LAB MANUAL

Design and Analysis of Algorithm

Block -1

The problems set for the block is divided into three sections. Section 1 problems covers basic characteristic of algorithm and various method to solve recurrence. Problem on recurrence covers unit -1 of block-1 that describe, Substitution method, iteration method and Master method. Section -2 problems are designed in such a way that student will be able to identify whether a function belongs to or does not belong to any of the basic asymptotic notations. Basic asymptotic notations (Big Oh, Theta, Big Omega, small o, small omega) definition and examples are discussed in unit-2 of block 1. Section-3 of this problem set is to write a C program for algorithms discussed in unit-3 of block-1. After writing the C program students are required to find number of comparisons and number of time a referred loop executes. This will help them in understanding the complexity analysis for various algorithms.

SECTION 1

Introduction

In this section recurrence problems are given that are to be solved by Substitution method, Iteration Method and Master method. The brief strategy followed by various methods is as follows:

Substitution method uses induction approach for solving recurrence. Guess the form of the answer, and then use induction to find the constants. After that show that assumed / guessed solution works.

Iteration method expands the recurrence by iteration approach. Then basic algebra and summation form are applied to convert the recurrence into a summation and find the solution of given recurrence.

Master method is used to solve the divide and conquer form of recurrences. This method is applicable when recurrence variables a,b and f(n) are fixed. Further we have to identify particular case of master method under which the given recurrence falls to find the solution.

Objective

For understanding design and analysis of algorithm various methods for solving recurrences such as Substitution method, iteration method and master method must be known to students. By studying these methods students will be able to understand how to eliminate recursion from the recurrence. It will be very useful while studying algorithm analysis when a function can not be easily written without recursion. Solution to recurrence will give time complexity estimate information.

Questions:

1. Write the characteristics of good algorithm.
2. Solve the following recurrences by Master method
    a. $T(n) = 8T(n/2) + cn^2$
    b. $T(n) = 3T(n/5) + n$
    c. $T(n) = 2T(n/2) + n$ [Hint: case 3 of Master Method]
    d. $T(n) = 2T(n/2) + 1$
    e. $T(n) = 3T(n/4) + nlgn$ [Hint: case 3 of Master Method]

Solution to 2(a)

For applying master method first identify that the given recurrence falls under which case. For the same identify a and b from the given recurrence. It gives a=8 and b=2 and $f(n) = cn^2$. Here, cn^2 is $O(n^{\log_2 8 - \varepsilon}) = O(n^{3-\varepsilon})$ for any $\varepsilon \leq 1$, so this falls into case 1 of Master Method definition. Hence, solution to the given recurrence T(n) is $\theta(n^3)$.

3. Solve the following recurrences by Substitution method
    a. $T(n) = 3T(n/4) + cn^2$

b. $T(n) = T(n-1) + n$

Solution to 3(b)

Assumtion: $T(n) = O(n^2)$

Induction goal: $T(n) \leq c\, n^2$, for some c and $n \geq n0$

Induction hypothesis: $T(n-1) \leq c(n-1)^2$ for all $k < n$

Proof of induction goal:

$T(n) = T(n-1) + n \quad \leq c\,(n-1)^2 + n$

$= cn^2 - (2cn - c - n) \leq cn^2$

if: $2cn - c - n \geq 0 \Leftrightarrow c \geq n/(2n-1) \Leftrightarrow c \geq 1/(2 - 1/n)$

For $n \geq 1 \Rightarrow 2 - 1/n \geq 1 \Rightarrow$ any $c \geq 1$ will satisfy inequality of induction goal.

Hence, assumed solution is correct and solution to the given recurrence is $O(n^2)$

4. $T(1)=1$

   $T(n)=T(n/2) + \sqrt{n}$,

   Prove that $T(n)=O(\sqrt{n})$ using the substitution method.

5. $T(1) = 1$

   $T(n) = 2T(n-1) + 1$

   Prove that $T(n) = O(2^n)$ by iteration method

Solution to 5:

$T(n) = 2T(n-1) + 1$

$T(n) = 4T(n-2) +2+ 1$

$T(n) = 8T(n-3) +2^2+2+ 1$

……..

