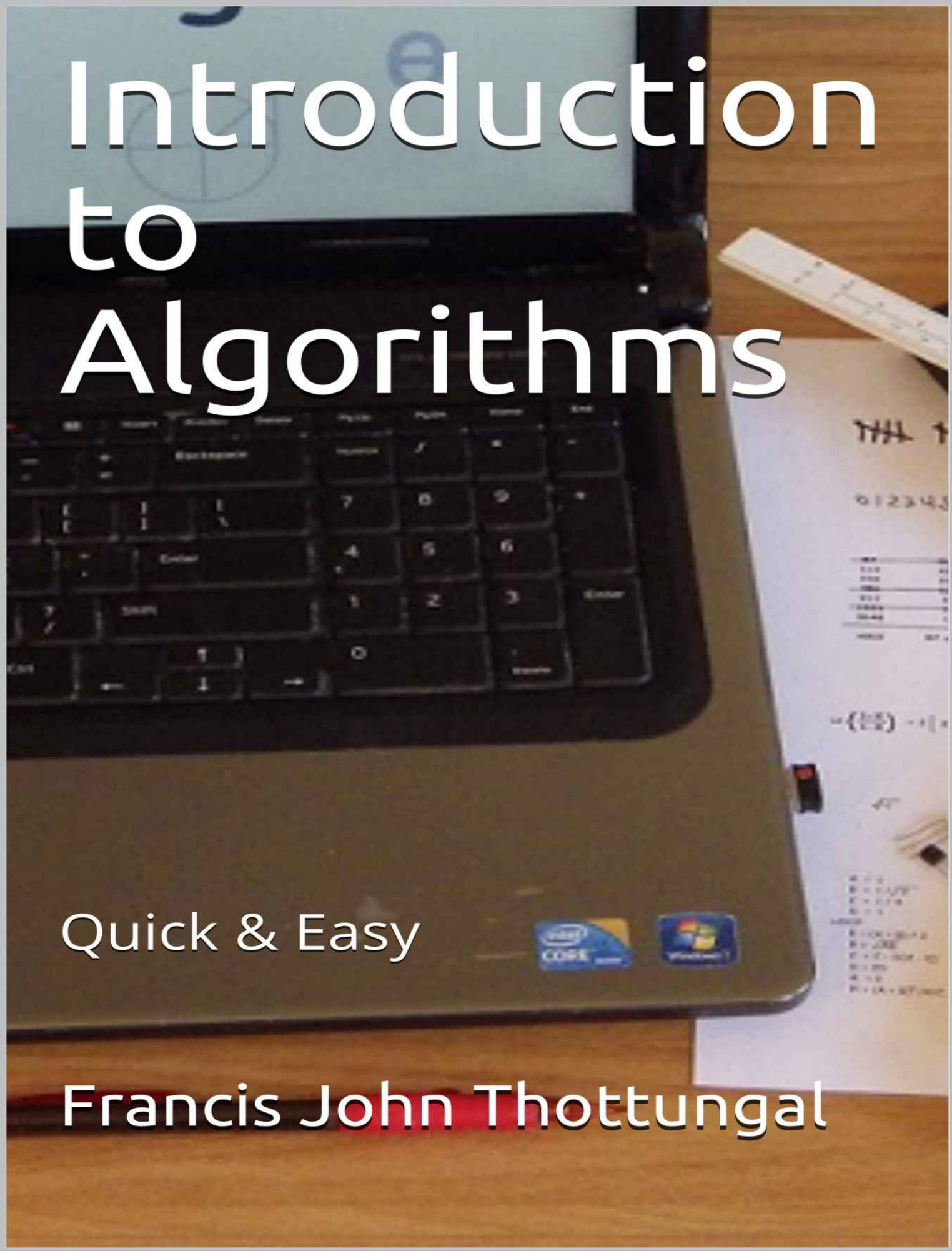
Introduction to Algorithms



Quick & Easy

Francis John Thottungal

Table of Contents

Overview

Algorithms as a subject are a fantastic part of computer science. It provides for a disciplined way of writing efficient programs. The prerequisite to understanding algorithms is knowledge of data structures such as arrays, lists, stacks, queues, graphs etc.

Algorithms are key to writing programs. It is the backbone of any program. In fact, the difference between software and programs for the same purpose is how they differ in their algorithms.

Thus algorithms in simple English terms can be considered as the way to do something. When algorithms are married to the correct programming language for the purpose in hand then an efficient and accurate program can be created.

The objective of this book is to illustrate various algorithms in a way that the reader can easily grasp their working. In addition to the twenty algorithms and the additional ones in chapter twenty-one, the appendix gives insight into various aspects of Java relevant to writing programs for algorithms.

The code for each of the twenty algorithms are for those who would like

to use the book for a laboratory or project or just for programming as a hobby. The best way to understand the illustrated algorithms is to work out the examples on paper with pen or pencil. I always prefer the latter. The code where available can be run to test if the answer obtained manually is correct or not.

In addition to knowledge of data structures, this book assumes a basic knowledge of Java and calculus. The book can be considered as a quick guide to various algorithms.

The code shown in this book was tested on various java editors including Jcreator, Netbeans and Eclipse among

others. The recommended Java JDK is 1.6 or higher for the code in this book.

The website for this book is - http://beam.to/fjtbooks

Before we start looking at understanding various algorithms:

An algorithm can be defined as a sequence of unambiguous instructions for solving a problem. I believe we have stated above that all computer programs are based on algorithms. Each feature of a software is based on an algorithm and when put together a collection of algorithms becomes a software package. But algorithms are not just for computer programs. Long before the computer of today came around humans have been

using algorithms to solve mathematical problems or describe solutions to various challenges in the means of steps which we now term as algorithms. Thus algorithms are something common to many disciplines.

In programming this means an output can be obtained over a finite period of time given certain correct inputs.

The term algorithm design is a general approach to solving problems. Algorithms differ for the same problem in terms of their efficiencies. There are two main measurements of efficiencies- Time and spatial. Time efficiency is based on the speed at which an

algorithm achieves its goal. Spatial or space efficiency is about the memory units required by the algorithm in addition to that required for its input and output. Thus we can say that when we start designing an algorithm we are looking at maximizing time and spatial efficiencies.

The best way to measure time efficiency is by counting the number of times the algorithm's basic operation is executed on inputs of size n. Let 't' be the execution time of an algorithm's basic operation on a particular computer, and let 'c' be the number of times this operation needs to be executed for this algorithm. Then we can estimate the total

time on that particular computer to be given by t C. The challenge for algorithm designers then would be to identify the correct basic operation.

Efficiency of an algorithm can be affected by the specifics of the input and its size. We can classify worst case, best case and average case scenarios for algorithms based on their efficiencies.

In a linear search or sequential search where each element is checked against the key being searched for the worst case is when there are no matching elements or it is the last element in the list. This increases the time needed to find the element. Therefore, the longer

the search time or conclusion time we can state that as the worst case scenario of an algorithm.

The best case efficiency for our example of the sequential search is when what we are looking for is the first element in the list or near to the beginning. In our example the average time needed to an element in the list is the average efficiency of the algorithm.

For further concepts check out Appendix I. Let us now look at various algorithms in terms of what they do.

Chapter 1. Euclid Algorithm

In this chapter, we will start with the Euclid method to find the greatest common denominator of two numbers. First let us understand or go through the method:

GCD: If we want to the find the GCD of two numbers m & n such that m is greater than n we do the following (this is called an algorithm):

1. Divide m by n and find the remainder

2. Replace m by n and n by r (the remainder)

3. Repeat the steps until n becomes 0.

If n is zero to start with then the answer is m. If n is greater than m, then just reorder them before doing the above steps.

Let us look at an example:

GCD of (75, 9)

Step 0: 75, 9

Step 1: 9,3

Step 2: 3,0

Answer is 3

Now try to find the GCD of the following by hand:

GCD of (60, 24)

Is the answer 12?

Let us look at the above steps. As algorithms can be considered as a collection of steps to accomplish a goal let us now look at the java code that will accomplish the steps show above.

```
import java.math.*;
```

```
public class lab1 {

public static void main(String[] args) {
BigInteger n1, n2, n3;
n1 = new BigInteger("60");
n2 = new BigInteger("24");
n3 = n1.gcd(n2);
String y = "GCD of " + n1 + " and " + n2
System.out.println( y );
}
}
```

Run the code and enter the values that were done by hand. Do we get the same answer?

Here we are using the BigInteger

variables in Java. The appendix at the end of this book has more features on it. The BigInteger was chosen because it will play important roles in Java with respect to GCD especially in the field of encryption and decryption.

Here three variables n1, n2 and n3 have been defined and n3 will hold the GCD value of n1 and n2. Note that the algorithm we mentioned in the previous page is being done by the inbuilt GCD function in Java's math package.

String y variable will then be used to print out the final value of n3. We could directly print out n3. Try it.

Exercises:

1. Find the GCD of 2024 and 54 by hand before running the program above.

2. Find another two ways by which GCD can be done? Write down their algorithms and see if they are more efficient than the one presented here.

3. As a programming exercise try to modify the program above to accept inputs from the keyboard using the scanner or similar method in Java?

Chapter 2. Fibonacci Numbers

Fibonacci numbers have a unique sequence as shown below:

0, 1, 1, 2, 3 , 5 , 8 , 13 , 21 , 34 , ……

This sequence follows the equation:

$F(n) = F(n-1) + F(n-2)$ for $n > 1$

With two initial conditions: $F(0) = 0$ and $F(1) = 1$

Thus the algorithm for this could be written in this manner:

a. Input a non-negative number n
b. Set F (0) =0 and F (1) =1
c. For i =2 to n do F (i) = F (i-1) + F (i-2) and return F (i)
d. Output the n Fibonacci numbers.

```
import java.util.*;
class lab2{
static int x1=0,x2=1,x3=0;
static void Fibonacci(int max){
if(max>0){
x3 = x1 + x2;
x1 = x2;
x2 = x3;
System.out.print(" "+x3);
```

```
Fibonacci(max-1);
}    }
public static void main(String args[]){
Scanner sc= new Scanner(System.in);
System.out.println("Please enter the max
fibonacci numbers to be created");
int max = sc.nextInt();
System.out.print(x1+" "+x2);
Fibonacci(max-2);
}  }
```

Chapter 3. Selection Sort

Selection sort is performed by scanning the entire given list to find the smallest element to exchange it with the first element. Then we scan the list, starting with the second element to find the smallest among the n-1 elements and exchange it with the second element and so on. After n-1 passes the list will be sorted.

Let look at an example by looking at the following list of numbers: 89, 45, 68, 90, 29, 23, 15.

89 45 68 90 29 23 15

15 45 68 90 29
23 89

15 23 68 90 29
45 89

15 23 29 90 68
45 89

15 23 29 45 68
90 89

15 23 29 45 68
89 90

The algorithm that can do this can be written as follows:

a. Input an array of numbers A(0..n-1)
b. for i = 0 to n-2 do min <= i
c. for j <= i + 1 to n -1 do if
   A [j] < A [min] min <= j swap A [i]

= A [min]

Based on our example above.

Let us try another example: 50, 19, 67, 23, 100, 90, 43, 75, 15, 7

50 19 67 23 100 90 43 75 15 7
7 19 67 23 100 90 43 75 15 50
7 15 67 23 100 90 43 75 19 50
7 15 19 23 100 90 43 75 67 50
7 15 19 23 43 90 100 75 67 50
7 15 19 23 43 50 100 75 67 90
7 15 19 23 43 50 67 75 100 90
7 15 19 23 43 50 67 75 90 100

Try the following:

89 12 67 13 15 19 21 31 41 1 5 34 31
68

```
import java.util.*;
public class lab3
{
public static void main(String args[]) {
int[] array = {50, 19, 67, 23, 100, 90, 43
for(int x = 0; x < array.length; x++) {
System.out.print( " " + array[x]);
}
System.out.println(" ");
int y;
int tmp;
int count=1;
for ( int i=array.length-1; i>0; i--,count+
y = 0;
for(int j=1; j<=i; j++) {
```

```
if( array[j] < array[y] )
y = j;
}
tmp = array[y];
array[y] = array[i];
array[i] = tmp;
System.out.print("Pass " + count + ": " )
for(int x = 0; x < array.length; x++)
{
System.out.print( " " + array[x]);
}
System.out.println(" ");
}   } }
```

Exercise:

1. Run the hand exercise 89 12 67 13 15 19 21 31 41 1 5 34 31 68 and see if you

get the correct sort.

2. Modify the above code to accept input via a scanner method using the java.util package. The program should ask for the size of the array to start with and then allow the user to input numbers on a single line.

Chapter 4. Bubble Sort

In this method we have to compare adjacent elements of the list and exchange them if they are out of order thus bubbling up the largest element to the end of the list. On the second pass the second largest element is bubbled up and thus after n-1 passes the list is sorted.

Let us take the same example as in chapter 3 on selection sort:

89 45 68 90 29 23 15
45 89 68 90 29 23 15 (compare adjacent)
45 68 89 90 29 23 15

45 68 89 29 90 23 15

45 68 89 29 23 90 15

45 68 89 29 23 15 90

Now we start from the left again.

45 68 89 29 23 15 90

45 68 29 89 23 15 90

45 68 29 23 89 15 90

45 68 29 23 15 89 90

Now we start from the left again

45 68 23 29 15 89 90

45 68 23 15 29 89 90

Now we start from left again

45 23 68 15 29 89 90

45 23 15 68 29 89 90

45 23 15 29 68 89 90

Now we start from the left again

23 45 15 29 68 89 90

23 15 45 29 68 89 90

Now we start from the left again

15 23 45 29 68 89 90

15 23 29 45 68 89 90

Now let us look at the algorithm for this sort:

a. Input an array A [0...n − 1] of orderable elements and for i ←0 **to** n − 2 **do**
for j ←0 **to** n − 2 − i **do if** A [j + 1] < A [j] swap A [j] and A [j + 1]

Now let us look at a program in Java for doing this:

```
 import java.util.*;
public class lab4
{
```

```
public static void main(String []args) {
int n, x, y, swap;
Scanner in = new Scanner(System.in);
System.out.println("Input number of inte
n = in.nextInt();
int array[] = new int[n];
System.out.println("Enter " + n + " integ
for (x = 0; x < n; x++)
array[x] = in.nextInt();
for (x = 0; x < ( n - 1 ); x++) {
for (y = 0; y< n - x - 1; y++) {
if (array[y] > array[y+1]) /* For descen
{
swap       = array[y];
array[y]   = array[y+1];
array[y+1] = swap;
}
}
```

```
}
System.out.println("Sorted list of numbe
for (x = 0; x < n; x++)
System.out.println(array[x]);
}
}
```

Exercise:

1. Do a bubble sort on the following: 5 12 52 8 18 19 21 32 25 1 5 72 31 92 by hand.

2. As a computer exercise:

a. Change the mode of entry to make it possible to enter the array on the same line.

b. Also use a for loop or some other mechanism to show the output for each pass of the sorting technique.

3. Do a bubble sort on the following: 23 92 2 18 12 56 65 25 78 6 35

Chapter 5. Sequential Search

The algorithm simply compares successive elements of a given list with a given search key until either a match is encountered or the list is exhausted without finding a match. Let us look at an example: If we are looking for the number 15 in the example:

89 45 68 90 29 23 15 then we will go one by one through the list until match is found. In this case in the last position. The algorithm can be written as follows:

a. Input an array of numbers to be searched.

b. A[n]←M
i ←0
while A[i] _= M do
i ←i + 1
if i < n return i
else return −1

Now let us look at a code in Java to do this:

```
import java.util.*;
public class lab5{
public static void main(String[] args){
int[] x={89,45,68,90,29,23,15};
int y;
Scanner sc = new Scanner(System.in);
System.out.println("Please enter the num
searching for.");
```

```
y= sc.nextInt();
int pos=seqsearch(x,y,x.length);
if(pos!=-1) System.out.println(" The val
position of "+ pos);
}
public static int seqsearch(int[] x,int tar
int found=0;
int i;
int pos=-1;
for(i=0;i<n && found!=1;i++)
if(target==x[i]){pos=i;found=1;}
return pos;  }}
```

Chapter 6. Insertion Sort

This algorithm is an example of a decrease by one technique to sorting an array. In this sort we assume that the smaller problem of sorting the array A[0..n − 2] versus the original A[0...n-1]has already been solved to give a sorted array of size n -1.

Then we need is to find an appropriate position for $A[n-1]$ among the sorted elements and insert it there.

This is usually done by scanning the sorted subarray from right to left until the first element smaller than or equal to $A[n-1]$ is encountered to insert $A[n-1]$ right after that element.

Let us look at an example:

Let us take the number we have used before: 89 45 68 90 29 23 15.

89]45 68 90 29 23 15 (the starting point is a bracket after 89)
45 89]68 90 29 23 15 (45 moved and sorted)
45 68 89] 90 29 23 15 (68 moved and sorted)
45 68 89 90] 23 15 (90 was moved but after sorting it remains in the same location)
23 45 68 89 90] 15 (23 moved and sorted)
15 23 45 68 89 90 (now it is sorted)

The bracket] represents the point across which the numbers will be shifted left and then sorted.

We could write the algorithm for this sort as follows:

a. Input: An array $A[0..n-1]$ of n orderable elements

for $i \leftarrow 1$ **to** $n-1$ **do**

$v \leftarrow A[i]$

$j \leftarrow i-1$

while $j \geq 0$ **and** $A[j] > v$ **do**

$A[j+1] \leftarrow A[j]$

$j \leftarrow j-1$

$A[j+1] \leftarrow v$

Now let us look at a code:

```
import java.util.*;
public class lab6
{
public static void main(String a[]){
int[] x = {89,45,68,90,29,23,15};
int[] y = InsertionSort(x);
for(int i:y){
System.out.print(i);
System.out.print(", ");
}
}
public static int[] InsertionSort(int[] inp
int temp;
for (int i = 1; i < input.length; i++) {
for(int j = i ; j > 0 ; j--){
if(input[j] < input[j-1]){
temp = input[j];
```

```
input[j] = input[j-1];
input[j-1] = temp;
}}
}
return input;
}
}
```

Exercise:

1. Solve the following sequence of numbers: 78 90 34 56 12 23 90 102 using insertion sort.

2. Modify the code to do the following:

a. Accept the array elements from the keyboard using java's util package.

b. Accept input in a single line

c. Show the result of each sort phase.

3. What are the other algorithms that use the decrease by one method?

Chapter 7. Binary search

Binary search works by comparing a search key K with the array's middle element A[m]. If they match, the algorithm stops; otherwise, the same operation is repeated recursively for the first half of the array if K < A[m], and for the second half if K > A[m]. The algorithm can be written as follows:

a. Input: An array $A[0..n-1]$ sorted in ascending order and search key K
l←0; r ←n − 1
while l ≤ r do
m←_(l + r)/2_
if K = A[m] return m
else if K <A[m] r ←m − 1

```
else l ←m + 1
return −1
```

```
Public class lab7 {
public int binarysearch(int[] inputx, int
int startpoint = 0;
int endpoint = inputx.length - 1;
while (startpoint <= endpoint) {
int midpoint = (startpoint + endpoint) /
if (k == inputx[midpoint]) {
return midpoint;
}
if (k < inputx[midpoint]) {
endpoint = midpoint - 1;
} else {
startpoint = midpoint + 1;
}   }
return -1;
}
public static void main(String[] args) {
lab7 ar = new lab7();
```

```
int[] ax = {15,23,45,67,89,90};
System.out.println("Position of key 45:
ar.binarysearch(ax, 45));
}}
```

In the above code the array is being input in ascending manner. Had the number been not in order the result will be correct also. Here the location of the number 45 will be 2 as arrays start with position 0.

Another way to do ascending would be to use the Arrays.sort feature of java to sort the array first and then to use the binary search method.

```
import java.util.Arrays;
public class lab7b {
public static void main(String[] args) {
int x[] = {67,89,90,15,23,45};
Arrays.sort(x);
int z= x.length;
```

```
int i;
System.out.println("The sorted array is:"
for (i=0;i<z;i++) {
System.out.println("Number = " + i + "T
is" +x[i]);
}
int k = 45;
int y = Arrays.binarySearch(x,k);
System.out.println("The index of elemen
   }
}
```

Here the array elements are hardcoded and given to array x and printed one by one after they have been sorted through the method in the util package of Java. Then we search for 45 using the binarySearch method of Arrays which is

defined in the util package.

Exercise:

1. See if you can do a descending order array and find the position of the various elements. Use the scanner method of util package of Java and input array elements. and modify the first and second code as is needed to accept input from the keyboard.

2. Do a binary search on the following sequence of numbers:

16,92,52,97,82,25,32,15,45,
56,78,64,12,8, 26 and we are searching

for 25.

Chapter 8. Knapsack Problem

The problem asks to find the maximum value for a container that can take a fixed amount of items due to weight limit. There are several items and each have different weights and different values. The objective is to find the combination of items that can be taken and optimal value that can be obtained.

Let us take an example: Suppose the knapsack or container can take only 8kgs of items and there are four items numbered 1 through 4 with the weights 4, 3,2 and 5 respectively. Assume that their values are 25,20,18 and 30 dollars

each. The question is which items to carry in the container to maximize the profit or effort.

Let us list the options out. This is an example of an exhaustive method:

Items	Weight	Value $
0	0	0
1	4	25
2	3	20
3	2	18
4	5	30
1,2	7	45
1,3	6	43
1,4	9	55
2,3	5	38
2,4	8	50

3,4	7	48

If we look at the subsets that we have, we can see combination of items 2 and 4 will be the best choice because the total weight comes to 8kg and the value is $50.

If we took 1 and 4 the value may be bigger but the weight restriction of 8kgs will be violated. If we took 3 and 4 then we would not have used the weight allowed to its maximum. Thus we take the final answer as items 2 and 4 with value of $50.

Now here the restriction was the weight if there was additional restriction on the

value also then exhaustive search may be slightly longer in terms of time to complete. One can see the practical application of this example in the travel industry or cargo industry.

