SIX WEEKS SUMMER TRAINING REPORT

On

DSA Self-paced

Swaraj Bandhu Prasad Kalwar

12111470

B. Tech CSE

Under the Guidance of

Mr. Sandeep Jain

School of Computer Science & Engineering
Lovely professional University, Phagwara

(May-July, 2023)

DECLARATION

I hereby declare that I have completed my six weeks summer training at DSA Self-paced from 25th May 2023 to 10th July 2023 under the guidance of Mr. Sandeep Jain. I declare that I have worked with full dedication during these six weeks of training and my learning outcomes fulfil the requirements of training for the award of degree of Bachelor of Technology, Lovely Professional university, Phagwara.

Swaraj Bandhu Prasad Kalwar

12111470

Acknowledgement

It is with sense of gratitude; I acknowledge the efforts of entire hosts of well-wishers who have in some way or other contributed in their own special ways to the success and completion of the Summer Training.

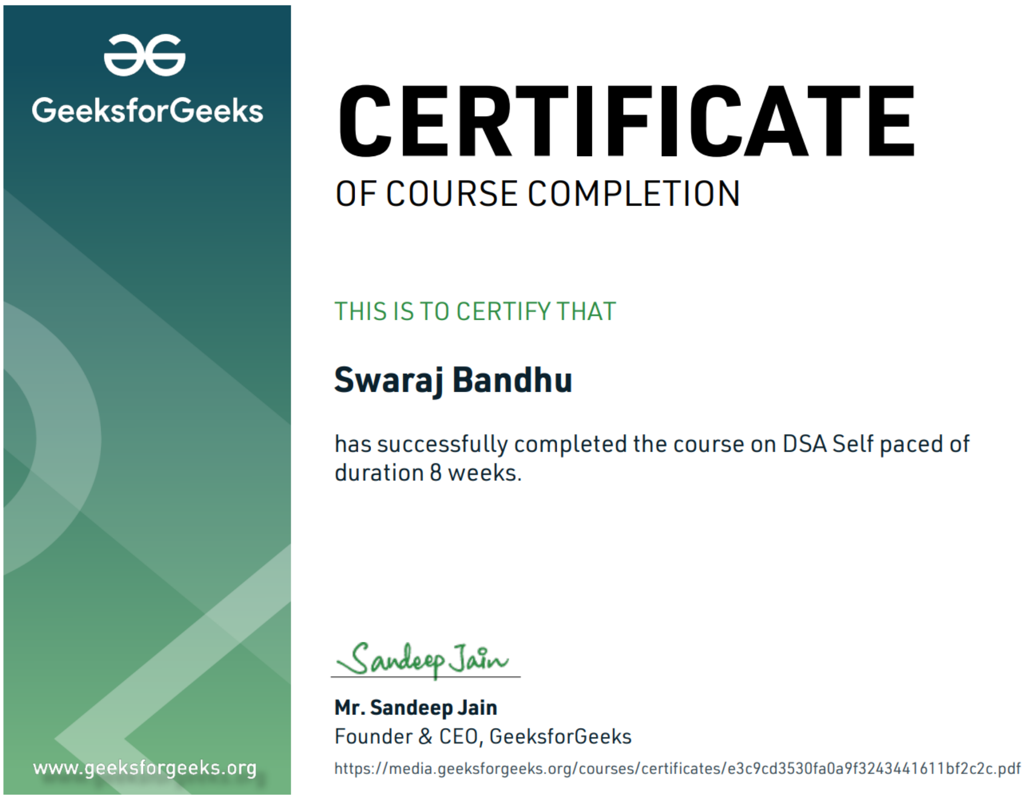
Successfully completion of any type of technology requires helps from several people. I have also taken help from different people for the preparation of the report. Now, there is little efforts to show my deep gratitude to those helpful people.

I would like to also thank my own college Lovely Professional University for offering such a course which not only improve my programming skill but also taught me other new technology.

Then I would like to thank my parents and friends who have helped me with their valuable suggestions and guidance for choosing this course.

Finally, I would like to thank my all classmates who have helped me a lot.

