Algorithm Analysis and Design

[Course Planner with Lab Exercises]

Academic Year 2011-12
Even Semester

Indira Gandhi Institute of Technology
Guru Gobind Singh Indraprastha University
Kashmere Gate, Delhi

Table of Contents

Version No.	Comments	Modification Date	Modification Done By
1.0	Created lab planner – added syllabus and lab exercises	2nd Jan 2012	Rishabh Kaushal Assistant Professor Department of Information Technology, IGIT

Table: Modification History

0. Guidelines

This document titled ‘Course Planner’ is written based on following guidelines.

1. Course work is divided into across weeks. Each week has certain objectives to be accomplished which shall be achieved by solving algorithmic problems of varying complexity.
2. Each week has 4 lectures hours supplemented with 2 hours scheduled Lab work.
3. We learn inner workings of an algorithm when it is at work. Therefore, it is highly recommended that each and every algorithm discussed in classroom is *implemented*. Being a course which forms bedrock for Computer Science, it is expected and strongly encouraged that **students not only utilize their Lab hours efficiently, but also implement algorithms in their personal computing environment**.
4. Students must ensure that each week, the stipulated lab problems as specified in this planner are to be completed irrespective of whether scheduled lab session is held or not.

1. Syllabus

As taken from www.ipu.ac.in website*.

**Academics → Academics → Scheme Syllabus (Affiliated Institutes, w.e.f. 2005-06)*

UNIT – I

Preliminaries: Review of growth of functions, Recurrences: The substitution method, The iteration method, The master method, Data Structures for Disjoint Sets.
Divide and Conquer Approach: Merge Sort, Quick sort, Medians and Order statistics, Strassen's algorithm for Matrix Multiplications. **[No. of Hrs.: 11]**

UNIT – II

Dynamic Programming: Elements of Dynamic Programming, Matrix Chain Multiplication, Longest common subsequence and optimal binary search trees problems.
Greedy Algorithms: Elements of Greedy strategy, An activity selection problem, Huffman Codes, A task scheduling problem. **[No. of Hrs.: 11]**

UNIT – III

Graph Algorithms: Representation of Graphs, Breadth First Search, Depth First Search, Topological Sort, Strongly Connected Components, Algorithm for Kruskal's and Prim's for finding Minimum cost Spanning Trees, Dijkstra's and Bellman Fort Algorithm for finding Single source shortest paths. All pair shortest paths and matrix multiplication, Floyd – Warshall algorithm for all pair shortest paths. **[No. of Hrs.: 11]**

UNIT – IV

String matching: The naïve String Matching algorithm, The Rabin-Karp Algorithm, String Matching with finite automata, The Knuth-Morris Pratt algorithm.
NP-Complete Problem: Polynomial-time verification, NP-Completeness and Reducibility, NP-Completeness Proof, NP-Complete problems. **[No. of Hrs.: 11]**

TEXT BOOKS:

1. T. H. Cormen, C. E. Leiserson, R. L. Rivest, Clifford Stein, "Introduction to Algorithms", 2nd Ed., PHI, 2004.

REFERENCES BOOKS:

1. A. V. Aho, J. E. Hopcroft, J. D. Ullman, "The Design and Analysis of Computer Algorithms", Addition Wesley, 1998.
2. Ellis Horowitz and Sartaz Sahani, "Computer Algorithms", Galgotia Publications, 1999.
3. D. E. Knuth, "The Art of Computer Programming", 2nd Ed., Addison Wesley, 1998

1.1 ***Course Coverage Plan***

The topics to be covered for various examinations (Minor-1, Minor-2 and End Semester Exam) are given in the table below.

Minor-1	Unit-I: Preliminaries & Divide and Conquer Approach
	Unit-II: Dynamic Programming
Minor-2	Unit-II: Greedy Algorithms
	Unit-III: Graph Algorithms
	Unit-IV: String Matching
End Term	Complete Syllabus

2. Lecture Plan

Given below is a tentative plan for the conduct of lectures in classroom for this course. Please note that this plan may change during the course progress.