Let us look at a code in Java for this:

```
import java.util.*;
import java.lang.*;
public class lab8
{
static int max(int x, int y)
{
        return (x > y)? x : y;
}
static int sack(int weight, int iw[], int va
{
```

```
int i, w;
int [ ][ ]K = new int[n+1][weight+1];
for (i = 0; i <= n; i++)
{
for (w = 0; w <= weight; w++)
{
if (i==0 || w==0)
K[i][w] = 0;
else if (iw[i-1] <= w)
K[i][w] = max(val[i-1] + K[i-1][w-iw[
else
K[i][w] = K[i-1][w];
} }
return K[n][weight];
}
public static void main(String args[])
{
Scanner sc = new Scanner(System.in);
```

```
System.out.println("Enter the number of
int n = sc.nextInt();
System.out.println("Enter the items weig
int []iw = new int[n];
for(int i=0; i<n; i++)
iw[i] = sc.nextInt();
System.out.println("Enter the items valu
int []val = new int[n];
for(int i=0; i<n; i++)
val[i] = sc.nextInt();
System.out.println("Enter the maximum c
int weight = sc.nextInt();
```

```
System.out.println("The maximum value
knapsack of capacity W is: " + sack(wei
        sc.close();
    }
```

```
}
```

Exercise:

1. Calculate manually the weight and values that can be taken in the sack of a cargo company for the following parameters:

Item1: 10 kg Item 2: 20 kg Item 3: 25 kg Item 4: 32 kg Item 5: 12 kg Item 6: 15kg

The values are: 150$, 800$, 560$, 900$, 650$ and 450$ respectively.

The maximum weight that can be taken is 60kg.

2. Run the code and see if your answer matches the result given by the code in terms of value.

3. The code gives the maximum value that can be taken-modify the code to show:

a. the combination of items that produces the value specified by the code

b. and the resulting weight that can be taken for the correct combination of items.

4. What happens if there are two possible solutions to a knapsack problem?

5. Run the code for the parameters in question one with the weight allowed to be taken increased to 90kgs.

6. Based on the values in question one calculate manually the items that can be taken if in addition to the weight restriction of 60kg there is a value restriction on each combination of shipment or carriage of $750.

7. Modify the code above to factor in a restriction on value also.

8. Name or find another algorithm that uses an exhaustive search method to obtain the final answer.

Chapter 9. Depth First

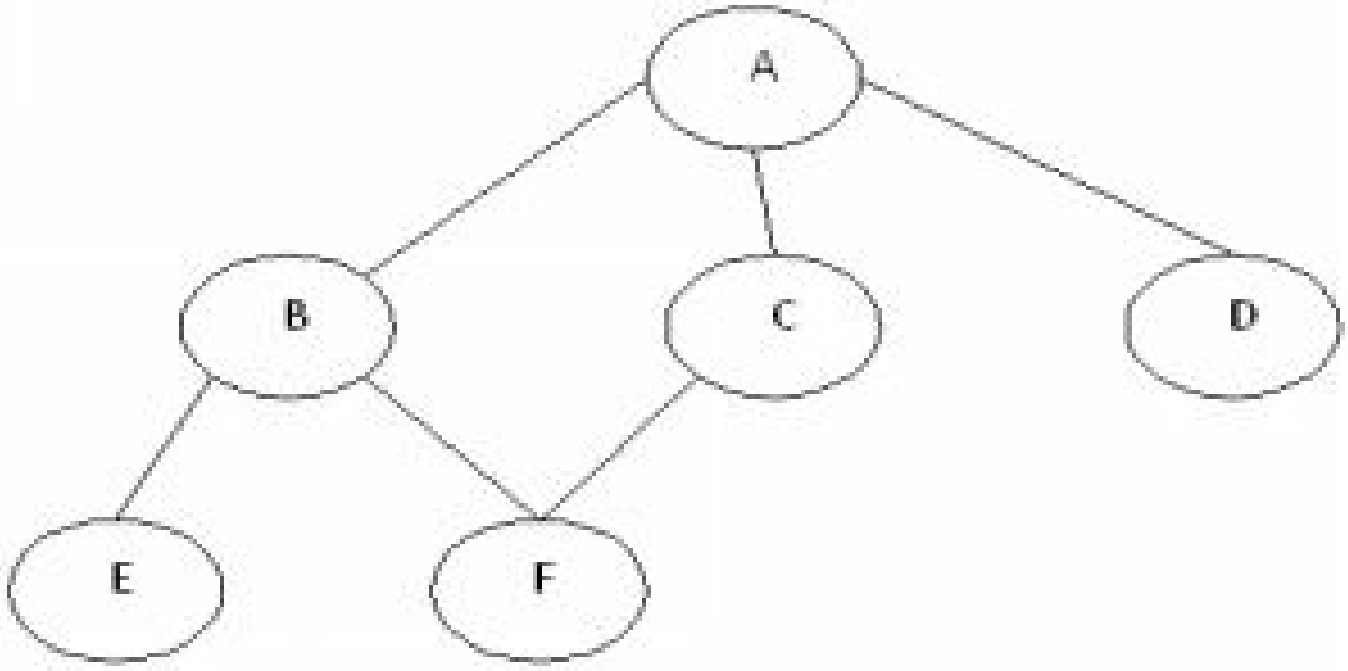
This applies to the data structure known as graphs. In this kind of search, the process of traversal of a graph starts at an arbitrary vertex by marking it as visited.

The search proceeds to an unvisited vertex that is adjacent to the one it is currently in, with ties resolved arbitrarily. In our examples and in similar examples ties are resolved by the alphabetical order of the vertices or logical order of numbers if nodes are numbered. The search goes on until a

dead end—a vertex with no adjacent unvisited vertices is reached. Then the search backs up one edge to the vertex it came from and tries to continue visiting unvisited vertices from there. The search eventually halts after backing up to the starting vertex.

If unvisited vertices still remain, the depth-first search must be restarted at any one of them.

Let us look at an example:



Here we have an undirected graph with six nodes: A B C D E F to put them in alphabetical order. While we can start at any node we assume for now that we will start at node A. Before we look into the depth first search let us build a matrix called an adjacency matrix. Simply put a matrix which shows which nodes are adjacent to each other.For the above diagram that matrix will be given

as follows:

	A	B	C	D
A	0	1	1	1
B	1	0	0	0
C	1	0	0	0
D	1	0	0	0
E	0	1	0	0
F	0	1	1	0

We created this matrix by taking each node A through and finding if the next node in the matrix column is adjacent to it or not.

We will not be visiting the same node. Starting at node A we go to B (LHS) and then from B to E. Since E has no

children we go back to node B which has been visited already so it will go to F and then to C and then to D. So the answer on this traversal of this graph is ABEFCD.

We use stacks when dealing with Depth First search. The logic to follow would be:

1. Push the root node in the Stack.
2. Loop until stack is empty.
   3. Peek the node of the stack.

If the node has unvisited child nodes, get the unvisited child node, mark it as traversed and push it on stack.

5. If the node does not have any

unvisited child nodes, pop the node from the stack.

So the nodes that get popped in our example above are: E, F, B C, D, A

```
import java.util.*;
public class lab9
{
private Stack<Integer> st;
public lab9()
{
st = new Stack<Integer>();
}
public void depthfirstsearch(int amatrix
startingpoint)
{
int numbernodes = amatrix[startingpoint
```

```
int v[] = new int[numbernodes + 1];
int e = startingpoint;
int i = startingpoint;
System.out.print(e + "\t");
v[startingpoint] = 1;
st.push(startingpoint);
while (!st.isEmpty())
{
e = st.peek();
i = e;
while (i <= numbernodes )
```

```
{
if (amatrix[e][i] == 1 && v[i] == 0)
{
st.push(i);
v[i] = 1;
e = i;
```

```
i = 1;
System.out.print(e + "\t");
}
i++;
}
st.pop();
}
}
public static void main(String []args)
{
System.out.println("Enter the number of
graph");
Scanner sc = new Scanner(System.in);
int numbernodes = sc.nextInt();
int amatrix[][] = new int[numbernodes +
1];
System.out.println("Enter the adjacency
for (int i = 1; i <= numbernodes; i++)
```

```
for (int j = 1; j <= numbernodes; j++)
amatrix[i][j] = sc.nextInt();
System.out.println("Enter the starting po
int startingpoint = sc.nextInt();
System.out.println("The DFS Traversal
lab9 xyz = new lab9();
xyz.depthfirstsearch(amatrix, startingpoi
}
}
```

The above code consists of two parts. The main function takes in the inputs to the program and calls the depthfirstsearch function to take in the adjacent matrix info and starting node to give the path of traversal.

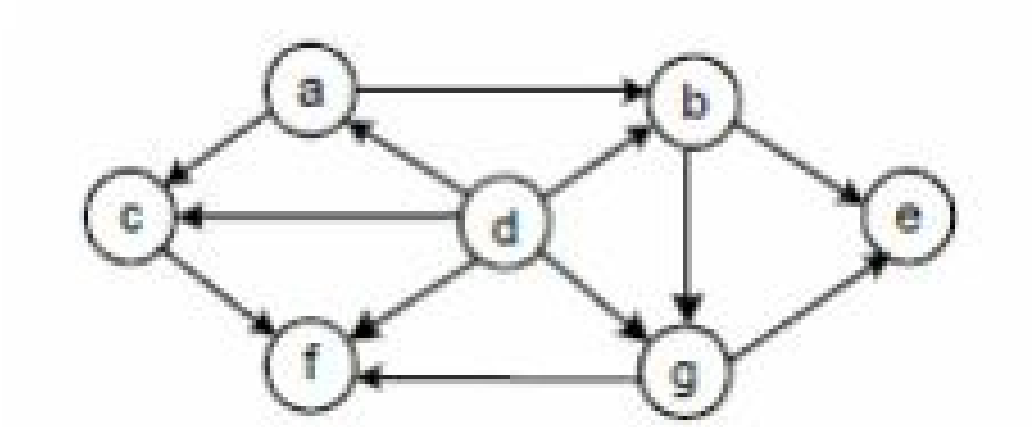
The starting node is referred to as 0 or 1

but this should not affect manual calculation of the traversal path.

For this code to work the adjacent matrix will have to be entered.

Exercise:

Calculate DFS path traversal for the following graph manually:

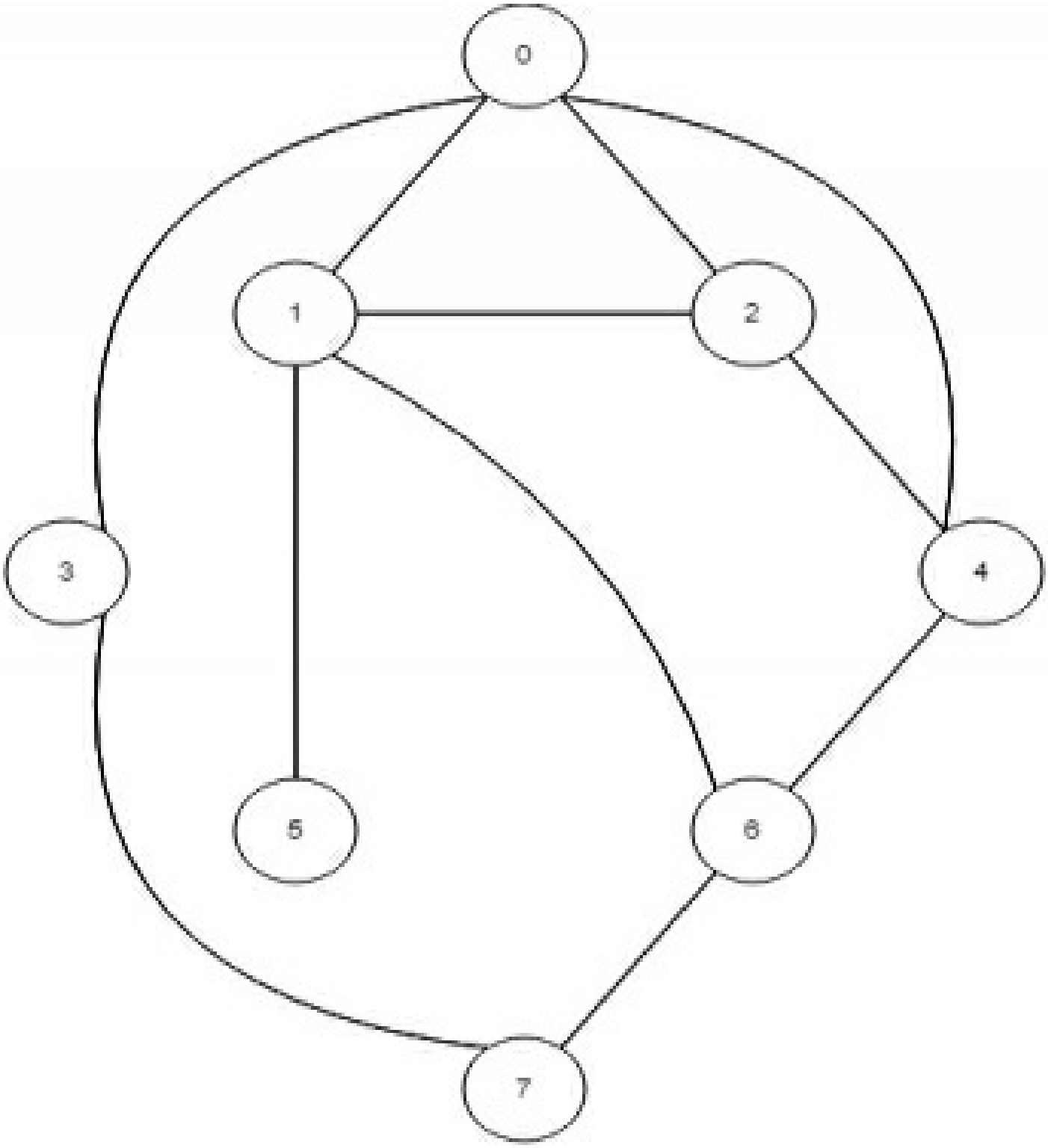


The graph here is directed graph. Follow the directions to get the correct answer.

- List the nodes that get popped out.

- Create an adjacent matrix for the above graph.
- Is there a difference between an adjacent matrix for a directed and undirected graph?
- What is the difference?
- After obtaining the adjacent matrix see if the code above will give you the same answer that you have obtained manually.

Use the code above for the following undirected graph and compare it with your manual solution (assume starting point as 0 you may want to try others):

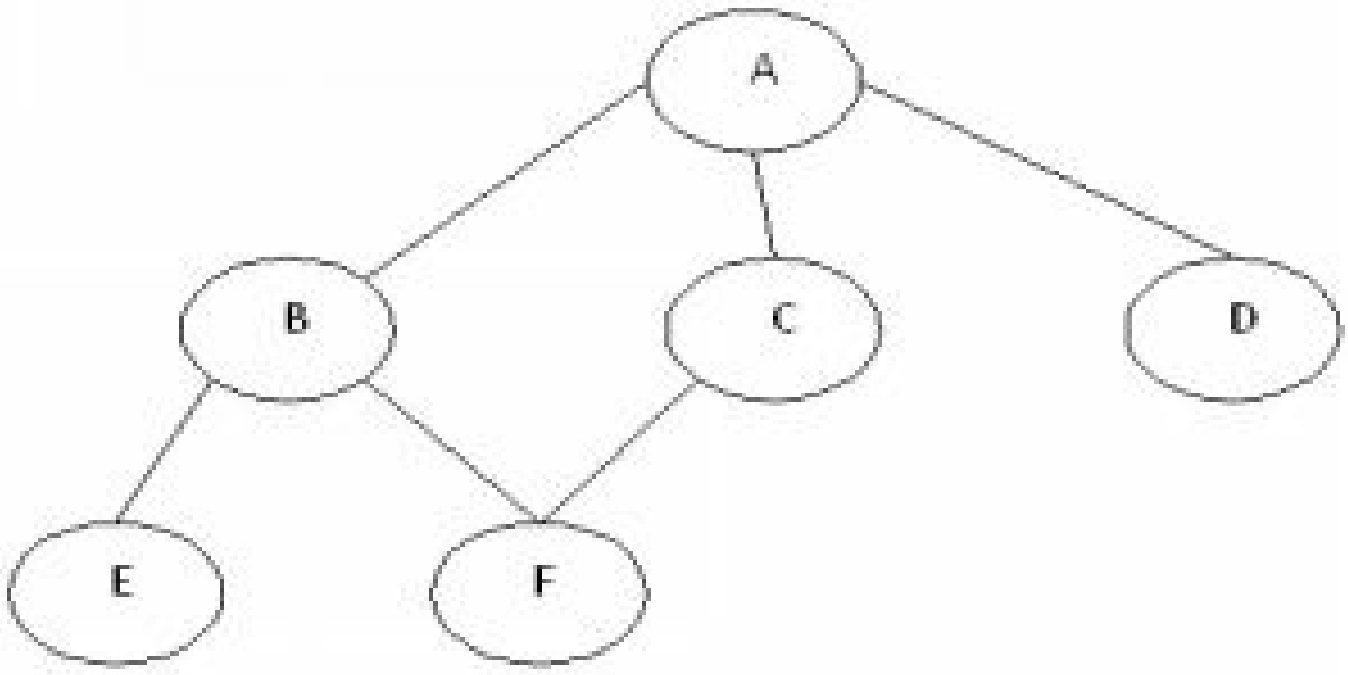


Here the nodes may not have been marked as A, B … but the meaning is the same

Chapter 10. Breadth First

The Breadth First search is based on queues versus the stacks that are used in Depth First search. Both stacks and queues are part of data structures.

Let us look at the same example as the one in the last chapter:



The adjacency matrix of the graph remains the same as in the last chapter. The aim of BFS algorithm is to traverse the graph as close as possible to the root node.

The BFS visits the nodes level by level, so it will start with level 0 which is the root node, and then it moves to the next levels which are B, C and D, then the last levels which are E and F. Thus the final traversal would be A B C D E F.

The logic we would follow will be:

1. Push the root node in the Queue.
2. Loop until the queue is empty.

3. Remove the node from the Queue.
If the removed node has unvisited child nodes, mark them as visited and insert the unvisited children in the queue.

```
import java.util.* ;
public class lab10
{
private Queue<Integer> q;
public lab10()
{
q = new LinkedList<Integer>();
}
public void breadthfirstsearch(int amatr
startingpoint)
{
```

```
int nodes = amatrix[startingpoint].length
```

```
int[] v = new int[nodes + 1];
int i, e;
v[startingpoint] = 1;
q.add(startingpoint);
while (!q.isEmpty())
{
e = q.remove();
i = e;
System.out.print(i + "\t");
while (i <= nodes)
{
if (amatrix[e][i] == 1 && v[i] == 0)
{
q.add(i);
v[i] = 1;
}
i++;
} }}
```

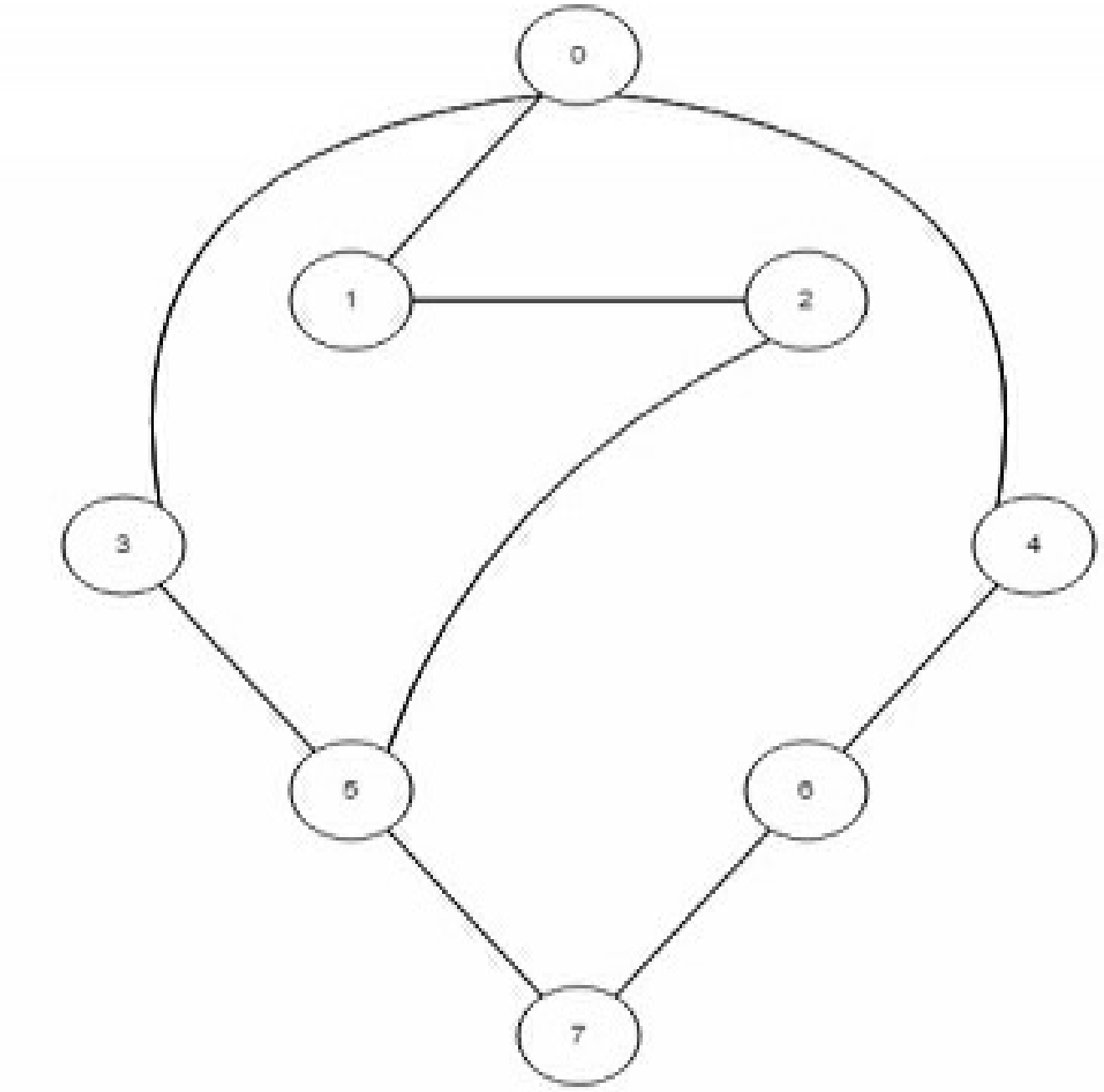
```
public static void main(String[]args)
{
System.out.println("Enter the number of
graph");
Scanner sc = new Scanner(System.in);
int nodes = sc.nextInt();
int amatrix[][] = new int[nodes + 1][nod
System.out.println("Enter the adjacency
for (int i = 1; i <= nodes; i++)
for (int j = 1; j <= nodes; j++)
amatrix[i][j] = sc.nextInt();
System.out.println("Enter the startingpoi
int startingpoint = sc.nextInt();
System.out.println("The breadthfirstsear
lab10 xyz = new lab10();
xyz.breadthfirstsearch(amatrix, startingp
}}
```

Like the Depth First search code this one needs the inputs which are the adjacent matrix and starting node and calculates the traversal in the breadthfirst search function. Only the traversal path is calculated.

