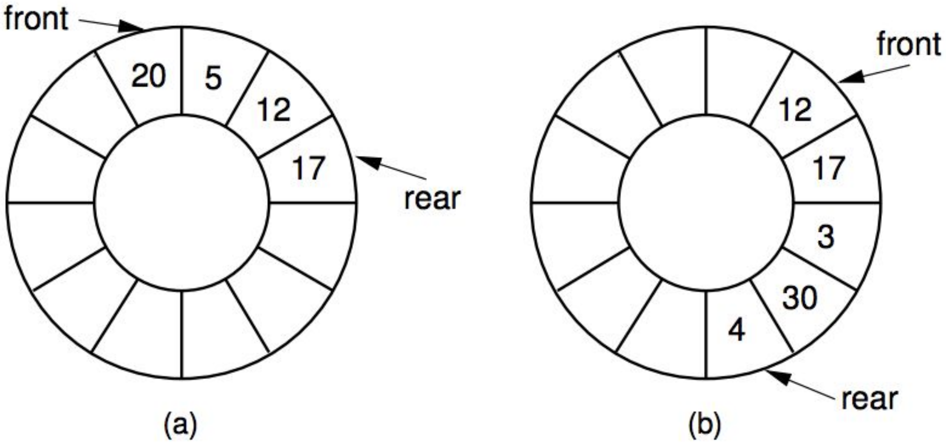
However, since a array stack requires a fixed size array at declaration, which means it will take up more space.

```
/** @return n! */
static long fact(int n) {
Stack<Integer> S = new AStack<Integer>(n);
long result = 1;
while (n > 1) {S.push(n--); }//make stack 1234….n-1
 while (S.length() > 0) result = result * S.pop(); return result;
```

}

Queues

The end and beginning of queues will slowly drift to higher numbers, however, this can be solved by implementing a circular queue by using %n-1;



The front always denotes the first non-empty node and the rear denotes the last non-empty node. if(front == (rear+2) % n-1), then the queue is full.

AQueue

- Has a max size
- Wastes spaces
- Empty queue can't be distinguished from full queues
- public void clear() { rear = 0; front = 1; }
- An astack of size n will always be of size n+1 so that the end and front can be distinguishable.

Lists

An abstract data type (ADT) is the realization of a data type as a software component.

interface of the ADT is defined in:

a type, a set of operations on that type.

The behavior of each operation is determined by its inputs and outputs. **An ADT does not specify how the data type is implemented**

E - Generics that can be anything ArrayList<E> can have anything.

Big O notation

Describes the worst case run time or upper bound

Big Omega notation

Describes the best case run time or lower bound

Theta notation
Theta of n is the tight bound when the lower bound is equal to the upper bound

Nodes - distincts objects

Array based list implementation
In order to insert something into a list, all other members after the insertion index must be shifted by one, therefore, the time it takes to insert at location a is equal to Θ(ArraySize - a)
In the average case, insertion or removal requires moving half of the elements, which is Θ(n).

Linked Lists
The linked list uses **dynamic memory allocation**, that is, it allocates memory for new list elements as needed.

The list's first node is accessed from a pointer named head, last node accessed from pointer tail. The position of the current node can be accessed through curr. Cnt stores the length of the list.
As the linked list does not require a size upon initialization, It is very important to give a linked list a minimum size to avoid a null linked list.

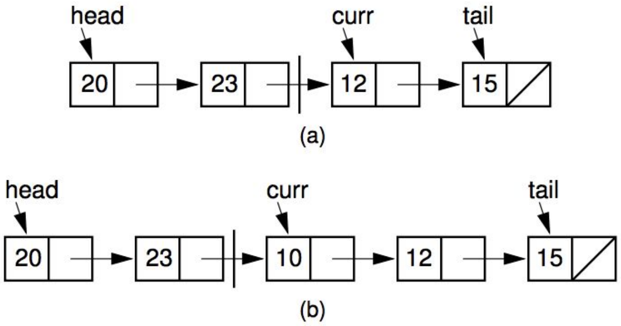


Figure 4.5 Illustration of a faulty linked-list implementation where **curr** points directly to the current node. (a) Linked list prior to inserting element with value 10. (b) Desired effect of inserting element with value 10.

insert

When inserting a new element to the list, in this example, to set curr to the preceding node(23), and insert 10. (This may be troubling when inserting a value to the beginning of the linked list, however, it could be avoided by the implementation of a **header node**)

```
public void insert(E it) {
curr.setNext(new Link<E>(it, curr.next())); // set the next node into value, index
if (tail == curr) tail = curr.next(); // Set new tail
cnt++;
}
```

Remove

```
public E remove() {
if (curr.next() == null) return null; // Nothing to remove
E it = curr.next().element(); // Remember value to remove
if (tail == curr.next()) tail = curr; // Removed last
curr.setNext(curr.next().next()); // set the value after next as the next value
cnt--; // Decrement count
return it; // Return value
}
```

Removing an element requires Θ(1) time

```
Public void swap(){
E holder = this.remove();
next();
insert(holder);
}
```

4.6 Add to the List class implementation a member function to reverse the order of the elements on the list. Your algorithm should run in Θ(n) time for a list of n elements.

```
Public void swap(){
        while(curr.next() !=)
        }
```

E.7 Write a function to merge two linked lists. The input lists have their elements in sorted order, from lowest to highest. The output list should also be sorted from lowest to highest. Your algorithm should run in linear time on the length of the output list.

```
(){


}
```

Test 1 Study Outline

ArrayList (operations from slides/reading)

Arraylist is an class that hosts an array which can hold reference types.