Week 1: Introduction to algorithms & Mathematical preliminaries	
L1	Definitions and need for algorithms
L2	Representation of algorithms
L3	Analyzing and Designing algorithms
L4	Asymptotic notations (growth of functions)
Week 2: Review on data structures	
L5	Asymptotic notations (growth of functions) contd.
L6	Recurrences (substitution method)
L7	Recurrences (master method)
L8	Exercises on Aymptotic notations & Recurrences
Week 3: Divide and Conquer approach	
L9	Binary search and analysis
L10	Merge sort
L11	Analysis of Merge sort
L12	Quick sort
Week 4: Divide and Conquer approach continued	
L13	Analysis of Quick sort
L14	Medians and Order statistics
L15	Strassen’s algorithm for matrix multiplication
L16	Analysis of Strassen’s algorithm
Week 5: Data Structures + Algorithms	
L17	Role of Data Structures in Algorithms
L18	Data structures for disjoint sets
L19	Application of disjoint sets
L20	Dynamic Programming: Introduction
Week 6: Dynamic Programming	
L21	Matrix chain multiplications
L22	Longest common subsequence
L23	Optimal binary search trees
L24	Elements of dynamic programming
MINOR EXAM - 1	
Week 7: Greedy Algorithms	
L25	Activity selection problem
L26	Elements of greedy strategy
L27	Huffman codes
L28	Task scheduling problem
Week 8: Graph Algorithms: Introduction	
W29	Representation of graphs
W30	Breadth First Search

W31	Depth First Search
W32	Topological Sorting
Week 9: Connected Components & Spanning Tree	
W33	Connected components in graph
W34	Kruskal's Algorithm
W35	Prim's Algorithm
W36	Comparative analysis between Kruskal and Prim algorithm
Week 10: Single Source Shortest Path	
W37	Introductory problem description
W38	Dijkstra's algorithm
W39	Bellman Ford algorithm
W40	Comparative analysis between Dijkstra and Bellman Ford algorithm
Week 11: All pair Shortest Path	
W41	Introductory problem description
W42	Matrix multiplication
W43	Floyd Warshall algorithm
W44	Analysis of Floyd Warshall algorithm
Week 12: String Matching	
W45	Naïve string matching algorithm
W46	Rabin Karp algorithm
W47	String matching with finite automata
W48	Knuth-Morris Pratt algorithm
MINOR EXAM – 2	
Week 13: NP Complete Problem	
W49	Polynomial-time verification
W50	NP-Completeness and Reducibility
W51	NP-Completeness Proof
W52	NP-Complete problems
Week 14: Revision & Miscellaneous Problem solving week	
W53 – W56	

3. Lab Work Guidelines and Plan

Guiding Principle:

- The way to learn algorithms is to write programs implementing them. It is only when we implement algorithm, we understand and appreciate its intricate workings.
- Algorithms follow the following life cycle:
    - a. Design / Proposal of the algorithm
    - b. Implementation (C/C++) of the algorithm
    - c. Analysis – Finding the asymptotic complexity of the algorithm

3.1 *Submission Guidelines*

Following are the modalities for evaluation:-

1. All programs for implementing algorithm have to be submitted **online to a web server** hosted in the Lab.
2. Programs submitted will undergo *online* compilation and *offline* (optional) testing along with necessary *plagiarism checks* in order to evaluate the program from time to time.
3. Students would maintain a **handwritten *lab file*** to keep a record of their ***weekly progress***. For each problem, write the following in *lab file*:-
    - a. Problem statement
    - b. Algorithm design (with optional relevant code snippets)
    - c. Asymptotic complexity analysis
4. **Printouts of algorithm implementation will NOT be accepted.**
5. There would be **zero tolerance** for plagiarism of any kind. In any event of detection of plagiarism, all students involved would be penalized including the perceived "original" writer. It is responsibility of a student to secure their work.

3.2 *Evaluation Policy*

Following shall be the break-up of the internal assessment.

Lab File work = 25%

Viva* = 25%

Online submission = 25%

Implementation = 25%

* Viva (oral examination) shall be conducted ***randomly*** during the stipulated lab hours. Each student shall undergo two such evaluation, one before Minor-1 and another before Minor-2.

Following shall be the remarks corresponding to the range of marks (in percentage) awarded eventually towards the end of the course.

Grade	**Range of Marks**	**Remarks**
EX	95-100	Excellent
A	85-94	Very Good
B	75-84	Good
C	65-74	Average
D	55-64	Poor

4. Lab Exercises

This section includes a consolidated list of programming exercises based on topics covered every week.

Note: Please ensure that you have installed the programming environment as mentioned in Appendix A.

4.1 Week-1: Review of STL in C++

1. **Merge sort.** Write a program in C++ using *vector* class that inputs two sets of *random* integers (size of each input set to be provided by user), stores (inserts) them in two *vectors* ensuring a *sorted order* (use STL's *sort* algorithm to perform sorting) and finally merges the two sorted sets into a single set (in a third *vector* class object).