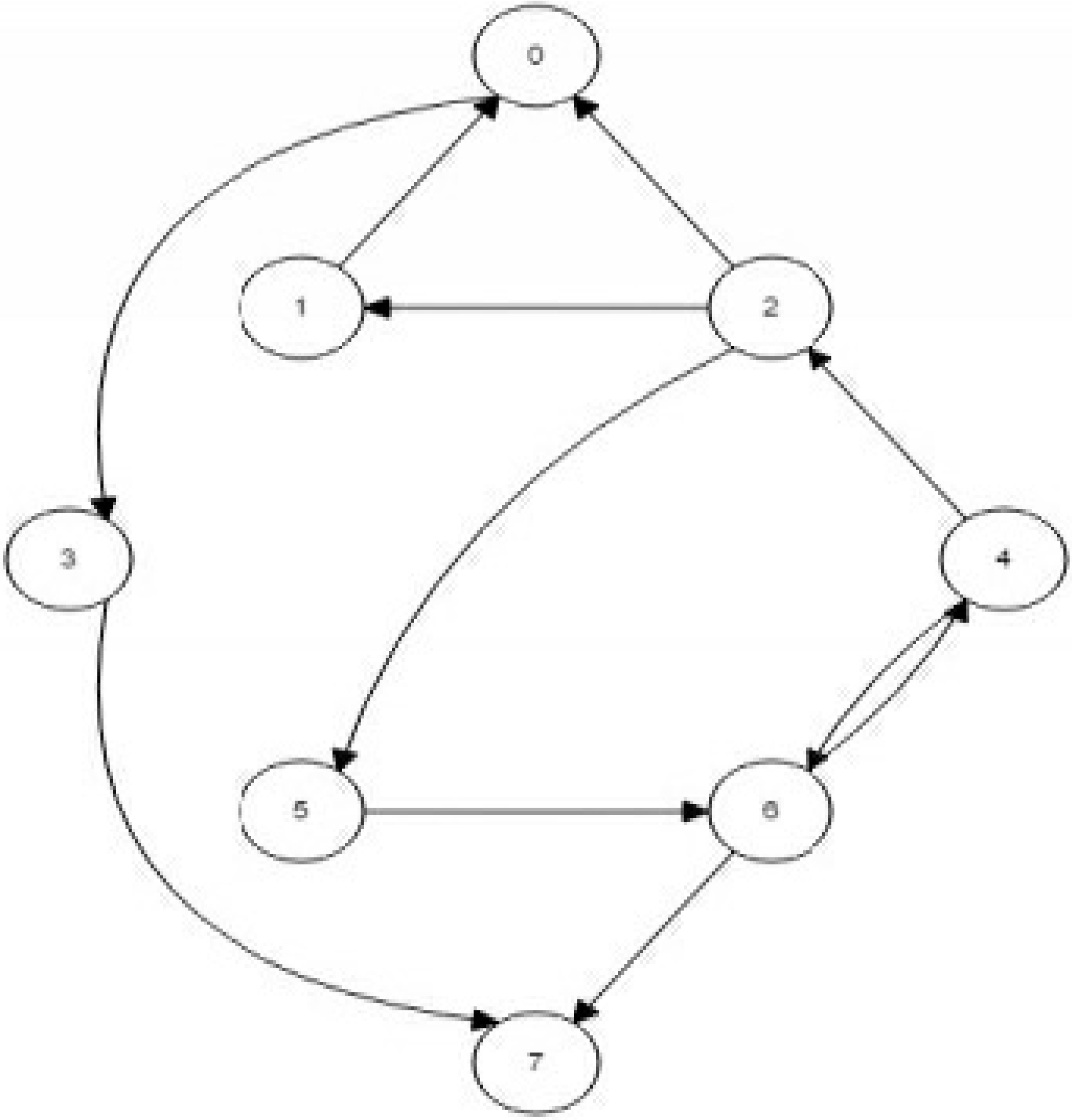
Exercise:

Calculate the BFS for the following graphs manually and then compare them with result produced by the program above:

1. Undirected graph (0 starting point)



2. Directed graph (0 starting point)



You may want to try with other starting nodes also. Then try using the program to verify your manual findings.

Chapter 11. Topological Sort

A directed graph, or digraph for short, is a graph with directions specified for all its
edges. The adjacency matrix and adjacency lists can be used to represent a digraph.

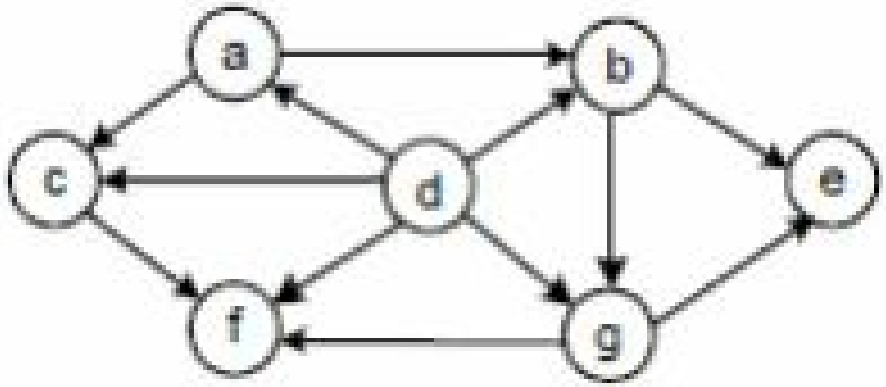
The main differences between undirected and directed graphs are:

(1) The adjacency matrix of a directed graph does not have to be symmetric.
(2) An edge in a directed graph has just one (not two) corresponding nodes in the digraph's adjacency list.

If we have a digraph in which we can list its vertices in such an order that for every edge in the graph, the vertex where the edge starts is listed before the vertex where the edge ends. This kind of a situation is called topological sorting.

Simply put to solve for a topological sorting perform depth first search first. Then from the stack as mentioned in the algorithm of Depth First search pop out the nodes and write them down in the correct order of popping. The reverse order of this pops will give the result of a topological sort.

Let us look at an example:



Let us look at this network of nodes. This is a directed graph.If we start from point a then the path of the traversal would be a □ b □ e □ g □ f □ c □ d. Now let us look at the pops. The first to pop is e as there are no children to it. Followed by f. So let us summarize the pops: e f g b c a d. Now the topological sort is the reverse of the pops in DFS so we get the final answer as d a c b g f e.

The example above is that of a directed graph. The rule is the same if the graph is not directed. The first thing to do to

run the program shown below is to create an adjacency matrix. There are 7 nodes in this above diagram. Number them from 1 through 7. The logical way would be to follow alphabetical order when numbering the nodes-thus node a will be 1 and node g will be 7. The starting point will be node 1 or node a.

Now the matrix for the above graph keeping in mind the direction of the arrows will look like this:

	a	b	c	d	e
a	0	1	1	0	0
b	0	0	0	0	1
c	0	0	0	0	0
d	1	1	1	0	0

e	0	0	0	0	0
f	0	0	0	0	0
g	0	0	0	0	1

The 0's state there is no direct connection to that node from that stated on the left or the node origin and destination is the same. The matrix states exactly what the diagram shows. The program requires the number of nodes to be entered followed by the adjacency matrix and the starting point which in this case is 1. Each row of the matrix has to be entered in a row with a single space in between each entry.

```
import java.util.Stack;
import java.util.*;
```

```
public class lab11
{
private Stack<Integer> st;

public lab11()
{
st = new Stack<Integer>();
}
public int[] tp(int a[][], int z)throws Nul
{
int nodes = a[z].length - 1;
int[] tp_sort = new int[nodes + 1];
int pos = 1;
int j;
int v[] = new int[nodes + 1];
int e = z;
int i = z;
v[z] = 1;
```

```
st.push(z);
while (!st.isEmpty())
{
e = st.peek();
while (i <= nodes)
{
if (a[e][i] == 1 && v[i] == 1)
{
if (st.contains(i))
{
System.out.println("not feasible");
return null;
}
}
if (a[e][i] == 1 && v[i] == 0)
{
st.push(i);
```

```
v[i] = 1;
e = i;
i = 1;
continue;
}
i++;
}
j = st.pop();
tp_sort[pos++] = j;
i = ++j;
}
return tp_sort;
}
public static void main(String... arg)
{
int nodes, z;
Scanner scanner = null;
int tp_sort[] = null;
```

```
System.out.println("Enter the number
graph");
scanner = new Scanner(System.in);
nodes = scanner.nextInt();
int a[][] = new int[nodes + 1][nodes + 1
System.out.println("Enter the adjacency
```

```
for (int i = 1; i <= nodes; i++)
for (int j = 1; j <= nodes; j++)
a[i][j] = scanner.nextInt();
System.out.println("Enter the source or
the graph");
z = scanner.nextInt();
System.out.println("The Topological so
given by ");
lab11 xyz = new lab11();
tp_sort = xyz.tp(a,z);
for (int i = tp_sort.length - 1; i > 0; i--)
```

```
{
if (tp_sort[i] != 0)
System.out.print(tp_sort[i] + "\t");
}
scanner.close();
}
}
```

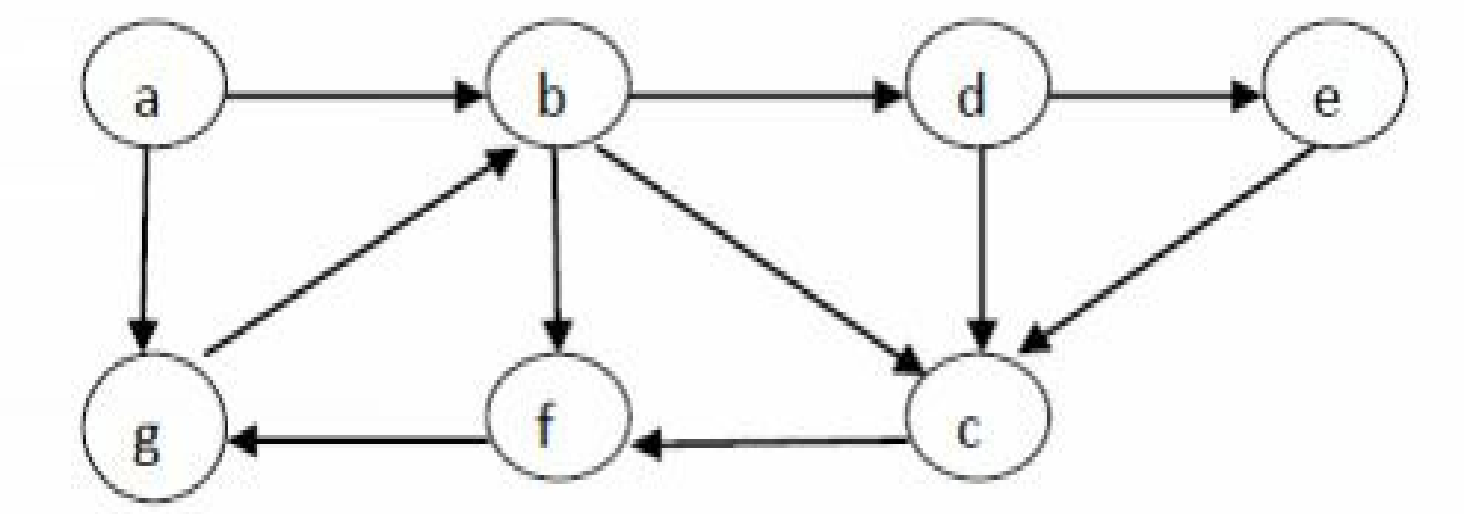
The result of running this code with the input adjacency matrix and starting point as 1 will be 1 3 2 7 6 5 which corresponds to a c b g f e which is what we got manually. The labeling of the nodes in the alphabetical order is important.

The program could be modified to start with node 0 as there is a preference for

some to denote starting point as 0 but in the overall context the meaning is the same.

Exercise:

1. Apply topological sorting to the following manually:



and verify the manually computed answer with the code . Note that this is a directed graph.

Chapter 12. Mergesort

This is an example of a divide and conquer technique.

In this technique we sort a given array $A[0..n-1]$ by dividing it into two halves $A[0..\lfloor n/2 \rfloor - 1]$ and $A[\lfloor n/2 \rfloor..n-1]$, sorting each of them recursively, and then merging the two smaller sorted arrays into a single sorted one.

Let us look at an example: We will use the same numbers we have used before: 89 45 68 90 29 23 15

This has 7 numbers so we need to split them into two groups. The midpoint is

3.5 so we will round them to 4 and 3 elements in each group. Thus we have:

89 45 68 90 29 23 15 (we will then divide each group until there is only 1 element in group)

[89 45] [68 90] [29 23] [15] (as you can see here I rounded 1.5 to 2)

[89] [45] [68] [90] [29] [23] [15] (now all of them are single in their groups)

(now we combine after sorting)
[45 89] [68 90] [23 29] [15]

further sorting on the left hand side and

right hand side we get:

[45 68 89 90] and [15 23 29] then we sort the two groups again

15 23 29 45 68 89 90.

Now let us look at a code: (as an exercise change array value in x and run)

```
import java.util.*;
public class lab12 {
private int[] x;
private int[] y;
public static void main(String a[]){
int[] x = {89,45,68,90,29,23,15};
lab12 xyz = new lab12();
```

```
xyz.sort(x);
```

```
for(int i:x){
System.out.print(i);
System.out.print(" ");
}}
public void sort(int x[]) {
this.x = x;
int length = x.length;
this.y = new int[length];
MergeSort(0, length - 1);
}
private void MergeSort(int li, int hi) {
if (li < hi) {
int middle = li + (hi - li) / 2;
MergeSort(li, middle);
MergeSort(middle + 1, hi);
merges(li, middle, hi);
}}
private void merges(int li, int middle, in
```

```
for (int i = li; i <= hi; i++) {
y[i] = x[i];
}
int i = li;
int j = middle + 1;
int k = li;
while (i <= middle && j <= hi) {
if (y[i] <= y[j]) {
x[k] = y[i];
i++;
} else {
x[k] = y[j];
j++;
}
k++;
}
while (i <= middle) {
x[k] = y[i];
```

```
k++;
i++;
} }}
```

Chapter 13. Quick Sort

This sort is another example of a divide-and conquer approach. Unlike mergesort in the previous chapter which divides its input elements according to their position in the array, this sort divides them according to their value.

Let us look at an example:

We will use the following numbers: 54 26 93 17 77 31 44 55 20

This algorithm requires a pivot point to start. The choice of the pivot will determine the complexity of the algorithm. Pivot can be chosen by taking

the first element, last element and middle element and finding the median. For now, let us choose in the above array of numbers the first number as the pivot.

So 54 is the pivot. Now we need two markers -left marker and a right marker. The right marker is set to 20 and the left marker is now pointing to 26. Now let us compare from the left marker with the pivot.

26 < 54 so move left marker to 93. 93 is > 54 so stop. Now look at right marker at 20. 20 is < 54 so swap 93 and 20. That is swap the numbers pointed to by left marker and right marker. The result will look like this:

54 26 20 17 77 31 44 55 93

Now continue from left marker:

17 < 54 move to 77. 77 > 54 so go to right marker. 93 > 54 move to 55. 55 > 54 move to next number 44. 44 < 54 stop. Swap value at left marker with that at right marker. So we swap 77 and 44 and get the result as follows:

54 26 20 17 44 31 77 55 93

Note left marker is at 44 and right marker at 77. Now the point where the two markers will cross is called the split point for the pivot 54 in this case.

This happens to be 31. So we can divide the numbers into two groups after swapping the pivot with the split point number.

[31 26 20 17 44] and [77 55 93] split point 54.
Now each of these two groups shown in [ ] has to be quick sorted in a similar way. Leading to the obvious sorted number which is:

17 20 26 31 44 54 55 77 93

When we have small array of numbers it is easy to get the final answer but if the array is a large one it is difficult.

Thus the algorithm in short can be written as:

a. Input: Subarray of array A [0...n − 1], defined by its left and right
indices l and r
if l < r
s ←Partition(A[l..r]) where s is a split position
Quicksort(A[l..s − 1])
Quicksort(A[s + 1..r])

Now let us look at the code:

```
public class lab13 {
private int x[];
public void sort(int[] x) {
if (x == null || x.length == 0) {
```

```
return;
}
this.x = x;
int length = x.length;
quickSort(0, length - 1);
}
private void quickSort(int li, int hi) {
int i = li;
int j = hi;
int p = x[li+(hi-li)/2];
while (i <= j) {
while (x[i] < p) {
i++;
}
while (x[j] > p) {
j--;
}if (i <= j) {
```

```
ex(i, j);
i++;
j--;
}
}
if (li < j)
quickSort(li, j);
if (i < hi)
quickSort(i, hi);
}
private void ex(int i, int j) {
int temp = x[i];
x[i] = x[j];
x[j] = temp;
}
public static void main(String [] args){
lab13 sx = new lab13();
int[] input = {54,26,93,17,77,31,44,55,2
```

```
sx.sort(input);
for(int i:input){
System.out.print(i);
System.out.print(" ");
}
}
}
```

In the above code the variable p is the pivot. The code above will not show line by line sorting and perhaps that can be thought as an exercise.

Exercise:

1. Quick sort the following manually: 67 12 34 45 90 89 77 13 15 23.

1. Modify the program to print out the pivots each time one is created.

2. What is the meaning of the following piece of code above:

```
int i = li;  int j = hi; int p = x[li+(hi-
li)/2];
while (i <= j) {
while (x[i] < p) { i++;}
```

Chapter 14. Binary Tree Traversals

There are three types of Binary tree traversals. They are:

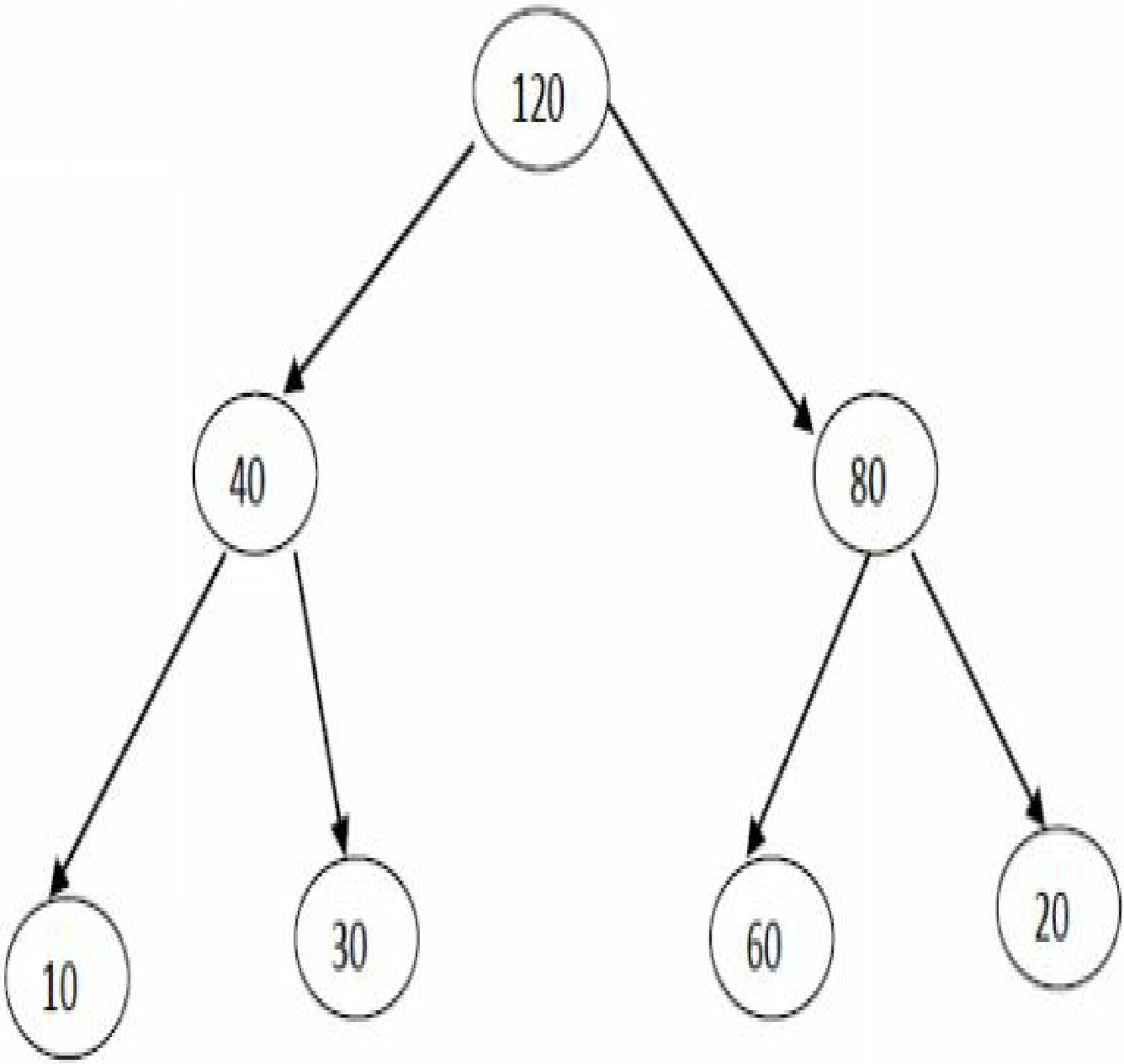
a. Preorder traversal, the root is visited before the left and right subtrees
are visited (in that order).

b. Inorder traversal, the root is visited after visiting its left subtree but
before visiting the right subtree.

c. Postorder traversal, the root is visited

after visiting the left and right subtrees (in that order).