Training certificate from organization



GeeksforGeeks

CERTIFICATE

OF COURSE COMPLETION

THIS IS TO CERTIFY THAT

Swaraj Bandhu

has successfully completed the course on DSA Self paced of duration 8 weeks.

Sandeep Jain

Mr. Sandeep Jain
Founder & CEO, GeeksforGeeks

https://media.geeksforgeeks.org/courses/certificates/e3c9cd3530fa0a9f3243441611bf2c2c.pdf

www.geeksforgeeks.org

Table Of Contents

Introduction

The course name DSA stands for “Data Structures and Algorithms” and Self-paced means, one can join the course anytime. All the content will be available once one gets enrolled. One can finish it at his own decided speed.

1. **What is Data Structure?**
   Data Structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage. For example, we have some data which has, player's name "Virat" and age 26. Here "Virat" is of String data type and 26 is of integer data type.
2. **What is Algorithm?**
   An algorithm is a finite set of instructions or logic, written in order, to accomplish a certain predefined task. Algorithm is not the complete code or program, it is just the core logic(solution) of a problem, which can be expressed either as an informal high-level description as pseudocode or using a flowchart.

This course is a complete package that helped me learn Data Structures and Algorithms from basic to an advanced level. The course curriculum has been divided into 8 weeks where one can practice questions & attempt the assessment tests according to his own pace. The course offers me a wealth of programming challenges that will help me to prepare for interviews with top-notch companies like Microsoft, Amazon, Adobe etc.

Technology Learnt

- Learn Data Structures and Algorithms from basic to an advanced level like:
- Learn Topic-wise implementation of different Data Structures & Algorithms as follows.

1. **Introduction**
    - **Asymptotic Notations**

      Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.

    - **Worst, Average and Best-Case Time Complexities**
      It is important to analyse an algorithm after writing it to find it's efficiency in terms of time and space in order to improve it if possible.

      When it comes to analysing algorithms, the asymptotic analysis seems to be the best way possible to do so. This is because asymptotic analysis analyses algorithms in terms of the input size. It checks how are the time and space growing in terms of the input size.

    - **Analysis of Loops**
2. **Mathematics**
    - **Finding number of Digits in a Number**
    - **Arithmetic and Geometric Progressions**
    - **Quadratic Equations**
    - **Mean and Median**
    - **Prime Numbers**
    - **LCM and HCF**
    - **Factorials**
    - **Permutation and Combinations Basics**
    - **Modular Arithmetic**
3. **Bit Magic**

- **Binary Representation**
- **Set and Unset**
- **Toggling**
- **Bitwise Operators**
- **Algorithms**

Objective: The objective of this track is to familiarize the learners with *Bitwise Algorithms* which can be used to solve problems efficiently and some interesting tips and tricks using Bit Algorithms.

4. **Recursion**

- **Recursion Basics**

  The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), In order/Preorder/Post order Tree Traversals, DFS of Graph, etc.
- **Basic Problems on Recursion**
- **Tail Recursion**

  A recursive function is said to be following Tail Recursion if it invokes itself at the end of the function. That is, if all the statements are executed before the function invokes itself then it is said to be following Tail Recursion.

  ```
  void print(int N)
  {     if(N==0)
  return;     else
  cout<<N<<" ";


     printN(N-1);
  }
  ```

  The above function call for **N = 5** will print:

  5 4 3 2 1
- **Explanation of Subset Generation Problem**
- **Explanation of Joesphus Problem**
- **Explanation of permutations of a string**

We iterate from first to last index. For every index I, we swap the I - th character with the first index. This is how we fix characters at the current first index, then we recursively generate all permutations beginning with fixed characters.
(By parent recursive calls). After we have recursively generated all permutations with the first character fixed, then we move the first character back to its original position so that we can get the original string back and fix the next character at first position.