2. **Priority queue.** Design a C++ class that captures the concept of a process in an operating system. Each process may have attributes as unique process ID, process name and process priority number (lower value means higher priority). Using *dequeue* class, write C++ program to perform the operations of inserting a new process (with random attributes) and deleting a process with highest priority.

3. **Student list.** Design a C++ class that captures the concept of a student with attributes student ID, student name and student marks. Use *list* class to maintain a collection of students and provide operations for adding, deleting and updating attributes of a student. Use STL's pre-defined algorithm functions to search for a particular student, arrange student list in sorted order w.r.t name/ID and to find student scoring maximum/minimum marks.

4. **Set operations**. Using *set* class, illustrate all set operations namely union, intersection, subset, disjoint and membership. Use collection of *random* integers as input in the sets.

5. **Telephone directory.** A telephone directory consisting of unique telephone numbers along with the name of the telephone holder is to be maintained. People with same names with different telephone numbers are possible but not the other way around. Write a C++ program using *map* class to maintain such a telephone directory and provide operations to add, delete, update and query (given a telephone number, report the telephone holder's name and vice versa) in such a directory. Store telephone records such that it facilitates such query operations, use STL's pre-defined algorithms to perform any computation.

6. **Dictionary.** A dictionary consisting of words with their meanings is to be maintained. Same word can have more than one meaning. Write a C++ program using *mmap* class to maintain such a dictionary and provide operations to add, delete, update and query (given a word find its meaning) in such a dictionary. Store dictionary records such that it facilitates such query operations, use STL's pre-defined algorithms to perform any computation.

4.2 Week-2: Introduction to Algorithms

1. **Time measurement.** There are various methods to ascertain the time taken by computational instructions when an algorithm is implemented. Some of the methods have been suggested in the underlying link. Write program(s) to measure time using all the methods mentioned in the link.

   Link: *http://www10.informatik.uni-erlangen.de/Teaching/Courses/WS2008/SiWiR/Time.pdf*

2. **Running time comparison.** Write a C/C++ program that sorts a given input set of random numbers using both merge sort (recursive procedure) and insertion sort (iterative procedure). Your program should take only one input, say N, i.e. size of the set of random numbers. Thereafter, it should generate the random numbers and use the same set as input for both merge sort and insertion sort. Computation time taken for sorting (please note generation of random number is to be excluded) N random numbers in each algorithm is to be compared. Use any one method for measuring time. Finally, report your observations in form of the following table.

Input size (N)	Computation Time (units) Measured	
	Merge Sort	Insertion Sort
10		
100		
1000		
10000		
100000		

3. **Recursive insertion sort.** The problem of sorting n numbers i.e. A(1...n) can be expressed using insertion sort using divide and conquer approach. As per this approach, n-1 numbers i.e. A(1...n-1) are recursively sorted and A[n] is inserted into the sorted sequence A(1....n-1). Write a program to implement a recursive version of insertion sort as explained above and find its asymptotic complexity.

4. **Find Sum.** Given a set of n integers (in sorted order) and another integer x. Design an algorithm that verifies whether there exist pairs of integers a and b such that $a + b = x$, your algorithm should have $\Theta(n \lg n)$ time bound.

5. **Inversion Count.** In an array $A[1 \ldots n]$, an inversion pair (i, j) is said to occur if $i < j$ and $A[i] > A[j]$. Design a $\Theta(n \lg n)$ time algorithm that finds all such inversion pairs in any given array.

4.3 Week-3: Divide and Conquer Algorithm

1. **Binary Search.** Implement a user defined recursive procedure to perform binary search. Use *pre-defined STL sort* function to ensure a sorted sequence is passed as input to the recursive binary search procedure. Comment upon asymptotic analysis of binary search implemented as above.

2. **Improved Insertion Sort.** Design an algorithm for an improved insertion sort which leverages the property that one half of input array remains sorted as the sorting proceeds from start towards completion. What is the asymptotic complexity of your improved algorithm.
   Hint: Use *binary search* to locate correct position for the number to be inserted in the sorted sub-array.

3. **Quick Sort with pivot selection.** Implement quick sort that sorts N random numbers using different pivot elements based on CTRL variable. Your program takes two inputs N and CTRL variables with possible values as $1 < N < 10000$ and CTRL = {1: pivot as last element, 2: pivot as first element, 3: pivot as middle element and 4: random pivot}. Comment on the asymptotic analysis of quick sort for the case when last element is taken as pivot. What is the running time of quick sort when elements are already sorted and when all elements are same?