Inorder traversal:



We go down the left sub tree starting with the node 120. Node 10 has no children below it. Thus this traversal

will print as follows: 10 40 30 120 60 80 20.

An iterative process can be used if one is writing a code to do this. Stacks (data structure) will be used for this purpose.

Let us look at a code.

```
import java.util.Stack;
public class lab14 {
public static class tn
{
int d;
tn left;
tn right;
tn(int d)
{
this.d=d;
```

```
}}
public void inorder(tn root) {
if(root == null)
return;
Stack<tn> s = new Stack<tn>();
tn cn=root;
while(!s.empty() || cn!=null){
if(cn!=null)
{
s.push(cn);
cn=cn.left;
}
else
{
tn n=s.pop();
System.out.printf("%d ",n.d);
cn=n.right;
}}}
```

```
public static void main(String[] args)
{
lab14 xyz=new lab14();
tn node=Tree();
xyz.inorder(node);
}
public static tn Tree()
{
tn node =new tn(120);
tn node40=new tn(40);
tn node10=new tn(10);
tn node30=new tn(30);
tn node80=new tn(80);
```

.

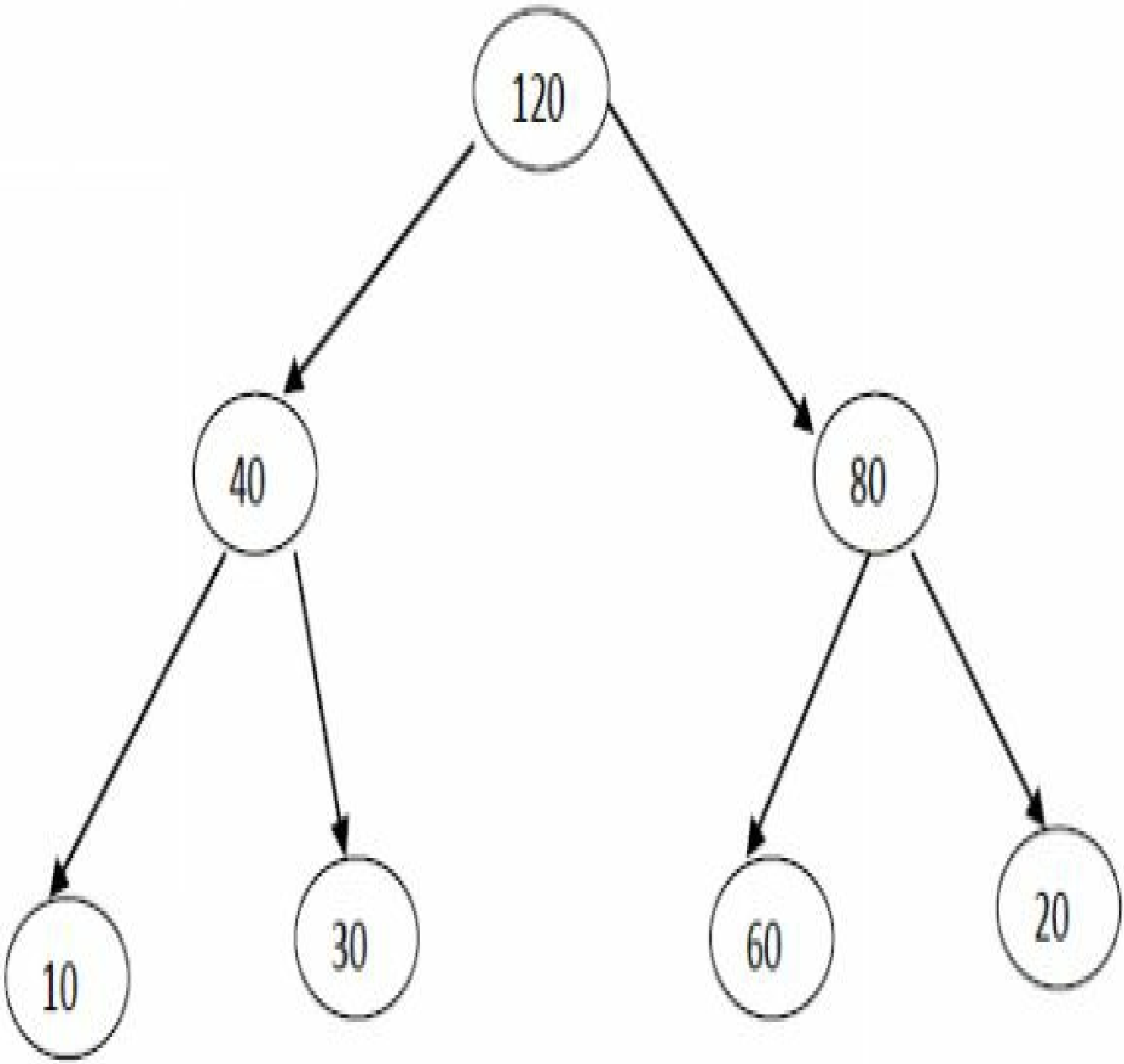
```
tn node60=new tn(60);
tn node20=new tn(20);
node.left=node40;
node.right=node80;
```

```
node40.left=node10;
node40.right=node30;
node80.left=node60;
node80.right=node20;
return node;
}}
```

Here the tree is defined in the Tree(). It is put in manually. A method to put it from the keyboard can be considered.

Preorder Traversal

Let use the same example:



Based on the definition of preorder traversal we have to visit the node and then traverse the left sub tree followed

by the right sub tree. Thus the traversal will be 120 40 10 30 80 60 20.

Let us look at a code for this:

```
import java.util.Stack;
public class lab14b {
public static class tn
{
int d;
tn left;
tn right;
tn(int d)
{
this.d=d;
} }
public void preorder(tn root) {
if(root == null)
return;
```

```
Stack<tn> st = new Stack<tn>();
st.push(root);
while(!st.empty()){
tn n = st.pop();
System.out.printf("%d ",n.d ) ;
if(n.right != null){
st.push(n.right);
}
if(n.left != null){
st.push(n.left);
} } }
public static void main(String[] args)
{
lab14b xyz=new lab14b();
tn node=Tree();
xyz.preorder(node);
}
public static tn Tree()
```

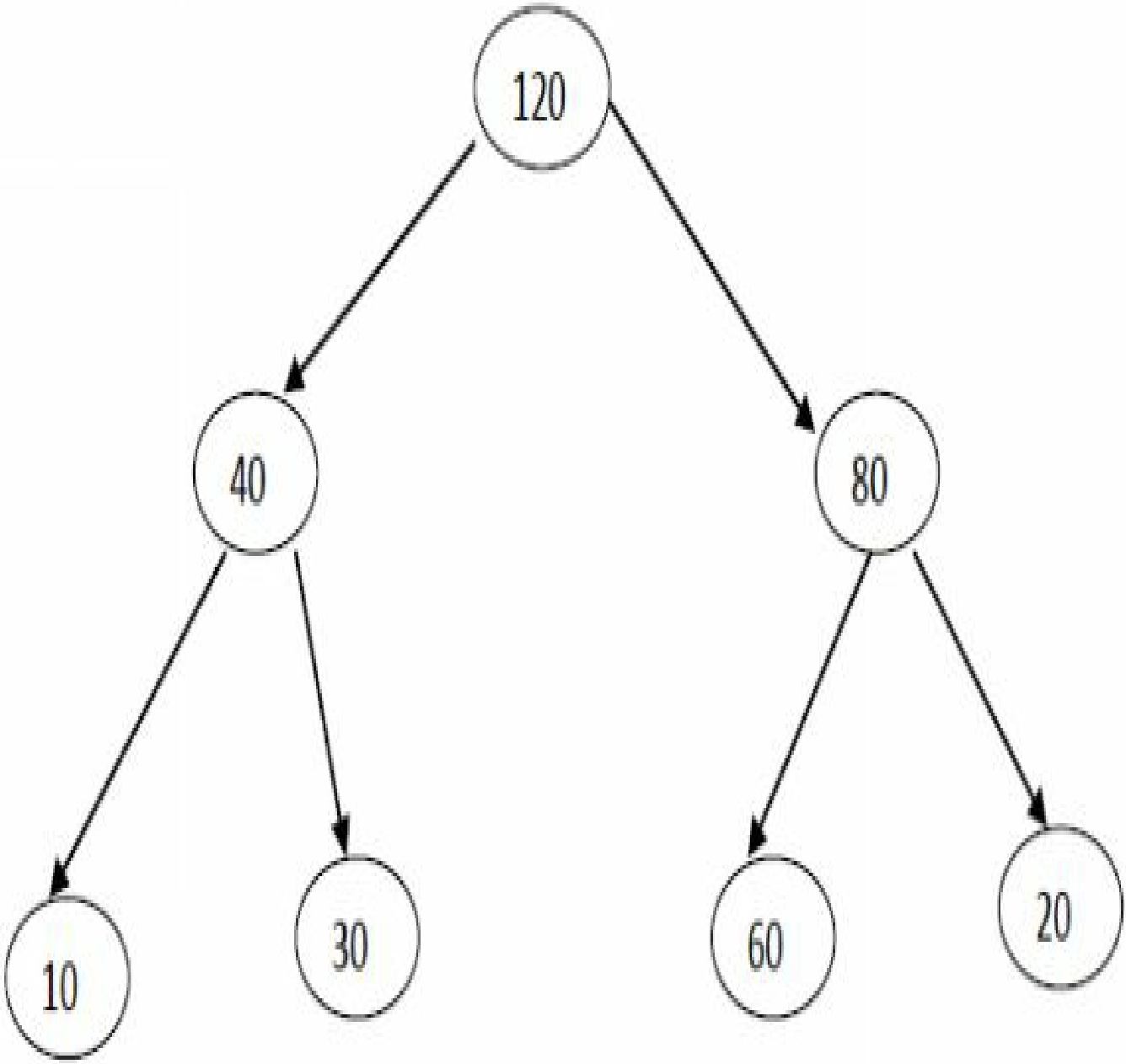
```
{
tn node =new tn(120);
tn node40=new tn(40);
tn node10=new tn(10);
tn node30=new tn(30);
tn node80=new tn(80);
tn node60=new tn(60);
tn node20=new tn(20);
node.left=node40;
```

```
node.right=node80;
node40.left=node10;
node40.right=node30;
node80.left=node60;
node60.right=node20;
return node;
} }
```

The code could be modified to accept input of nodes from the keyboard and this idea is to left as an exercise.

Postorder Traversal

We use the same example for this traversal:



Here by definition we have to traverse the left sub tree and then the right sub tree and then the node. So the traversal

for the above diagram would be 10 30 20 60 20 80 and finally the root which is 120.

You may have noticed that tree used on all three examples is the same and that it is a balanced tree.

Now let us look at a code for the post order traversal.

```
import java.util.Stack;
public class lab14c {
public static class tn
{
int d;
tn left;
tn right;
tn(int d)
```

```
{
this.d=d;
}  }
public void postOrder(tn rt) {
if(rt !=  null) {
postOrder(rt.left);
postOrder(rt.right);
System.out.printf("%d ",rt.d);
} }
public void postorder(tn rt) {
if( rt == null ) return;
Stack<tn> st = new Stack<tn>( );
tn z = rt;
while( true ) {
if(z != null ) {
if(z.right != null )
st.push(z.right );
st.push(z );
```

```
z =z.left;
continue;
}
if( st.isEmpty( ) )
return;
z = st.pop( );
if(z.right != null && ! st.isEmpty( ) &&z
) {
st.pop( );
st.push(z );
z =z.right;
} else {
System.out.print(z.d + " " );
z = null;
}  }
public static void main(String[] args)
```

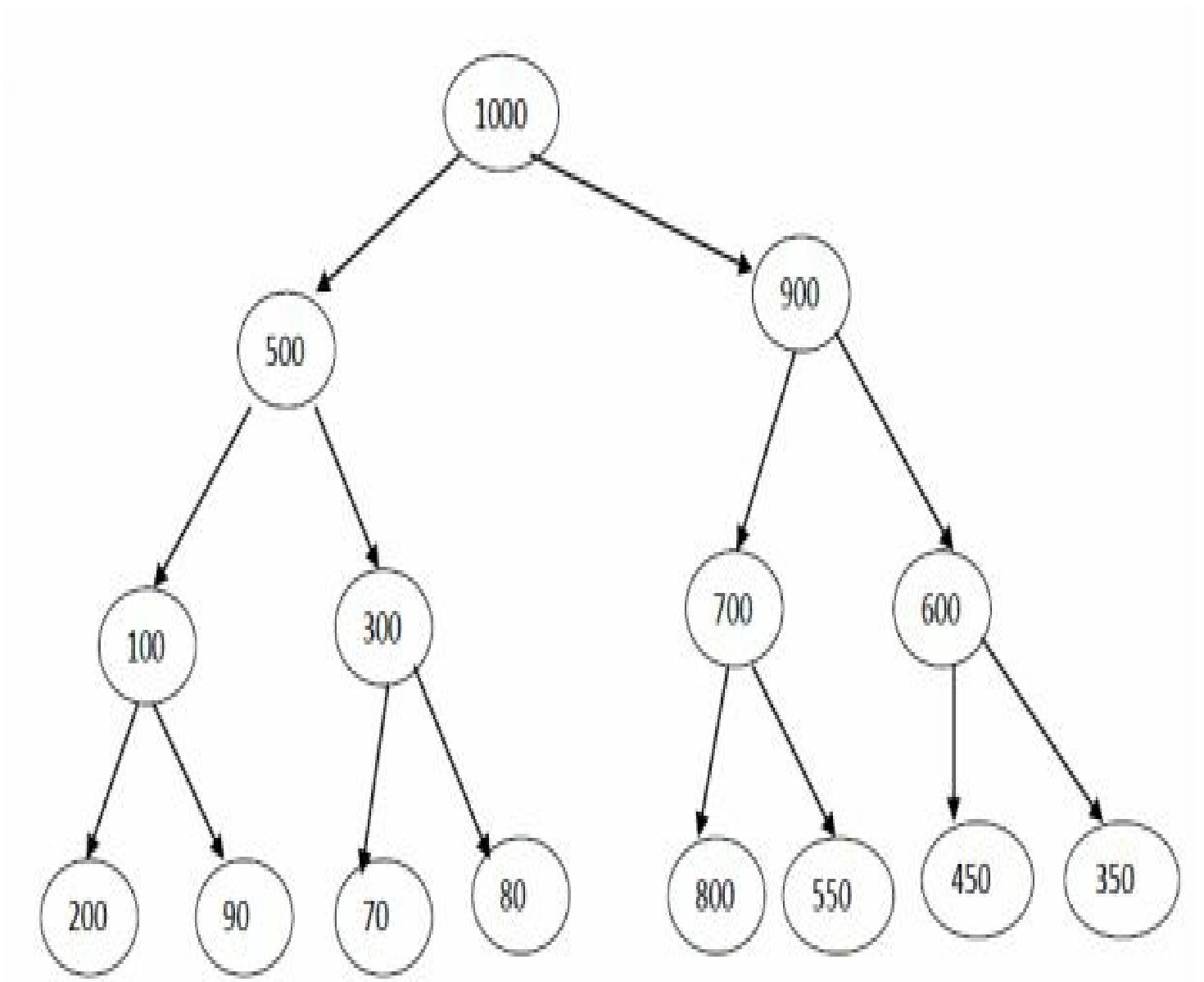
```
{
```

```
lab14c xyz=new lab14c();
tn node=Tree();
xyz.postorder(node);
}
public static tn Tree()
{
tn node =new tn(120);
tn node40=new tn(40);
tn node10=new tn(10);
tn node30=new tn(30);
tn node80=new tn(80);
tn node60=new tn(60);
tn node20=new tn(20);
node.left=node40;
node.right=node80;
node40.left=node10;
node40.right=node30;
node80.left=node60;
```

```
node80.right=node20;
return node; }   }
```

Exercises

1. Try practicing all three traversals on the following:



Chapter 15. Gaussian Elimination

Gaussian elimination deals with the solution of systems of linear equations.

The idea of Gaussian elimination is to transform a system of n linear equations in n unknowns to an equivalent system with an upper triangular coefficient matrix, that is a matrix with all zeros below its main diagonal.

Let us look at an example:

$$2x1 - x2 + x3 = 2$$
$$4x1 + x2 - x3 = 4$$

$x1 + x2 + x3 = 2$

let us create the first matrix:

2	-1	1	2
4	1	-1	4
1	1	1	2

taken from the equations above. The last column being the values of the equations after the = sign.

Now to get the answer for the values of x1, x2 and x3 if we can get the values of x1 and x2 to be zero in any of these three equations will help us to get the value of x3. In another words if two variables are zero the value the third can be found and

then we can find the value of the others by having value of at least one of the two remaining variables to be zero. To do this let us follow for the following rules.

Leave the first row and go to the second row. We will a_{11} as the pivot. That means here the pivot is 2. For the second row do a_{21}/a_{11}. or 4/2. Then the new row 2 will be row 2 - 4/2 * row 1 and row 3 will be row 3 - 1/2 * row 1. By doing this we will get the value of x1 in row 2 and 3 to become 0.

The new matrix will be:

2 -1 1 2
0 3 -3 0

0 3/2 1/2 1

Now x1 in row 2 and 3 are 0. Now we want to get x2 to be zero. To do this we will now use row 2 as the pivot. Division by 0 is not possible so the pivot element in row 2 will be 3. So we divide 3/2 in third row by 3 to get 1/2.

The first and second row will remain the same.

The new third row will be calculated by the same rule as in previous case and that is:

new third row - 1/2 * (row 2). This will make the value of x2 =0 in the third row.

The new matrix is :

2 -1 1 2
0 3 -3 0
0 0 2 1

Now that we have x1 and x2 =0 on the last row we have 2x3 = 1 and therefore x3= 1/2. Using this value we can calculate for the second equation.

3x2 -3 x3 =0
3x2 = 3 * 1/2 = 3/2
x2 = (3/2) / 3 = 1.5/3 = 1/2

Therefore, from first equation we find 2x1 - 1/2 +1/2 = 2

Thus 2x1 = 2 and then x1 = 2/2 = 1

The final answers are x1 = 1 x2 = 1/2 and x3 = 1/2

Now let us look at the code for this:

```
import java.util.*;
public class lab15 {
public static void main(String args[])thr
{
Scanner sc = new Scanner(System.in);
System.out.println("The number of equat
by the number of variables.");
System.out.println("No of variables?");
int n=sc.nextInt();
```

```
double value[ ][ ]=new double[n][n+1];
```

```
int i,j;
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
System.out.println("Enter the value of va
value[i][j]=sc.nextDouble();
}
System.out.println("\nEnter the value of
of equation ?"+(i+1));
value[i][j]=sc.nextDouble();
}
gs(value,n,n+1);
}
public static void gs(double val[][],int x
{
double t[]=new double[zq];
int i=0,j=0,k=0,m=0,n=0,x=0;
```

```
double y=0;
for(i=0;i<xy-1;i++)
{
y=val[i][i];
for(j=0;j<zq;j++)
{
t[j]=val[i][j]/y;
val[i][j]=t[j];
}
k=0;
for(m=i+1;m<xy;m++)
{
y=val[m][x];
for(n=0;n<zq;n++)
{
val[m][n]=val[m][n]-(t[n]*y);
}
}
```

```
for(k=0;k<zq;k++)
t[k]=0;
y=0;
```

```
x++;
}
for(i=xy-1;i>0;i--)
{
val[i][zq-1]=val[i][zq-1]/val[i][i];
val[i][i]=0;
for(j=i-1;j>=0;j--)
{
val[j][zq-1]=val[j][zq-1]-(val[j][i]*val|
val[j][i]=0;
}
}
System.out.println();
for(i=0;i<xy;i++)
```

```
{
System.out.println("The value of variabl
"+val[i][zq-1]);
}
}}
```

Exercise:

1. Perform the following elimination manually:

$x_1 + x_2 + x_3 = 2$
$2x_1 + x_2 + x_3 = 3$
$x_1 - x_2 + 3x_3 = 8$

2. Now run the same using the code.

3. Given the matrix in question 1 compute it's inverse manually.

4. Write a program to create an inverse of a matrix.

5. In the above code the scanner method has been used to input values from the keyboard. However the scanner has not been closed after use as it should be. Where in the code will that be done?

6. Run the code for the following equations: $x_1 + x_2 = 3$ $2x_1 + 3x_2 = 5$

Chapter 16. Heap Sort

This is an algorithm with two parts.

In part 1 we construct a heap for a given array and in part 2 we apply root deletion operations n-1 times to the remaining heap.

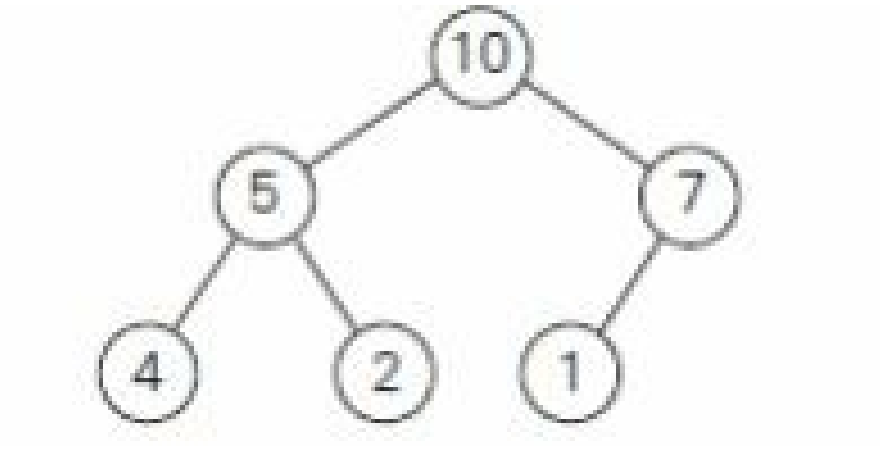
Let us look at an example: We will use the bottom to top method. But first what is a heap?

A heap is a binary tree with keys assigned to its nodes-one key for each node as long as two conditions are met.