……..

After ith iteration

$T(n) = 2^iT(n-i) +2^{i-1} +……2^2+2+ 1$

…..

After i= n-1 iterations, it will stop,

$$T(n)= 2^{n-1} T(1) + \sum_{i=0}^{n-2} 2^i$$

$$= 2^{n-1} + 2^{n-1} -1$$

$$= 2 * 2^{n-1} -1 = 2^n-1 = T(2^n)$$

6. Solve the following recurrences by iteration method

   $T(a) = \theta(1)$

   $T(n) = T(n-a) + T(a) + n$

7. Solve $T(n) = n + 2T(n/2)$ using iteration method.

8. Write the recurrence for binary search method and solve the same by recursion tree approach.
9. Write a program to find factorial of a number.

SECTION 2

Introduction

Asymptotic notations are fundamentals required for understanding various bounds and comparisons of efficiency of algorithms. These notations provide an idea about how much time an algorithm will take to solve a given problem. Basically there are five asymptotic notations which give different measures like maximum time requirement, minimum time requirement or maximum and minimum time requirement for running an algorithm. The problem set in this section will provide insight to the student to identify that a function will fall under which asymptotic notation. This further provides time limits/bounds information. This will help the students to understand algorithm analysis in a better way.

Objective

After solving various problems on asymptotic notations student will learn how to find a function will belong to which of the basic asymptotic notation. It also tells how to represent complexity for an algorithm. These notations will help in comparing algorithms time estimates. As a problem may have many solutions. In order to find the good solution we are required to compare the time requirement of set of solutions. These notations give a way to compare one algorithm against other one. This will further provide information regarding the provided solution will be feasible or not.

Questions:

Solve the following:

1. Is $2n^2 + 4n + 4 = \theta(n^2)$ or not?
2. Prove that $n^2/2-3n= \theta (n^2)$
3. Show that $3n^3=O(n^4)$ for appropriate c and n_0.
4. Is $15n^3 + n^2 + 4 = O(n^3)$.or not?
5. Is $n^3= \Omega(n^2)$ or not?
6. Show that $\sqrt{n}=\Omega(\log n)$ by definition.
7. Is $n= \Omega(n^2)$ or not?
8. Write an iterative algorithm and C program for generating Fibonacci series and derive its complexity?
9. Write a program to find maximum of 10 numbers and derive its running time complexity by computing complexity of each statement in the program.
10. Write a program to find minimum of 8 numbers and find the total number of comparison in the program.
11. Write a program to perform linear search. Take input data set of 10 numbers.
    a. Consider data set in ascending order. Perform linear search on that data set and find the total number of comparison operations.
    b. Consider data set in descending order and find the total number of comparison operations.

c. Consider random data set. Find the total number of comparison operations.

12. For the following abstract find the complexity expression

i)
```
a=5;
for(i=0;i<a;i++)
{
        printf("Inside the for loop");
}
```

ii)
```
x=7;
y=x;
```

iii)
```
a=7;
b=5;
c=1;
    if( a> b)
            if( a>c)
                    max=a;
```

iv)
```
for(i=0;i<n;i++)
    for(j=0;j<n;j++)
    printf("inner for loop");
```

Solution to 1:

To verify $2n^2 + 4n + 4 = \theta(n^2)$ or not, make use of definition of theta(θ) asymptotic notation given as below.

θ (Theta): Let g(n) be given function. f(n) is the set of function defined as

θ (g(n))={f(n): if there exist positive constant c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all n, $n \geq n_0$}

$f(n)= 2n^2 + 4n + 4$ and $g(n)= n^2$

=> to verify $f(n) \leq c_2 g(n)$

We can write $f(n)= 2n^2 + 4n + 4$ as $f(n)= 2n^2 + 4n + 4 \leq (2+4+4)\ n^2$ (write f(n) in terms of g(n) such that mathematically ineqaulity should be true)