4. **Insertion based Quick sort.** Perform quick sort only until the given sequence of N random numbers have been divided into partitions, each of size not more than a input value K, where $K < N$. Finally perform improved insertion sort over the entire sequence of N random numbers. Perform asymptotic analysis of the procedure described.

5. **Quick Sort with Hoare Partition.** Perform quick sort using *Hoare Partition* as described in one of the exercises in the Chapter titled 'Quick Sort' in textbook 'Introduction to Algorithms' by Cormen.

4.4 Week-4: Divide and Conquer continued

1. **Finding Maximum-Minimum.** Write a program that takes as input N and CTRL, where N is number of *distinct* random numbers and CTRL takes values 1 and 2, corresponding for finding maximum and minimum, respectively among the input random numbers.

2. **Simultaneous Maximum-Minimum.** Write a program that calculates *both* minimum and maximum out of given N *distinct* random numbers. The calculation is to be performed using two methods (described below) and performance of each method is to calculated based on the number of comparisons involved.
   * **Method 1:** Take one number from the input at a time and compare it with running minimum and running maximum.
   *** Method 2:** Take two numbers at a time, compare them with each other. Then, compare larger number with running maximum and smaller number with running minimum.

Input size (N)	Number of Comparisons performed	
	Method 1	Method 2
10		
100		
1000		
10000		
100000		

3. **Matrix Multiplication.** Write a program that performs matrix multiplication using conventional algorithm and using Strassen's algorithm. Calculate the performance of each method based upon number of scaler matrix multiplications involved. Your program should take input as N i.e. order N X N of the sqaure matrix. Values of N are to be taken in powers of 2. Matrices are to be filled with random numbers in the range {0 … 9}. **Note:** Output should be properly formatted depicting input and output matrices.

Input size (N)	Number of scalar matrix multiplications involved	
	Conventional Algorithm	Strassen's Algorithm
2		
4		
8		
16		
32		

4.5 Week-5: Role of Data Structures in Algorithms

1. **Well formed expression parser.** Design and Implement an algorithm for parsing a given expression containing parenthesis which verifies whether a given expression is well formed or not. What data structure is most appropriate in such an algorithm?

   Note: *A well formed expression always has a pair of parenthesis matching each other. eg. (x+5)*((4+y)/5) is well formed but ((x+5)*(4+y)/5 is not well formed.*

2. **Function call tracker.** Suppose that you are compiler writer and you need to track the function call mechanism. What data structure shall you use to perform the tracking of recursive function? What shall be the information stored in each node of your data structure? Write program for insertion, deletion and traversal of such a data structure, perform time and space complexity analysis for each of these operations.

   Note: *Assume that a pre-written parser code having received as input as a C program with nested function calls and has extracted all relevant information required for your* *data structure and placed it into a file (which your program reads as input). You may have your own format for this file.*

3. **Undo-Redo feature support.** Suppose that you are developer of text editor software and you are given a task to provide support of *undo-redo operation* into the software. Design and implement an algorithm to provide such a support, what data structure should you use to assist in your operation.

4. **Efficient Searching.** Telephone directory is to be maintained in which each record has two attributes namely telephone number (uniquely identifies a record) and name of the telephone owner. What data structure (and provide corresponding implementation) to be used to *optimize* search in each of the scenario given below:-
   i. Searching based on telephone number, returns the name of owner if search succeeds.
   ii. Searching based on owner name, returns the telephone number(s) if search succeeds.

5. **Collection of disjoint sets.** A dynamic collection of disjoint sets is to be maintained providing following operations:-
   a. Make Set, creates a set with an input element.
   b. Find Set, returns a representative element of the set containing the input element.
   c. Union, unites two sets represented by given input elements into a new set and destroys the previous two sets.

   What data structure shall be used to optimize time? Implement each of the above operation as well. Will our data structure change if we wish to optimize on space rather than time? If yes, mention the data structure and its implementation.

4.6 Week-6: Dynamic Programming

1. **Matrix Chain Multiplication.** Print all possible *valid* parenthesizations of a given input chain of matrices (their orders are given) along with their cost. Among all these solution, highlight the one with optimal solution and optimal value (cost).