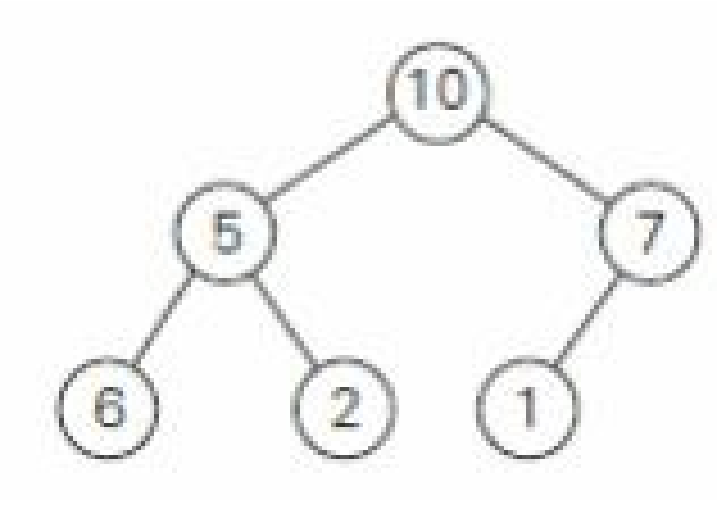
They are:

The binary tree must be complete at all levels. This is called the shape property

The key in each node is equal to or greater than that in its child nodes. This is called the heap property or parental dominance property.

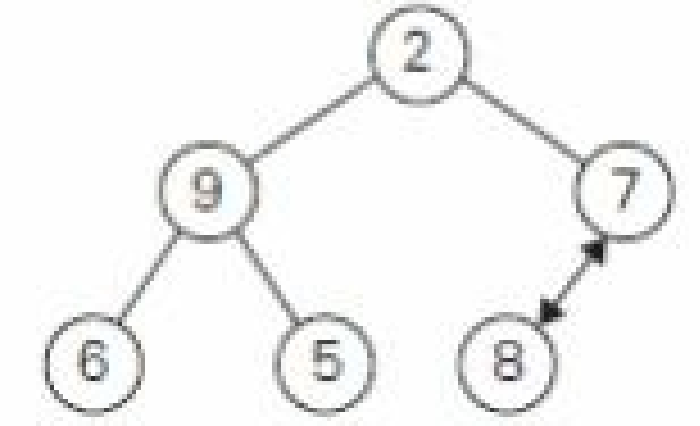
So 

this is a heap.



This second one is not based on the second condition.

Suppose we have the following:



The process of heap sort starts with making the heap first. There are two types of heaps. The condition above

leads to a max heap. In such a heap the largest element of the tree is always at top or at root node. The other type of heap is a min heap where the keys of parent nodes less than or equal to those of the children. In a min heap the smallest element of the tree is always at top or at root node.

In the picture above 2 which is the smallest of the values is at the root node but node 9 is bigger than 6 and 5 so it does not satisfy the requirements of a min heap.

For now, let us look at an example of an array say: 4, 10, 3, 5, 1. We can arrange it as follows from 0 to 4 as it is an array:

Node	4	10	3
Index	*0*	*1*	*2*

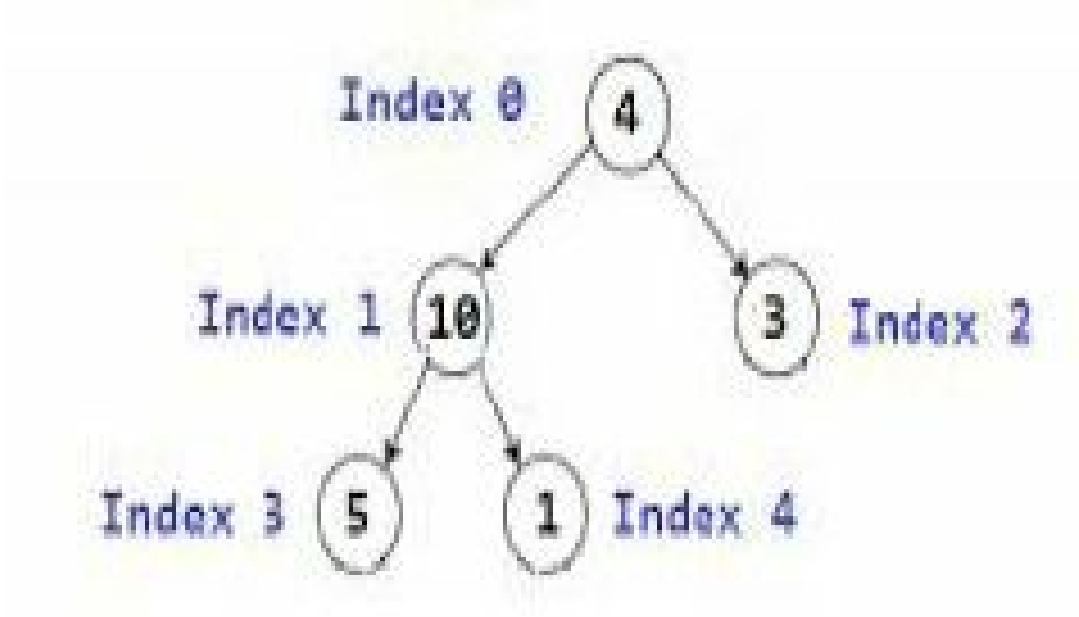
Assume for now that our intention to make a heap and then sort or heap sort in other words with a max heap in mind.

Let us start with the node in index position 0 which is node 4. We will use a formula to create a model starting at node at index level 0.

Left child = 2 * i + 1

Right child = 2 * i + 2

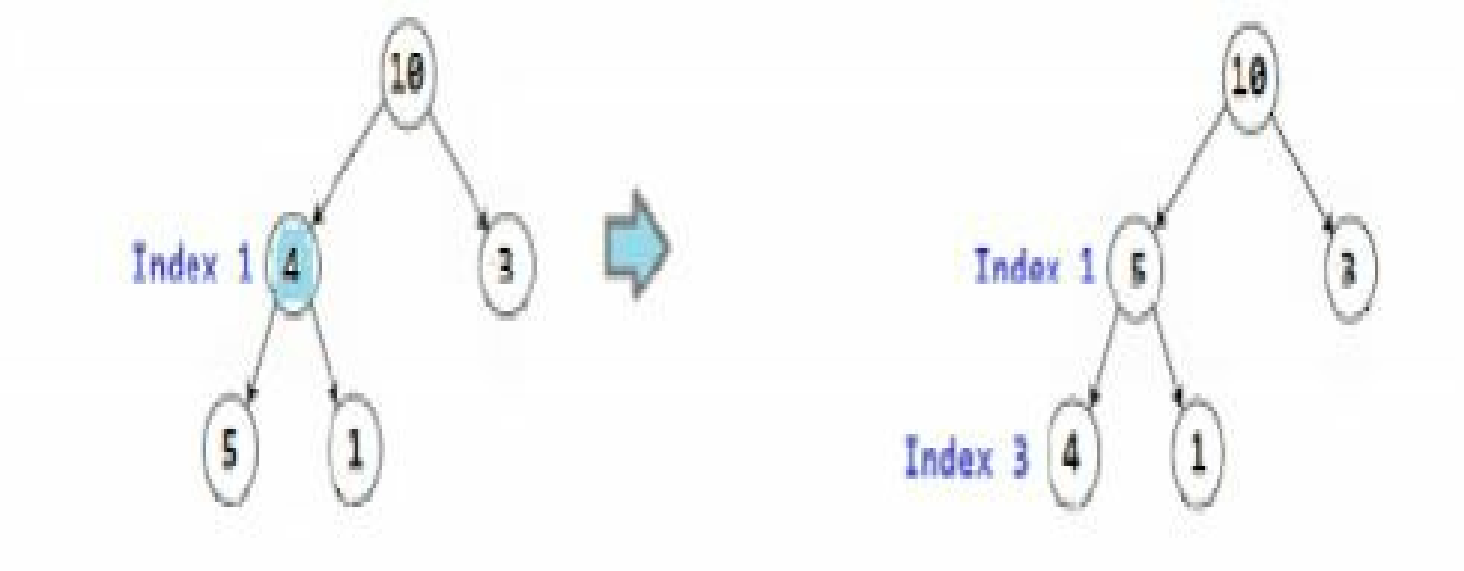
Thus we have the following:



We can use the formula to start the process at index = array size/2 -1 = 5/2 -1. Here if we take 5/2 as 2.5 -1 = 1.5 and take node index 2 we see that it has no children and it itself is smaller than node at index level 0. Thus we will start at node index level 1. 10 is bigger than 4 and greater than 5 and 1. Swap 10 and 4.

Then we will find that node at index

level 3 which is 5 is greater than node at index level 1 which is 4. Thus swap again to get the following with the following table:

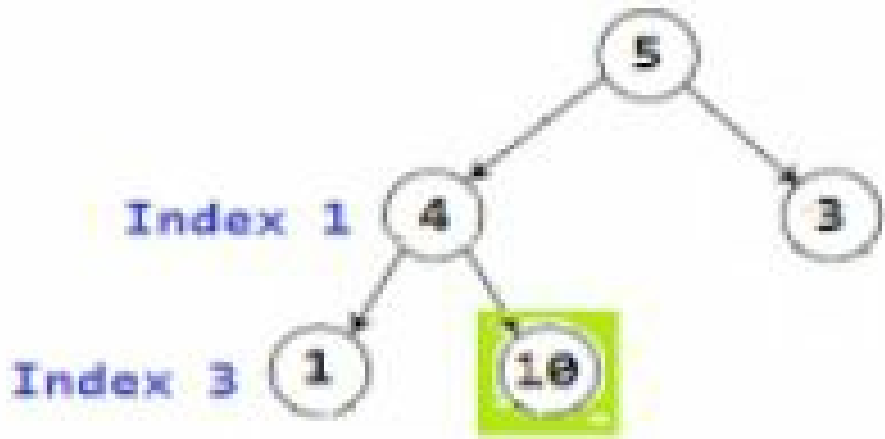


Node	10	5	3
Index	*0*	*1*	*2*

Now swap the value at the last and first indexes.

Node	1	5	3
Index	*0*	*1*	*2*

10 will not be part of the heapify process anymore. But by placing 1 at index 0 has disturbed the max heap structure so we do the same as before. We see that node at 1 is bigger than node at index 2. So let us swap 1 and 5. Then we swap 1 and 4 to get the following:



The new array table will look like this:

Node	5	4	3
Index	*0*	*1*	*2*

Swap the last and the first value of the array. Since 10 has been taken out from consideration we have the following:

Node	1	4	3
Index	*0*	*1*	*2*

Now we do not consider 5 and move on a similar way as before:

Since 4 is greater than 3 we will start with it and swap 1 and 4. We will get the array table as follows:

Node	4	1	3
Index	*0*	*1*	*2*

Now we will swap the last and first values of the array. As 5 and 10 have been taken out of consideration, we get the following table:

Node	3	1	4
Index	*0*	*1*	*2*

We continue the process with 4 taken out consideration. We note that in terms of a max heap of the remaining nodes the structure is correct as 3 is greater than 1. So final step to complete the sorting is just to start the last element of the array

and first to get the following sorted array.

Node	1	3	4
Index	*0*	*1*	*2*

Now let us look at the code:

```
import java.util.*;
public class lab16
{
private static int x;
public static void hsort(int a[])
{
hy(a);
for (int i = x; i > 0; i--)
{
swap(a,0, i);
```

```
x = x-1;
mh(a, 0);
}
}
public static void hy(int a[])
{
x = a.length-1;
for (int i = x/2; i >= 0; i--)
```

```
mh(a, i);
}
public static void mh(int a[], int z)
{
int left = 2*z ;
int right = 2*z + 1;
int max = z;
if (left <= x && a[left] > a[z])
max = left;
```

```
if (right <= x && a[right] > a[max])
max = right;
if (max != z)
{
swap(a, z, max);
mh(a, max);
}
}
public static void swap(int a[], int i, int
{
int tmp = a[i];
a[i] = a[j];
a[j] = tmp;
}
public static void main(String[] args)
{
Scanner scan = new Scanner( System.in
int n, i;
```

```
System.out.println("Enter number of inte
n = scan.nextInt();
int a[] = new int[ n ];
System.out.println("Enter integer eleme
space between them.");
for (i = 0; i < n; i++)
a[i] = scan.nextInt();
hsort(a);
System.out.println("The answer is: ");
for (i = 0; i < n; i++)
System.out.print(a[i]+" ");
System.out.println();
}    }
```

Chapter 17. Horner's Rule

Horner's rule is used to solve a polynomial. Suppose we have a polynomial in the form:

$$p(x) = a_n x^n + an_{-1} x^{n-1} + ... + a_1 x + a$$

according Horner's rule one can solve this polynomial for any value of x by the following formula:

$$p(x) = (... (a_n x + a_{n-1})x + ...)x + a_0$$

Let us look at an example:

$2x^4 - x^3 + 3x^2 + x - 5$ can be written as:

x(x(x(2x-1)+3)+1)-5 by factoring x one by one from the original equation:

Now to evaluate the value at any particular value of x we do not need the equation according to Horner. Suppose we want to evaluate the above equation at x=3. All we have to do is to make two rows. The top row with coefficients of the polynomial as follows:

2 -

1 3 1 -5

2 3*2 +(-1)= 5 3 * 5 + 3= 18 3 * 18 + 1= 55 3* 55 + (-5) = 160

next we do as above for x=3 to get the final answer as 160.

What we are doing is for the first column we keep the same coefficient value in the second row but for every other column we multiply x=3 with the previous column result and then add the value of the column coefficient in row 1 respectively.

Thus we do not have to even factor the polynomial as shown in the example. We can get the answer even from that but no need to.

Now let us look at the code. A point in this code is that values are being hardcoded in the program. Further the coefficient values are being entered in

reverse that is because of the way the for loop has been made.

Remember that in the calculation above the first value remains the same as the coefficient.

```
public class lab17 {
public static double computePolynomial
double x) {
int n = val.length - 1;
double y = val[n];
for (int i = n-1; i >= 0; i--) {
y = (y * x) + val[i];
}
return y;
}
public static void main(String[] args) {
double[] p = new double[]{ -5, 1, 3, -1,
System.out.println("Polynomial: 2x^4 -1
5");
System.out.print("x=3: ");
System.out.println(HornersRule.compute
}
}
```

Run the code. Do you get the same answer 160?

Exercise

1. Modify the program so that we can enter the coefficients can be entered in the order of the polynomial from 2..-5. (Hint: You may want to check on the for loop)

2. Using the scanner method enter the coefficients and value of x from the keyboard.

3. Run the code for the following polynomial:

$2x^5 - x^2 + x - 5$. at $x=5$

Will the code work. How is this done manually?

4. Work out the answer for the following manually:

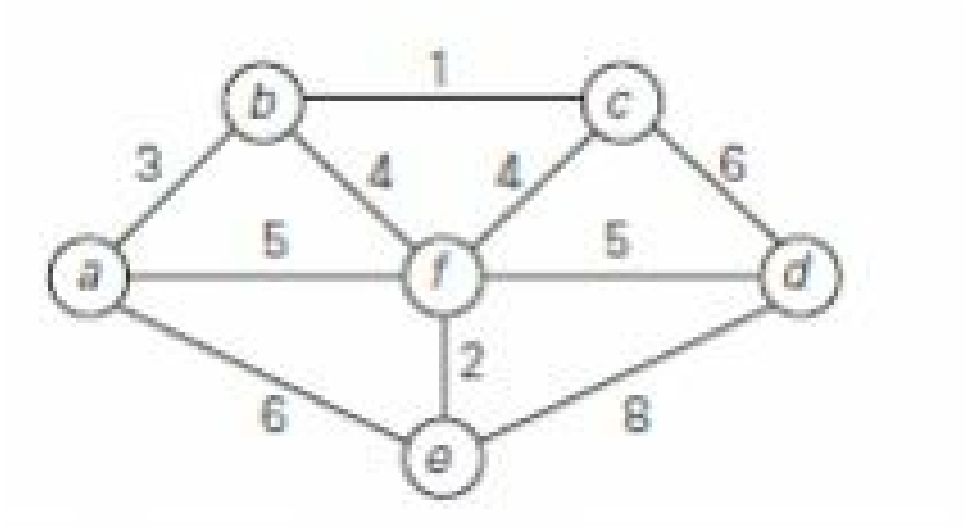
$x=2$

$5x^4 + 2x^3 - 3x^2 + x - 7$ Run the code to see if the answers match.

Chapter 18. Prim's Algorithm

A spanning tree of an undirected connected graph is its connected acyclic sub graph containing all the vertices of the graph. If such a graph has weights assigned to its edges, a minimum spanning tree is its spanning tree of the smallest weight, where the weight of a tree is defined as the sum of the weights on all its edges.

Let us look at an example:



If we assume that vertex a is the starting point, then the remaining vertices from a are as follows:

a (--) b(a,3) c (-,-), d(-,-) e(a,6), f(a,5)

the shortest is b so the next node is b.

b(a,3) and remaining are: c (b,1) d(-,-), e(a,6), f(b,4)(note that c and f are connected to b)

we select c based on the shortest path or lowest weight.

c (b,1) and remaining are: d(c,6) f(b,4) e (a,6)

we select f (b,4) and the remaining are: d(f,5) e (f,2)

we select e (f,2) and the remaining is d(f,5)

Lastly we select d(f,5) and there are no more remaining nodes or vertices. We can represent this graph in the form of a matrix. Assume for the sake of the code that follows that A is node or vertex 1

and the others in alphabetical order are 2,3,4,5 and 6 respectively.

Then we get the following matrix.

	A	B	C	D	E	F
A	0	3	0	0	6	5
B	3	0	1	0	0	4
C	0	1	0	6	0	4
D	0	0	6	0	8	5
E	6	0	0	8	0	2
F	5	4	4	5	2	0

The numbers are the respective weights or path lengths as shown in the graph in the previous page.

```
import java.util.*;
```

```
public class lab18
{
public  int iv[] = new int[15];
public  int cost[][] = new int[10][10];
public int minimum_cost;
public void calc(int n)
{
int flag[] = new int[n+1];
int
i,j,min=999,num_edges=1,a=1,b=1,minp
while(num_edges < n)
{
for(i=1,min=999;i<=n;i++)
for(j=1;j<=n;j++)
if(this.cost[i][j]<min)
if(this.iv[i]!=0)
{
min=this.cost[i][j];
```

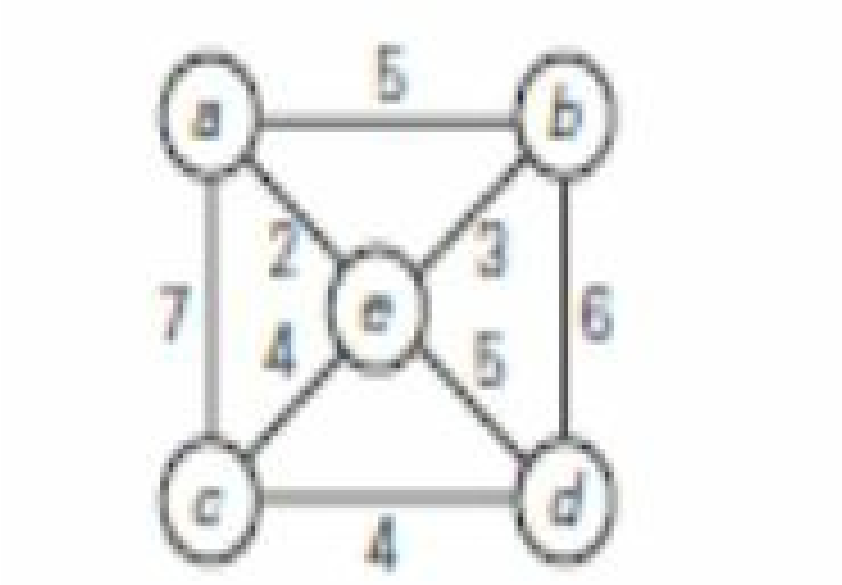
```
a=minpos_i=i;
b=minpos_j=j;
}
if(this.iv[minpos_i]==0 || this.iv[minpos
{
System.out.println("Edge Number \t"+nu
Vertex \t"+a+"\t  to Vertex \t"+b+"-minco
this.minimum_cost=this.minimum_cost+
num_edges=num_edges+1;
this.iv[b]=1; }
```

```
this.cost[a][b]=this.cost[b][a]=999;
}
}
public static void main(String args[])
{
int nodes,i,j;
Scanner in = new Scanner(System.in);
```

```
System.out.println("Enter the Number of
nodes = in.nextInt();
lab18 x = new lab18();
System.out.println("Enter the Weights in
matrix: \n");
for(i=1;i<=nodes;i++)
for(j=1;j<=nodes;j++)
{
x.cost[i][j]=in.nextInt();
if(x.cost[i][j]==0)
x.cost[i][j]=999;
}
x.iv[1]=1;
x.calc(nodes);
}   }
```

Exercises:

1. Apply the prim's algorithm manually to the following graph:



2. Create the matrix of weights for the above graph.

3. See if the program above produces the same result obtained through a manual process.

Chapter 19. Kruskal's Algorithm

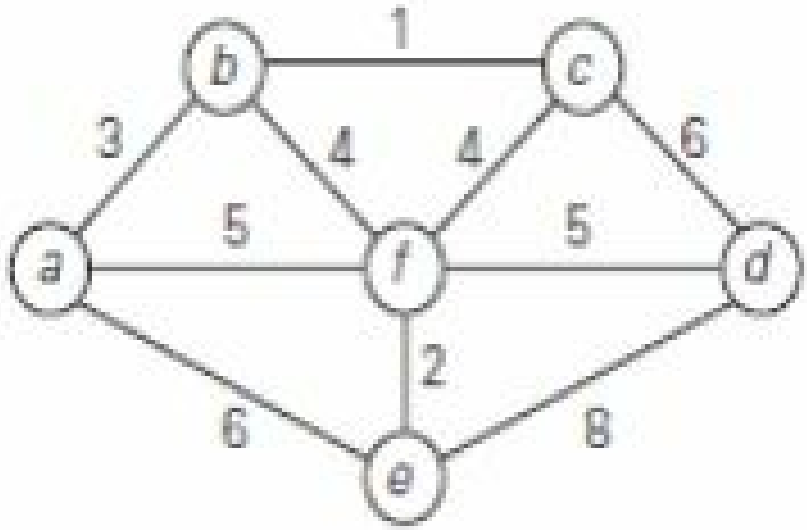
Kruskal's algorithm is for finding an optimal solution for a minimum spanning tree. This type of an algorithm is called a greedy algorithm.

