PUNE INSTITUTE OF COMPUTER TECHNOLOGY

DHANKAWADI, PUNE –43

LIST OF LAB EXPERIMENTS

ACADEMIC YEAR: 2022- 2023

Department: Computer Engineering **Date:** 18/07/2021
Class: B.E. **Semester:** II
Subject: Laboratory Practice III (410246) **Examination scheme:**TW-50, PR-50

LAB EXP. NO	PROBLEM STATEMENT (Any 05 assignments Design and Analysis of Algorithms, Machine Learning & Blockchain Technology AND 01 Mini- project per course)
GROUP A	**Based on Design and Analysis of Algorithms (410241)**
1.	Write a program non-recursive and recursive program to calculate Fibonacci numbers andanalyze their time and space complexity.
2.	Write a program to implement Huffman Encoding using a greedy strategy.
3.	Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch andbound strategy.
4.	Design n-Queens matrix having first Queen placed. Use backtracking to place remainingQueens to generate the final n-queen
5.	Write a program for analysis of quick sort by using deterministic and randomized variant.
6	**Mini-Project on DAA** **1. Mini Project -** Write a program to implement matrix multiplication. Also implement multithreaded matrix multiplication with either one thread per row or one thread per cell. Analyze and compare their performance. **2. Mini Project -** Implement merge sort and multithreaded merge sort. Compare time requiredby both the algorithms. Also analyze the performance of each algorithm for the best case and the worst case. **3. Mini Project -** Implement the Naive string matching algorithm and Rabin-

	Karp algorithm forstring matching. Observe difference in working of both the algorithms for the same input. **4. Mini Project -** Different exact and approximation algorithms for Travelling-Sales-PersonProblem
GROUP B	**Based on Machine Learning (410242)**
1	Predict the price of the Uber ride from a given pickup point to the agreed drop-off location.Perform following tasks: 1. Pre-process the dataset. 2. Identify outliers. 3. Check the correlation. 4. Implement linear regression and random forest regression models. Evaluate the models and compare their respective scores like R2, RMSE, etc. Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset
2	Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State Not Spam, b) Abnormal State Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance. Dataset link: The emails.csv dataset on the Kaggle https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv
3	Given a bank customer, build a neural network-based classifier that can determine whetherthey will leave or not in the next 6 months. Dataset Description: The case study is from an open-source dataset from Kaggle.The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc. Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling Perform following steps: 1. Read the dataset. 2. Distinguish the feature and target set and divide the data set into training and test sets. 3. Normalize the train and test data. 4. Initialize and build the model. Identify the points of improvement and implement the same. Print the accuracy score and confusion matrix (5 points).
4	Implement Gradient Descent Algorithm to find the local minima of a function. For example, find the local minima of the function $y=(x+3)^2$ starting from the point x=2
5	Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset.Determine the number of clusters using the elbow method. Dataset link : https://www.kaggle.com/datasets/kyanyoga/sample-sales-data

6	**Mini-Project on Machine Learning** **1. Mini Project** - - Use the following dataset to analyze ups and downs in the market and predictfuture stock price returns based on Indian Market data from 2000 to 2020. Dataset Link: https://www.kaggle.com/datasets/sagara9595/stock-data 2. **Mini Project** - Build a machine learning model that predicts the type of people who survivedthe Titanic shipwreck using passenger data (i.e. name, age, gender, socio-economic class, etc.).Dataset Link: https://www.kaggle.com/competitions/titanic/data
GROUP C	**Based on Blockchain Technology (410243)**
1.	Installation of MetaMask and Create your own wallet using Metamask for crypto transactions.
2.	Write a smart contract on a test network, for Bank account of a customer for following operations: a. Deposit money b. Withdraw Money c. Show balance
3.	Write a program in solidity to create Student data. Use the following constructs: a. Structures b. Arrays c. Fallback Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values.
4.	Study spending Ether per transaction.
5.	Write a survey report on types of Blockchains and its real time use cases
6	**Mini-Project on Blockchain Technology** 1.Develop a Blockchain based application dApp (de-centralized app) for e- voting system. 2.Develop a Blockchain based application for transparent and genuine charity 3.Develop a Blockchain based application for health-related medical records 4.Develop a Blockchain based application for mental health

Subject Coordinator
(Yogesh Handge)

Head, Dept. of CE
(Dr. G. V. Kale)

Assignment No.	**1**
Title	Recursive and Iterative algorithms
PROBLEM STATEMENT/ DEFINITION	Write a non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.
Objectives	• Learn to implement procedures and to pass parameters. • Understand how to use recursion to define concepts. • Learn to implement recursive functions and handle a stack using low level instructions. • Learn how to write iterative programs • Analyze algorithm in tern of time and space Complexity
Software packages and hardware apparatus used	PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15";Color Monitor, Keyboard, Mouse with eclipse installed
References	http://en.wikipedia.org/wiki/Fibonacci_number http://www.ics.uci.edu/~eppstein/161/960109.html
STEPS	Refer details
Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

AssignmentNo.1

Title : Recursive an Iterative algorithms

Objectives:

- Learn to implement procedures and to pass parameters.
- Understand how to use recursion to define concepts.
- Learn to implement recursive functions and handle a stack using low level instructions.
- Learn how to write iterative program
- Analyze algorithm in tern of time and space Complexity

Theory:

Aim: Write recursive & iterative programme which computes the nth Fibonacci number, for appropriate values of n. Analyze behavior of the programme in their time and space complexity.

Recursive function

Simply put, a recursive function is one which calls itself. The typical example presented when recursion is first encountered is the factorial function. The factorial of n is defined in mathematics as the product of the integers from 1 to n .

$$n! = 1 \times 2 \times 3 \times \ldots \times (n-2) \times (n-1) \times n$$

For example:

$$\begin{aligned} 3! &= 1 \times 2 \times 3 \\ 4! &= 1 \times 2 \times 3 \times 4 \\ 5! &= 1 \times 2 \times 3 \times 4 \times 5 \end{aligned}$$

Factorials are useful in counting ordered outcomes. For example, consider a race run by five people. Assuming no ties, there are five possibilities as to who will cross the finish line first; there are subsequently four remaining possibilities for second place, three for third, two for second and finally only one possibility for last place. The total number of possible outcomes for the race is $5 \times 4 \times 3 \times 2 \times 1$, or $5!$.

A Recursive Factorial Function

Any iterative function can be implemented recursively and vice versa. The recursive function will always be slower due to the added overhead needed by function calls, but it is often times easier to state the solution of a problem recursively. The factorial function can be defined recursively by the following.

$$\begin{aligned} 5! &= 5 \times 4 \times 3 \times 2 \times 1 \\ 4! &= 4 \times 3 \times 2 \times 1 \\ 5! &= 5 \times 4! \\ n! &= n \times (n-1)! \end{aligned}$$

As is the case for every recursive definition, there must be some stopping point at which the function will no longer call itself. For factorial, this point is when zero is reached; by definition, 0!=1. This definition can be interpreted using the earlier race analogy to mean that there is only one way for a race with zero participants to end. The full recursive definition for factorial follows.

$$n! = \begin{cases} 1 & \text{if} \quad n = 0 \\ n \times (n-1)! & \text{if} \quad n > 0 \end{cases}$$

Add the function recursive Factorial, which implements this definition, to the program

```
1. int recursiveFactorial (int n)
2. {
3. int result;
4. if (n == 0)
5. result = 1;
6. else // n != 0
7. result = n * recursiveFactorial (n-1);
8. return result;
9. }
```

An Iterative Function

It is easy enough to write a function which uses a counting loop to multiply successive numbers together in order to obtain a factorial, which accepts a value parameter n and returns an integer; note how the work of the function is done by its loop.

```
1. int iterativeFactorial (int n)
2. {
3. int product = 1;
4. for (int i = 2; i <= n; i++)
5. product = product * i;
6. return product;
7. }
```

Fibonacci numbers

The Fibonacci numbers or Fibonacci series are the numbers in the following integer sequence: 0,1,1,2,3,5,8,13,21,…. .
By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two.
In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation Fn= Fn-1 + Fn-2

Algorithm

Input: Read n value
Output: Prints the nth Fibonacci term

Step1: Start
Step2: Read n value for computing nth term in Fibonacci series
Step3: call Fibonacci (n)
Step4: Print the nth term
Step5: End Fibonacci(n)

Algorithm: Recursive Fibonacci computes the nth Fibonacci number, for appropriate values of n.

The sequence is conventionally defined as follows:

$$Fib_n = \begin{cases} 1 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ Fib_{n-1} + Fib_{n-2} & \text{if } n > 1 \end{cases}$$

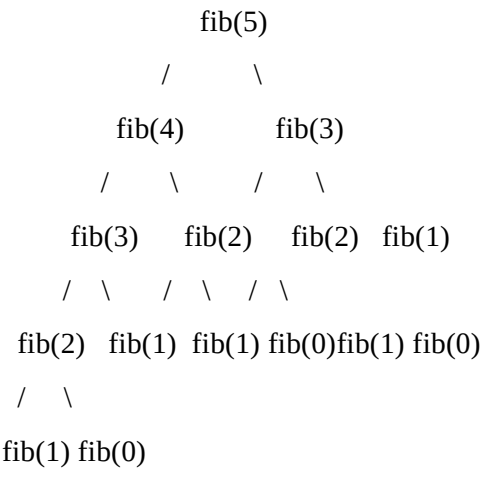
Step1: If n = 0 then go to step2 else go to step3
Step2: return 0
Step3: If n = 1 then go to step4
else go to step5
Step4: return 1
Step5: return(Fibonacci (n-1) + Fibonacci (n-2))

Analysis

Time Complexity: Exponential, as every function calls two other functions.

If the original recursion tree were to be implemented then this would have been the tree but now for n times the recursion function is called

Original tree for recursion



Space Complexity: O(n) if we consider the function call stack size, otherwise O(1).

An Iterative Fibonacci Function

In order to reduce the execution time for Fibonacci numbers, an iterative algorithm can be developed. The iterative factorial-for-n algorithm provides a good starting point. Time permitting, develop this algorithm and execute it with an extension to function main.

Step1: If n = 0 then go to step2 else go to step3
Step2: return 0
Step3: If n = 1 then go to step4
else go to step5
Step4: return 1
Step 5: for(i=3;i<=n;i++)

```
{ c=a+b
   a=b;
   b=c;
  return c;
 }
```

Time Complexity:O(n)
Extra Space: O(1)

Sample Input: n= 8 (Fibonacci Number Computation For which term you want to compute the Fibonacci number)

Observed Output:
Fibonacci Number of 9 is 21

Conclusion: We have successfully seen how recursive and iterative Fibonacci numbers are implemented and how analysis of same is done

FAQ

1. What is Fibonacci series?
2. What is the Logic of Fibonacci series?
3. Give example of Fibonacci series:
4. What is recursion?
5. What are the advantages of using functions?
6. What is data structure used in recusive fuction of Fibonacci series?
7. What is the complexity(space and time of recursive & non recursive Fibonacci numbers algorithm

AssignmentNo.	**2**
Title	To Implement Huffman Encoding using a greedy strategy.
PROBLEM STATEMENT/DEFINITION	Write a program to implement Huffman Encoding using a greedy strategy.
Objectives	Understand and implement the concept of Huffman encoding
Software packages and hardware apparatus used	Technology : Java/Python Ubuntu /Linux Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
References	1. http://en.wikipedia.org/wiki/Huffman_coding 2. https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/#:~:text=Steps%20to%20build%20Huffman%20Tree&text=Create%20a%20new%20internal%20node,heap%20contains%20only%20one%20node.
STEPS	1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root) 2. Extract two nodes with the minimum frequency from the min heap. 3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept, Architecture, Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

Concepts Related Theory

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.

The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.

Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is the prefix of codes assigned to a and b. If the compressed bit stream is 0001, the de-compressed output may be “cccd” or “ccb” or “acd” or “ab”.

There are mainly two major parts in Huffman Coding

1. Build a Huffman Tree from input characters.
2. Traverse the Huffman Tree and assign codes to characters.

Steps to build Huffman Tree

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

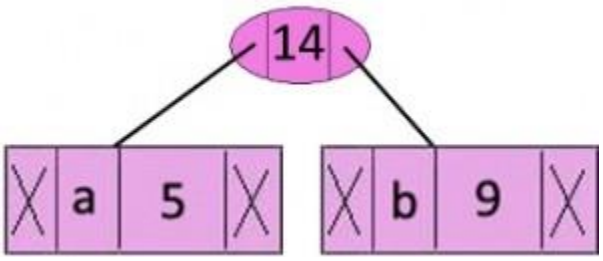
4. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
5. Extract two nodes with the minimum frequency from the min heap.

6. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
7. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.
   Let us understand the algorithm with an example:

character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

Step 1. Build a min heap that contains 6 nodes where each node represents root of a tree with single node.

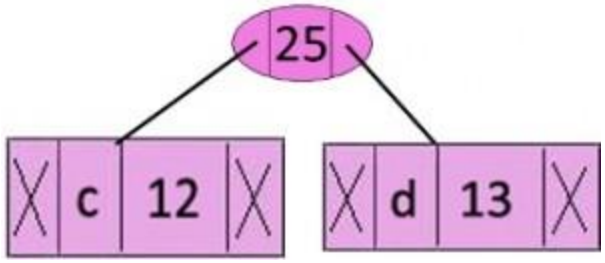
Step 2 Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 5 + 9 = 14.