$\leq 10\ n^2$ for all $n > 0$

$c_2=10$ for all $n > 0$ i.e $n_0=1$

To verify $0 \leq c_1 g(n) \leq f(n)$

We can write $f(n)= 2n^2 + 4n + 4 \geq 2n^2$ (again write f(n) in terms of g(n) such that mathematically ineqaulity should be true)

$c_1=2$ for all n, $n_0=1$

$\Rightarrow\ 2n^2 \leq 2n^2 + 4n + 4 \leq 10\ n^2$ for all $n \geq n_0$, $n_0=1$

i.e we are able to find, $c_1=2$, $c_2=10$ and $n_0=1$ such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all n, $n \geq n_0$

So, $f(n)= \theta(g(n))$ for all $n \geq 1$

$2n^2 + 4n + 4 = \theta(n^2)$ Holds.

Solution to 7:

Ω (Big Omega): For a given function g(n), Ω (g(n))), f(n) is the set of functions defined as Ω (g(n))={f(n): if there exist positive constant c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all n, $n \geq n_0$}

To check $n= \Omega(n^2)$ or not , we will be using Big Omega definition given as above.

$f(n)= n$

$g(n)= n^2$

To show $0 \leq cg(n) \leq f(n)$ for all n, $n \geq n_0$

For given $f(n)= n$, we can not find any positive constant c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all n, $n \geq n_0$

This can be verfied as we can not find c and n_0 such that $0 \leq cn^2 \leq n$ holds for $n \geq n_0$

As inequality does not hols and c and n_0 can not be identified proves

$n \neq \Omega(n^2)$

SECTION 3

Introduction

In this section programming approach is considered so that student can easily identify key operations of an algorithm. The key operations of the algorithm are required for computing complexity of an algorithm. Key operations in the algorithm could be comparing or swapping etc. Further it provides the total running time complexity of an algorithm.

Objective

After solving the given problem set, student will learn how to design a solution for a problem. While designing solution what operations should be considered so that it will take minimum time as much as minimum it can take. As time efficiency is the key factor towards studying design and analysis of the algorithm. Design a solution such that it will have less number of comparisons or swapping operations. This will reduce the total running time complexity of an algorithm.

Questions:

1. Write a C program to evaluate a polynomial using Horner's rule and print the count for '**for loop'** execution.
2. Write a C program to compute a^n by left to right binary exponentiation method and print the total number of comparison take place during execution.
3. Write a C program to sort a data list using selection sort. Consider ascending order , descending order and mixed list as input and print how many number of times **'for loop'** executing for the respective data set.
4. Write a C program to perform linear search and write the number of comparison for worst case, best case and average case by considering an appropriate data set.

5. Write a C program to sort a data set using insertion sort. Write the number of times **for loop** executing and the total number of comparison during execution. Also write the output of data set after each iteration in the program.
6. Write a C program to find multiplication of two matrices of order 3x3 and find the total number of comparison, total number of assignment, total number of multiplication and total number of addition.
7. Write a C program for performing bubble sort for 8 numbers and evaluate the total number of comparison and total number of assignment for the same.
8. Write an algorithm and C program for computing gcd by simple approach (studied in school) and Euclid's approach. Also find gcd of (12, 48) by both methods and analyze the difference in terms of operation and computation.

Answer to Question 4

```
#include<stdio.h>
#define MAX 10
void main()
{
        int a[MAX],item,i;

        int found=0;

        for(i=0;i<MAX;i++)

        {
                if (a[i]==item)
                        found = 1;
                break;
        }
        if (i==MAX)
                found = 0;
        if (found==1)
```

```
            printf("Search successful");
        else
            printf("Element not found");
}
```

Worst Case :

2	4	5	9	14	12	1	7	3	6

Element to be searched= 8

Number of comparison inside the for loop construct = 10
Number of comparison for first if construct =1
Number of comparison for second if construct to print the message =1

Total number of comparison in the programme = 12

Element to be searched = 6

Number of comparison inside the for loop construct = 10
Number of comparison for first if construct =1
Number of comparison for second if construct to print the message =1

Total number of comparison in the programme = 12

Best Case :

Element to be searched = 2

Number of comparison inside the for loop construct = 1
Number of comparison for first if construct =1

Number of comparison for second if construct to print the message =1

Total number of comparison in the programme = 3

Average Case:

Element to be searched = 14

Number of comparison inside the for loop construct = 5
Number of comparison for first if construct =1
Number of comparison for second if construct to print the message =1

Total number of comparison in the programme = 7

For finding the complexity of any algorithm, important operations are considered. Here important operation is comparison inside the for loop construct.
So maximum time taken by the algorithm will be O(n).