The algorithm constructs a minimum spanning tree as an expanding sequence of subgraphs that are always acyclic but are not necessarily connected on the intermediate stages of the algorithm.

The algorithm begins by sorting the graph's edges in non-decreasing order of their weights. Then, starting with the

empty subgraph, it scans this sorted list, adding the next edge on the list to the current subgraph if such an inclusion does not create a cycle and simply skipping the edge otherwise.

Let us look at an example:



Given the graph with edges as shown let us apply the Kruskal's algorithm. First we need to sort the edges by their

weight. Thus we get:

1 2 3 4 4 5 5 6 6 8
bc ef ab bf cf af df ae cd de (Notice that the edges were taken in alphabetical order)

We start with selecting bc (lowest weight). The next would ef. Then ab followed by bf. Now at this point we have the following verices – a b f e. Now we have a choice between af and df. We cannot choose af because if we do it will create a cycle and that is not allowed as per the algorithm.

The same problem if we choose any of the remaining edges- ae cd or de. It will

create a cycle. So we stop. The final answer are the edges: bc ab bf ef df

Now let us look at the code below:

```
import java.util.*;
import java.lang.*;
import java.io.*;
 class lab19
{
class Edge implements Comparable<Ed
{
int src, dest, weight;
public int compareTo(Edge compareEdg
{
return this.weight-compareEdge.weight;
}};
class subset
```

```
{
int parent, rank;
};
int V, E;
Edge edge[];
lab19(int v, int e)
{
V = v;
E = e;
edge = new Edge[E];
for (int i=0; i<e; ++i)
edge[i] = new Edge();
}
int find(subset subsets[], int i)
{
if (subsets[i].parent != i)
subsets[i].parent = find(subsets, subsets
return subsets[i].parent;
```

```
}
void Union(subset subsets[], int x, int y)
{
int xroot = find(subsets, x);
int yroot = find(subsets, y);
if (subsets[xroot].rank < subsets[yroot].r
subsets[xroot].parent = yroot;
else if (subsets[xroot].rank > subsets[yr
```

```
subsets[yroot].parent = xroot;
else
{
subsets[yroot].parent = xroot;
subsets[xroot].rank++;
}
}
void Kruskal()
{
```

```
Edge result[] = new Edge[V];
int e = 0;
int i = 0;
for (i=0; i<V; ++i)
result[i] = new Edge();
Arrays.sort(edge);
subset subsets[] = new subset[V];
for(i=0; i<V; ++i)
subsets[i]=new subset();
for (int v = 0; v < V; ++v)
{
subsets[v].parent = v;
subsets[v].rank = 0;
}
i = 0;
while (e < V - 1)
{
Edge next_edge = new Edge();
```

```
next_edge = edge[i++];
int x = find(subsets, next_edge.src);
int y = find(subsets, next_edge.dest);
if (x != y)
{
result[e++] = next_edge;
Union(subsets, x, y);
}}
System.out.println("Following are th
constructed MST");
for (i = 0; i < e; ++i)
System.out.println(result[i].src+" -- "+
"+ result[i].weight);
```

```
}
public static void main (String[] args)
{
int V = 6;  // Number of vertices in grapl
```

```
int E = 10;  // Number of edges in graph
lab19 graph = new lab19(V, E);
// add edge 0-1
graph.edge[0].src = 0;
graph.edge[0].dest = 1;
graph.edge[0].weight = 3;
// add edge 0-4
graph.edge[1].src = 0;
graph.edge[1].dest = 4;
graph.edge[1].weight = 6;
// add edge 0-5
graph.edge[2].src = 0;
graph.edge[2].dest = 5;
graph.edge[2].weight = 5;
// add edge 1-2
graph.edge[3].src = 1;
graph.edge[3].dest = 2;
graph.edge[3].weight = 1;
```

```
// add edge 1-5
graph.edge[4].src = 1;
graph.edge[4].dest = 5;
graph.edge[4].weight = 4;
// add edge 2-3
graph.edge[5].src = 2;
graph.edge[5].dest = 3;
graph.edge[5].weight = 6;
// add edge 2-5
graph.edge[6].src = 2;
graph.edge[6].dest = 5;
graph.edge[6].weight = 4;
// add edge 3-4
graph.edge[7].src = 3;
graph.edge[7].dest = 4;
graph.edge[7].weight = 8;
// add edge 3-5
graph.edge[8].src = 3; graph.edge[8].de
```

```
graph.edge[8].weight = 5;
// add edge 4-5
graph.edge[9].src = 4;
graph.edge[9].dest = 5;
graph.edge[9].weight = 2;
graph.Kruskal();
}
}
```

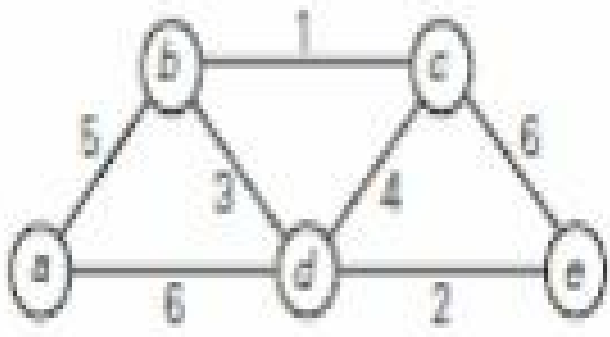
Run the code and it will be seen that the result will match that done by hand. The code for this is very long. The code requires each edge of the graph be entered assuming but there is no need to double back- meaning if edge a-b is of weight 3 there is no need to define b-a as 3 again. That is why in the above we

got 10 edges (0-9) for 6 vertices.

Exercise:

1. See if there is a way to take the number of vertices and edges from the keyboard using the scanner method and then using a for loop to enter the edges.

2. Apply the Kruskal's algorithm to the following manually:

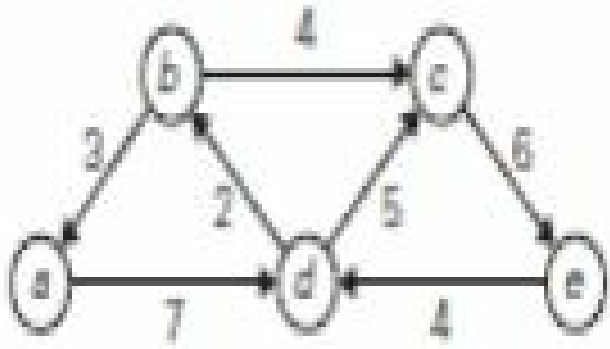


3. Enter the edges in the code above and

run it. Does the answers match?

4. What are the applications of Kruskal's algorithm?

5. Apply the Kruskal's algorithm to the following manually:

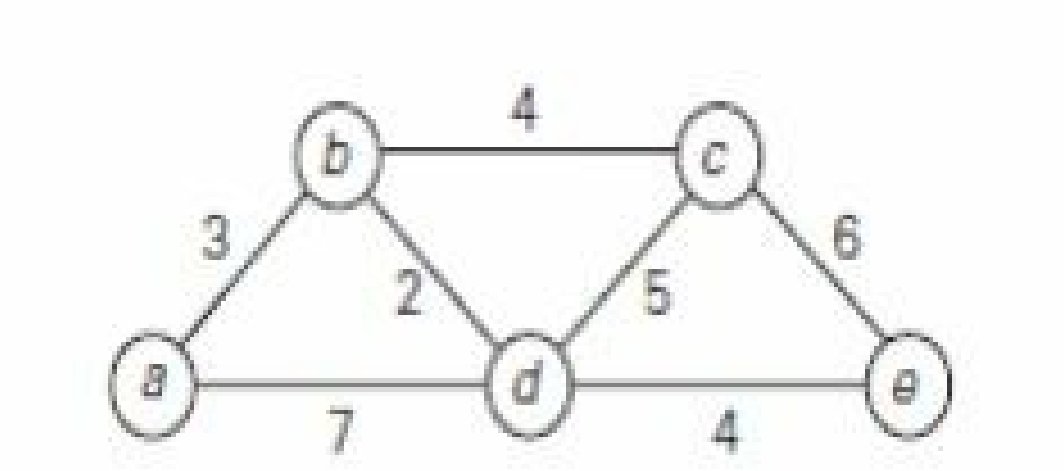


6. Run the code and see if the answer matches that of question 5.

Chapter 20. Dijkstra's Algorithm

This is an example of a greedy algorithm like the Kruskal's and Prim's algorithms.

Let us look at an example:



If the starting point is a then we have the

following:

a (-,0) = b(a,3) c(-,-) d(a,7) e (-,-)

In this case the shortest path is b(a,3)

b (a,3) = c (b, 3 +4=7) d(b, 3 +2 =5) e(-,-)

and we select d(b,5)

d(b,5) = c (b,7) e(d,5+4=9)

c (b,7) = e(d,9)

e(d,9) is the final step.

Thus the edges are:

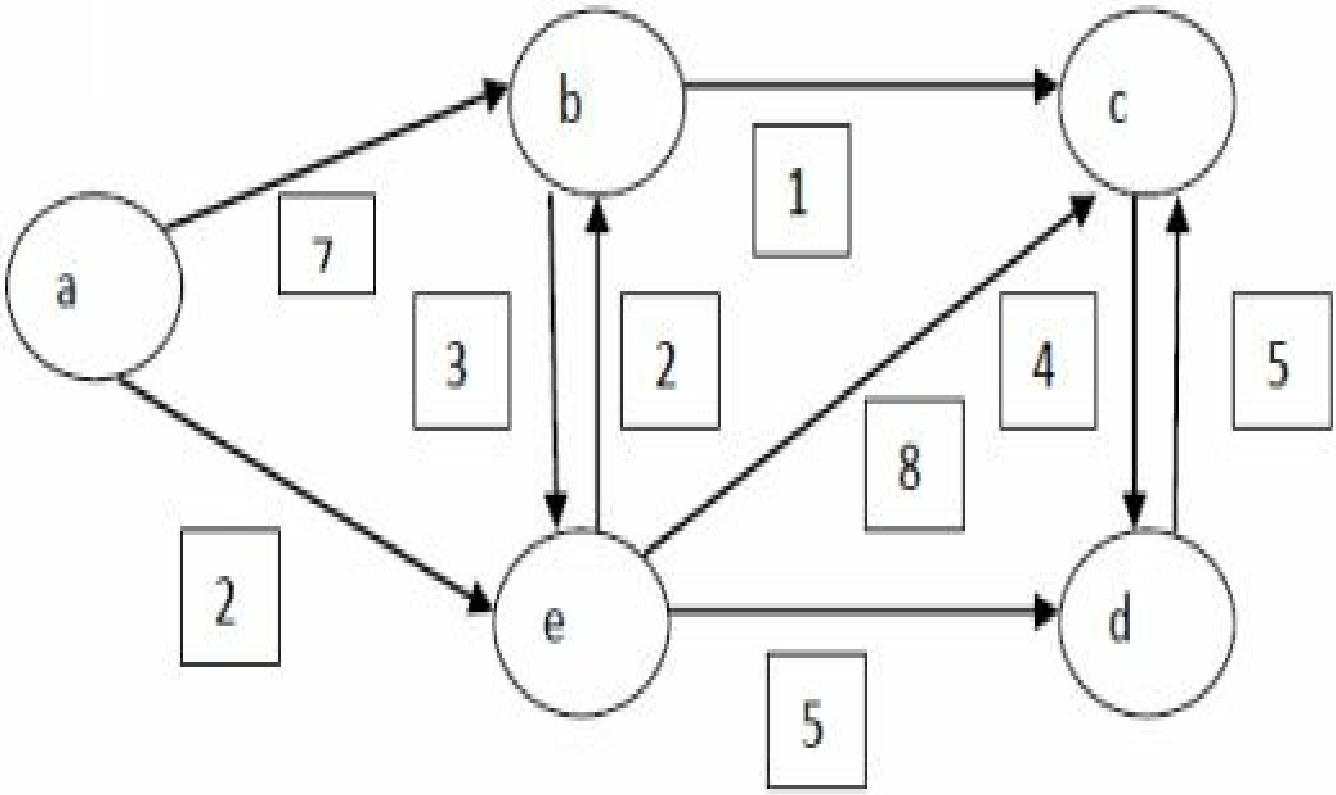
a-b of length 3

a-b-d of length 5

a-b-c of length 7

a- b -d -e of length 9

Let us look at another example:



Let the source be a. From here we can find the following:

a (0,-) = b(a,7) and e(a,2)

We select the shortest path that is e so we have:

e(a,2) = b(e,5) , c(e,10), d(e,7) (the

node preceding e is a)

We select b(e,5) so we have:

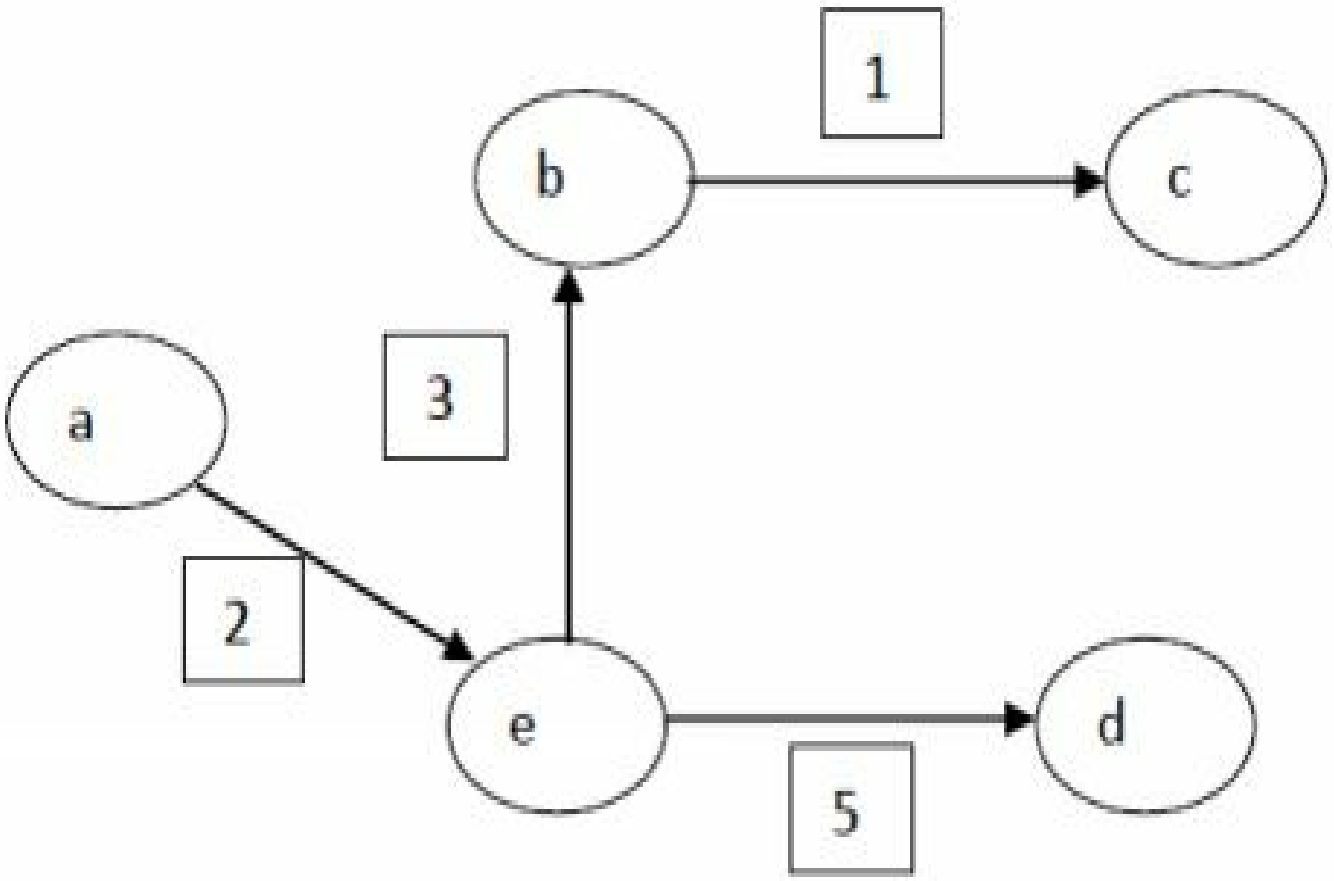
b(e,5) = c(b,6) , d(e,7) (the node preceding b is e)

We select c(b,6) and so we have:

c(b,6) = d(e,7) (we have reached the end. The node preceding c is b)

d(e,7) (the node preceding d is e).

Thus path is as follows:


1
b
c
3
a
2
e
d
5

Chapter 21. More Algorithms

Here we look at some more algorithms or cases where we can develop algorithms.

Fake coin problem

This is an example of the decrease-by-a-constant-factor type of algorithms. We have seen in the previous chapters, examples of such algorithms – Insertion sort and Topological sort.

In this problem let there are “n” identical-looking coins with one being fake. A logical approach would be to compare any two sets of coins using a balance scale to see if the left or the right hand of the scale will tip. We are assuming that there is a difference in weight between the fake and non-fake coins. Logically the fake coin should be lighter but need not be.

For a moment let us assume we know that the fake coin is lighter than the rest of the coins. We need an efficient algorithm for detecting the fake coin.

The most natural idea for solving this problem is to divide n coins into two piles of n/2 coins each, leaving one extra coin aside if n is odd, and put the two piles on the scale. If the piles weigh the same, the coin put aside must be fake; otherwise, we can proceed in the same manner with the lighter pile, which must be the one with the fake coin.

We can easily set up a recurrence relation for the number of times we have

to weigh the different batches of coins. If W(n) represents the number of times we have to weight, then the worst case is given by:

$W(n) = W(n/2) + 1$ for $n > 1$, $W(1) = 0$.

This kind of search is similar to the binary search we have seen before except for the starting condition. The similarity is because of the fact that the binary search is also an example of a decrease by constant factor type of algorithm. If we solve recursively the answer to the above would be

$W(n) = Log_2 n$ is it efficient to divide into two piles or three? Try it out and see if

the problem indicated here can be written in a programming language such as Java or one of your choice.

Coin row problem

There is a row of n coins whose values are some positive integers c1, c2 ... c (n), not necessarily distinct. The goal is to pick up the maximum amount of money subject to the constraint that no two coins adjacent in the initial row can be picked up.

If F(n) is the maximum amount that can be picked up from the row of n coins. To derive a recurrence for F(n), let us divide into two groups- a) coins that include the last coin and b) without it.

The largest amount we can get from the first group is given by:

C(n) + F (n − 2)—the value of the nth coin plus the maximum amount we can pick up from the first n − 2 coins.

The maximum amount we can get from the second group is equal to F (n − 1) by the
definition of F(n). Thus, we have the following recurrence subject to the obvious
initial conditions:

$$F(n) = \max \{c(n) + F(n-2), F(n-1)\} \text{ for } n > 1,$$

$$F(0) = 0, F(1) = c1$$

So let us look at an example:

Suppose we have the following coins in a row: 5 1 2 10 6 2 (denominations)

Put them in a table as such:

Index	0	1	2	3	4
Coin(value)		5	1	2	1
F	0	5	5	7	1

F (0) =0 F (1) = C1 = 5 Thus F (2) = max {1 + 0, 5} = 5 F (3) = max {2 +5 ,5} =7

F (4) = max {10+5, 7} = 15 F (5) = max {6+7, 15} = 15 F (6) = {2 + 15, 15} = 17

The largest value is 17 which came from C6 + F (4) thus coin 6 or 2 is one of the solutions. Now let us look at F (4). The maximum is 15 made by c4 + F (2) so coin 4 or 10 is part of the solution. Let us look at F (2). Here the maximum is 5 given by F (1) and not c2 thus c2 is not part of the solution but c1 is. That is the value of 5.

So the final answer is {c1, c4, c6} or 5 + 10 + 2 = 17 will be the highest amount that can be picked from this combination of denominations of coins.

This type of a problem is considered as a dynamic programming type of

algorithms.

See if this can be programmed in Java or a similar language of your choice.

Coin Change Problem

Suppose we have some coins c1, c2… c(n) and we want to give change with the minimum number of coins.

Let F(n) be the minimum number of coins whose value adds to up to n. Let F (0) =0

The amount n can only be obtained by adding one coin of denomination c_j to the amount $n - c_j$ for $j = 1, 2, \ldots, m$ such that $n \geq c_j$.

Therefore, we can consider all such denominations and select the one minimizing:

$F(n - c_j) + 1$. We can find the smallest F $(n - c_j)$ and add 1 to it. Hence, we get the following:

$F(n) = \min_{j:n>=cj} \{F(n - c_j)\} + 1$ for $n > 0$

$F(0) = 0$.

Let us look at an example:

Let us have three denominations 1, 3 and 4 and the amount to be given is 6.

n	0	1	2	3
F	0			

F (0) =0 as per definition above.

F (1) = min {F [1 -1]} + 1 = 1 (This is by taking n=1 and seeing that the only denomination that can be taken is 1 out of the three 1,3 and 4)

n	0	1	2	3
F	0	1		

F (2) = min {F [2-1]} + 1 = 2 (Here the n=2 and the highest denomination we can take is 1)

n	0	1	2	3
F	0	1	2	

F (3) = min {F [3-1], F [3-3]} + 1 (Here

n =3 and we have two denominations 1 and 3 that can be taken.)

Why can't we take 4? Well, let us assume n=3 is the maximum at this stage that can be given. If that is the case cannot a denomination 4 which is bigger than n=3.

n	0	1	2	3
F	0	1	2	1

F (4) = min {F [4-1], F [4-3], F [4-4]} +1 = 1 (n=4 and all three denominations can be subtracted from it.)

n	0	1	2	3
F	0	1	2	1

F (5) = min {F [5-1], F [5-3], F [5-4]} + 1 = 2

n	0	1	2	3
F	0	1	2	1

Similarly, F (6) = min {F [6-1], F [6-3], F [6-4]} + 1 = 2

To get the solution we can start with the last column n=6 we see that the minimum for this was obtained from second coin c2 whose value is 3. Since 6 is the change we need to give- 6-3 =3 is the remaining amount and F (3) minimum was also given by the second coin of

value 3. Thus in this case the answer is two 3's should be given.