You can Change the size of an arraylist

equals vs. ==

== only considers two objects the same if they point to the same object

== only compares primitives, and only compares the same object in reference types, for example:
String str = new String(“HELLO”);
String str1 = new String(“HELLO”);
The expression str1 == str would return false because they are not the same object.

casting (when possible/appropriate)

Upcast
When you store a child object in a parent class type.
You can’t call child class specific variables using that parent variable.
Polymorphism means you can store the child in a parent class type.

Downcast
When you cast a parent type object back into the child type;

abstract classes

Contains Abstract method, a method that does nothing (a declaration but not definition)
EX : abstract void(int n);
You can use interfaces to force a class to implement certain methods.
Contain abstract methods that must be overridden and implemented in child classes.

up Interfaces are for when you want to say "I don't care how you do it, but here's what you need to get done."

Abstract classes are for when you want to say "I know what you should do, and I know how you should do it in some/many of the cases."

Basically pointless if never extended, can’t create other methods or other objects in the same name of the abstract class.

Object class
Every Class created in java extends object class.
Thereby, when you define a toString method in your class, you're overriding the toString method in Object.
is-A vs. has-A relationships
Is-a: If a class extends another, it is that class. EX: if car extends machine, it is a machine.
Has-A: another class that has an instance variable of type machine.
It's usually safer to use Has-A correlations to avoid casting problems.
method overriding
Child class methods with the same name will override parents class method unless specified by super.

class hierarchies and how extended classes relate (think the 'valid/invalid' HW exercises)

Class/Generics Terminology

Interface - the beginning code of a class only containing of parameters or variables and no method

Instance Variable - variable defined in a class where each instantiated object of the class has a seperate copy(instance), similar to a class variable.

Constructor - special type of method that is used to initialize the object.
Name = class name
No explicit return type

Generic classes/Interface - Class or Interface or more than one type.
Type variables are delimited by angle brackets and follow the class name

Covariant - array is said to be covariant, because it contains both a type and subclasses.

Contravariance - Does not preserve the ordering of types

A type - is a collection of values

Aggregated, or composite types - a type containing many values

Upcast
When you store a child object in a parent class type.
You can’t call child class specific variables using that parent variable.

Downcast
When you cast a parent type object back into the child type;

```
Fruit mystery = new Fruit();
Fruit mystery = (Apple) Fruit;
```

Access Modifiers in Java
Public - default - could be accessed anywhere
Private - only the class the method is declared in can access this method(Cannot even be accessed through super)
Protected - can be accessed through the class itself, or any other class in that package.

Equals vs ==
== only compares primitives, and only compares the same object in reference types, for example:

```
String str = new String(“HELLO”);
String str1 = new String(“HELLO”);
```

The expression str1 == str would return false because they are not the same object.

Equals can be overridden based on specific implementations in Eclipse to compare only certain attributes of different objects;

Return type covariance
.equals can be overridden to return type B for .equals of A id B is a child of A. So to return certain classes Apple in type Apple but not Apple in type Object

Abstract Classes
Abstract method is one that does nothing
You can use interfaces to force a class to implement certain methods.
Contain abstract methods that must be overridden and implemented in child classes.

up Interfaces are for when you want to say "I don't care how you do it, but here's what you need to get done."

Abstract classes are for when you want to say "I know what you should do, and I know how you should do it in some/many of the cases."

Basically pointless if never extended, can’t create other methods or other objects in the same name of the abstract class.

Advantage to interface - there are no multiple inheritance in java, so declaring classes as interfaces could allow multiple class inheritance.

Inheritance

Creating child classes using extend, if fuji is an apple, then fuji can do everything an apple can do, in addition to what’s specific to a fuji apple.

An instance variable will have a copy in Inherited child classes depending on its access modifiers.
Same applies to constructors, when not specified, default constructor in child class is constructor in parents class.

Polymorphism

Once method is defined in child’s class when using parents class.

```
Fruit b = new Banana();
b.grow banana();
```

Parents only know about parents methods, children know both child and parent methods.

Generics

Saves casting operations.

```
List box = new ArrayList<>;
Apple apple = new (Apple) box.get(0);
```

Type variables are delimited by angle brackets and follow the class name

```
void eat(List<? extends Fruit> fruits); applies to all types extended from fruits
```

You only need to cast if running a method not belonging to the parent class, such as loading in the vehicle case

Abstract Classes and ADT

An abstract data type (ADT) is the realization of a data type as a software component. The interface of the ADT is defined in terms of a type and a set of operations on that type. The behavior of each operation is determined by its inputs and outputs. An ADT does not specify how the data type is implemented. These implementation details are hidden from the user of the ADT and protected from outside access, a concept referred to as encapsulation

Subclasses of abstract classes can also be abstract.

toString = default method when nothing happens

there is a golden rule of programming = don't start planning the progarm until every aspect is crystal clear