Block-2

Introduction

The set of problems for Block-2 is divided into three sections. Section 1 has problem covering Greedy approach to solve knapsack problem. Greedy strategy to find minimum spanning tree by kruskal's or prim's algorithm for better understanding of complexity analysis. Then single pair shortest path algorithms dijkastra's and Bellmanford problem is considered under greedy approach. Programming strategy and output of each iteration during program execution will help them in understanding complexity analysis. Section 2 has problem based on Divide and conquer approach. Divide and conquer approach is very powerful approach. In this large problem is divided in to smaller set of sub problems. Various algorithms like merge sort, binary search and matrix multiplication by strassen's algorithm are

considered that is to be solved by divide and conquer approach. Also exercise for solving the recurrence form of divide and conquer technique is taken to analyze their complexity. This analysis will help in understanding divide and conquer approach in a better way. Whereas section 3 covers the problems on topics covered in unit-3 of the block. Graph representation methods and search algorithm viz BFS and DFS are very common. These searching algorithms in the graph have wide application.

SECTION -1

Introduction

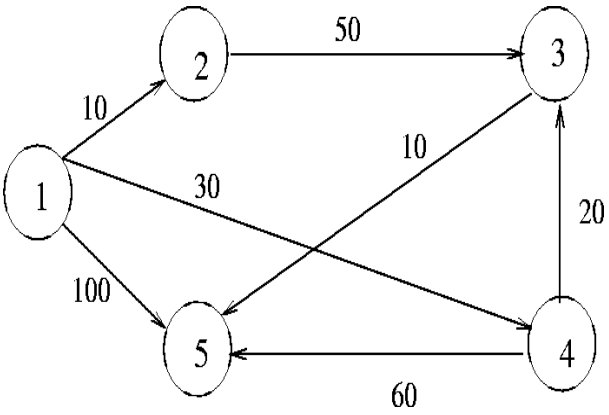
This section will help in exploring how greedy technique approach is applied to solve different types of problems. Generally greedy techniques are used for optimization problems. It has wide range of domain where it can be applied viz spanning tree, shortest path problem etc. To start with greedy approach for solving a problem, identify elements of greedy strategy in respect of given problem. Then step by step process should be applied towards finding a feasible solution of the given problem.

Objective

After attempting the following exercises, student will be able to understand how to use greedy technique for solving a problem. What are the key elements of greedy strategy that we need to identify in respect of our problem domain. What will be the approach for applying greedy technique for solving a problem and how to derive an algorithm time complexity. By considering programming approach for the problem where greedy technique is applied viz minimum spanning tree student will get number of key operations through which algorithm complexity can be derived. Single pair shortest path problem is also taken into consideration where greedy technique can be applied. After this student will have a fair idea where and how to apply greedy technique.

Questions:

1. What are the elements of the Greedy strategy?
2. Derive complexity expression for the function Greedy Fractional-Knapsack (P[1..n], W[1..n], X [1..n], M) defined under section Knapsack(fractional) problem.
3. Write C program to generate Minimum Cost Spanning Tree using kruskal's algorithm. Also draw the minimum spanning tree by using prim's algorithm. Also write the output after each iteration of the program. Find out the cost of minimum spanning tree by using both algorithm's i.e kruskal's and prim's.
   [Hint: Take an example of graph and on that graph find minimum cost spanning tree by both kruskal's algorithm and prim's algorithm. Make an observation that cost of spanning tree by both algorithms is same. But Minimum spanning tree may be different.]
4. Define shortest path problem. Differentiate between dijkastra's and Bellmanford algorithm.
5. Consider a weighted graph of 6 vertices that represents company stores of an electronic item viz. T.V in a city. Weights on edges will represent distance between the stores. All stores are connected with head office by one or the other path. One of the stores is designated as head office. Head office will be distributing required number of TV to different stores. Find shortest path from head office to different stores by using Dijkastra's algorithm such that the required number of TV's can be supplied to a store by shortest path i.e in minimum time.
6. For the above problem draw adjacency matrix and adjacency list.
7. Consider vertex 1 as start vertex and find shortest path using Dijkastra' algorithm and show the result for each iteration.