Now min heap contains 5 nodes where 4 nodes are roots of trees with single element each, and one heap node is root of tree with 3 elements

character	Frequency
c	12
d	13
Internal Node	14
e	16
f	45

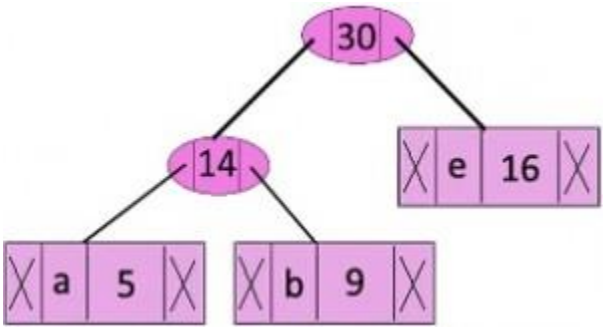
Step 3: Extract two minimum frequency nodes from heap. Add a new internal node with frequency 12 + 13 = 25



Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes

character	Frequency
Internal Node	14
e	16
Internal Node	25
f	45

Step 4: Extract two minimum frequency nodes. Add a new internal node with frequency 14 + 16 = 30



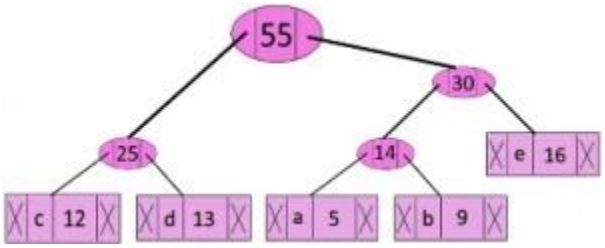
Now min heap contains 3 nodes.

character	Frequency
Internal Node	25

Internal Node 30

f 45

Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency 25 + 30 = 55



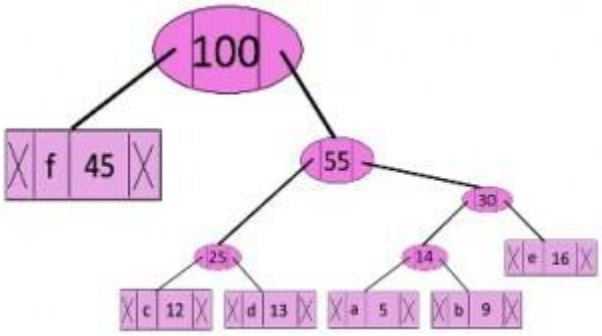
Now min heap contains 2 nodes.

character Frequency

f 45

Internal Node 55

Step 6: Extract two minimum frequency nodes. Add a new internal node with frequency 45 + 55 = 100



Now min heap contains only one node.

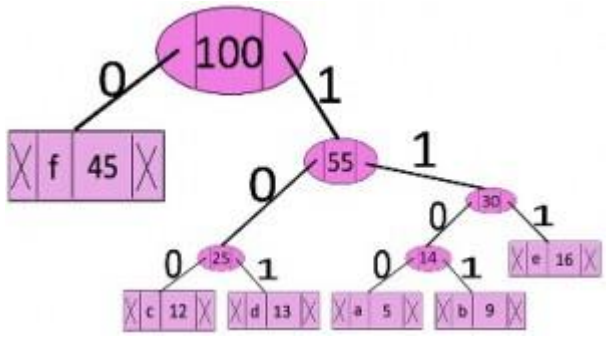
character Frequency

Internal Node 100

Since the heap contains only one node, the algorithm stops here.

Steps to print codes from Huffman Tree:

Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.



The codes are as follows:

character code-word

f 0

c 100

d 101

a 1100

b 1101

e 111

Pseudocode

```
# A Huffman Tree Node
class node:
        def __init__(self, freq, symbol, left=None, right=None):
```

```
        # frequency of symbol
        self.freq = freq

        # symbol name (character)
        self.symbol = symbol

        # node left of current node
        self.left = left

        # node right of current node
        self.right = right

        # tree direction (0/1)
        self.huff = ''

# utility function to print huffman
# codes for all symbols in the newly
# created Huffman tree


def printNodes(node, val=''):
    # huffman code for current node
    newVal = val + str(node.huff)

    # if node is not an edge node
    # then traverse inside it
    if(node.left):
        printNodes(node.left, newVal)
    if(node.right):
        printNodes(node.right, newVal)
```

```
                # if node is edge node then
                # display its huffman code
        if(not node.left and not node.right):
                print(f"{node.symbol} -> {newVal}")


# characters for huffman tree
chars = ['a', 'b', 'c', 'd', 'e', 'f']

# frequency of characters
freq = [ 5, 9, 12, 13, 16, 45]

# list containing unused nodes
nodes = []

# converting characters and frequencies
# into huffman tree nodes
for x in range(len(chars)):
        nodes.append(node(freq[x], chars[x]))

while len(nodes) > 1:
        # sort all the nodes in ascending order
        # based on their frequency
        nodes = sorted(nodes, key=lambda x: x.freq)

        # pick 2 smallest nodes
        left = nodes[0]
        right = nodes[1]

        # assign directional value to these nodes
        left.huff = 0
```

```
        right.huff = 1

        # combine the 2 smallest nodes to create
        # new node as their parent
        newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)

        # remove the 2 nodes and add their
        # parent as new node among others
        nodes.remove(left)
        nodes.remove(right)
        nodes.append(newNode)

# Huffman Tree is ready!
printNo**des(nodes[0])**
```

Output:

f: 0

c: 100

d: 101

a: 1100

b: 1101

e: 111

Time complexity*:* O(nlogn) where n is the number of unique characters. If there are n nodes, extractMin() is called 2*(n – 1) times. extractMin() takes O(logn) time as it calles minHeapify(). So, overall complexity is O(nlogn).

If the input array is sorted, there exists a linear time algorithm. We will soon be discussing in our next post.

Applications of Huffman Coding:

1. They are used for transmitting fax and text.
2. They are used by conventional compression formats like PKZIP, GZIP, etc.
3. Multimedia codecs like JPEG, PNG, and MP3 use Huffman encoding(to be more precise the prefix codes).

It is useful in cases where there is a series of frequently occurring characters.

FAQ's

Q-1.Which of the following is true about Huffman Coding?

(A)Huffman coding may become lossy in some cases

(B)Huffman Codes may not be optimal lossless codes in some cases

(C)In Huffman coding, no code is prefix of any other code.

(D) All of the above

Q-2. How many bits may be required for encoding the message 'mississippi'?

Q-3 How can you generate Huffman codes using greedy method?

Q-4 What is the greedy choice in Huffman coding?

Q-5 Is Huffman algorithm A greedy algorithm?

Q-6 Is Huffman coding lossy or lossless?

Q-7 Which technique is applied to compress the given message using greedy?

Q-8 What is the running time of Huffman encoding algorithm?

Q-9 What are the various applications of Huffman coding?

Q-10 What is the advantage of Huffman code over variable length code?

AssignmentNo.	**3**
Title	Knapsack problem using dynamic programming or branch and bound strategy.
Problem Statement /Definition	a. To understand 0-1 knapsack problem in design and analysis algorithm b. Implement dynamic programming or branch and bound strategy as optimization problem solution of knapsack problem
Objectives	• Understand & implement the divide and conquer method. • Understand optimal solution using dynamic and branch - bound algorithms
Software packages and hardware apparatus used	**Programming tool recommended -** Eclipse IDE **Programming Language**-C++/Python/Java **PC with the configuration-** as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
References	**e-Books :http://103.47.12.35/bitstream/handle/1/2884/PS76%20Algorithms_%20Design%20Techniques%20and%20Analysis%20%28%20PDFDrive%20%29.pdf?sequence=1&isAllowed=y**
Steps	Refer to details
Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory- a. Introduction knapsack problem,its types b.0/1 Knapsack problem solution by using methods - dynamic programming- overlapping problem using bottom up solution(tabulation) OR top down solution (memoization) ,time and space complexity. OR c. use another solution for 0/1 Knapsack problem-branch and bound method, time and space complexity • Program Listing • Output • Conclusion

AssignmentNo.3

Title : Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

Objectives: a) To understand the 0-1 Knapsack Problem have optimization problems. b) Apply dynamic programming solution c)Apply branch and bound strategy solution.

Theory: the knapsack problem in the analysis and design of algorithms.

Introduction

Knapsack Problem may be of 2 types:

- [A] 0/1 Knapsack problem
- [B] Fractional Knapsack problem

a. **0/1 Knapsack problem**

In this problem, either a whole item is selected(1) or the whole item not to be selected(0).Here, the thief can't carry a fraction of the item.In the LPP(Linear programming problem) form, it can be described as:

Formula

Maximum profit = $\sum_{i=1}^{n} \binom{n}{i} \text{Pi Xi}$

Constraint subject to = $\sum_{i=1}^{n} \binom{n}{i} \text{Wi Xi} \leq \text{m}$

Where Xi = either 0 or 1

Xi = selection value of item i

b. **Fractional Knapsack problem**

In this problem, a whole item can be selected (1) or a whole item can't be selected (0) or a fraction of item can also be selected (between 0 or 1).The fractional knapsack problem is solved by the Greedy approach.In the LPP(Linear programming problem) form, it can be described as:

Maximum profit = $\sum_{i=1}^{n} \binom{n}{i} \text{Pi Xi}$

Constraint subject to = $\sum_{i=1}^{n} \binom{n}{i} \text{Wi Xi} \leq \text{m}$

Where $0 \leq \text{Xi} \leq 1$

Xi = selection value of item i

- **Advantage of optimization problem solution-**

The knapsack problem also tests how well we approach combinatorial optimization problems.There are many practical applications in the workplace, as all combinatorial optimization problems seek maximum benefit within constraints.

For example, combinatorial optimization is used in solutions like:

- ❖ Determine the best programs to run on a limited resource cloud system
- ❖ Optimize water distribution across a fixed pipe network
- ❖ Automatically plan the best package delivery route
- ❖ Optimize the company's supply chain

What is 0/1 Knapsack Problem-

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

- ➔ In other words, given two integer arrays val[0..n-1] and wt[0..n-1] which represent values and weights associated with n items respectively.
- ➔ Also given an integer W which represents knapsack capacity, find out the maximum value subset of val[] such that the sum of the weights of this subset is smaller than or equal to W.
- ➔ You cannot break an item, either pick the complete item or don't pick it (0-1 property).

You are given the following-

- A knapsack (kind of shoulder bag) with limited weight capacity.
- Few items each having some weight and value.

The problem states-

Which items should be placed into the knapsack such that-

- The value or profit obtained by putting the items into the knapsack is maximum.
- And the weight limit of the knapsack does not exceed.

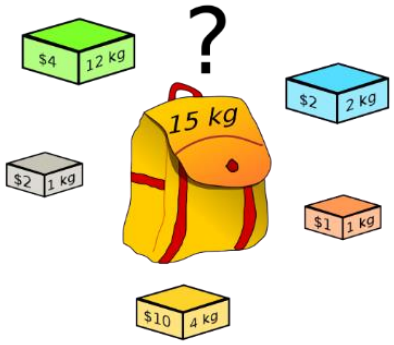


Figure 1 Knapsack Problem

0/1 Knapsack problem can be solved by using these two following methods:

Dynamic Programming and Branch and Bound Both methods are used for solving optimization problems. Optimization methods find the best solution out of a pool of feasible solutions. The aim of optimization methods is to minimize or maximize the given cost function.

1. Dynamic Programming method
2. Branch & Bound

1. 0/1 Knapsack Problem Using Dynamic Programming Method-

Dynamic Programming is also used in optimization problems. Like divide-and-conquer methods, Dynamic Programming solves problems by combining the solutions of subproblems. Moreover, the Dynamic Programming algorithm solves each sub-problem just once and then saves its answer in a table, thereby avoiding the work of re-computing the answer every time.

Two main properties of a problem suggest that the given problem can be solved using Dynamic Programming. **These properties are overlapping subproblems and optimal substructure**.

a. Overlapping Subproblems

Similar to the Divide-and-Conquer approach, Dynamic Programming also combines solutions to sub-problems. It is mainly used where the solution of one sub-problem is needed repeatedly. The computed solutions are stored in a table, so that these don't have to be re-computed. Hence, this technique is needed where overlapping sub-problem exist.

For example- Binary Search does not have overlapping sub-problem. Whereas recursive programs of Fibonacci numbers have many overlapping subproblems.

There are following two different ways to store the values so that these values can be reused:

i) Memoization (Top Down)

The memoized program for a problem is similar to the recursive version with a small modification that looks into a lookup table before computing solutions. We initialize a lookup array with all initial values as NIL. Whenever we need the solution to a subproblem, we first look into the lookup table. If the precomputed value is there then we return that value, otherwise, we calculate the value and put the result in the lookup table so that it can be reused later.

```
/* C/C++ program for Memoized version for nth Fibonacci number */
#include<stdio.h>
#include<time.h>
#define NIL -1
#define MAX 100
int lookup[MAX];
/* Function to initialize NIL values in lookup table */
void _initialize()
{
int i;
for (i = 0; i < MAX; i++)
lookup[i] = NIL;
```

```
}
/* function for nth Fibonacci number */
int fib(int n)
{
if (lookup[n] == NIL)
{
if (n <= 1)
        lookup[n] = n;
else
        lookup[n] = fib(n-1) + fib(n-2);
}
return lookup[n];
}
int main ()
{
int n = 40;
clock_t begin, end;
double time_spent;
_initialize();
begin = clock();
printf("Fibonacci number is %d \n", fib(n));
end = clock();
time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
printf("\nTime Taken %lf", time_spent);
return 0;
}
```

ii) Tabulation (Bottom Up): The tabulated program for a given problem builds a table in a bottom-up fashion and returns the last entry from the table. For example, for the same Fibonacci number, we first calculate fib(0) then fib(1) then fib(2) then fib(3), and so on. So literally, we are building the solutions to subproblems bottom-up.

```
/* C program for Tabulated version */
#include<stdio.h>
#include<time.h>
int fib(int n)
```

```
{
int f[n+1];
int i;
double time_spent ;
f[0] = 0; f[1] = 1;
for (i = 2; i <= n; i++)
f[i] = f[i-1] + f[i-2];
return f[n];
}
int main ()
{
int n = 40;
clock_t begin, end;
double time_spent;
begin = clock(); // Time before calculating Fib number
printf("Fibonacci number is %d \n", fib(n));
end = clock();  // Time before calculating Fib number
time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
printf("\nTime Taken %lf ", time_spent);
return 0;
}
```

b. Optimal Substructure

A given problem has Optimal Substructure Property, if the optimal solution of the given problem can be obtained using optimal solutions of its sub-problems.

