CSN-212: Design and Analysis of Algorithms
(Spring 2015-2016)

Coding Project -2 (CP-2) Duration: Apr 11 – Apr 17, 2016

➢ Instructions:

- Objective of this Project:

  To know implementation of algorithms of different paradigms like divide-and-conquer, dynamic programming, greedy, graph.

- Tasks to do in CP-2:

1. Write the code in any programming language (C, C++, Java, Python or Perl) for the problem statements assigned to your group in separate files (name the files according to the algorithm name):

2. Write the main() function in a top-level file <filename.c> or <filename.cpp> (a wrapper code) to take the input from the user, to take the input from a file (file format can be decided yourself) and to generate the input randomly in runtime (different for different runs). Your code (main function) should have all of three ways of taking input.

3. Write separate function for solving your problem and call that function from the main() function to demonstrate the solution to your problem.

4. While writing a file for your code, you MUST INCLUDE the following information (for example) in the file before coding the actual program:

   ```
   ## GroupID-1 (14114XXX_14114YYY) - Name1 & Name2
   ## Date: March 26, 2016
   ## <filename>.c - State which algorithm is coded in this file.
   …
   Here you start writing your code.
   …
   ```

5. Run your code for two extreme cases: (a) small size input data (e.g., 10 numbers) and (b) larger input data (e.g., 5000 numbers). Display the input and output on the screen for the first case (a). Calculate the CPU time within your code (excluding the random number generation part). Then report the CPU times for both the cases (a) and (b) of your solutions to the problem.

6. Take the screenshot (by PrintScreen key) of one such run for case (a) of each of the algorithms you implemented. Include those screenshot images in your report.

- Submission Method (Strictly Follow These):

1. You must submit a zipped folder (<filename>.zip or <filename>.tar.gz) containing the following items:
   (a) All the code files named properly as mentioned above.
   (b) A report file (<filename>.DOC or <filename>.PDF) should contain your details like Group-ID, Name(s) and Enrollment Number(s) of the group member(s). The report should contain the following:
      i. The very brief descriptions of the algorithms or codes.
      ii. The code you have written for the top-level file <filename.c> or <filename.cpp> (the wrapper code).
      iii. The screenshots as described above.

2. Very Important: If the enrollment numbers of two members of your group are 14114XXX and 14114YYY, then replace <filename> with "14114XXX_14114YYY". Strictly follow this convention of you filename while submitting. If you do not follow this conventions, you (the group) will be given ZERO for CP-2.

3. Submit your zipped folder (<filename>.zip or <filename>.tar.gz) through your account in Moodle. We have created a submission link in Moodle course site.

4. Hard (Strict) deadline for submission: Apr 17, 2016 (10:00 am Indian Time). The submission site in Moodle will be LOCKED after this deadline.

5. For any submission after Final Deadline, 5 marks will be deducted for every 24 hours of extra time.

6. The key to success is starting early. You can always take a break, if you finish early.

- Evaluation Process:

1. Group information is same as you followed for CP-1. If any discrepancy is found, the responsible students will get ZERO for CP-2 without any discussion with the course instructor.

2. We will simulate your codes contained in your zipped folder and will regenerate the results you had included in the report file.

3. Your submission will be checked with others' submission to identify any copy case. If we detect that your code is a copy (partially or fully) of other's code, then the total marks obtained by one group will be divided by the number of groups sharing the same code and will be given to all those groups who have duplicities.

4. You may be asked to demonstrate and explain your submission after the submission deadline.

- List of Problems: [Check the Problem Number for your Group from the Random Assignment File]

1. Take a pattern and a text as input from the user. Implement the brute-force algorithm and KMP algorithm for pattern matching. Show the steps of pattern matching for a small-size problem instance. Report the CPU times for finding a pattern using both the algorithms with a different size inputs.

2. Implement the recursive algorithm to solve Tower-of-Hanoi problem with N disks and three pegs. Show the steps of solving the problem for a small problem instance with four disks and three pegs. Report the CPU times for solving this problem with three pegs and varying number of disks.

3. Take two input matrices of any dimensions. Implement the brute-force algorithm for matrix multiplication and Strassen's matrix multiplication. Compare CPU times of both the algorithms with varying dimensions of the two matrices.

4. Take two large integers (decimal) as input in an array. Implement the brute-force algorithm for large integer multiplication using divide-and-conquer paradigm. Implement Karatsuba's algorithm for large integer multiplication. Compare CPU times of both the algorithms with varying length of integers.

5. Take two long (> 128 bits) binary bit strings as input in an array. Implement the brute-force algorithm for large integer multiplication using divide-and-conquer paradigm. Implement Karatsuba's algorithm for large integer multiplication. Use Booth's algorithm for binary multiplication. Compare CPU times of both the algorithms with varying length of integers.