As an exercise try to find the value to be given as change is 9 and the denominations 1, 3, 4, 5. Try to see if this dynamic programming example can be programmed in Java or a similar language of your choice. What happens when there are two min points? How to select. Select the coin with a higher value.

Huffman Coding

Initialize n one-node trees and label them with the symbols of the alphabet given. Record the frequency of each symbol in its tree's root to indicate the tree's weight. The weight of a tree will be equal to the sum of the frequencies in the tree's leaves.

Repeat the following operation until a single tree is obtained. Find two trees with the smallest weight. Make them the left and right sub tree of a new tree and record the sum of their weights in the root of the new tree as its weight.

A tree created in this manner is called a

Huffman Tree and the procedure of getting the tree is called Huffman coding.

Let us look at an example:

Consider four symbols – A B C D - with the following frequencies: 0.30 0.1 0.2 0.25 and 0.15 respectively.

We will construct the Huffman tree as follows:

Take lowest two: 0.1 and 0.15 and put them into a group. Let this be the left group and therefore the right group would contain 0.2 0.25 and 0.30.

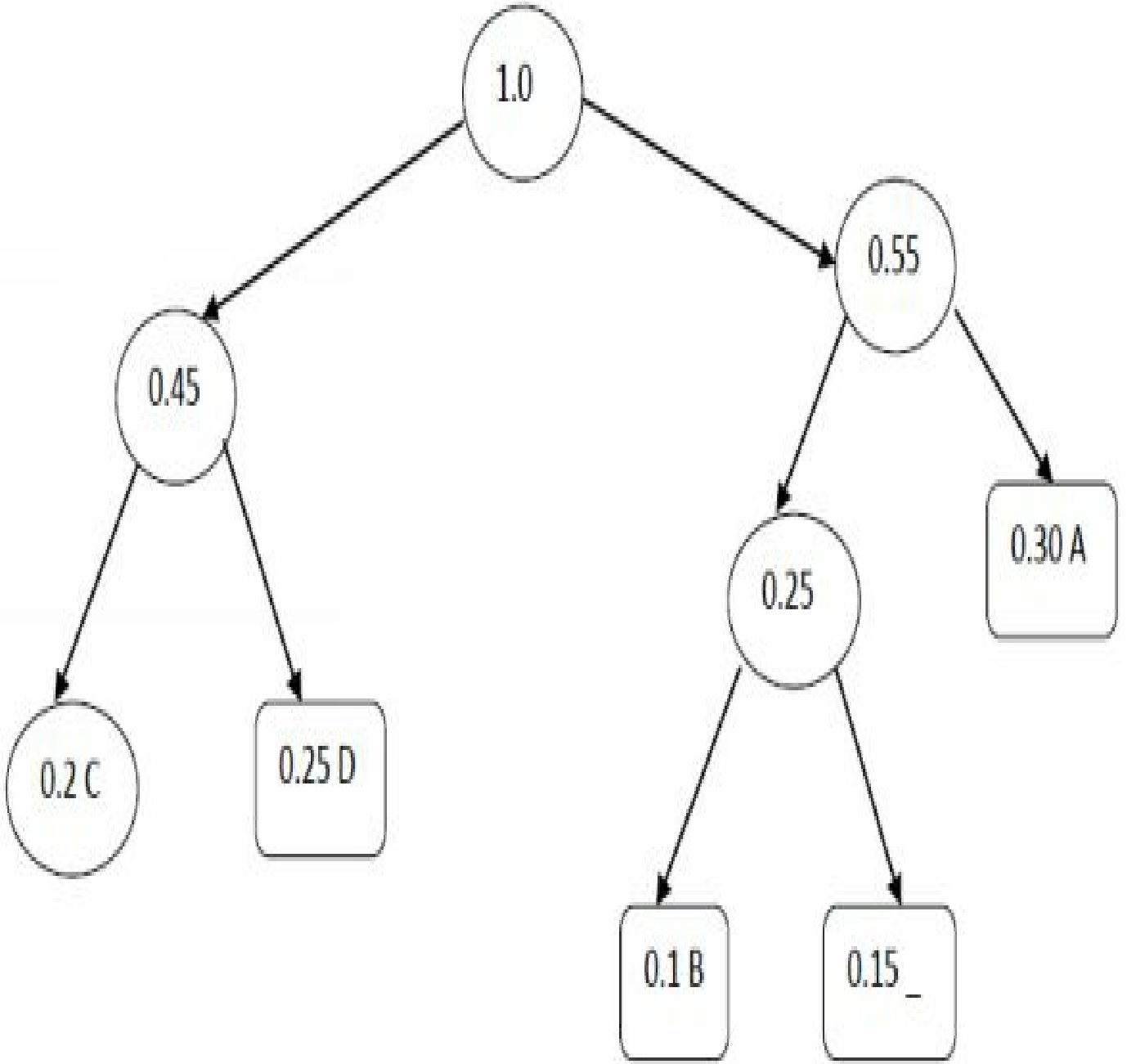
Now take from the right group another

group of lowest numbers and that would be 0.2 and 0.2 to form a subgroup of the right hand side. Now put the number together as follows:

We now select two lowest weights and group them together as above. This time we find that 0.25 + 0.30 = 0.55 and the other is 0.45. Since we cannot get another group of 2 we keep the lowest to

the left and highest to the right as follows:

As all probabilities have to add up to 1 we have the following: Number each left side as 0 and right side 1



Now we have the resulting code words as follows:

A B C D _ the frequency is 0.30 0.1 0.2 0.25 and 0.15 and code words are: 11, 100, 00 and 01 respectively. So if we want to encode the word BAD it will be 1001101.

Try to code this in Java or a similar language of your choice. The Huffman coding is an example of a greedy algorithm similar to the Prim's, Kruskal and Dijkstra's algorithms.

So try out the following manually:

H E L O A – with frequencies of 0.1 0.2 0.3 0.1 0.2 0.1 respectively.

If the word HELLO has to be encoded

how that would be achieved through Huffman coding?

Closest pair concept

In this concept the closest two point among n in a set is being sought after. This concept is very important in geometry and used in statistics for cluster analysis. On a practical basis finding two closest branches of bank etc is of importance to decision makers as they may be thinking of closing one of them to save on cost.

If we have two points A and B represented by (x,y) (w,z) then the distance between these two points is given by the Euclidean method and can be represented as follows:

Distance between A and B = square root of $[(x-w)^2 + (y-z)^2)]$

Brute force techniques can be used to solve the problem. Here the basic operation is the square root.

Convex Hull Concept

In this concept by definition a set of points in a plane is considered to convex if for any two points x and y in the set the entire line segment with endpoints x and y belongs to the set.

Thus a triangle, square, rectangle, circle etc are convex. The vertices of a triangle form what mathematicians call the extreme points. For a circle the extreme points are all points on the circumference.

Further to the definition of convex, the convex hull of a set S of points is the smallest convex set containing S.

Extreme points are very important because they have unique characteristics which are helpful to find solutions in linear programming. The Simplex method to solve linear equations bound by some constraints is an example where convex hull concept is very useful.

Simplex Method

Simplex algorithm is used to solve linear equations bound by some constraints. The main purpose of this method is to calculate the maximum or minimum of an objective based on some given constraints.
Let us look at an example:

Maximize $Z = 80x1 + 55x2$

Subject to

$4x1 + 2x2 \leq 40$
$2x1 + 4x2 \leq 32$
and $x1 \geq 0, x2 \geq 0$

Let us assume for now Z represents the profit that needs to be maximized based on two items to be produced. These two items are x1 and x2. The constraints let us assume are hours on two machines A and B whose working hours are given as 40 and 32 respectively. Assume product x1 requires 4 hours on A and 2 hours on machine B. Product x2 requires 2 hours on Machine A and 4 hours on Machine B.

We want to know how many of x1 and x2 to produce to maximize the profit given the constraints.

To do this the algorithm first asks the <= to be replaced by =. This is done by

adding slack variables 's' to each variable. If the sign of the constraint equations were >= then we will be adding surplus variables. The difference is we add (+) slack variables and subtract (-) surplus variables.

So the above equations will be modified as follows:

Maximize $Z = 80x1 + 55x2 + 0s1 + 0s2$

Subject to

$4x1 + 2x2 + s1 = 40$
$2x1 + 4x2 + s2 = 32$
$x1 \geq 0, x2 \geq 0, s1 \geq 0, s2 \geq 0$

If the objective was a minimization, then we have to convert the minimization to maximization by multiplying the objective statement by -1. After finding the answer one has to by -1 to get the correct answer.

Now we have to make a table to compute the values of x1 and x2. We will be calculation the pivots etc to find which variables will produce the optimum value for the profit.
The first table will look like this:

	$C_j \rightarrow$		80	55	0	0	
Basic Variables	C_B	X_B	X_1	X_2	S_1	S_2	Min ratio X_B / X_k
s_1	0	40	4	2	1	0	40 / 4 = 10 → outgoing
s_2	0	32	2	4	0	1	32 / 2 = 16
		$Z = C_B X_B = 0$	↑incoming $\Delta_1 = -80$	$\Delta_2 = -55$	$\Delta_3 = 0$	$\Delta_4 = 0$	

Notice the s1 and s2 are slack variables initialized to 0. We now have to find the outgoing and incoming. To do this we have to calculate the value of z as follows:

$z1 = 0 * 4 + 0 * 2 = 0$

$z2 = 0 * 2 + 0 * 4 = 0$

$z3 = 0 * 1 + 0 * 0 = 0$

$z4 = 0 * 0 + 0 * 1 = 0$

This is to be expected in all starting points because s1 and s2 =0.

Now do z -c for each column. This will give -80, -55 etc as shown above. The highest negative value is -80 so that will be the incoming. This represents x1 column, so take each member of x1 column 4 and 2 and divide 40 and 32 respectively. The lowest value will indicate the outgoing. Thus the outgoing variable is x1 and is the first row with pivot as 4.

Using the pivot calculate the new row1and row 2. The new row1 will be:

4/4 2/4 1/4 and 0/4 = 1 1/2 1/4 and 0

and 10 for the right hand side.

To calculate the second row, follow this rule:

old number in row - (number above or below pivot * number in pivot row matching the old number)

Thus we will get the second row as follows:

2 - (2 * 1) =0
4- (2 * 1/2) = 3
0 - (2 * 1/4) = - 1/2
1- (2 * 0) = 1
Calculate the right hand side of row 2 which is

32 - (2 * 10) = 12

x_1	80	10	1	1/2	1/4	0	10/1/2 = 20
s_2	0	12	0	3	-1/2	1	12/3 = 4→ outgoing
	Z = 800		↑incoming $\Delta_1=0$	$\Delta_2=-15$	$\Delta_3=40$	$\Delta_4=0$	
x_1	80	8	1	0	1/3	-1/6	
x_2	55	4	0	1	-1/6	1/3	
	Z = 860		$\Delta_1=0$	$\Delta_2=0$	$\Delta_3=35/2$	$\Delta_4=5$	

Thus we remove s1 and replace it with x1 and its value and we calculate z

again.

$z1 = 80 * 1 + 0 * 0 = 80$
$z2 = 80 * 1/2 + 0 * 3 = 40$
$z3 = 80 * 1/4 + 0 * -1/2 = 20$
$z4 = 80 * 0 + 0 * 1 = 0$

Now do z-c which will give you the values as show in table above which are: 0 -15 40 and 0 respectively.

By the same rules in the last page choose the largest negative number to be incoming and find the lowest right hand side for outgoing. So here it is the x2 column this time and the outgoing is x2 based on row 2.

We calculate as before to get the third row as shown above. In this row we notice the values of z-c have become positive. That means no need to go further. The maximum profit is given by Z= 860 and x2 which has replaced s2 in third row is equal to 4 and x1 =8.

As mentioned before if the objective is minimization the procedure is the same except for converting the objective statement to maximization by multiplying by -1 and then finding the answer and multiplying that by -1 to get the correct answer.

Assignment Algorithm

The assignment algorithm assumes that if there are x number of people and y number of tasks then each task should be assigned to a single person only. The practicality of that may be in question as sometimes some jobs do require a team of workers. But then each team can be considered as a single entity thus rolling back to the definition of the assignment algorithm.

Let us look at an example:

	J1	J2	J3	J4	J5
M1	7	5	9	8	11
M2	9	12	7	11	10
M3	8	5	4	6	9
M4	7	3	6	9	5
M5	4	6	7	5	11

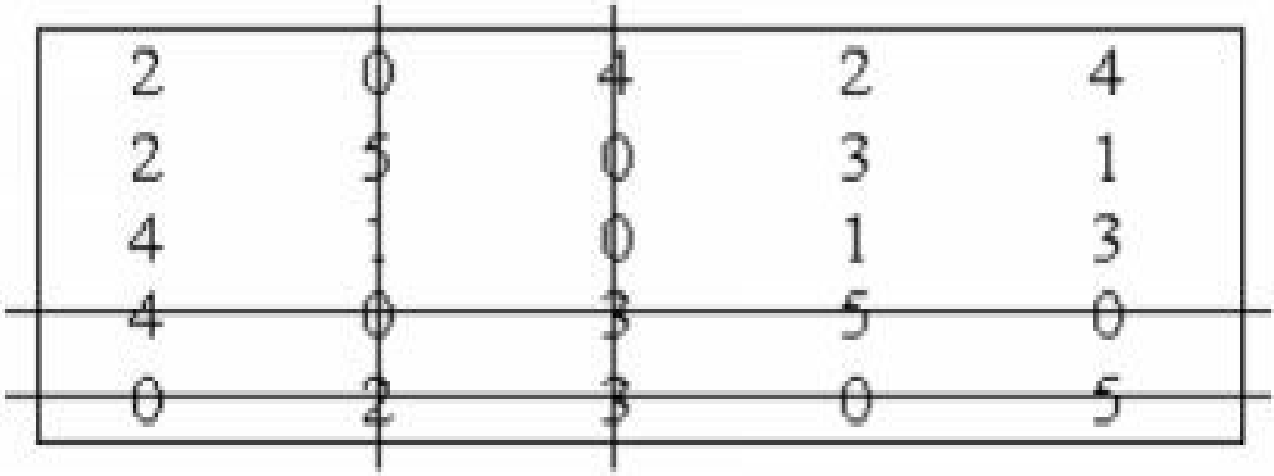
Let us assume M1 -M5 are machines and J1-J5 represent jobs.

We first do row reduction by taking the smallest in each row and subtracting the row numbers with it. We will get the following:

Row Reduced Matrix

2	0	4	3	6
2	5	0	4	3
4	1	0	2	5
4	0	3	6	2
0	2	3	1	7

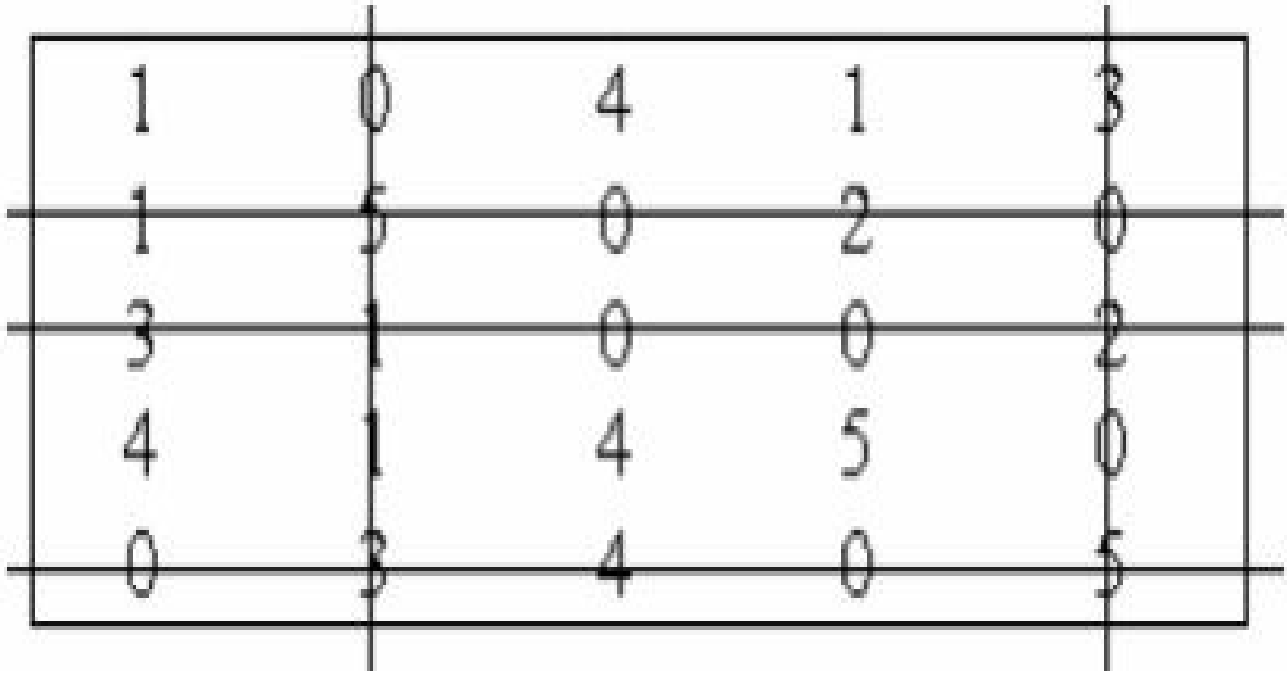
Now we have to reduce each column by taking the smallest number in each column and subtracting the column numbers with it. After this we will draw the minimum number of lines (horizontal or vertical) to cover all the zeros we got in the column reduced matrix. We will get the following:



Count the number of lines. We see that

we got 4 lines. The size of the matrix is 5 by 5. Thus the number of lines is less than 5. In this case we need to modify one more time. If the number of lines is equal to the size of the matrix then we do not need to modify it further.

The rule for modification is to find the smallest number in the above modified table which is not covered by a line. In this case it is 1. Then subtract all numbers which are not covered with it. Those numbers on a line are to be left untouched. If a number falls on the intersection of two lines, then add the smallest number to it. So we get the new modified table as follows:



Notice we have already redrawn new lines (horizontal or vertical) on the new modified table to cover all the zeros. We also have the number of lines equal to size of that matrix that is 5. So we can stop and do the assignment.

1	[0]	4	1	3
1	5	[0]	2	~~0~~
3	1	~~0~~	[0]	2
4	1	4	5	[0]
[0]	3	4	~~0~~	5

So the assignment will be as follows: M1 - J2 because that row has only 1 zero. M2 will be give J3 because even though M3 could do it- no task can be given two machines as per the rules of assignment. Thus M3 will be given J4, M4 will be given J5 and M5 will be give J1.

Decision Trees

Decision trees allow for managers to decide on options to select.
Decision trees provide a graphical approach to the various options with respect to each decision taken based on as much an accuracy on probability of that option being taken.

An example of a decision tree could be the following:

A decision to buy, rent or lease a vehicle. If person A is thinking of getting a vehicle, he could either buy the vehicle which at the end of eight years of its life could fetch only 10% of its purchase

price or lease the vehicle for three years and decide to buy the vehicle or get it replaced with a new vehicle on similar terms of the lease. Alternatively, the person could just rent a vehicle on a monthly or annual basis. The result of each option needs to be seen before a decision can be made on how to go forward.

Algorithms help create programs or steps to calculate values to help make decisions in this type of situations.

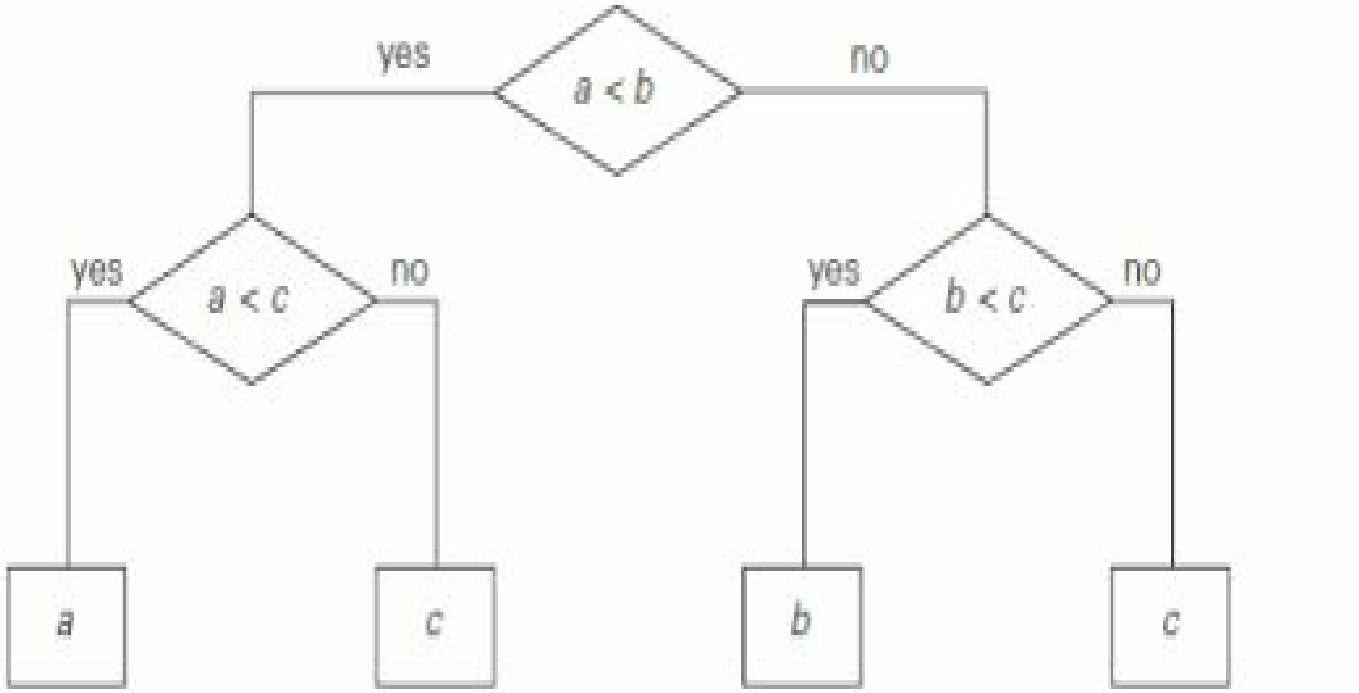
So if the vehicle costs $20,000 in eight years it is worth only $2000. Assuming no additional cost for selling and keeping in mind the maintenance cost

over the eight years is selling the vehicle of any benefit? Well keeping it might only add cost. But as for any profit at selling that needs to be calculated. There is also the issue of the vehicle was purchased through a loan or on cash basis which will add to the cost of the vehicle.