For example The standard All Pair Shortest Path algorithms like Floyd-Warshall and Bellman-Ford are typical examples of Dynamic Programming.

How to Solve Knapsack Problem using Dynamic Programming with Example

The basic idea of Knapsack dynamic programming is to use a table to store the solutions of solved subproblems. If you face a subproblem again, you just need to take the solution to the table without having to solve it again. Therefore, the algorithms designed by dynamic programming are very effective.

To solve a problem by dynamic programming, you need to do the following tasks:

1. Find solutions to the smallest subproblems.
2. Find out the formula (or rule) to build a solution of a subproblem through solutions of even smallest subproblems.
3. Create a table that stores the solutions of subproblems. Then calculate the solution of the subproblem according to the found formula and save it to the table.
4. From the solved subproblems, you find the solution of the original problem.

Steps to follow -Consider-

- Knapsack weight capacity = w
- Number of items each having some weight and value = n

0/1 knapsack problem is solved using dynamic programming in the following steps-

Step-01:

- Draw a table say 'T' with (n+1) number of rows and (w+1) number of columns.
- Fill all the boxes of 0^{th} row and 0^{th} column with zeros as shown-

	0	1	2	3	.....	W
0	0	0	0	0	.....	0
1	0					
2	0					
	.....					
n	0					

T-Table

Step-02:

Start filling the table row wise top to bottom from left to right.

Use the following formula-

$$T(i, j) = \max \{ T(i-1, j), value_i + T(i-1, j - weight_i) \}$$

Here, T(i , j) = maximum value of the selected items if we can take items 1 to i and have weight restrictions of j.

- This step leads to completely filling the table.
- Then, the value of the last box represents the maximum possible value that can be put into the knapsack.

Step-03:

To identify the items that must be put into the knapsack to obtain that maximum profit,

- Consider the last column of the table.
- Start scanning the entries from bottom to top.
- On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.
- After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.

Time and Space Complexity-

- Each entry of the table requires constant time $\theta(1)$ for its computation.
- It takes $\theta(nw)$ time to fill $(n+1)(w+1)$ table entries.
- It takes $\theta(n)$ time for tracing the solution since the tracing process traces the n rows.
- Thus, overall $\theta(nw)$ time and $\theta(nw)$ space is taken to solve the 0/1 knapsack problem using dynamic programming.

PRACTICE PROBLEM BASED ON 0/1 KNAPSACK PROBLEM-

Problem-For the given set of items and knapsack capacity = 5 kg, find the optimal solution for the 0/1 knapsack problem making use of dynamic programming approach.

Item	Weight	Value
1	2	3
2	3	4
3	4	5
4	5	6

OR

Find the optimal solution for the 0/1 knapsack problem making use of a dynamic programming approach. Consider-

n = 4

w = 5 kg

(w1, w2, w3, w4) = (2, 3, 4, 5)

(b1, b2, b3, b4) = (3, 4, 5, 6)

OR

A thief enters a house to rob it. He can carry a maximal weight of 5 kg into his bag. There are 4 items in the house with the following weights and values. What items should the thief take if he either takes the item completely or leaves it completely?

Item	Weight (kg)	Value ($)
Mirror	2	3
Silver nugget	3	4
Painting	4	5
Vase	5	6

Solution- Given-

- Knapsack capacity (w) = 5 kg
- Number of items (n) = 4

Step-01:

- Draw a table say 'T' with (n+1) = 4 + 1 = 5 number of rows and (w+1) = 5 + 1 = 6 number of columns.
- Fill all the boxes of 0^{th} row and 0^{th} column with 0.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

T-Table

Step-02:

Start filling the table row wise top to bottom from left to right using the formula-

$$T(i, j) = \max \{ T(i-1, j), value_i + T(i-1, j - weight_i) \}$$

Finding T(1,1)- We have,

- i = 1
- j = 1
- $(value)_i = (value)_1 = 3$
- $(weight)_i = (weight)_1 = 2$

Substituting the values, we get-

T(1,1) = max { T(1-1 , 1) , 3 + T(1-1 , 1-2) }

T(1,1) = max { T(0,1) , 3 + T(0,-1) }

T(1,1) = T(0,1) { Ignore T(0,-1) }

T(1,1) = 0

Finding T(1,2)-We have,

- i = 1
- j = 2
- $(value)_i = (value)_1 = 3$
- $(weight)_i = (weight)_1 = 2$

Substituting the values, we get-

T(1,2) = max { T(1-1 , 2) , 3 + T(1-1 , 2-2) }

T(1,2) = max { T(0,2) , 3 + T(0,0) }

T(1,2) = max {0 , 3+0}

T(1,2) = 3

Finding T(1,3)-We have,

- i = 1
- j = 3
- $(value)_i = (value)_1 = 3$
- $(weight)_i = (weight)_1 = 2$

Substituting the values, we get-

T(1,3) = max { T(1-1 , 3) , 3 + T(1-1 , 3-2) }

T(1,3) = max { T(0,3) , 3 + T(0,1) }

T(1,3) = max {0 , 3+0}

T(1,3) = 3

Finding T(1,4)-We have,

- i = 1
- j = 4
- $(value)_i = (value)_1 = 3$
- $(weight)_i = (weight)_1 = 2$

Substituting the values, we get-

T(1,4) = max { T(1-1 , 4) , 3 + T(1-1 , 4-2) }

T(1,4) = max { T(0,4) , 3 + T(0,2) }

T(1,4) = max {0 , 3+0}

T(1,4) = 3

Finding T(1,5)-We have,

- i = 1
- j = 5
- $(value)_i = (value)_1 = 3$
- $(weight)_i = (weight)_1 = 2$

Substituting the values, we get-

T(1,5) = max { T(1-1 , 5) , 3 + T(1-1 , 5-2) }

T(1,5) = max { T(0,5) , 3 + T(0,3) }

T(1,5) = max {0 , 3+0}

T(1,5) = 3

Finding T(2,1)- We have,

- i = 2
- j = 1
- $(value)_i = (value)_2 = 4$
- $(weight)_i = (weight)_2 = 3$

Substituting the values, we get-

T(2,1) = max { T(2-1 , 1) , 4 + T(2-1 , 1-3) }

T(2,1) = max { T(1,1) , 4 + T(1,-2) }

T(2,1) = T(1,1) { Ignore T(1,-2) }

T(2,1) = 0

Finding T(2,2)- We have,

- i = 2
- j = 2
- $(value)_i = (value)_2 = 4$
- $(weight)_i = (weight)_2 = 3$

Substituting the values, we get-

T(2,2) = max { T(2-1 , 2) , 4 + T(2-1 , 2-3) }

T(2,2) = max { T(1,2) , 4 + T(1,-1) }

T(2,2) = T(1,2) { Ignore T(1,-1) }

T(2,2) = 3

Finding T(2,3)-We have,

- i = 2

- j = 3
- $(value)_i = (value)_2 = 4$
- $(weight)_i = (weight)_2 = 3$

Substituting the values, we get-

T(2,3) = max { T(2-1 , 3) , 4 + T(2-1 , 3-3) }

T(2,3) = max { T(1,3) , 4 + T(1,0) }

T(2,3) = max { 3 , 4+0 }

T(2,3) = 4

Finding T(2,4)-We have,

- i = 2
- j = 4
- $(value)_i = (value)_2 = 4$
- $(weight)_i = (weight)_2 = 3$

Substituting the values, we get-

T(2,4) = max { T(2-1 , 4) , 4 + T(2-1 , 4-3) }

T(2,4) = max { T(1,4) , 4 + T(1,1) }

T(2,4) = max { 3 , 4+0 }

T(2,4) = 4

Finding T(2,5)-We have,

- i = 2
- j = 5
- $(value)_i = (value)_2 = 4$
- $(weight)_i = (weight)_2 = 3$

Substituting the values, we get-

T(2,5) = max { T(2-1 , 5) , 4 + T(2-1 , 5-3) }

T(2,5) = max { T(1,5) , 4 + T(1,2) }

T(2,5) = max { 3 , 4+3 }

T(2,5) = 7

Similarly, compute all the entries.

After all the entries are computed and filled in the table, we get the following table-

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

T-Table

- The last entry represents the maximum possible value that can be put into the knapsack.
- So, maximum possible value that can be put into the knapsack = 7.

2. 0/1 Knapsack Problem Using Branch and Bound Method

Used to find optimal solution to many optimization problems, especially in discrete and combinatorial optimization.Systematic enumeration of all candidate solutions, discarding large subsets of fruitless candidates by using upper and lower estimated bounds of quantity being optimized.

Features :

1. Constructs the solution in the form of a tree.
2. Only solves promising instances from the set of instances at any given point.
3. Needs to compute and apply a bounding function at each node.
4. The solution sequence is implicit, a leaf node of the tree is the final solution.
5. Ex.Knapsack problem

Branch and Bound Terminology

i. Live Node

A node that has been generated and all of whose children have not yet been generated is called a live node. The live node whose children are currently being generated is called the E-node.

ii. E-node

A live node whose children are currently being explored. In other words, an E-node is a node currently being expanded.

iii. Dead node

The dead node is a generated node that is not to be expended further or all of whose children have been generated.

iv. Bounding functions

Bounding functions are used to kill live nodes without generating all their children. This is done carefully so that at the conclusion of the process at least one answer node is always generated or all answer nodes are generated if the problem requires finding all solutions.

Complete Algorithm

1. sort all items in decreasing order of ratio of value per unit weight so that an upper bound can be computed using greedy approach
2. initialize maximum profit ,maxprofit = 0
3. create an empty queue , Q.
4. create a dummy node of the decision tree and enqueue it to Q. Profit and weight of the dummy node is 0.

   Extract an item from Q. Let the extracted item be u.

   Compute profit of next level node. If the profit is more than maxProfit, then update maxProfit.

   Compute bound of next level node. If bound is more than maxProfit, then add the next level node to Q.

   Consider the case when the next level node is not considered as part of the solution and add a node to queue with level as next, but weight and profit without considering next level nodes.

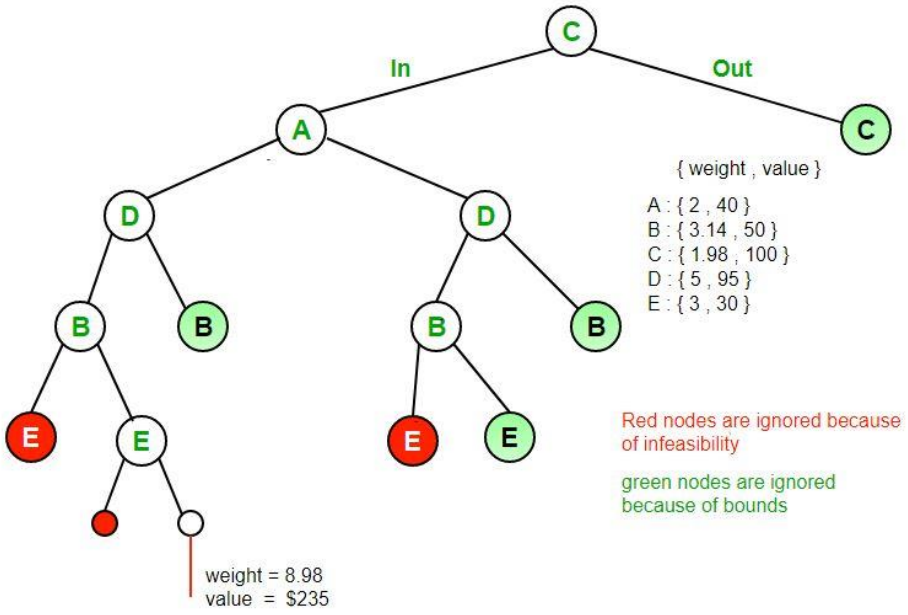
Input:

```
// First thing in every pair is weight of item
// and second thing is value of item
Item arr[] = {{2, 40}, {3.14, 50}, {1.98, 100},
              {5, 95}, {3, 30}};
Knapsack Capacity W = 10
```

Output:

The maximum possible profit = 235

Below diagram shows illustration. Items are considered sorted by value/weight.



Complexity Analysis -

- **time complexity: O (2^n)**

  As there are redundant subproblems

- **Space complexity: O (1) + O(N)**

  No extra data structure has been used for storing values but O(N) auxiliary stack space has been used for recursive stack.

Conclusion: Thus we have implemented a knapsack problem using dynamic programming or branch and bound strategy.

FAQ?

1. Compare: dynamic programming Vs Branch bound method.
2. Which algorithm is best for knapsack problems?
3. What exactly is the goal of the knapsack problem?
4. What is the Time Complexity of 0/1 Knapsack Problem?
5. What are the 2 categories of knapsack problems?
6. Is knapsack divide and conquer?
7. What common problems are solved and not solved with dynamic programming?
8. What are 2 things required in order to successfully use the dynamic programming technique?
9. Which type of complexity is often seen in dynamic programming algorithms?
10. Which problem can be solved by branch and bound?
11. Which data structure is most suitable for implementing the best first branch and bound strategy?

AssignmentNo.	4
Title	Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen
Problem Statement /Definition	c. To understand n-Queens matrix problem in design and analysis algorithm d. Implement backtracking strategy to place remaining Queens to generate the final n-queen
Objectives	• Understand & implement a backtracking algorithm. • Understand & implement N -Queen problem using backtracking w.r.t time complexity
Software packages and hardware apparatus used	**Programming tool recommended -** Eclipse IDE **Programming Language**-C++/Python/Java **PC with the configuration-** as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
References	**e-Books :http://103.47.12.35/bitstream/handle/1/2884/PS76%20Algorithms_%20Design%20Techniques%20and%20Analysis%20%28%20PDFDrive%20%29.pdf?sequence=1&isAllowed=y**
Steps	Refer to details
Instructions for writing journal	• Date • Title

	• Problem Definition • Learning Objective • Learning Outcome • Theory- Define backtracking, when to use backtracking algorithm, example, define N Queens Problem with example, time complexity of n queen problem. • Program Listing • Output • Conclusion

AssignmentNo.4

Title : Write a program to design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen

Objectives: a**)** To understand the N-Queens matrix. b) Apply a backtracking solution to generate the final n-queen .