Section 2

Introduction

Divide and conquer approach is top down strategy for designing an algorithm for solving some problem. In this approach a given problem is divided into smaller sub problems. This division will be till the point that problem is solvable directly. Then the solution obtained from sub problems is combined to have the final solution of the given problem. It can be framed in a way that it is three step process. One is that divide given problem into smaller sub problems. Solve smaller sub problems. Combine solutions of sub problems to obtain the final solution. This approach is very commonly known and used in many applications like searching, sorting etc.

Objective

After going through with the following exercises student will be able to learn the application of divide and conquer strategy. The understanding of basic components required to apply this approach. The form of recurrence for a problem where divide and conquer approach will be used. The methods of solving the recurrences are already taken up in section-1 of block-1. A searching problem that is solved without divide & conquer and with divide & conquer is taken for better understanding of its

application. Programming aspect for solving a sorting and matrix multiplication problem where use of divide & conquer technique is required.

Questions:

1. Explain divide and conquer strategy. Also give a recurrence to where divide and conquer strategy can be applied.
2. Write recurrence for merge sort and solve this recurrence by master method.
3. Differentiate between linear search and binary search.
4. Write a C program for merge sort. Also discuss its complexity analysis.
5. Consider a data set of 8 numbers and apply merge sort algorithm, give the out put of each iteration.
6. Compare performance of merge sort and quick sort in respect of best case, average case and worst case.
7. Write a C program to perform basic matrix multiplication (order 2 x 2). Find the total number of addition operation and multiplication operation take place while computing matrix multiplication,
8. Write a C program to perform matrix multiplication (order 2 x 2) using strassen's algorithm. Find the total number of addition operation and multiplication operation take place while computing matrix multiplication.
9. Write a C program to perform quick sort. Write its recurrence and solve it to find the running time complexity. Also write the output of each iteration for data set of 8 numbers that is to be sorted.

Answer to Question 5

Consider the following data set of 8 numbers.

4	7	3	1	9	8	6	17

As per merge sort algorithm, split the given data set into two halves till every sub lists have only one numbers. Splitting into two halves are indicated by underlying the sub lists of numbers of data set as follows:

4, 7, 3, 1, 9, 8, 6, 17

4, 7, 3, 1, 9, 8, 6, 17

4, 7, 3, 1, 9, 8, 6, 17

Now apply merge process to get the required sorted list of given data set. Sub list merging is also indicated by underlying after merge process.

4, 7, 1, 3, 8, 9, 6, 17

1, 3, 4, 7, 6, 8, 9, 17

1, 3, 4, 6, 7, 8, 9, 17

It is required sorted list of the given data set.

In each step of above process it can be observed that in the first step problem is divided into two halves by splitting and in the second phase merging the solution of sub list. It can be observed that above algorithm follows divide and conquer approach. This way analyzing the output and problem solving approach, it can be found that algorithm falls under which category. How to formulate recurrence and solve the same by solution methods discussed in Unit-1 of Block-1.