- **Check if the element is present in a row and column-wise sorted matrix.**
- **Boundary Traversal**
- **Spiral Traversal**
- **Matrix Multiplication**
- **Search in row-wise and column-wise Sorted Matrix**

8. **Hashing**
    - **Introduction and Time complexity analysis**
    - **Application of Hashing**
    - **Discussion on Direct Address Table**
    - **Working and examples on various Hash Functions**
    - **Introduction and Various techniques on Collision Handling**
    - **Chaining and its implementation**
    - **Open Addressing and its Implementation.**
    - **Chaining V/S Open Addressing**
    - **Double Hashing**
    - **C++**
    - Unordered Set
    - Unordered Map
    - **Java**
    - HashSet
    - HashMap
9. **Strings**
    - **Discussion of String DS**
    - **Strings in CPP**
    - **Strings in Java**
    - **Rabin Karp Algorithm**
    - **KMP Algorithm**
10. **Linked List**
    - **Introduction Implementation in CPP**
    - Implementation in Java
    - Comparison with Array DS
    - **Doubly Linked List**
    - **Circular Linked List**

- **Loop Problems**
- Detecting Loops
- Detecting loops using Floyd cycle detection
- Detecting and Removing Loops in Linked List

11. Stack

- **Understanding the Stack data structure**
- **Applications of Stack**
- **Implementation of Stack in Array and Linked List**
- In C++
- In Java

12. Queue

- **Introduction and Application**
- **Implementation of the queue using array and LinkedList**
- In C++ STL
- In Java
- Stack using queue.

13. Deque

- **Introduction and Application**
- **Implementation**
- In C++ STL 14
- In Java
- **Problems (With Video Solutions)**
- Maximums of all subarrays of size k
- Array Deque in Java
- Design a DS with min max operations.

14. Tree

- **Introduction**
- Tree
- Application
- Binary Tree • Tree Traversal
- **Implementation of:**
- In order Traversal
- Preorder Traversal

- Post order Traversal
- Level Order Traversal (Line by Line)
- Tree Traversal in Spiral Form

15. Binary Search Tree

- **Background, Introduction and Application**
- **Implementation of Search in BST**
- **Insertion in BST**
- **Deletion in BST**
- **Floor in BST**
- **Self-Balancing BST**
- **AVL Tree**

16. Heap

- **Introduction & Implementation**
- **Binary Heap**
- Insertion
- Heapify and Extract
- Decrease Key, Delete and Build Heap
- **Heap Sort**
- **Priority Queue in C++**
- **PriorityQueue in Java**

17. Graph

- **Introduction to Graph**
- **Graph Representation**
- Adjacency Matrix
- Adjacency List in CPP and Java
- Adjacency Matrix VS List
- **Breadth-First Search**
- Applications
- **Depth First Search**
- Applications
- **Shortest Path in Directed Acyclic Graph**
- **Prim's Algorithm/Minimum Spanning Tree**
- Implementation in CPP

- **Range Query**
- **Update Query**

24. Disjoint Set

- **Introduction**
- **Find and Union Operations**
- **Union by Rank**
- **Path Compression**
- **Kruskal's Algorithm**

1. Improved my problem-solving skills by practicing problems to become a stronger developer.
2. Developed my analytical skills on Data Structures to use them efficiently
3. Solved problems asked in product-based companies' interviews.
4. Solved problems in contests similar to coding round for SDE role

Reason for choosing this technology.

With advancement and innovation in technology, programming is becoming a highly in-demand skill for Software Developers. Everything you see around yourself from Smart TVs, ACs, Lights, Traffic Signals uses some kind of programming for executing user commands.

Data Structures and Algorithms are the identity of a good Software Developer. The interviews for technical roles in some of the tech giants like Google, Facebook, Amazon, Flipkart is more focused on measuring the knowledge of Data Structures and Algorithms of the candidates. The main reason behind this is Data Structures and Algorithms improves the problem-solving ability of a candidate to a great extent.

1. This course has video lectures of all the topics from which one can easily learn. I prefer learning from video rather than books and notes. I know books and notes and thesis have their own significance but still video lecture or face to face lectures make it easy to understand faster as we are involved Practically.
2. It has 200+ algorithmic coding problems with video explained solutions.
3. It has track based learning and weekly assessment to test my skills.