Theory: Design n-Queens matrix in the analysis and design of algorithms.

Introduction-

What is backtracking?

Think about the problems like finding a path in a maze puzzle, assembling lego pieces, sudoku, etc. In all these problems, backtracking is the natural approach to solve them because all these problems require one thing - if a path is not leading you to the correct solution, come back and choose a different path.

Thus, we start with a sub-solution of a problem (which may or may not lead to the correct solution) and check if we can proceed further with this sub-solution or not. If not, then we just change this sub-solution. So, the steps involved are

- Start with a sub-solution.
- Check if this sub-solution will lead to a solution or not.
- If not, then change the sub-solution and continue again.

Take note that even tough backtracking solves the problem but yet it doesn't always give us a great running time.

Definition -Backtracking is an algorithmic technique where the goal is to get all solutions to a problem using the brute force approach. It consists of building a set of all the solutions incrementally. Since a problem would have constraints, the solutions that fail to satisfy them will be removed.

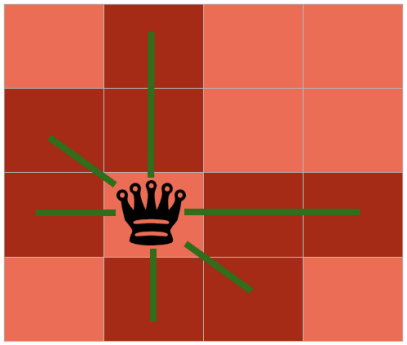
features -

1.It uses recursive calling to find a solution set by building a solution step by step, increasing levels with time.

2.A backtracking algorithm uses the depth-first search method.

N Queens Problem

N queens problem is one of the most common examples of backtracking. Our goal is to arrange N queens on an NxN chessboard such that no queen can strike down any other queen.Following figure shows a queen can attack horizontally, vertically, or diagonally.



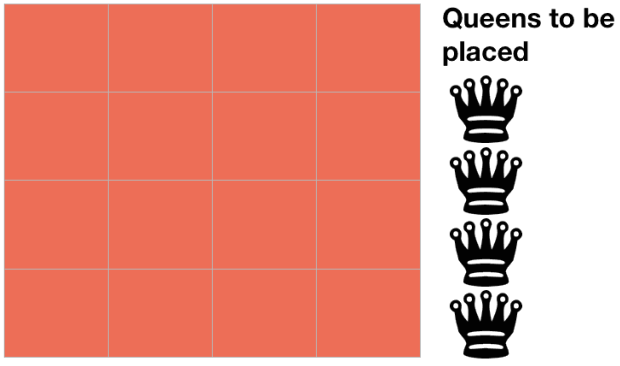
Cells attacked by the queen

Follow the steps -
Step 1-

So, we start by placing the first queen anywhere arbitrarily and then place the next queen in any of the safe places. We continue this process until the number of unplaced queens becomes zero (a solution is found) or no safe place is left. If no safe place is left, then we change the position of the previously placed queen.

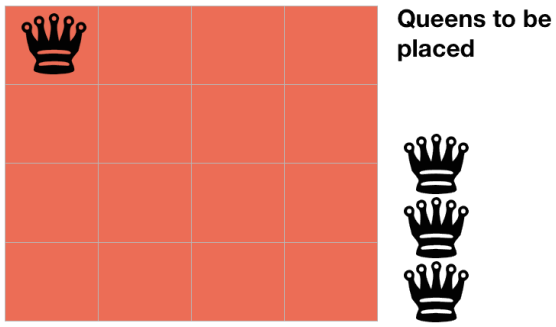
Step 2-

a. Let's test this algorithm on a 4x4 chessboard shown in the following figure.(Using Backtracking to Solve N Queens)

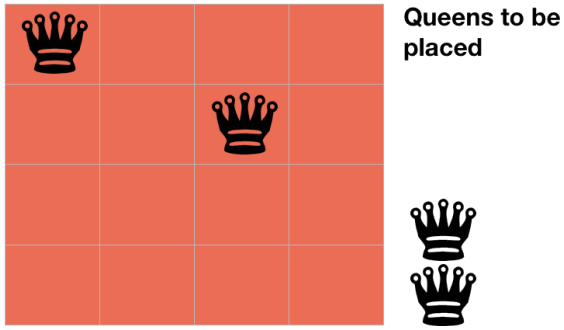


4x4 Chessboard

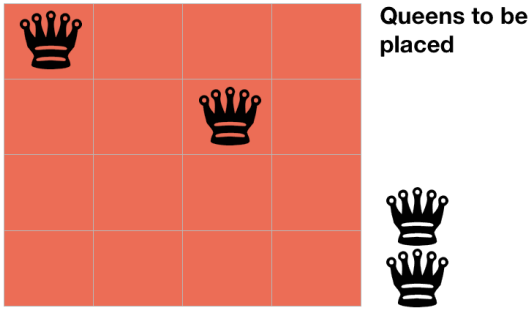
b. The above picture shows a 4x4 chessboard and we have to place 4 queens on it. So, we will start by placing the first queen in the first row as shown in the following figure.



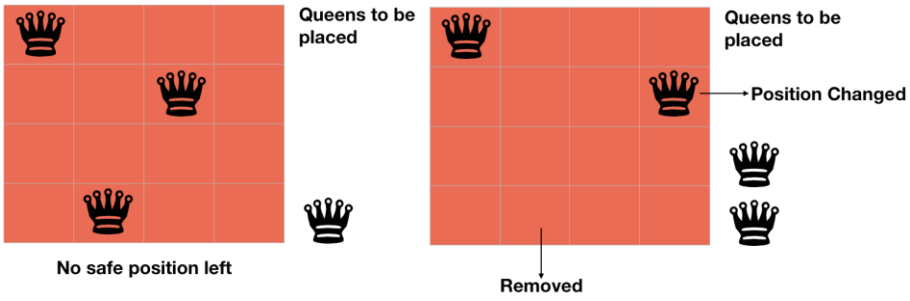
c. Now, the second step is to place the second queen in a safe position. Also, we can't place the queen in the first row, so we will try putting the queen in the second row this time.



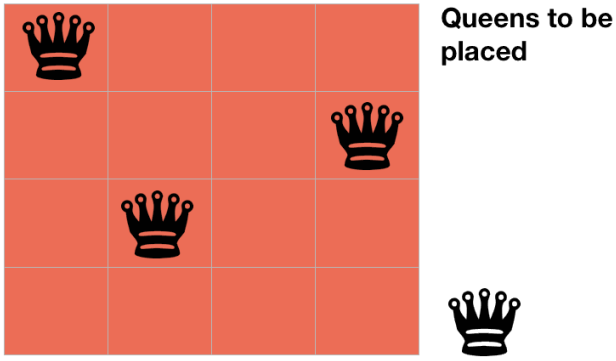
d. Let's place the third queen in a safe position, somewhere in the third row.



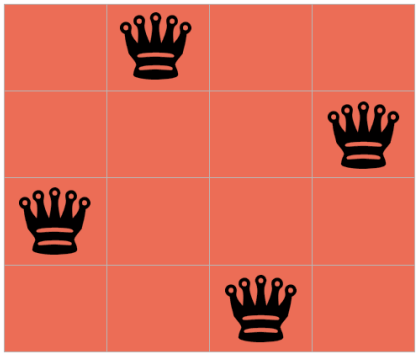
e. Now, we can see that there is no safe place where we can put the last queen. So, we will just change the position of the previous queen i.e., backtrack and change the previous decision.Also, there is no other position where we can place the third queen, so we will go back one more step and change the position of the second queen.



f. And now we will place the third queen again in a safe position other than the previously placed position in the third row.

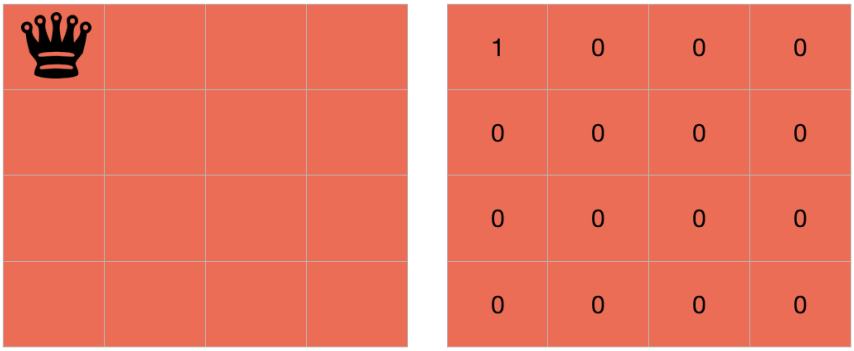


g. We will continue this process and finally, we will get the solution as shown below.

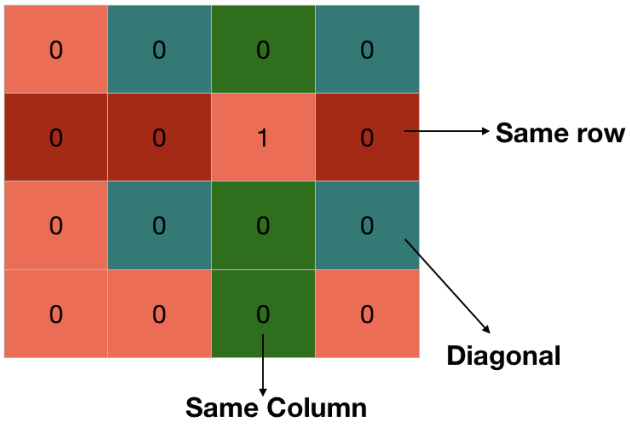


Step 3- Code for N Queens

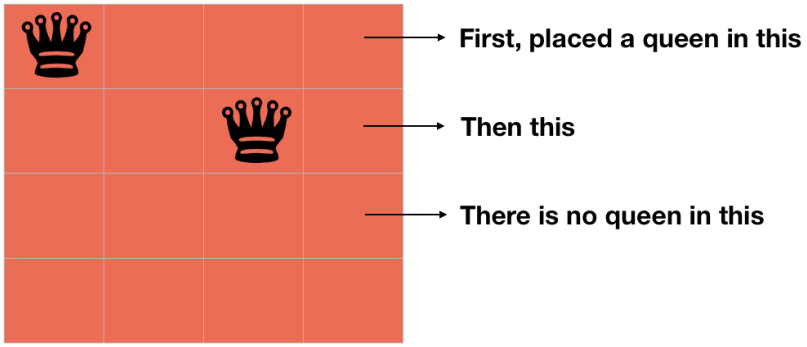
- A function to check if a place is safe to put a queen or not. We need to check if a cell (i, j) is under attack or not. For that, we will pass these two in our function along with the chessboard and its size - IS-ATTACK(i, j, board, N).
- If there is a queen in a cell of the chessboard, then its value will be 1, otherwise, 0.



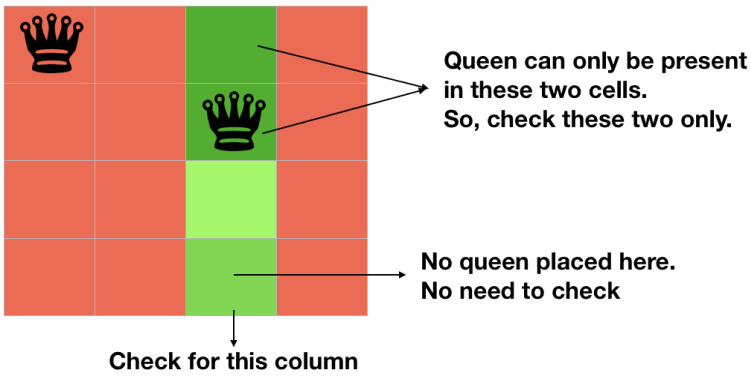
- The cell (i,j) will be under attack in three conditions - if there is any other queen in row i, if there is any other queen in column j or if there is any queen in the diagonals.



- We are already proceeding row-wise, so we know that all the rows above the current row(i) are filled but not the current row and thus, there is no need to check for row i.

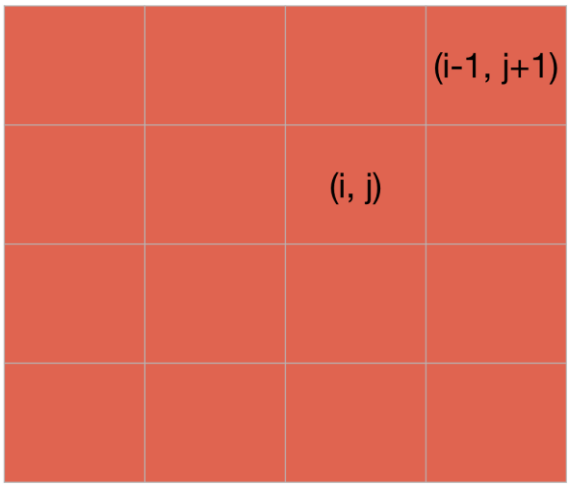


- We can check for the column j by changing k from 1 to i-1 on board[k][j] because only the rows from 1 to i-1 are filled.




```
  for k in 1 to i-1

  if board[k][j]==1

    return TRUE
```

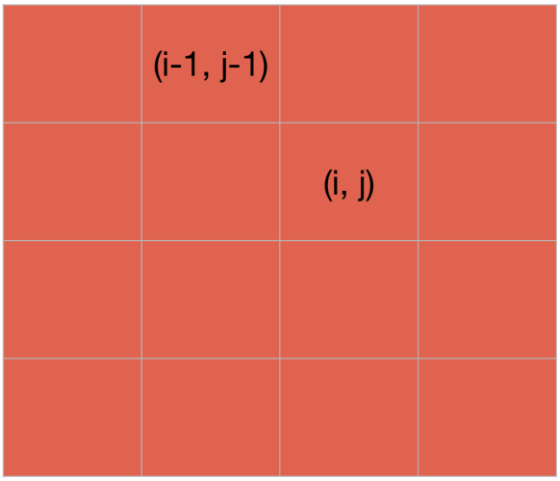
- Now, we need to check for the diagonal. We know that all the rows below the row i are empty, so we need to check only for the diagonal elements above the row i. If we are on the cell (i, j), then decreasing the value of i and increasing the value of j will make us traverse over the diagonal on the right side, above the row i.




```
k = i-1
l = j+1

while k>=1 and l<=N

      if board[k][l] == 1

 return TRUE
```

```
    k=k-1
    l=l+1
```

- Also if we reduce both the values of i and j of cell (i, j) by 1, we will traverse over the left diagonal, above the row i.




```
k = i-1
l = j-1

while k>=1 and l>=1

  if board[k][l] == 1

return TRUE

  k=k-1

  l=l-1
```

- At last, we will return false as it will be returned true if it is not returned by the above statements and the cell (i,j) is safe.