6. Take an integer to generate the Fibonacci sequence. Implement recursive algorithms (both divide-and-conquer approach and dynamic programming approach). Compare CPU times with varying input integers (smaller to larger).

7. Take an input integer for its factorial computation. Implement recursive algorithms (both divide-and-conquer approach and dynamic programming approach). Compare CPU times with varying input integers (smaller to larger).

8. Take two integers n and k as input. Compute the binomial coefficient (nCk) with that input. Implement recursive algorithms (both divide-and-conquer approach and dynamic programming approach). Compare CPU times with varying inputs.

9. Construct an instance of 0-1 Knapsack problem with n items and K=100 as the size of the bag. Implement the brute-force algorithm for the 0-1 Knapsack problem. Implement the dynamic programming approach to solve the same 0-1 Knapsack problem. Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.

10. Create a text file S containing about 5000 random characters from the program. Take any string (pattern) from user as input P. Implement the brute-force algorithm for solving the longest common subsequence problem. Implement the dynamic programming approach for solving the longest common subsequence problem with S and P.

11. Randomly generate n (> 50) points in 2D plane. Implement the brute-force algorithm for solving the problem of finding closest-pair of points. Implement the divide-and-conquer algorithm for solving the problem of finding closest-pair of points. Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.

12. Take n matrices from the user with their dimensions as input. Implement the dynamic programming based algorithm for minimizing the number of multiplications in the chained matrix multiplication. Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.

13. Take weights (W) and prices (P) of n items and the capacity of a knapsack (C) from the user as input. Implement the dynamic programming based algorithm for maximizing the price of selecting items from those n items (0-1 Knapsack Problem). Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.

14. Take the number of cities (n) from the user as input. Randomly generate a graph with n vertices and some undirected edges with randomly assigned edge-weights. Implement the dynamic programming based algorithm for solving the Travelling Salesman Problem for the weighted graph. Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.

15. Take a number N from the user as input. Implement the dynamic programming based algorithm for solving the Number Partitioning Problem with N to find out all possible integer partitions of N. Show the outputs for smaller size problems and compare the CPU times for varying size (N) problems.

16. Take the number of integers in a set S as n and a number T from the user as input. Randomly generate the set S of n numbers such that each of the elements is less than T. Implement the dynamic programming based algorithm for solving the Subset-Sum Problem with S and T to find out all possible subsets of S, whose sum total is equal to T. Show the outputs for smaller size problems and compare the CPU times for varying size (N) problems.

17. Take two random strings (of different lengths m and n) from the user as input. Implement the dynamic programming based algorithm for solving the Edit-distance Problem with those two strings. Show the outputs for smaller size problems and compare the CPU times for varying size (m and n) problems.

18. Take the number of vertices (n) from the user as input. Randomly generate a graph with n vertices and some undirected edges with randomly assigned edge-weights. Implement the greedy technique based on Prim's algorithm for finding the minimum spanning tree in the weighted graph. Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.

19. Take the number of vertices (n) from the user as input. Randomly generate a graph with n vertices and some undirected edges with randomly assigned edge-weights. Implement the greedy technique based on Kruskal's algorithm for finding the minimum spanning tree in the weighted graph. Use Union-Find data structure in your implementation. Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.

20. Take the number of vertices (n) from the user as input. Randomly generate a graph with n vertices and some undirected edges with randomly assigned edge-weights. Randomly assign any two vertices as the source and end vertices. Implement the greedy technique based on Dijkstra's algorithm for finding the single source shortest path between those two vertices in the weighted graph. Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.

21. Take the number of vertices (n) from the user as input. Randomly generate a graph with n vertices and some directed edges with randomly assigned edge-weights. Randomly assign any vertex as the source vertex. Implement Bellman-Ford algorithm for computing the shortest paths from the single source vertex to all of the other vertices in that weighted digraph. Also, consider negative edge weights. Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.

22. Take the number of vertices (n) from the user as input. Randomly generate a graph with n vertices and some directed edges with randomly assigned edge-weights. Implement Floyd-Warshall algorithm for computing all-pair shortest paths in that weighted digraph. Also, consider negative edge weights. Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.

23. Take the number of vertices (n) from the user as input. Randomly generate a graph with n vertices and some directed edges with randomly assigned edge-weights as the edge-capacities. Randomly assign a source vertex and a sink vertex. Implement Ford-Fulkerson algorithm for computing maximum-flow in that weighted digraph. Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.

24. Randomly generate n (> 50) points in 2D plane. Implement an algorithm for constructing a simple polygon with those n points. Implement an algorithm for computing the convex hull with those n points. For each of above cases, output the sequence of points in counter-clock wise order. Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.