4. It was a great opportunity for me to invest my time in learning instead of wasting it here and there during my summer break in this Covid-19 pandemic.
5. This was a lifetime accessible course which I can use to learn even after my training whenever I want to revise.

Implementation code

```
#include<iostream>
#include<fstream>
using namespace std;

/* This class provides a data structure which can hold and manipulate the values in a
sudoku puzzle.
  *       In this file, we shall call this data structure the 'Sudoku Frame'.
*/
class SudokuFrame{

        int sudokuFrame[9][9]; //This pointer will hold all the values in the matrix.
        int editableFrame[9][9]; //This pointer will tell us all the values which are editable.

        /* This constructor calls the menu() func to provide the menu.
        */
        public:SudokuFrame(){
                menu();
        }

        /* Displays a menu to the user when the SudokuFrame objects in instantiated
          *       (which is basically at the start of the program) to display all possible options
          *       from the user.
        */
        private:void menu(){

                cout<<"\n======================\n";
                cout<<"    Sudoku Solver\n";
                cout<<"======================\n\n";

                cout<<"Welcome to Sudoku Solver!\n";

                readFrameValues();

        }

        /* Reads the values for the Sudoku Frame cell-by-cell.
        */
        private:void readFrameValues(){
                cout<<"\nEnter the specified value when prompted.\n";
                cout<<"Enter 0 if cell is empty.\n\n";

                int rowIter, colIter;

                for(rowIter=0; rowIter<9; rowIter++){ //Iterating over cells to read vals.
                        for(colIter=0; colIter<9; colIter++){
                                int enteredValue;

                                cout<<"Enter value for
cell["<<rowIter+1<<"]["<<colIter+1<<"] --> ";
                                cin>>enteredValue;

                                if(!(enteredValue>=0 && enteredValue<=9)){ //Checking for
bounds in input.
                                        while(true){ //We loop until valid input is read from
user.
```

```
                                        cout<<"Oops! You seem to have entered a
wrong value! Try again.\n";
                                        cout<<"Enter value for cell
["<<rowIter+1<<"]["<<colIter+1<<"] --> ";
                                        cin>>enteredValue;

                                        if(enteredValue>=0 && enteredValue<=9)
break;
                                   }
                              }

                              sudokuFrame[rowIter][colIter]=enteredValue;

                              if(enteredValue==0) editableFrame[rowIter][colIter]=0;
                              else editableFrame[rowIter][colIter]=1;
                         }
                         cout<<"-------\n"; //Display a break after every row for convenience.
                    }
          }

          /* Assigns the passed-in number to the specified row and col.
          */
          public:void setCellValue(int row, int col, int num){
                    if(editableFrame[row][col]==0) sudokuFrame[row][col]=num;
          }

          /* Returns the value of the cell at the specified row and col.
          */
          public:int getCellValue(int row, int col){
                    int cellValue=sudokuFrame[row][col];
                    return cellValue;
          }

          /* Returns 0/1 depending on editableFrame values.
```

```
	*/
	public:int isEditable(int row, int col){
		return (editableFrame[row][col]-1)*(-1);
	}

	/* Clears frame of all values, other than the question values, from
	 *	the specified cell to the last cell.
	*/
	public:void clearFrameFrom(int row, int col){
		int jcount=0;
		int rowIter, colIter;

		for(rowIter=row; rowIter<9; rowIter++){

			if(jcount==0) colIter=col;
			else colIter=0;

			for(; colIter<9; colIter++){
				if(editableFrame[rowIter][colIter]==0)
sudokuFrame[rowIter][colIter]=0;
			}

			jcount++;

		}
	}

	/* Displays the values stored in the frame with designs. We also use
	 *	ANSI colors, using escape sequences, to display the frame.
	*/
	public:void displayFrame(){
		cout<<"\n[++=====================================================
++]";
		int rowIter, colIter;

		for(rowIter=0; rowIter<9; rowIter++){
			int cellIter=1;

			cout<<"\n ||";
			for(colIter=0; colIter<9; colIter++){
				if(cellIter==3){
					cout<<"  "<<sudokuFrame[rowIter][colIter]<<" ";