Step 4- Now involving backtracking to solve the N Queen problem.Our function will take the row, number of queens, size of the board and the board itself **.**

`N-QUEEN(row, n, N, board)`.

a. If the number of queens is 0, then we have already placed all the queens**.**

```
if n==0

return TRUE
```

b. Otherwise, we will iterate over each cell of the board in the row passed to the function and for each cell, we will check if we can place the queen in that cell or not. We can't place the queen in a cell if it is under attack.

```
for j in 1 to N

  if !IS-ATTACK(row, j, board, N)

    board[row][j] = 1
```

c. After placing the queen in the cell, we will check if we are able to place the next queen with this arrangement or not. If not, then we will choose a different position for the current queen.

```
for j in 1 to N

  ...
```

```
 if N-QUEEN(row+1, n-1, N, board)
 return TRUE
board[row][j] = 0
if N-QUEEN(row+1, n-1, N, board)
```

d. We are placing the rest of the queens with the current arrangement. Also, since all the rows up to 'row' are occupied, we will start from 'row+1'. If this returns true, then we are successful in placing all the queen,

e. If not, then we have to change the position of our current queen. So, we are leaving the current cell `board[row][j] = 0` and then iteration will find another place for the queen and this is backtracking.

f. Take a note that we have already covered the base case `if n==0 → return TRUE` It means when all queens will be placed correctly, then `N-QUEEN(row, 0, N, board)` will be called and this will return true.

g. At last, if true is not returned, then we didn't find any way, so we will return false.

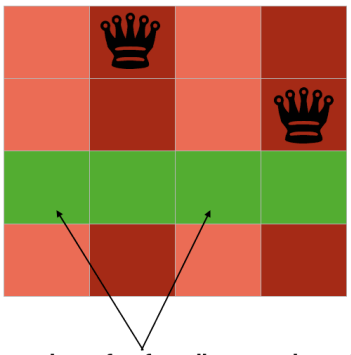
```
N-QUEEN(row, n, N, board)
…
return FALSE
```

- **sample code-**

```
N-QUEEN(row, n, N, board)
 if n==0
 return TRUE
 for j in 1 to N
 if !IS-ATTACK(row, j, board, N)
 board[row][j] = 1
 if N-QUEEN(row+1, n-1, N, board)
  return TRUE
  board[row][j] = 0 //backtracking, changing current decision
  return FALSE
```

Analysis of N Queens Problem

- The for loop in the N-QUEEN function is running from 1 to N (N, not n. N is fixed and n is the size of the problem i.e., the number of queens left) but the recursive call of `N-QUEEN(row+1, n-1, N, board)` (T(n−1)T(n−1)) is not going to run N times because it will run only for the safe cells. Since we have started by filling up the rows, there won't be more than n (number of queens left) safe cells in the row in any case.



Maximum number of safe cells = number of queens left

So, this part is going to take $n * T(n-1)$ time.

Also, the for loop is making N calls to the function IS-ATTACK and the function has a $O(N-n)$ worst case running time.

Since $(N-n) \leq N$, therefore, $O(N-n) = O(N)$.

Thus,

$$T(n) = O(N^2) + n * T(n-1)$$

Replacing $T(n-1)$ with $O(N^2) + (n-1)T(n-2)$,

$$T(n) = O(N^2) + n * \left(O(N^2) + (n-1)T(n-2)\right)$$

$$= O(N^2) + nO(N^2) + n(n-1)T(n-2)$$

Replacing $T(n-2)$,

$$T(n) = O(N^2) + nO(N^2) + n(n-1)\left(O(N^2) + (n-2)T(n-3)\right)$$

$$= O(N^2) + nO(N^2) + n(n-1)O(N^2) + n(n-1)(n-2)T(n-3)$$

Similarly,

$$T(n) = O(N^2)\left(1 + n + n(n-1) + n(n-2) + \ldots\right) + n * (n-1) * (n-2) * (n-3) * (n-4) * \ldots * T(0)$$

$$T(n) = O(N^2)\left(O((n-2)!)\right) + n * (n-1) * (n-2) * (n-3) * \ldots * T(0)$$

$$= O(N^2)\left(O((n-2)!)\right) + O(n!)$$

The above expression is dependent upon both the size of the board (N) and the number of queens (n). One can think that the term $O(N^2)\left(O((n-2)!)\right)$ will dominate if N is large enough but this is not going to happen.

Think about placing 1 queen on a 4x4 chessboard. Even if the size of the board (N) is quite greater than the number of queen (n), the algorithm will just find a place for the queen and then terminate (`if n==0 → return TRUE`). So it is not going to depend on N and thus, the running time will be $O(n!)$.

Another case is when the term $O(n!)$ will dominate, i.e., when the number of queens is larger than N, this will happen when there won't be any solution. In this case, the algorithm will take $O(n!)$ time.

In our case, the number of queens is also equal to N. In this case, we can write $O(N^2)\,(O((n-2)!))$ as $(O((n-2)! * n * n))$ which is just $\frac{n}{n-1}$ times larger than $n!$ as $\left(\frac{(n-2)!*n*n}{n*(n-1)*(n-2)!} = \frac{n}{n-1}\right)$. According to the definition of Big-Oh, we can choose the value of constant $c > \frac{n}{n-1}$

$$(f(n) = O(g(n)),\ if\ f(n) \leq c.\,g(n))$$

and thus, say $O(N^2)\,(O((n-2)!)) = O(n!)$. You can use our discussion forum to get your doubt cleared.

So by analyzing the equation, we can say that the algorithm is going to take $O(n!)$.

Conclusion: Thus we have studied and implemented n-Queens matrix having first Queen placed by Using backtracking method .

FAQ:

1. In which Problem can we use backtracking?
2. Which data structure is useful in a backtracking algorithm?
3. What is the time complexity of backtracking?
4. Is backtracking same as recursion?
5. Is backtracking DFS or BFS?
6. What are real applications where backtracking is used?
7. How many possible solutions exist for the N-queen problem ?
8. which data structures is used in the N-queen problem
9. What is the best way to solve the N-queen problem?

AssignmentNo.	**5**
Title	Quick Sort Algorithm
PROBLEM STATEMENT/DEFINITION	Write a program for analysis of quick sort by using deterministic and randomized variant.
Objectives	To implement algorithms that follow divide and conquer design Strategy To analysis quick sort by using deterministic and randomized variant. To understand concept of recursion.
Software packages and hardware apparatus used	MySQL/Oracle PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
References	Quicksort - Wikipedia QuickSort - GeeksforGeeks http://en.wikipedia.org/wiki/Quicksort
STEPS	Refer to details
Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

AssignmentNo.1

Title : Quick sort Algorithm

Objectives:

- To implement algorithms that follow divide and conquer design Strategy
- To analysis quick sort by using deterministic and randomized variant.
- To understand concept of recursion.

Theory: **Deterministic Quicksort**

Algorithm

```
Quicksort(A,p,r)

{

if p < r then

q := Partition(A,p,r); // rearrange A[p..r] in place

Quicksort(A, p,q-1);

Quicksort(A,p+1,r);

}
```

Divide-and-Conquer

The design of Quicksort is based on the divide-and-conquer paradigm.

Divide: Partition the array A[p..r] into two (possibly empty) subarrays A [p..q-1] and A[q+1,r] such that

A[x] <= A[q] for all x in [p..q-1] A[x] > A[q] for all x in [q+1,r]

b) Conquer: Recursively sort A[p..q-1] and A[q+1,r]

c) Combine: nothing to do here

Example:

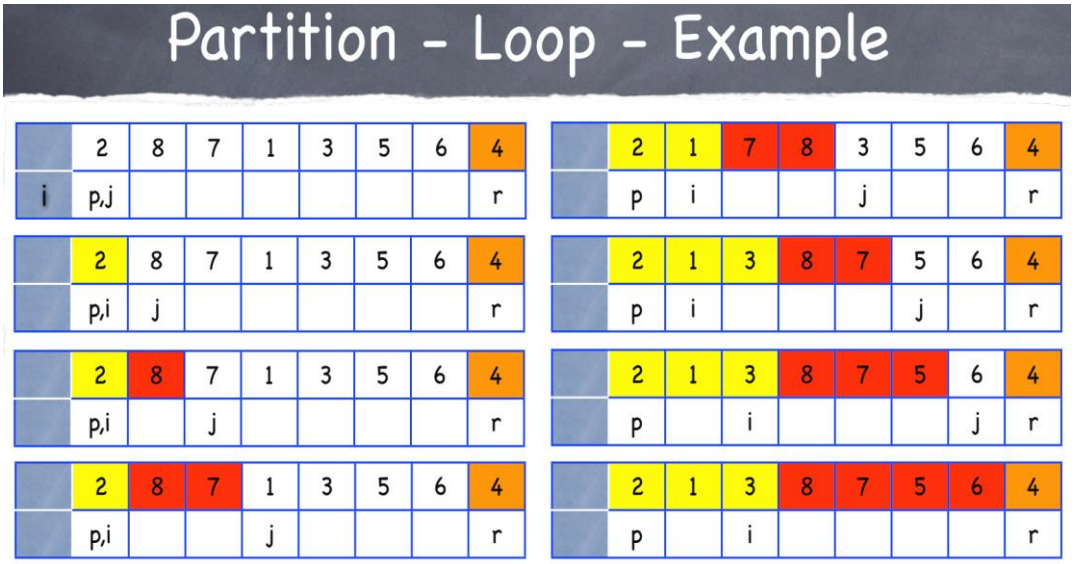
Partition Select pivot (orange element) and rearrange:

2	1	3	4	7	5	6	8	
p		i						r

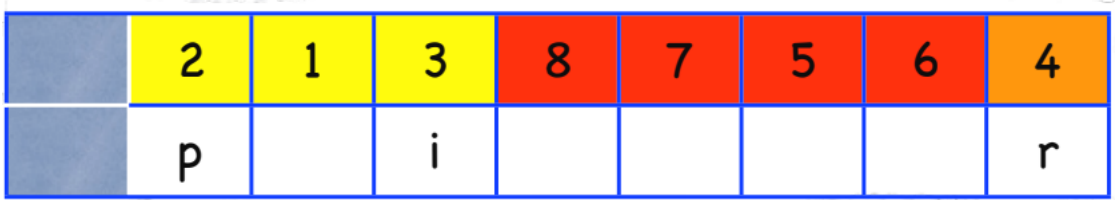
larger elements to the left of the pivot (red)

elements not exceeding the pivot to the right (yellow)

```
Partition(A,p,r)
x := A[r]; // select rightmost element as pivot
i := p-1;
for j = p to r-1
{ if A[j] <= x then
 i := i+1;
swap(A[i], A[j]);
 }
swap(A[i+1],A[r]);
return i+1
```



After the loop, the partition routine swaps the leftmost element of the right partition with the pivot element



swap(A[i+1],A[r])

now recursively sort yellow and red parts.

Analysis

Worst-Case Partitioning
The worst-case behavior for quicksort occurs on an input of length n when partitioning produces just one subproblem with n-1 elements and one subproblem with 0 elements. Therefore the recurrence for the running time

T(n) is: $T(n) = T(n-1) + T(0) + \theta(n) = T(n-1) + \theta(n) = \theta(n^2)$

Best-case partitioning:
If partition produces two subproblems that are roughly of the same size, then the recurrence of the running time is
$T(n) <= 2T(n/2) + \theta(n)$ so that
$T(n) = O(n \log n)$

Randomized Quicksort

Randomized Quicksort Algorithm

Randomized-Quicksort(A,p,r)

{

if p < r then

{

q := Randomized-Partition(A,p,r);

Randomized-Quicksort(A, p,q-1);

Randomized-Quicksort(A,p+1,r);

}

}

Randomized-Partition(A,p,r)

{

i := Random(p,r);

swap(A[i],A[r]);

Partition(A,p,r);

Almost the same as Partition, but now the pivot element is not the rightmost element, but rather an element from A[p..r] that is chosen uniformly at random.

}

Analysis

It follows that the expected running time of RandomizedQuicksort is O(n log n).

It is unlikely that this algorithm will choose a terribly unbalanced partition each time, so the performance is very good almost all the time.

Conclusion: We have successfully seen how deterministic and randomized quick sort is implemented

FAQ?

1. How quick sort work?
2. Which algorithm strategy quick sort uses?
3. How deterministic quicksort is different from randomize quick sort?
4. Explain with example deterministic quick sort?
5. What is the best case ,worst case ,average case complexity of Deterministic quick sort?
6. Comment on complexity of randomized quicksort.

Assignment No.	**1**
Title	Uber ride price prediction using linear regression and random forest regression models.
PROBLEM STATEMENT/DEFINITION	Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks: 1. Pre-process the dataset. 2. Identify outliers. 3. Check the correlation. 4. Implement linear regression and random forest regression models. Evaluate the models and compare their respective scores like R2, RMSE, etc.
Objectives	Implementation of linear regression and random forest regression models.
Software packages and hardware apparatus used	Jupyter Notebook, PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
References	**Dataset link:** **https://www.kaggle.com/datasets/yasserh/uber-fares-dataset**
STEPS	Refer to details
Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

AssignmentNo.1

Title: Uber ride price prediction using linear regression and random forest regression models.

Objectives: Implementation of linear regression and random forest regression models.