Answer to Question 6

```
#include<stdio.h>

void main()
```

```
{
    int a[2][2],b[2][2],c[2][2],sum,i,j,k;

    printf("Enter elements of first matrix");

    for(i=0;i<2;i++)
            for (j=0;j<2;j++)
                    scanf("%d",&a[i][j]);

    printf("Enter elements of Second matrix");

    for(i=0;i<2;i++)
            for (j=0;j<2;j++)
                    scanf("%d",&b[i][j]);

    for(i=0;k<2;k++)
    {
            for(j=0;i<2;i++)
            {
                    c[i][j]=0;
                    for (k=0;j<2;j++)

                            c[i][j]= c[i][j]+a[i][k]*b[k][j];
            }
    }

    printf("Matrix multiplication of two matrices a[2][2] and b[2][2]\n");
    for(i=0;i<2;i++)
    {
            for (j=0;j<2;j++)
                    printf(" %d ",c[i][j]);
            printf("\n");
```

```
    }

}
```

Total number of multiplication = 8
Total number of addition operation = 4

Section 3

Introduction

This section has coverage for representation method of a graph and searching in the graph. Searching is a one of the important operation in graph algorithms. Commonly used graph search algorithms are breadth first search (BFS) and depth first search (DFS). For each searching algorithm student should know order of vertices visited and path in which they are visited. Each searching algorithm has different areas of applications.

Objective

After solving the problems in this section student will be able to understand different ways by which a graph can be represented. Basic searching algorithms BFS and DFS will be thorough to the students. Computation of BFS and DFS running time complexity will be clear. Also order of vertices visited in graph searching algorithm in each iteration will help the students to understand graph searching algorithms effectively.

Questions:

1. Define graph. Describe different method for representation of graph.
2. Consider an example graph with minimum of 5 vertices and give its adjacency matrix and adjacency list representation.

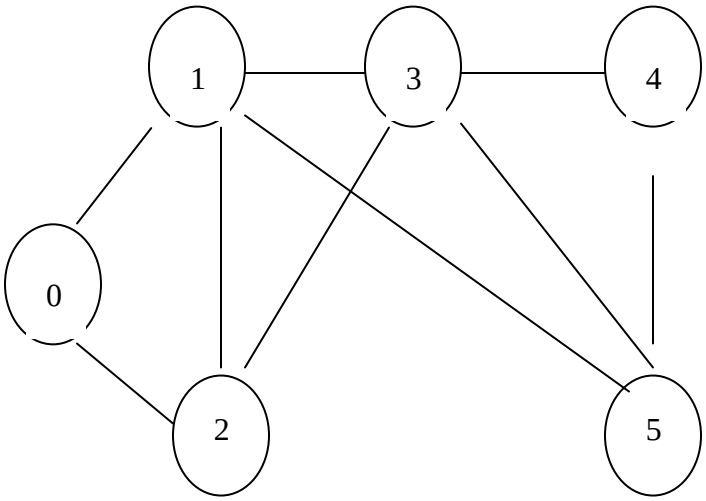
3. Explain complexity analysis of Breadth first search algorithm.

[Hint: For computing the complexity of an algorithm, write the algorithm along with line number. Find complexity of each line in the algorithm and by summing up all lines complexities, you can evaluate total running time complexity of the algorithm]

4. Illustrate complexity analysis of Depth First Search algorithm.

[Hint: For computing the complexity of an algorithm, write the algorithm along with line number. Find complexity of each line in the algorithm and by summing up all lines complexities, you can evaluate total running time complexity of the algorithm]

5. Explain each iteration of DFS and BFS algorithm for the following graph:



[Hint: Show the status of each vertex with color marking scheme as grey, white or black as discussed in the example given in the unit-3 of Block-2 for DFS and BFS search algorithms]

6. Differentiate between BFS and DFS.

[Hint: List the differences in respect of how vertices are explored, Data structure required to implement the search algorithm, Application area where these search algorithm can be used, Complexity expression if adjacency matrix data structure is used and if adjacency list data structure is used]

7. Consider a connected graph with 4 vertices. Write a program to store it by adjacency matrix and adjacency list. Find out the memory used in both the representation.

[Hint: Consider a sparse graph where number of edges will be less. In this graph adjacency matrix, number of zero's will be more as compare to number of one's]

8. Consider a complete connected graph with 4 vertices. Write a program to store it by adjacency matrix and adjacency list. Also find the memory used in both the representation.

[Hint: Complete connected graph is where each vertex is connected to every other vertex in the graph.]