					cout<<" ||";
					cellIter=1;
				}
				else{
					cout<<"  "<<sudokuFrame[rowIter][colIter]<<"  ";
					cellIter++;
				}
			}

			if(rowIter%3!=2) cout<<"\n[++---------------++---------------++--------
-------++";
			else
cout<<"\n[++=====================================================++]";
		}

	}
};

/**
  *	This class provides the programmer a very simple way to iterate through
  *	the possibilities of a specified cell. This object utilises linked lists.
*/
```

```
class Possibilities{

	struct node{
		int value;
		struct node* next;
	}; //The struct for the node

	typedef struct node* Node;

	Node head; //The head node
	Node pos; //A node iterator variable



	/* This constructor initialises the head (or sentinel) node
	  *	@param none
	*/
	public:Possibilities(){
		head=new struct node;
		head->next=NULL;
	}

	/*This destructor destroys the linked list once the object
	  *	has finished its lifespan. Calls the destroy() function.
	*/
	public:~Possibilities(){
		destroy();
	}

	/* This function takes in a number and adds it as a node in the linked list.
	*/
	public:void append(int number){
		Node temp=new struct node;

		temp->value=number;
```

```
        temp->next=NULL;

        pos=head;
        while(pos!=NULL){
                if(pos->next==NULL){
                        pos->next=temp;
                        break;
                }
                pos=pos->next;
        }
}

/* An operator overload function which overloads the [] operator.
*/
public:int operator[](int index){
        int count=0;
        pos=head->next;

        while(pos!=NULL){
                if(count==index)
                        return pos->value;
                pos=pos->next;
                count++;
        }

        return -1;
}

/* Prints the values inside all the nodes of the linked list.
*/
public:void print(){
        pos=head->next;
        while(pos!=NULL){
                cout<<pos->value<<",";
                pos=pos->next;
            }
            cout<<"\b";
        }

        /** Returns the length of the linked list.
        */
        public:int length(){
            pos=head->next;
            int count=0;

            while(pos!=NULL){
                count++;
                pos=pos->next;
            }

            return count;
        }

        /** This function takes in a possibilities object and copies
          *     the contents into THIS object.
        */
        public:void copy(Possibilities possibilities){ //Need to make this clear the old list if
exists
            int oldLength=possibilities.length();
            int iter=0;

            pos=head;
            for(iter=0; iter<oldLength; iter++){
                Node temp=new struct node;

                temp->next=NULL;
                temp->value=possibilities[iter];
```

```
                    pos->next=temp;
                    pos=pos->next;
          }
     }

     /* Frees all the nodes in the linked list.
     */
     private:void destroy(){
          Node temp;
          pos=head;
          while(pos!=NULL){
                    temp=pos;
                    pos=pos->next;
                    delete temp;
          }
     }

};


/*   Takes in the SudokuFrame object and solves the Sudoku Puzzle.
*/
class SudokuSolver{

     int recursiveCount; //Statistics variable
     SudokuFrame frame; //The frame object

     /**
       The constructor initialises the recursiveCount variable and also calls
       *    the solve() function which solves the puzzle. It also displays the frames
       *    before and after the solving.
     */
     public:SudokuSolver(){
          recursiveCount=0;
```

```
        cout<<"\nCalculating possibilities...\n";
        cout<<"Backtracking across puzzle....\n";
        cout<<"Validating cells and values...\n\n";

        solve();
        cout<<"QED. Your puzzle has been solved!\n\n";
        displayFrame();

        cout<<"\n\n";
}

/**
  *     Checks if the value in the specified cell is valid or not.
*/
private:bool cellValueValid(int row, int col, int currentValue){
        int rowIter, colIter;

        //Checking if value exists in same column
        for(rowIter=0; rowIter<9; rowIter++){
                if(rowIter!=row){
                        int comparingValue=frame.getCellValue(rowIter,col);
                        if(comparingValue==currentValue) return false;
                }
        }

        //Checking if value exists in same row
        for(colIter=0; colIter<9; colIter++){
                if(colIter!=col){
                        int comparingValue=frame.getCellValue(row,colIter);