Theory:

Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables. More specifically, Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed. It predicts continuous/real values such as **temperature, age, salary, price,** etc.
Regression is a supervised learning technique
which helps in finding the correlation between variables and enables us to predict the continuous output variable based on the one or more predictor variables. It is mainly used for **prediction, forecasting, time series modeling, and determining the causal-effect relationship between variables**.

In Regression, we plot a graph between the variables which best fits the given datapoints, using this plot, the machine learning model can make predictions about the data. In simple words, **"Regression shows a line or curve that passes through all the datapoints on target-predictor graph in such a way that the vertical distance between the datapoints and the regression line is minimum."** The distance between datapoints and line tells whether a model has captured a strong relationship or not.

Some examples of regression can be as:

- Prediction of rain using temperature and other factors
- Determining Market trends
- Prediction of road accidents due to rash driving.

Types of Regression

- **Linear Regression**
- **Logistic Regression**
- **Polynomial Regression**
- **Support Vector Regression**
- **Decision Tree Regression**
- **Random Forest Regression**
- **Ridge Regression**

- **Lasso Regression:**

1.Linear Regression:

- Linear regression is a statistical regression method which is used for predictive analysis.
- It is one of the very simple and easy algorithms which works on regression and shows the relationship between the continuous variables.
- It is used for solving the regression problem in machine learning.
- Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), hence called linear regression.
- If there is only one input variable (x), then such linear regression is called **simple linear regression**. And if there is more than one input variable, then such linear regression is called **multiple linear regression**.
- The relationship between variables in the linear regression model can be explained using the below image. Here we are predicting the salary of an employee on the basis of **the year of experience**.

- Below is the mathematical equation for Linear regression

$$Y= aX+b$$

Here, Y = dependent variables (target variables),
X= Independent variables (predictor variables),
a and b are the linear coefficients

Some popular applications of linear regression are:

- **Analyzing trends and sales estimates**
- **Salary forecasting**
- **Real estate prediction**
- **Arriving at ETAs in traffic.**

1.R-squared method:

- R-squared is a statistical method that determines the goodness of fit.
- It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.

- The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a **coefficient of determination,** or **coefficient of multiple determination** for multiple regression.
- It can be calculated from the below formula:

$$\text{R-squared} = \frac{\text{Explained variation}}{\text{Total Variation}}$$

Introduction:

The datset contains the following fields:

key - a unique identifier for each trip

fare_amount - the cost of each trip in usd

pickup_datetime - date and time when the meter was engaged

passenger_count - the number of passengers in the vehicle (driver entered value)

pickup_longitude - the longitude where the meter was engaged

pickup_latitude - the latitude where the meter was engaged

dropoff_longitude - the longitude where the meter was disengaged

dropoff_latitude - the latitude where the meter was disengaged

- **Import libraries**

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pylab
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn import metrics
import math
from statsmodels.tools.eval_measures import rmse
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV
```

```
#importing the dataset
df  = pd.read_csv("uber.csv")
```

1. Pre-process the dataset.

```
df.head()
df.info() #To get the required information of the dataset
df.columns #TO get number of columns in the dataset
df = df.drop(['Unnamed: 0', 'key'], axis= 1) #To drop unnamed column as it
 isn't required
df.head()
df.shape #To get the total (Rows,Columns)
df.dtypes #To get the type of each column
df.info()
df.describe() #To get statistics of each columns
```

2. Filling Missing values

```
df.isnull().sum()
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace
= True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inpl
ace = True)
df.isnull().sum()
df.dtypes
```

3. Column pickup_datetime is in wrong format (Object). Convert it to DateTime Format

```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce')
df.dtypes
```

4.To segregate each time of date and time

```
df= df.assign(hour = df.pickup_datetime.dt.hour,
              day= df.pickup_datetime.dt.day,
              month = df.pickup_datetime.dt.month,
              year = df.pickup_datetime.dt.year,
              dayofweek = df.pickup_datetime.dt.dayofweek)
df.head()
# drop the column 'pickup_daetime' using drop()
# 'axis = 1' drops the specified column

df = df.drop('pickup_datetime',axis=1)
```

```
df.head()
df.dtypes
```

5. Checking outliers and filling them

Importance of detecting an outlier

An outlier is an observation that appears to deviate distinctly from other observations in the data. If the outliers are not removed, the model accuracy may decrease. Let us detect the extreme values in the data.

The following can be considered as outliers in this case study:

1. Amount < -52
2. Trips with travel distance less than or equal to 0, and more than 130Kms
3. Trips where 90< latitude <-90, 180 < longitude < -180

1. We have seen that there are instances of fare_amount less that 0 as well in the data set, where as the minimum fare for any trip is -52 dollars, hence we will remove such observations. We have already seen that the max fare is 499 in the data set.

```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxp
lot to check the outliers
#Using the InterQuartile Range to fill the values
def remove_outlier(df1 , col):
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1-1.5*IQR
    upper_whisker = Q3+1.5*IQR
    df[col] = np.clip(df1[col] , lower_whisker , upper_whisker)
    return df1

def treat_outliers_all(df1 , col_list):
    for c in col_list:
        df1 = remove_outlier(df , c)
    return df1
df = treat_outliers_all(df , df.iloc[: , 0::])
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxp
lot shows that dataset is free from outliers
#pip install haversine
```

```
import haversine as hs  #Calculate the distance using Haversine to calcula
te the distance between to points. Can't use Eucladian as it is for flat s
urface.
travel_dist = []
for pos in range(len(df['pickup_longitude'])):
        long1,lati1,long2,lati2 = [df['pickup_longitude'][pos],df['pickup_
latitude'][pos],df['dropoff_longitude'][pos],df['dropoff_latitude'][pos]]
        loc1=(lati1,long1)
        loc2=(lati2,long2)
        c = hs.haversine(loc1,loc2)
        travel_dist.append(c)

print(travel_dist)
df['dist_travel_km'] = travel_dist
df.head()
#Uber doesn't travel over 130 kms so minimize the distance
df= df.loc[(df.dist_travel_km >= 1) | (df.dist_travel_km <= 130)]
print("Remaining observastions in the dataset:", df.shape)
#Finding inccorect latitude (Less than or greater than 90) and longitude (
greater than or less than 180)
incorrect_coordinates = df.loc[(df.pickup_latitude > 90) |(df.pickup_latit
ude < -90) |
                                    (df.dropoff_latitude > 90) |(df.dropoff
_latitude < -90) |
                                    (df.pickup_longitude > 180) |(df.pickup
_longitude < -180) |
                                    (df.dropoff_longitude > 90) |(df.dropof
f_longitude < -90)
                                     ]
df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')
df.head()
df.isnull().sum()
sns.heatmap(df.isnull()) #Free for null values
corr = df.corr() #Function to find the correlation
corr
fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True) #Correlation Heatmap (Light values mea
ns highly correlated)
```

6.Dividing the dataset into feature and target values

```
x = df[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_
latitude','passenger_count','hour','day','month','year','dayofweek','dist_
travel_km']]
```

```
y = df['fare_amount']
```

7.Dividing the dataset into training and testing dataset

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size = 0.33)
```

8.Linear Regression

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train,y_train)
regression.intercept_ #To find the linear intercept
regression.coef_ #To find the linear coeeficient
prediction = regression.predict(X_test) #To predict the target values
print(prediction)
y_test
```

9.Metrics Evaluation using R2, Mean Squared Error, Root Mean Sqared Error

```
from sklearn.metrics import r2_score
r2_score(y_test,prediction)
from sklearn.metrics import mean_squared_error
MSE = mean_squared_error(y_test,prediction)
MSE
RMSE = np.sqrt(MSE)
RMSE
```

10. Random Forest Regression

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators=100) #Here n_estimators means numb
er of trees you want to build before making the prediction
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
y_pred
```

11.Metrics evaluatin for Random Forest

```
R2_Random = r2_score(y_test,y_pred)
R2_Random
MSE_Random = mean_squared_error(y_test,y_pred)
```

```
MSE_Random
RMSE_Random = np.sqrt(MSE_Random)
RMSE_Random
```

Assignment No.	2
Title	Classify the email using the binary classification K-Nearest Neighbors and Support Vector Machine and Analyze their performance.
PROBLEM STATEMENT/DEFINITION	Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance
Objectives	Classification of the email using the binary classification K-Nearest Neighbors and Support Vector Machine and Analyze their performance.
Software packages and hardware apparatus used	Jupyter Notebook, PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
References	**Dataset link:** **https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv**
STEPS	Refer to details
Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

AssignmentNo.2

Title: Classify the email using the binary classification K-Nearest Neighbors and Support Vector Machine and Analyze their performance.

Objectives: Classification of the email using the binary classification K-Nearest Neighbors and Support Vector Machine and Analyze their performance.

Theory:

1. K-Nearest Neighbour

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

The K-NN working can be explained on the basis of the below algorithm:

Step-1: Select the number K of the neighbors

Step-2: Calculate the Euclidean distance of **K number of neighbors**

Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4: Among these k neighbors, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

Step-6: Our model is ready.

2. Support Vector Machine

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

Introduction:

Import libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics


df=pd.read_csv('emails.csv')
df.head()
df.columns
df.isnull().sum()
df.dropna(inplace = True)
df.drop(['Email No.'],axis=1,inplace=True)
X = df.drop(['Prediction'],axis = 1)
y = df['Prediction']
from sklearn.preprocessing import scale
X = scale(X)
# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
 random_state = 42)
```

1.KNN classifier

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
```

```
y_pred = knn.predict(X_test)
print("Prediction",y_pred)
print("KNN accuracy = ",metrics.accuracy_score(y_test,y_pred))
print("Confusion matrix",metrics.confusion_matrix(y_test,y_pred))
```