On the other hand, if the vehicle is leased a down payment has to be paid an annual or monthly lease amount will have to be paid. The vehicle cannot be sold unless it is bought. Renting the vehicle means there is no obligation to buy and no down payment in most cases.

Each of these scenarios will need to be

looked into with details. The creation of a chart with branches makes it appear in the form of a tree and hence the name decision trees. Let us look at another example of finding the minimum of three numbers. The decision tree of this will look like this:



The diagram above says that if there are three numbers a, b and c and we need to find the minimum of these of numbers then we start with the first two numbers

a and b. If a < b then check if a is less than c. If yes, the answer is a otherwise it is c. Similarly, if b is less than a then check b with c. If b is less than c then answer is b otherwise it is c.

Transportation Model

In this model, we are transporting from multiple origin to various destinations with different costs. The total availabilities will meet the total requirements at the same time minimizing the total cost of transportation. There are many methods to do this. One of the models is called the least cost method. Let us look at an example:

	To					Availability
From	2	11	10	3	7	4
	1	4	7	2	1	8
	3	9	4	8	12	9
Requirement	3	3	4	5	6	

Find the lowest number in the matrix and then check the availability and requirement values for it. For example, in the example above the lowest value is 1 and there are two of them. So select arbitrarily. If we take 1 in the first column, then the availability is 8 and the requirement is 3. Select the minimum of (8,3) which is 3 and assign it to the cell where 1 is. Make requirement to 0 and availability to 8-3= 5. Since we selected the requirement amount as the minimum the column will now be removed from further consideration. Go for the next smallest number which is 1 in the last the column and do the same. Here the minimum is (5,6) and thus 5. Assign 5 to

cell containing 1 and remove the row from consideration since the minimum we have chosen came from availability and adjust the values of availability and requirements accordingly. Get the next smallest number and continue this way until all rows and columns are removed from consideration. Note if availability and requirements are equal either the row or the column can be removed but not both. The final answer is 78.

			4 (3)		4 0
3 (1)				5 (1)	8 5 0
	3 (9)	4 (4)	1 (8)	1 (12)	9 5 4 1 0
3 0	3 0	4 0	5 1 0	6 1 0	

The value is obtained by 3 *1 + 3 *9 + 4 *4 + 1*8 +4*3 + 5*1 + 1 *12.

Appendix I- Algorithms

The order of growth of an algorithm's basic operation count is considered as the
principal indicator of the algorithm's efficiency.

There are three notations that one should be familiar with in this respect when it comes to algorithms -*O* (big oh), Ω (big omega), and Θ (big theta).

Let us assume t (n) and g(n) can be any nonnegative functions defined on the set of natural numbers. Let t(n) will be an algorithm's running time based on its count operation and g(n) will be some

simple function to compare it with.

Big O notations says that O (g(n) is the set of all functions with a lower or same order of growth as g(n). Thus n is member of $O(n^2)$ but n^3 is not.

The big Omega notation stands for all the set of functions with higher or same order of growth as g(n). So n^3 is a member of omega Ω (n^2).

The big Theta notation stands for all the set of functions with same order of growth as g(n). So $an^2 + 1$ is a member of theta $\Theta(n^2)$.

The orders of growth in these notations

can be calculated using comparing the limits of the two functions involved t(n) and g(n).

$\text{Lim}_{n->\text{infinity}}$ (t(n)/g(n)) =0 implies that t(n) has a smaller order of growth than g(n)

= c implies that t(n) has same order of growth than g(n)

= infinity implies that t(n) has a larger order of growth than g(n)

Thus the first will be a case of big O and second theta and the last Omega. Limit based comparison is more convenient and is based on Calculus techniques such as L'Hopital's rule and Stirling's

formula.

Let us look at an example: Compare 1/2 $n(n-1)$ and n^2. We put it in the limit formula format above:

$\lim_{n==>infinity}$ $1/2n(n-1)/ n^2 = 1/2$ $\lim_{n==>infinity}$ $(n^2 -n)/n^2 = 1/2$ thus we have the same order of growth and therefore big theta.

Appendix II – Big Integer

It was seen that in Chapter 1, on finding the GCD of two numbers the data type used was Big Integer versus Integer. Generally the BigInteger in the math package of Java is used when we have to deal with integers that are larger than 64 bit (the size of a long.

It's use in Euclid's theorem has been seen and I would like to point out that if one is doing a class in crypto logical algorithms many programs in Java will use the Big Integer.In summary let us look at some of the things that the Big Integer can be used for:

Constructors, constants, and static factor

bi =	new BigInteger(s);	C w re de
bi =	BigInteger.ONE;	Pr
bi =	BigInteger.ZERO;	Pr
bi =	BigInteger.**valueOf**(lng);	U m B nu A w pr
Arithmetic operations		
bi1 =	bi2.**abs**();	R ab
bi1 =	bi2.**add**(bi3);	R

		of
bi1 =	bi2.**divide**(bi3);	R of
bia =	bi2.**divideAndRemainder**(bi3);	R B re re re of
bi1 =	bi2.**gcd**(bi3);	R co of
bi1 =	bi2.**max**(bi3);	R of

bi1 =	bi2.**min**(bi3);	Returns mini of bi2 and bi
bi1 =	bi2.**mod**(bi3);	Returns rema dividing bi2
bi1 =	bi2.**multiply**(bi3);	Returns proc
bi1 =	bi2.**pow**(bi3);	Returns bi2
bi1 =	bi2.**remainder**(bi3);	Returns rema dividing bi2 negative.
i =	bi.**signum**();	-1 for neg nu and +1 for p
bi1 =	bi2.**subtract**(bi3);	Returns bi2
Conversion to other values		
d =	bi.**doubleValue**();	Returns doul

		equivalent o
f =	bi.**floatValue**();	Returns floa of bi.
i =	bi.**intValue**();	Returns int v of bi.
lng =	bi.**longValue**();	Returns long of bi.
s =	bi.**toString**();	Returns deci representatio
s =	bi.**toString**(i);	Returns strin of bi in radi
Other		
b =	bi1.**compareTo**(bi2);	Returns nega if bi1<bi2, 0 positive num

Big Integer variables are declared as

follows:

BigInteger c, c1, c2 etc

or in the form of a single dimension array as follows:

BigInteger [ ] c

We have already seen the code for finding the GCD in chapter 1 so we will not repeat that here. Let us look at some code for some of the other features listed in the summary table above.

Specifically, we will look at those for:

add, max, pow, abs and shift left.

Code for doing an add using Big Integer:

```
import java.math.BigInteger;
import java.util.Scanner;

public class BigIntegerAdd {
public static void main(String[] args) {
System.out.print("Enter the first value:")
Scanner sc = new Scanner(System.in);
String firstnumber = sc.nextLine();
System.out.print("Enter the second value
String secondnumber = sc.nextLine();
sc.close();
BigInteger x = new BigInteger(firstnumb
BigInteger y = new BigInteger(secondnu
BigInteger z = x.add(y);
System.out.println("Sum of x and y ="+z
```

```
}}
```

Here we are using the scanner method to input two values. Note here the firstnumber and secondnumber variables are accepting input in the first of a string. This in itself could prove to be wrong because if someone entered an alphabet the program will give an exception error. Nevertheless the variables x and y are getting values after converting the input into BigInteger datatype. How can you prevent someone from entering alphabets as inputs?

```
import java.math.BigInteger;
import java.util.Scanner;
public class BigIntegerAdd {
```

```
public static void main(String[] args) {
System.out.print("Enter the first value:")
Scanner sc = new Scanner(System.in);
BigInteger x = sc.nextBigInteger();
System.out.print("Enter the second value
BigInteger y = s.nextBigInteger();
sc.close();
BigInteger z = x.add(y);
System.out.println("Sum of x and y ="+z
}}
```

Thus we have modified the original program to define x and y as Big Integers right from the start. Now if you try to type in a character as the input it will give an exception error right from the start before even accepting the second input. The code still lacks the code for testing for input and looping back but

that is left as an exercise.

Now the code for maximum two numbers.

```
import java.math.BigInteger;
import java.util.Scanner;
public class BigIntegerMax {
public static void main(String[] args) {
System.out.print("Enter the first number:
Scanner sc = new Scanner(System.in);
BigInteger x = sc.nextBigInteger();
System.out.print("Enter the second numb
BigInteger y = sc.nextBigInteger();
sc.close();
BigInteger z = x.max(y);
System.out.println("The result is "+z);
}}
```

The two variables x and y has been defined as Big Integers and the result stored in z. Now let us the look at the code for pow.

```
import java.math.BigInteger;
import java.util.Scanner;
public class BigIntegerPow {
public static void main(String[] args) {
System.out.print("Enter the base number
Scanner sc = new Scanner(System.in);
BigInteger x = sc.nextBigInteger();
System.out.print("Enter the exponent:");
int y = sc.nextInt();
sc.close();
BigInteger z = x.pow(y);
System.out.println("The result is "+z);
```

```
}
}
```

One important point to note in this code is that the exponent to which we raising the base has to be entered as an integer and then when the result is found it will be converted to a Big Integer before display. Try to keep y as a Big Integer and run the code. What message do you get?

Now the code for abs or absolute.

```
import java.math.BigInteger;
import java.util.Scanner;
public class BigIntegerAbs {
public static void main(String[] args) {
```

```
System.out.print("Enter a value:");
Scanner sc = new Scanner(System.in);
BigInteger x = sc.nextBigInteger();
sc.close();
System.out.println("Absolute value ="+x
}}
```

Here a value either positive or negative can be given and its absolute value will be given out. Try a negative number. The absolute of a negative is positive.

Now the code for shifting left.

```
import java.math.BigInteger;
import java.util.Scanner;
public class BigIntegerShiftLeft {
public static void main(String[]args){
```

```
Scanner sc = new Scanner(System.in);
System.out.println("Please enter the num
BigInteger x = sc.nextBigInteger();
System.out.print("Enter the shift distanc
int y = sc.nextInt();
sc.close();
BigInteger z = x.shiftLeft(y);
System.out.println("Result of the operati
}
}
```

Here we enter a value in decimal. Say 17 and it will be seen in binary as 10001 and if we do a shift by 1 to the left the value of the result will be 34. 34 is obtaining by shifting the binary 10001 to the left. This will give binary as 100010 = $2^5 + 2^1 = 34$. Try with other shift

distances and see the result. Compare the binary values.

Appendix III- Arrays

As you saw in most of the scripts in this book we have used arrays to input values into the various programs.

So let us look at arrays.

An array, stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the

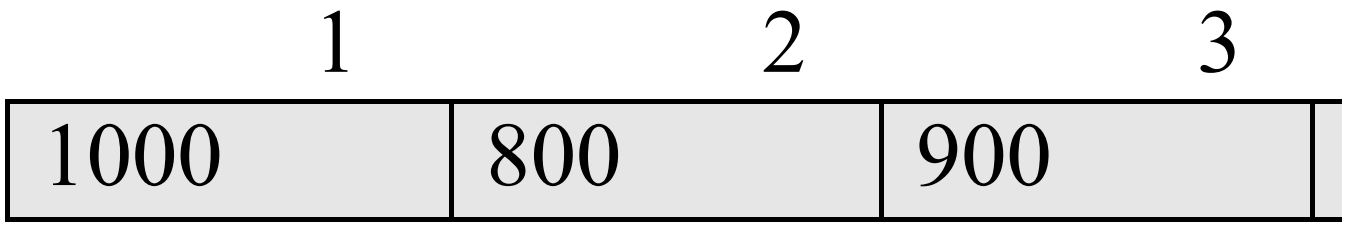
highest address to the last element. The lowest is represented by the index value 0.

In Java, we declare arrays as follows:

```
int x[];

x = new int[5];
```

This is called a single or one-dimensional array of 5 integers. Similarly, we can do the same for other data types. We could look at the above declaration in the following manner



The fifth position represented by index 4 will hold the value of 600 in this one-dimensional array.

In Java, all arrays store the allocated size in a variable called length.

We can get this length by using array name.length

In the case of our example:

int xsize = x.length;

Now let us look at the code for an array and its elements:

```
import java.util.*;
```

```
public class first
{
public static void main(String[]args)
{
int number[] = {55,10,20,30};
int n = number.length;
for (int i=0; i<n;i++)
{
System.out.println(""+number[i]);
}
        }
}
```

This will print the elements of this one dimensional array called number. The number.length will give the size of the array.

The format for a multidimensional array is as follows:

type name[size1][size2]...[size n];

The simplest form of the multidimensional array is the two-dimensional array. To declare a two-dimensional integer array of size x, y, you would write something as follows:

int x [] [] = new int [3] [2]

Thus, this means three rows and 2 columns

int a [2] [3] is an array with two rows and three columns. For example:

int a [2] [3] = {0, 1, 2, 3, 4, 5};

So here, the row 1 has the index 0 or in simple words, we count rows starting with 0 and not 1. Thus the row number for the above array is 0 & 1 and column numbers are 0 … 2.

Column 0 Column 1 Column 2

Row 0	0	1
Row 1	3	4

Now we access the elements of a multi-dimensional array in the same manner as we did for a single array by using a

loop. Here we will take a 2 row by 3 column array. To print out the values of this, we will use the 'for' loop.

```
import java.util.*
public class first
{
public static void main(String[]args)
{       int number[][]= {{0,0,0},{1,1,1}}
        for (int i=0; i<2;i++)
        {
        for (int j=0; j<3; j++)
        {
        System.out.println(number[i][j]);

        }
}}}
```

The output of this program would look like this:

0
0
0
1
1
1

Notice that in both these examples we are importing the util package of Java- Specifically, the Arrays methods under the util package.

The following lines of code represent how one can use the Arrays.sort method to sort out the output of an array list. It

also shows how to convert the output to a string.

```
int[]   ints = new int[10];
for(int i=0; i < ints.length; i++){
ints[i] = 10 - i;
}
System.out.println(java.util.Arrays.toStr
java.util.Arrays.sort(ints);
System.out.println(java.util.Arrays.toStr
```

Here is an example of the Array.sort method as applied to objects.

```
Employee[] employeeArray = new Emp
employeeArray[0] = new Employee("M
employeeArray[1] = new Employee("Jo
employeeArray[2] = new Employee("A
java.util.Arrays.sort(employeeArray, ne
```

```
Comparator<Employee>() {
@Override
public int compare(Employee e1, Emplo
return e1.name.compareTo(e2.name);
}
});
for(int i=0; i < employeeArray.length; i+
System.out.println(employeeArray[i].na
```

First a Java array is declared. The array is of type Employee. Next three employee objects are inserted into the array.

The Arrays.sort() method is called to sort the array. The parameter to the Arrays.sort we pass the employee array, and a Comparator implementation which

can determine the order of Employee objects.

Another thing we can do with arrays is to check if two arrays are equal.

```
int[] ints1 = {0,2,4,6,8,10};
int[] ints2 = {0,2,4,6,8,10};
int[] ints3 = {10,8,6,4,2,0};
boolean ints1EqualsInts2 = Arrays.equa
boolean ints1EqualsInts3 = Arrays.equa
System.out.println(ints1EqualsInts2);
System.out.println(ints1EqualsInts3);
```

This compares the array ints1 to the arrays ints2 and ints3. The first comparison will result in the value true since ints1 and ints2 contains the same

elements in the same order. The second comparison will result in the value false. Array ints1 contains the same elements as ints3 but not in the same order. Therefore, the two arrays are not considered equal.

For a two dimensional array the following code snippet can be used to read in values using the scanner method. Note the same carried on other multi-dimensional arrays.

```
public static void readArray(int[][] x, ir
Scanner input = new Scanner(System.in)
for (int r = 0; r < row; r++) {
for (int c = 0; c < col; c++) {
x[r][c] = input.nextInt();
}}}
```

Here the array name is x. Another way of doing exactly the same thing as above is as follows:

```
public static void readArray(int[][] x) {
Scanner input = new Scanner(System.in)
for (int r = 0; r < x.length; r++) {
for (int c = 0; c < x[r].length; c++) {
x[r][c] = input.nextInt();
}}}
```

The .length method which was used in single dimension to find the length of an array is being used here. The difference is that when there are two dimensions to an array a row and a column we cannot use the .length alone. The first line x.length handles the rows while the

second .length line handles the columns.We have already seen one method of printing out the elements in a two dimensional array in the beginning of this appendix. Another way to do the same using the length property is as follows:

```
public static void printArray(int[][] x) {
for (int r = 0; r < x.length; r++) {
for (int c = 0; c < x[r].length; c++) {
System.out.printf("%5d", x[r][c]);
System.out.println();
} }}
```

We can do sum of a 2 dimensional array using the code snippet such as:

```
public static int sumArray(int[][] x, int r
int sum = 0;
for (int r = 0; r < row; r++){
for (int c = 0; c < col; c++) {
sum = sum + x[r][c];
} return sum; }}}
```

Or by the following code snippet:

```
public static int sumArray(int[][] x) {
int sum = 0;
for (int r = 0; r < x.length; r++){
for (int c = 0; c < x[r].length; c++) {
sum = sum + x[r][c];
}
return sum;
}}}
```

To find the maximum in a 2 dimensional

array we could use the following code snippet:

```
public static int maxArray(int[][] x, int r
int max = x[0][0];
for (int r = 0; r < row; r++){
for (int c = 0; c < col; c++){
if(x[r][c] > max) {
max = x[r][c];
}
return max;
}}}
```

Or

```
public static int maxArray(int[][] x) {
int max = x[0][0];
for (int r = 0; r < x.length; r++) {
for (int c = 0; c < x[r].length; c++) {
if(x[r][c] > max)
```

```
{
max = x[r][c];
return max;
}}}
```

The minimum can be found in a similar manner replacing the max for min in the above code snippets. Please note that code snippets will need some more code in Java to fully work such as the definition of a class and a main class and remember to check on the brackets {. For every {a closing} is required.

Random numbers can be created in Java using the random feature which is Random x = new Random();
It is possible to fill a matrix with

random numbers in two ways similar to the ones shown before. The following two code snippets will show the logic:

```
public static void randArray(int[][] mat
Random rand = new Random();
for (int r = 0; r < matrix.length; r++)  {
for (int c = 0; c < matrix[r].length; c++)
matrix[r][c] = rand.nextInt(up + 1);
}}}
```

```
public static void randArray(int[][]
matrix, int row, int col, int up) {
Random rand = new Random();
for (int r = 0; r < row; r++)  {
for (int c = 0; c < col; c++) {
matrix[r][c] = rand.nextInt(up + 1);
```

```
}}}
```

Let us now try to run this code which depicts some of the snippets we have seen:

```
import java.util.Scanner;
public class a1 {
public static final int ROW = 2;
public static final int COL = 2;
public static void main(String[] args) {
int[][] a = new int[ROW][COL];
int[][] b = new int[ROW][COL];
System.out.println("Initial values/first ar
printA(a, ROW, COL);
System.out.println("Initial values/second
printA(b, ROW, COL);
System.out.print("Enter " + ROW * COL
```

```
getA(a, ROW, COL);
System.out.println("The first array is: ")
```

```
printA(a, ROW, COL);
System.out.println("The sum of the eler
" + sumA(a, ROW, COL));
System.out.println("Max value/first arra
ROW, COL));
System.out.println("Min value/first arra
ROW, COL));
copyA(a, b, ROW, COL);
System.out.println("After copy, the seco
printA(b, ROW, COL);
System.out.println();
}
public static void getA(int[][] x, int row
Scanner input = new Scanner(System.in)
```

```
for (int r = 0; r < row; r++)
for (int c = 0; c < col; c++)
x[r][c] = input.nextInt();
}
```

```
public static void printA(int[][] x, int ro
for (int r = 0; r < row; r++){
for (int c = 0; c < col; c++)
System.out.print( x[r][c]);
System.out.println();
}
}
public static int sumA(int[][] x, int row,
int sum = 0;
for (int r = 0; r < row; r++){
for (int c = 0; c < col; c++)
sum = sum + x[r][c];
```

```
}
return sum;
}
```

The next part of this code is as shown in the table below:

```
public static int maxA(int[][] x, int row,
int max = x[0][0];
for (int r = 0; r < row; r++){
for (int c = 0; c < col; c++)
if(x[r][c] > max)
max = x[r][c];
}
return max;
}
```

```
public static int minA(int[][] x, int row,
int min = x[0][0];
for (int r = 0; r < row; r++){
for (int c = 0; c < col; c++)
if(x[r][c] < min)
min = x[r][c];
}
return min;
}
public static void copyA(int[][] source
int col) {
for (int r = 0; r < row; r++)
for (int c = 0; c < col; c++)
d[r][c] = source[r][c];
} }
```

The code shows the result of a 2 by 2 array. The size of the array can be

changed and some of the concepts of arrays such as insertion, filling, printing, finding a maximum and minimum are shown in this code.

In this code we have used the method in which the length of the array is assessed through a loop.

There is another method as shown in the snippets where the size of the array is found through the .length method.

It may be worth taking the time to try this same code above with that method.

Appendix IV – References.

Anany Levitin,"Introduction to Design and Analysis of Algorithms" 3rd Edition ISBN: 978-0-13-231681-1.

Aho, A.V., Hopcroft, J.E., and Ullman, J.D. "The Design and Analysis of Computer Algorithms. Addison-Wesley", 1974.

Hamdy A Taha; “Optimization in Operations Research”, 9nd Edition, 2011, Prentice Hall.

Neapolitan, Jones & Bartlett Learning, 4th E., "Foundations of Algorithms" , 2009

Thomas H. Cormen,"Introduction to Algorithms" MIT Press, 2009.

Computer code was referred from the following sources and modified as needed for the chapters:

http://www.java67.com

http://www.java2novice.com

http://docs.oracle.com/javase/7/

http://www.java2blog.com

http://www.sanfoundary.com