                        if(comparingValue==currentValue) return false;
                }
        }
		//Checking if value exists in the same 3x3 square block
		if(ThreeByThreeGridValid(row,col,currentValue)==false) return false;

		return true;
	}

	/*Checks if the same value is also present in the same 3x3 grid block
	*/
	private:bool ThreeByThreeGridValid(int row, int col, int currentValue){
		int rowStart=(row/3)*3;
		int rowEnd=(rowStart+2);

		int colStart=(col/3)*3;
		int colEnd=(colStart+2);

		int rowIter, colIter;

		for(rowIter=rowStart; rowIter<=rowEnd; rowIter++){
			for(colIter=colStart; colIter<=colEnd; colIter++){
				if(frame.getCellValue(rowIter,colIter)==currentValue) return
false;
			}
		}

		return true;
	}

	/*	Gets the possible values and assigns them to the possibilities list.
	*/
	private:Possibilities getCellPossibilities(int row, int col){
		int iter=0;

		Possibilities possibilities;
		for(iter=1; iter<=9; iter++){
			if(cellValueValid(row,col,iter)==true)
				possibilities.append(iter);
		}

		return possibilities;
	}

	/**
	 *	The recursive function which does all the work, this iterates over the
	 *	possible values for the specified cell one-by-one. Once a value has been
filled, it
	 *	goes to the next cell. Here, the same thing happens. If none of the possible
values
	 *	work out, then the function backtracks to the previous cell.
	*/
	private:int singleCellSolve(int row, int col){

		recursiveCount++; //This is used to see how many times the func is called.

		if(frame.isEditable(row,col)==true){

			Possibilities possibilities;
			possibilities.copy(getCellPossibilities(row,col));

			int posLength=possibilities.length();
			int posIter=0, newRow=row, newCol=col;

			for(posIter=0; posIter<posLength; posIter++){		//We iter over
the possible values
				int possibility=possibilities[posIter];
				frame.setCellValue(row,col,possibility);

				//We now increment the col/row values for the next recursion
```

```
                    if(col<8) newCol=col+1;
                    else if(col==8){
                         if(row==8) return 1;                    //this means
success
                         newRow=row+1;
                         newCol=0;
                    }

                    {

                         if(singleCellSolve(newRow,newCol)==0){          //If
wrong, clear frame and start over
                              frame.clearFrameFrom(newRow,newCol);
                         }
                         else return 1;

                    }

               }

               return 0;

          } //The isEditable() if block ends here.
          else{

               int newRow=row, newCol=col;

               //Same incrementing of the col/row values
               if(col<8) newCol=col+1;
               else if(col==8){
                    if(row==8) return 1;
                    newRow=row+1;
                    newCol=0;
               }
```

```
                        return singleCellSolve(newRow,newCol);

                }
                return 0;
        }

        /* Calls the singleCellSolve() func and prints a success/fail mesg.
        */
        private:void solve(){
                int success=singleCellSolve(0,0);
        }

        private:void displayFrame(){
                frame.displayFrame();
        }

        public:void statsPrint(){
                cout<<"\nThe singleCellSolve() function was recursively called
"<<recursiveCount<<" times.\n";
        }

};

int main(){
        SudokuSolver ss;
        return 0;
}
```

Learning Outcomes

Programming is all about data structures and algorithms. Data structures are used to hold data while algorithms are used to solve the problem using that data.

Data structures and algorithms (DSA) goes through solutions to standard problems in detail and gives you an insight into how efficient it is to use each one of them. It also teaches you the science of evaluating the efficiency of an algorithm. This enables you to choose the best of various choices.

For example, you want to search your roll number in 30000 pages of documents, for that you have choices like Linear search, Binary search, etc.

So, the more efficient way will be Binary search for searching something in a huge amount of data. So, if you know the DSA, you can solve any problem efficiently.

The main use of DSA is to make your code scalable because.

- Time is precious.
- Memory is expensive.

Bibliography

- Google • GeeksforGeeks
- Learn C++ – Skill up with our free tutorials (learncpp.com)
- Stack overflow
- YouTube

https://github.com/Swaraj468/report.git