2. SVM classifier

```
# cost C = 1
model = SVC(C = 1)

# fit
model.fit(X_train, y_train)

# predict
y_pred = model.predict(X_test)
metrics.confusion_matrix(y_true=y_test, y_pred=y_pred)
print("SVM accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

AssignmentNo.	B4
Title	Machine Learning Gradient Descent Algorithm
PROBLEM STATEMENT/DEFINITION	Implement Gradient Descent Algorithm to find the local minima of a function. For example, find the local minima of the function $y=(x+3)^2$ starting from the point x=2
Objectives	To understand Gradient Descent Algorithm.
Software packages and hardware apparatus used	Programming Languages Java, Python
References	https://builtin.com/data-science/gradient-descent
STEPS	1. Define the initial parameter's values 2. Use calculus to iteratively adjust the values so they minimize the given cost-function.
Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

Aim:
Implement Gradient Descent Algorithm to find the local minima of a function.
For example, find the local minima of the function $y=(x+3)^2$ starting from the point x=2

Objective:
To understand Gradient Descent Algorithm.

Theory:
Gradient Descent is known as one of the most commonly used optimization algorithms to train machine learning models by means of minimizing errors between actual and expected results. Further, gradient descent is also used to train Neural Networks.

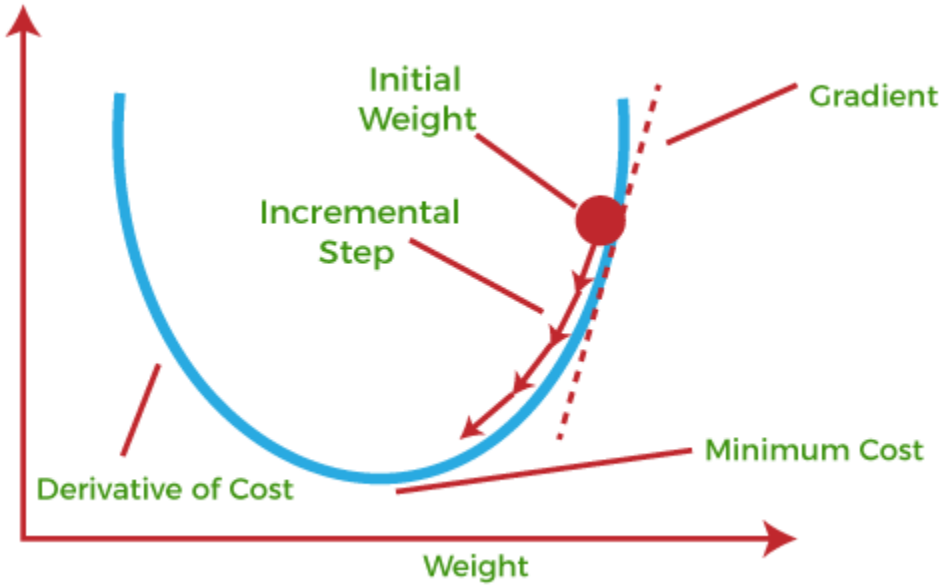
In mathematical terminology, Optimization algorithm refers to the task of minimizing/maximizing an objective function f(x) parameterized by x. Similarly, in machine learning, optimization is the task of minimizing the cost function parameterized by the model's parameters. The main objective of gradient descent is to minimize the convex function using iteration of parameter updates. Once these machine learning models are optimized, these models can be used as powerful tools for Artificial Intelligence and various computer science applications.

What is Gradient Descent or Steepest Descent?

Gradient descent was initially discovered by "Augustin-Louis Cauchy" in mid of 18th century. Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models. It helps in finding the local minimum of a function.

The best way to define the local minimum or local maximum of a function using gradient descent is as follows:

If we move towards a negative gradient or away from the gradient of the function at the current point, it will give the local minimum of that function.
Whenever we move towards a positive gradient or towards the gradient of the function at the current point, we will get the local maximum of that function.



This entire procedure is known as Gradient Ascent, which is also known as steepest descent. The main objective of using a gradient descent algorithm is to minimize the cost function using iteration. To achieve this goal, it performs two steps iteratively:

- Calculates the first-order derivative of the function to compute the gradient or slope of that function.
- Move away from the direction of the gradient, which means slope increased from the current point by alpha times, where Alpha is defined as Learning Rate. It is a tuning parameter in the optimization process which helps to decide the length of the steps.

Conclusion:
Students will understand Gradient Descent Algorithm.

AssignmentNo.	B3
Title	Machine Learning neural network-based classifier
PROBLEM STATEMENT/DEFINITION	Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months. Dataset Description: The case study is from an open-source dataset from Kaggle.The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc. Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling Perform following steps: 1. Read the dataset. 2. Distinguish the feature and target set and divide the data set into training and test sets. 3. Normalize the train and test data. 4. Initialize and build the model. Identify the points of improvement and implement the same. Print the accuracy score and confusion matrix (5 points).
Objectives	Understand basic concepts of neural network-based classifier
Software packages and hardware apparatus used	Programming Languages: Java, Python
References	https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling
STEPS	1. Read the dataset. 2. Distinguish the feature and target set and divide the data set into training and test sets. 3. Normalize the train and test data. 4. Initialize and build the model. Identify the points of improvement and implement the same.
Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective

	• Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

Aim:

Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.
Dataset Description: The case study is from an open-source dataset from Kaggle.The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc.
Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling Perform following steps:

1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.

Print the accuracy score and confusion matrix (5 points).

Objective:

Understand basic concepts of neural network-based classifier

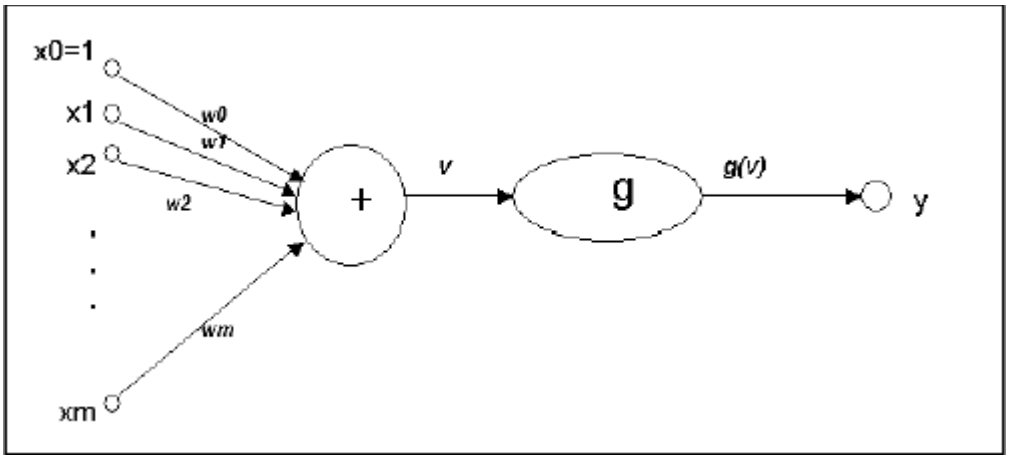
Theory:

Artificial neural networks are relatively crude electronic networks of neurons based on the neural structure of the brain. They process records one at a time, and learn by comparing their classification of the record (i.e., largely arbitrary) with the known actual classification of the record. The errors from the initial classification of the first record is fed back into the network, and used to modify the networks algorithm for further iterations.

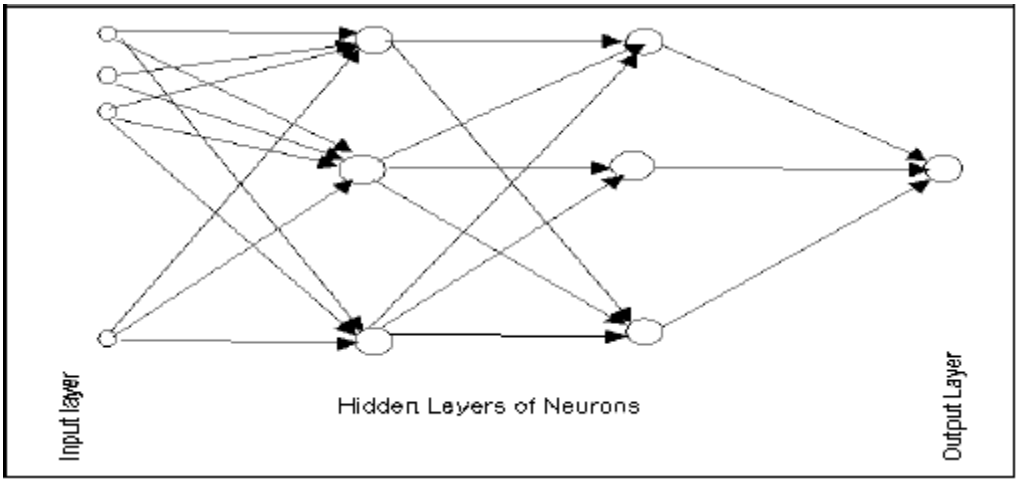
A neuron in an artificial neural network is

1. A set of input values (xi) and associated weights (wi).

2. A function (g) that sums the weights and maps the results to an output (y).



Neurons are organized into layers: input, hidden and output. The input layer is composed not of full neurons, but rather consists simply of the record's values that are inputs to the next layer of neurons. The next layer is the hidden layer. Several hidden layers can exist in one neural network. The final layer is the output layer, where there is one node for each class. A single sweep forward through the network results in the assignment of a value to each output node, and the record is assigned to the class node with the highest value.



Training an Artificial Neural Network

In the training phase, the correct class for each record is known (termed supervised training), and the output nodes can be assigned correct values -- 1 for the node corresponding to the correct class, and 0 for the others. (In practice, better results have been found using values of 0.9 and 0.1, respectively.) It is thus possible to compare the network's calculated values for the output nodes to these correct values, and calculate an error term for each node (the Delta rule). These error terms are then used to adjust the weights in the hidden layers so that, hopefully, during the next iteration the output values will be closer to the correct values.

The Iterative Learning Process

A key feature of neural networks is an iterative learning process in which records (rows) are presented to the network one at a time, and the weights associated with the input values are adjusted each time. After all cases are presented, the process is often repeated. During this learning phase, the network trains by adjusting the weights to predict the correct class label of input samples. Advantages of neural networks include their high tolerance to noisy data, as well as their ability to classify patterns on which they have not been trained. The most popular neural network algorithm is the back-propagation algorithm proposed in the 1980s.

Once a network has been structured for a particular application, that network is ready to be trained. To start this process, the initial weights (described in the next section) are chosen randomly. Then the training (learning) begins.

The network processes the records in the Training Set one at a time, using the weights and functions in the hidden layers, then compares the resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights for application to the next record. This process occurs repeatedly as the weights are tweaked. During the training of a network, the same set of data is processed many times as the connection weights are continually refined.

Note that some networks never learn. This could be because the input data does not contain the specific information from which the desired output is derived. Networks also will not converge if there is not enough data to enable complete learning. Ideally, there should be enough data available to create a Validation Set.

Feedforward, Back-Propagation

The feedforward, back-propagation architecture was developed in the early 1970s by several independent sources (Werbor; Parker; Rumelhart, Hinton, and Williams). This independent co-development was the result of a proliferation of articles and talks at various conferences that stimulated the entire industry. Currently, this synergistically developed back-propagation architecture is the most popular model for complex, multi-layered networks. Its greatest strength is in non-linear solutions to ill-defined problems.

The typical back-propagation network has an input layer, an output layer, and at least one hidden layer. There is no theoretical limit on the number of hidden layers but typically there are just one or two. Some studies have shown that the total number of layers needed to solve problems of any complexity is five (one input layer, three hidden layers and an output layer). Each layer is fully connected to the succeeding layer.