   Note: *Input is to be read from a file mentioning the order of matrices, each separated with a comma on a single line. eg. 5, 10, 4, 15 means that there are three matrices labeled A_1, A_2, A_3 of order 5 X 10, 10 X 4 and 4 X 15, respectively.*

2. **Longest Common Subsequence.** DNA strands consist of string of molecules called *bases.* Possible values of bases are adenine, guanine, cytosine and thymine, each represented by letters A, G, C and T, respectively. Given two input DNA strands, print the longest common subsequence and its length value.

***Note:** Input is to be read from a file mentioning the two DNA strands, each consisting of sequences of bases (each base value separated by a space from other base value and each DNA strand on separate lines.*
eg.
A, A, C, G, T, T, A, C
G, A, C, T, T, A, A, C

3. **Similarity Index*.** Extending the concept of longest common subsequence from characters to words, find the similarity index between two given input files.

   ***Note:** Similarity index between two files is calculated as the number of words which are common across two files and occurring in relatively same order.*

4. **Optimal Binary Search Tree.** A paragraph consist of words from English alphabet is given as input. Some of the words in it are repeated in varying degrees. Assuming that each *unique* word of the paragraph is a ***key*** which is to be searched and each *unique* word's selection as a key is equally likelihood, organize the words of the paragraph such that searching is optimized. Print the in-order traversal of optimized binary search tree and the optimized cost.

   ***Note:** Input is to be read from a file containing the paragraph. A paragraph consist of unique words, say, W_1, W_2, … W_n.*

   Frequency based approach is used to calculate probabilities of words appearing as keys as below:-

   Probability(W_i) = Frequency(W_i) / 2
   where,
   Frequency(W_i) = Number of occurrence(W_i) / Total Words

   Probability of "dummy nodes" is calculated as below:-
   Probability (D_i) = Probability(W_{n-i}), such that Probability(W_0) = 0

4.7 Week-7: Greedy Algorithms

1. **Activity Selection Problem.** During a college festival, various events are proposed in form of guest talks, technical presentations, etc are being planned. Each event is of varying duration with a start and end time. However the problem is that there is only a single Seminar Room with adequate infrastructure where these events can take place. Prepare a schedule of such events to be held in Seminar Room such that maximum number of events could be held.

***Note:** Input is to be read from a file mentioning the proposed events (with their start-end time pair). eg. (0,5), (2,3), (6,9) means that three events are proposed with 0, 2 & 6 are their respective start times and 5, 3 & 9 as their end times.*

2. **File compression using Huffman Code.** An input file consisting of only alphabets (small letters) of English language is to be sent across the Internet. The lesser is its encoded size, the lesser bandwidth (network resource), will be required to send it across. Compare the two encoding schemes, namely, fixed length code and variable length code (using Huffman coding) and find the percentage compression achieved.
3. **Task Scheduling Problem.** A program in execution is referred as a process and modern day operating systems can capable of managing multiple processes at same time. Assume that each process takes unit time to execute, generate a process schedule such that minimum penalty is incurred.

   ***Note:** Input is to be read from a file mentioning the list of processes (with their deadline-penalty pair). eg. (1,5), (2,3), (6,9) means that three processes are to finish by times 1, 2 & 6, respectively otherwise the penalty incurred shall be 5, 3 & 9, respectively.*

4.8 Week-8: Graph Algorithms

1. **Adjacency Matrix representation.** Represent a given input graph in its adjacency matrix form and print the contents of the matrix. Your program should provide support for both directed as well as undirected graphs.

   ***Note:** Input about the graph is to be read from a file mentioning the number of vertices (each separated by a comma) on first line, after which each edge is represented in a vertex-pair in a separate line that follows.*
   eg. Input files for graph with five vertices and four edges.
   1,2,3,4,5
   1,2
   2,3
   4,2
   5,1

2. **Adjacency List representation.** Represent a given input graph in its adjacency list form and print the contents of the matrix. Your program should provide support for both directed as well as undirected graphs.

   ***Note:** Input about the graph is to be read from a file as per the format suggested in Question No. 1 above.*

3. **Graphical representation*.** Given input graph information in form of an input file as explained in Note above, can you suggest and implement an algorithm that actually draws the input graph in its diagrammatic form, the *graphics.h* library learnt as part of Computer Graphics course may be used to draw the vertices, edges and labels.

4. **Breadth First Search.** Given an input graph (in form of an input file as explained in Note above), perform breadth first search and print its result.

5. **Depth First Search.** Given an input graph (in form of an input file as explained in Note above), perform breadth first search and print its result.
6. **Topological Sorting.** In a university curriculum, often each course has a set of pre-requisites. Given a set of courses along with their respective set of pre-requisites, prepare a curriculum such that no course appears before its pre-requisite.

