Department of Computer Science and Engineering, SVNIT, Surat

MTech I (1^{st} semester)

Design and Analysis of Algorithms

Lab Assignment No 2 - Divide and Conquer Design Technique

AUTUMN Semester 2022-23

Dated Uploaded: 13th Sep 2022

Instructions:

1. *The date of submission **will be specified only two days before**. Therefore, right from the time the assignment is uploaded the students must start implementing the assignments.*
2. *For every delayed submission beyond the deadline, 10 marks per day will be deducted from the maximum marks of the assignment, without any exception, whatsoever may be the scapegoat.*
3. *You can use any programming language, except Python.*
4. *For every program whether specified or not, it is necessary to time your program on input dataset of a large size and give a critique of the theoretical asymptotic analysis of the algorithm that your program is based on and the empirical timing analysis of the program that you have written. The a-priori estimates and the a-posteriori analysis must be in sync with each other.*
5. *All the assignments must be submitted in the form of a zip file containing the Program Source, the screenshot of the output that you obtained, the DataSet/Input Test data used and a ReadMe .txt file explaining what platform to use, what are the input parameters required for execution and how to execute it. Also write your conclusion in ReadMe file. For the theoretical questions in Part B, write the answers using Latex in a .Tex file and submit the .tex as well as .pdf files - to be included in the zip file as above.*
6. *Perform usual error checking. Don't go overboard on this, but don't let your program die because of divide by zero.*
7. *Remember, your programs could be checked by a code-cheating program, so please follow the code of academic integrity.*
8. *There will be a viva for each assignment. This viva would be conducted for this assignment on a future date as specified.*
9. ***Maximum Points: 100***

---

1. Design a divide-and-conquer algorithm for computing the number of levels in a binary tree. (In particular, the algorithm must return 0 and 1 for the empty and single-node trees, respectively.) Analyze the time efficiency class of your algorithm. Now, implement the algorithm you designed and find out the time taken an for empty, one-level, two-level and three-level binary tree. Compare it with the theoretical analysis that you would have done before.

2. Implement the Strassen's matrix multiplication algorithm (may or may not be discussed in the class). Test your program on matrices of different sizes and time your program. As the matrix size becomes smaller, does the algorithm yield efficiency better than the brute force ? If yes, argue how is it so and show it with consistent results. If NO, there must be a crossover point beyond which the Strassen's algorithm yields better efficiecny as compared to the bruteforce method. Run an experiment to determine such a crossover point on your computer system.

3. Write programs to count the number of inversions in a given array using the brute force approach and using the algorithm applying the merge-sort discussed in the class (with time complexities $O(n^2)$ and $O(n\ lg\ n)$ respectively). Compare both the programs with timing analysis with input size starting with $n = 10$ upto $n = 10^5$. Comment on the results obtained.

4. Observe that the algorithm PARTITION() for Quicksort given in the text is different from the one discussed in the class. Implement two different Quicksort routines that use the two different PARTITION routines. Test your implementation on an input vector of size at least a 2048 integers and time the programs. Analyze the result and document it.

5. Implement MEDIAN-OF-THREE-PARTITION routine for Quicksort as discussed in the class. Time the program on ten different randomly generated integer vectors of size 1024 each and time the programs. Analyze the result and document it.

6. Implement Probabilistic-Quicksort routine discussed in the class. Time the program on ten different randomly generated integer vectors of size 1024 each and time the programs. Analyze the result and document it.

7. Compare the times in executing the programs in problems at Sr nos 4,5,6 and analyze your results. Give appropriate inferences.

8. Topological ordering of a graph is defined as follows: *Let G = (V,E) be a directed graph. A topological ordering of G is an assignment f(v) of every vertex vϵV to a different number such that: for every (v, w) ϵ E, f(v) < f(w).* Topological ordering is typically useful when one is given, say, a bunch of tasks to complete, and there are precedence constraints, meaning that one cannot start some of the tasks until he'she has completed others. As for example, the courses in a university degree program, some of which are prerequisites for others. One application of topological orderings is to sequencing tasks so that all precedence constraints are respected. Depth-first search is perfectly suited for computing a topological ordering of a directed acyclic graph.

   Does every graph have a topological ordering ? Reason your answer with appropriate justifications. Now, implement a program that uses DFS of a given Directed Acyclic Graph and outputs topological ordering of the graph. Time your program and analyze the same.

9. A light bulb is connected to n switches in such a way that it lights up only when all the switches are closed. Each switch is controlled by a push button; pressing the button toggles the switch, but there is no way to know the state of the switch. Design an algorithm to turn on the light bulb with the minimum number of button pushes needed in the worst case. Use the concept of the Binary Gray codes and imagine the switches as a collection of bits, each bit representing a switch. Implement the program and analyse the time taken by your program to execute.

10. Implement the Counting sort covered in the class and analyze the running time of your code.