4.9 Week-9: Spanning Tree Algorithms

1. **Strongly connected components.** Given an input graph, write a program that prints its strongly connected components.

   Note: *Input about the graph is to be read from a file mentioning the number of vertices (each separated by a comma) on first line, after which each edge is represented in a vertex-pair in a separate line that follows.*
   eg. Input files for graph with five vertices and four edges.
   1,2,3,4,5
   1,2
   2,3
   4,2
   5,1

2. **Kruskal algorithm.** Generate minimum spanning tree (print it in an in-order traversal order) from a given input graph as per Kruskal algorithm.

   Note: *Input about the graph is to be read from a file mentioning the number of vertices (each separated by a comma) on first line, after which each edge is represented in a vertex-pair in a separate line that follows.*
   eg. Input files for graph with five vertices and four edges.
   1,2,3,4,5
   1,2
   2,3
   4,2
   5,1

3. **Prim algorithm.** Generate minimum spanning tree (print it in an in-order traversal order) from a given input graph as per Prim algorithm.

   Note: *Input about the graph is to be read from a file mentioning the number of vertices (each separated by a comma) on first line, after which each edge is represented in a vertex-pair in a separate line that follows.*
   eg. Input files for graph with five vertices and four edges.
   1,2,3,4,5
   1,2
   2,3
   4,2
   5,1

4.10 Week-10: Single source Shortest Path

1. **Shortest Route Map.** A city's map with all the roads along with distances (in kilometers) between various places in the city is given. Using *Bellman Ford algorithm* and *Dijikstra's algorithm*, find the shortest path from a given input place fed as a source in the algorithm.

2. **Dynamic Shortest Route Map.** Suppose that in addition to the distance in the routes, real time status of the traffic conditions are also being provided at regular time intervals. Find the shortest route from a given input place fed as source. You can use any of the algorithm either *Bellman Ford algorithm* and *Dijikstra's algorithm.*

4.11 Week-11: All pair Shortest Path

1. **Floyd Warshall Algorithm.** Given an input graph, using *Floyd Warshall algorithm*, find all pair shortest path.

4.12 Week-12: String Matching

1. **Naïve String Matching Algorithm.** Find the occurrence of a given string in an input file using the naïve string matching algorithm.

2. **Rabin Karp Algorithm.** Using *rabin-karp algorithm*, compare the two input files. Output whether the two input files are same or not.

3. **String matching using Automata.** Implement string matching algorithm using the finite automata approach.

4. **Knuth-Morris Pratt Algorithm.** Perform pattern matching given input file and input pattern using the *Knuth-Morris Pratt algorithm.*

Appendix A: Setting up programming environment

Aim:

Setting up Linux environment in Windows Operating System using Virtual Box Software

Step 1: Installation of Virtual Box

1. Download Virtual Box Software
http://download.virtualbox.org/virtualbox/4.1.2/VirtualBox-4.1.2-73507-Win.exe

2. Run the .exe file downloaded in Step 1 and install Virtual Box Software using the default options appearing during the installation procedure.

Note: By default, Virtual Box will install in C: drive, recommended free space that should be available in C: drive is around 15GB, if you don't have this much free space, select a different drive during the Virtual Box installation.

Step 2: Creating a Virtual Machine and installing Guest Operating System (ubuntu-10.04)

1. Download guest operating system (ubuntu-10.04 desktop i386)
http://www.psychocats.net/ubuntu/getting

2. Follow the steps in link below for creating virtual machine and installing guest operating system
http://www.psychocats.net/ubuntu/virtualbox

Step 3: Installing Virtual Box Guest Additions

1. Download virtual box guest additions
http://download.virtualbox.org/virtualbox/4.1.2/VBoxGuestAdditions_4.1.2.iso

2. Attach the guest addition iso file to CD/DVD device as explained in Step 2's 2. Next follow steps are given below
http://helpdeskgeek.com/linux-tips/install-virtualbox-guest-additions-in-ubuntu/

Step 4: Installing Software Packages (compiler, editor, etc.)

1. Go to Applications → Accessories → Terminal and type following at command prompt to install packages build-essential and vim editor.

```
sudo apt-get install build-essential
sudo apt-get install vim
```

Reference Online and Books

- **Textbook:** Introduction to algorithms by Thomas H. Cormen
- **Reference:** Algorithm design by Jon Kleinberg
- Online: ***http://algorithmguru.com/***
- Online: ***http://www.thealgorithmist.com/***