The training process normally uses some variant of the Delta Rule, which starts with the calculated difference between the actual outputs and the desired outputs. Using this error, connection weights are increased in proportion to the error times, which are a scaling factor for global accuracy. This means that the inputs, the output, and the desired output all must be present at the same processing element. The most complex part of this algorithm is determining which input contributed the most to an incorrect output and how must the input be modified to correct the error. (An inactive node would not contribute to the error and would have no need to change

its weights.) To solve this problem, training inputs are applied to the input layer of the network, and desired outputs are compared at the output layer. During the learning process, a forward sweep is made through the network, and the output of each element is computed by layer. The difference between the output of the final layer and the desired output is back-propagated to the previous layer(s), usually modified by the derivative of the transfer function. The connection weights are normally adjusted using the Delta Rule. This process proceeds for the previous layer(s) until the input layer is reached.

Structuring the Network

The number of layers and the number of processing elements per layer are important decisions. To a feedforward, back-propagation topology, these parameters are also the most ethereal -- they are the art of the network designer. There is no quantifiable answer to the layout of the network for any particular application. There are only general rules picked up over time and followed by most researchers and engineers applying while this architecture to their problems.

Rule One: As the complexity in the relationship between the input data and the desired output increases, the number of the processing elements in the hidden layer should also increase.

Rule Two: If the process being modeled is separable into multiple stages, then additional hidden layer(s) may be required. If the process is not separable into stages, then additional layers may simply enable memorization of the training set, and not a true general solution.

Rule Three: The amount of Training Set available sets an upper bound for the number of processing elements in the hidden layer(s). To calculate this upper bound, use the number of cases in the Training Set and divide that number by the sum of the number of nodes in the input and output layers in the network. Then divide that result again by a scaling factor between five and ten. Larger scaling factors are used for relatively less noisy data. If too many artificial neurons are used the Training Set will be memorized, not generalized, and the network will be useless on new data sets.

Conclusion:
Students will learn neural network-based classifier.

PUNE INSTITUTE OF COMPUTER TECHNOLOGY, PUNE

ACADEMIC YEAR: 2022-23

DEPARTMENT of COMPUTER ENGINEERING DEPARTMENT

CLASS: T.E. **SEMESTER: I**

SUBJECT: LP-III

ASSINGMENT NO.	2
TITLE	Create your own wallet using Metamask for crypto transactions
PROBLEM STATEMENT /DEFINITION	Create your own wallet using Metamask for crypto transactions
OBJECTIVE	• Students must be able to understand concepts related to crypto transactions. • Students will be able to understand Metamask wallet.
OUTCOME	• Students will be able to make use of Wallet related to crypto transactions.
S/W PACKAGES AND HARDWARE/ APPARATUS USED	• Meta mask wallet api • Operating System- IOS,Andriod • Browser Extension • Mobile App
REFERENCES	• The crypto wallet for Defi, Web3 Dapps and NFTs | MetaMask • Cryptocurrency - Wikipedia • Blockchain Technology-Chandramouli Subramanian,Asha A George,Abhilash K A and Meena Karthikeyan • MetaMask - Wikipedia
STEPS	See After Concepts
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases

	9. Conclusion/Analysis

Prerequisites: Understanding of Cryptocurrency basics, Wallets related with it.

Concepts related Theory:

Cryptocurrency –

Crypto Means "secret" and currency means "system of money".

Cryptocurrency is nothing but the Digital Currency especially developed for faster, cheaper and more reliable than money. In the Cryptocurrency world, the need for government to create money and banks to store and facilitate the money transactions is practically redundant hence it makes the transactions using cryptocurrency quicker and affordable transactions.

Cryptocurrency Wallets

Bitcoin or any other transactions can be done with the help of Digital Wallet. Cryptocurrency wallet is a Digital wallet which stores the users private and public key enabling the user to do truncations of crypto assets.

It allows user to interact with various blockchain users by sending and receiving digital currency also they can track their details including Profile, Balances etc.

Types of Cryptocurrency wallet –

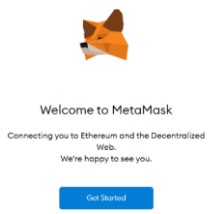
- Hot Wallet
    - Online/Web Wallet
    - Software Wallet
        - Mobile Wallet (e.g Coinomi,Mycelium)
        - Desktop Wallet (e.g Electrum,Armory)
- Cold Wallet
    - Paper Wallet
    - Hardware wallet (e.g Ledger,Trezor)

Meta Mask

Meta mask is Software cryptocurrency wallet used to interact with Ethereum Blockchain. It helps user to access their Ethereum through Mobile app or browser extension.

Steps To Install Meta Mask [On Google Chrome]

1. Go to Chrome Web Store Extension option
2. Search Meta Mask
3. Check For the Proper Meta Mask only by considering number of Downloads for that app.
4. Click on "Add to chrome"
5. Once Installation is complete below page will be displayed.



6. Set up Wallet with you required information by Clicking on “Create Wallet”, Set user ID and Passwords.
7. Now you can access all the functions of Meta Mask

Conclusion: Thus, Students are able to learn and use Meta Mask as Cryptocurrency wallet and are able to make transactions related to it.

Review Questions:

1. What is Cryptocurrency?
2. Explain Evolution of Currency.
3. What is Cryptocurrency wallet? Explain with example.
4. Explain Meta Task. Explain steps to install Meta Mask.

Assignment No.	C1
Title	Installation of MetaMask and Create your own wallet using Metamask for crypto transactions.
PROBLEM STATEMENT/DEFINITION	Installation of MetaMask and Create your own wallet using Metamask for crypto transactions.
Objectives	Study Metamask and its application for Blockchain transaction
Software packages and hardware apparatus used	Any device with chrome browser.
References	**https://metamask.io/** **https://ethereum.org/en/developers/docs** **https://www.ceilidhswhisky.com/documents/Tutorial-on-How-to-use-Metamask.pdf** **https://ethereum.org/en/developers/docs/transactions/**
STEPS	1. Install MetaMask from metamask.io 2. Create account and setup password 3. Secure your account using 12 word secret recovery phrase 4. Get test network ETH using any one of the test network. 5. Perform ETH transaction
Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Test cases • Program Listing • Output • Conclusion

Concept Related Theory:

Digital Wallet: A digital wallet, just like your traditional leather wallet, in blockchain terms, is an interface that allows you to store your private key and interact with multiple digital ledgers depending on the wallet type. Each public blockchain has its wallet. The MetaMask wallet is native to Ethereum, supporting the storage of ETH and its tokens. It's like a bridge, bringing the Ethereum blockchain closer to users through browsers.

MetaMask:

MetaMask is a wallet that connects the user to the Ethereum blockchain. The objective of its creators back in 2016 was to make ETH transactions simple and as intuitive as possible. The project grew much more than that and is today one of the biggest web3 companies. Through MetaMask, you can also interact and use various Ethereum dapps easily – and you can do it from your desktop or mobile app.

MetaMask is a web browser add-on which enables anyone to run the Ethereum, DApps without running the Ethereum full node. An Ethereum full node installation will take a lot of memory as well as time; so Metamask is a tool that eliminates the overburden of this hectic installation task. Initially, Metamask was available only for Google Chrome, but now it is available for Firefox and other popular web browsers.

MetaMask add-on for chrome can be added from chrome web store or from 'metamask.io' website. This MetaMask add-on provides a user interface for interacting with the blockchain. The user can connect to the Ethereum main network or 'testnet' or he may create his own private network and run DApps on the blockchain.

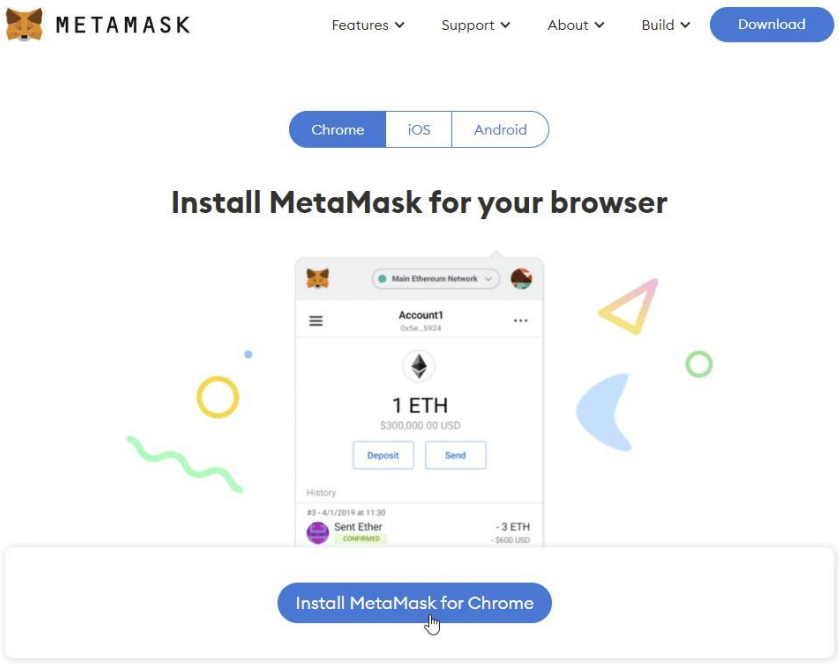
After adding the Metamask, the user can interact with Ethereum blockchain. The user can create an account, access Ethereum DApps, or deploy once own DApp. MetaMask retrieves data from the blockchain and allows the users to manage the data securely.

Wallet Seed :
The Metamask will provide a group of 12 words known as "wallet seed" while installing it. It is the user credential and it must be stored somewhere safe. The users can also create passwords for their account. The wallet seed or the password is necessary to log in to the MetaMask. The vault account will encrypt the user metadata and securely store it in the browser itself.

How to Install MetaMask?

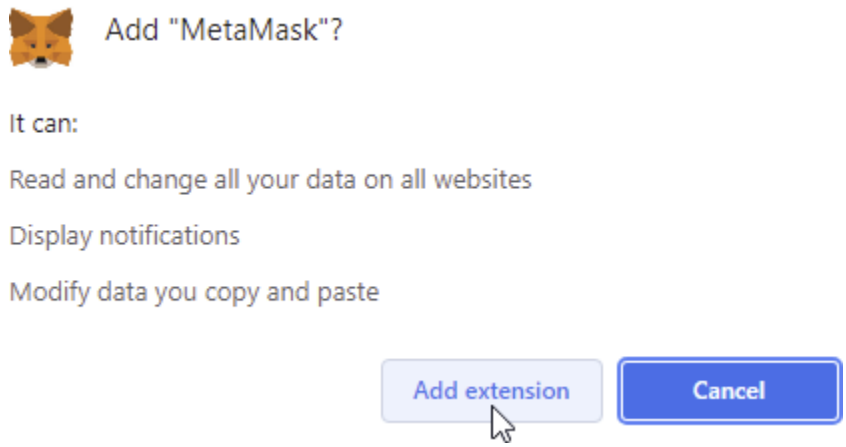
To install MetaMask, you need your computer and Google Chrome.

Go to metamask.io and click Install MetaMask for Chrome:



The Chrome web store will open and click “Add to Chrome”

A confirmation popup will open. And Click Add extension:



A MetaMask welcome screen will open.

Welcome to MetaMask

Connecting you to Ethereum and the Decentralized Web.

We're happy to see you.

Get Started

Click Create a Wallet:

METAMASK

New to MetaMask?

No, I already have a Secret Recovery Phrase

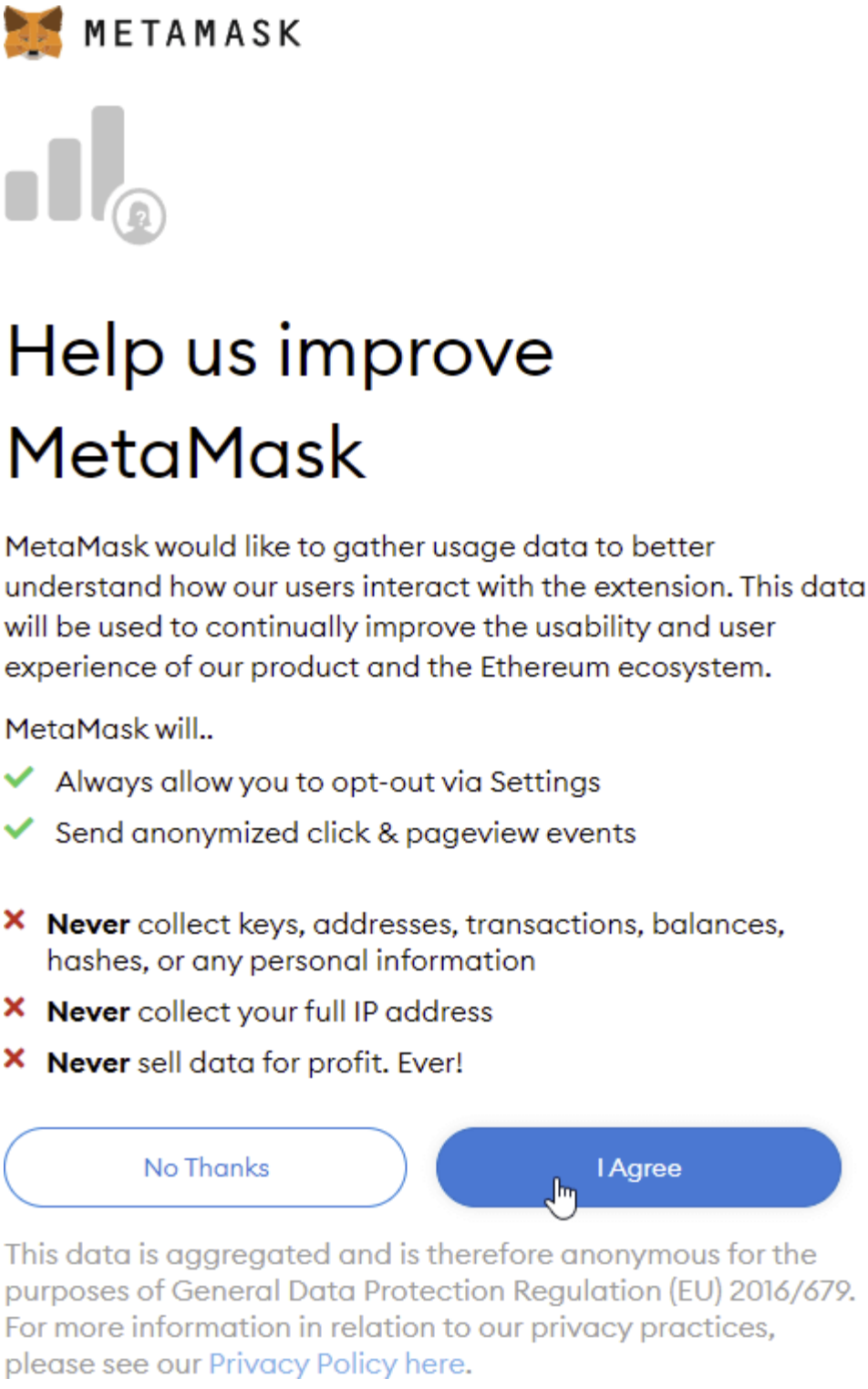
Import your existing wallet using a Secret Recovery Phrase

Import wallet

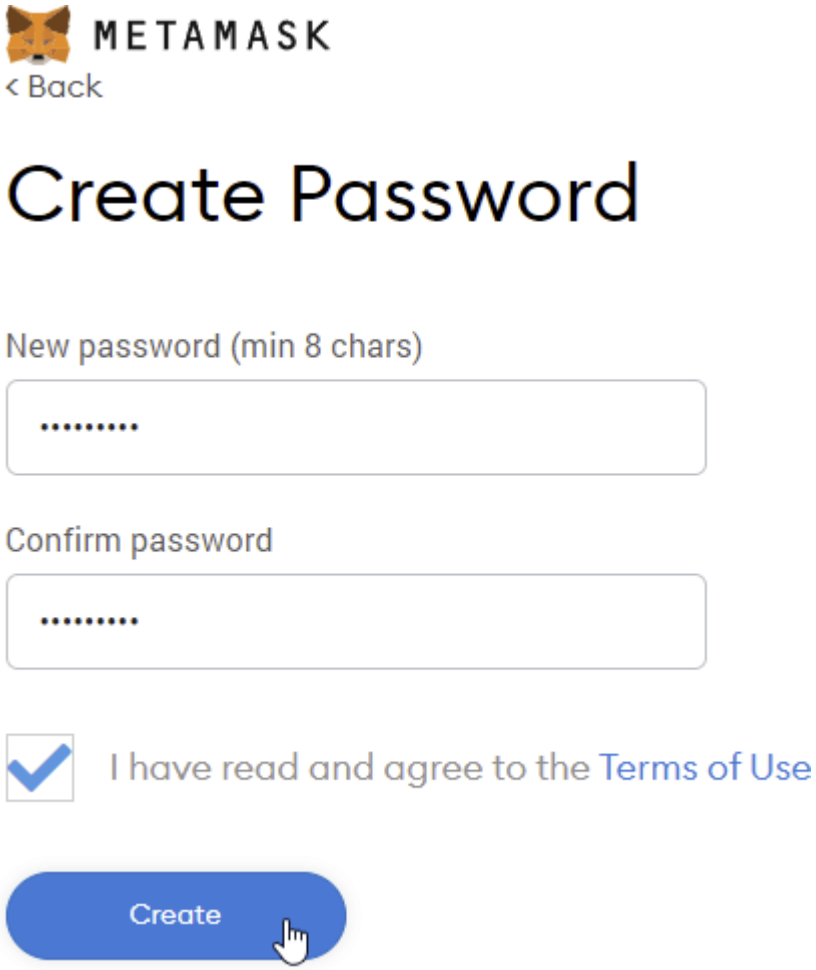
Yes, let's get set up!

This will create a new wallet and Secret Recovery Phrase

Create a Wallet

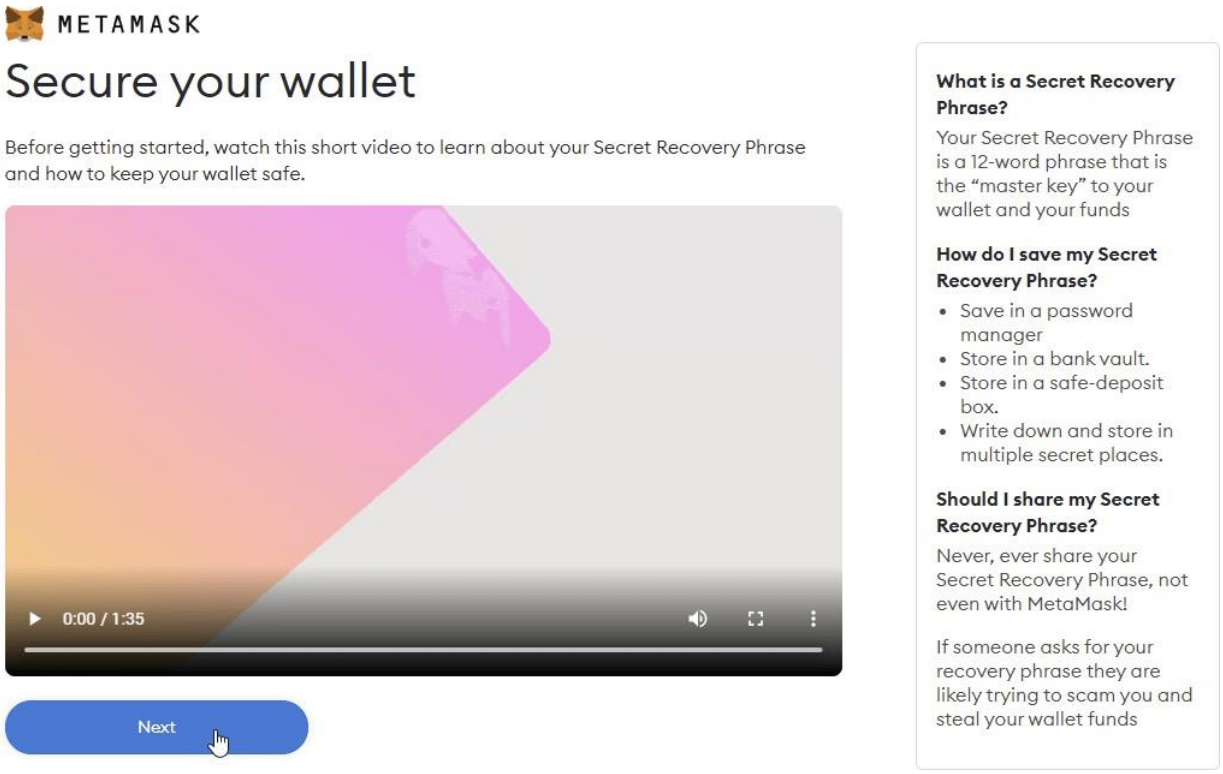


Click I Agree:

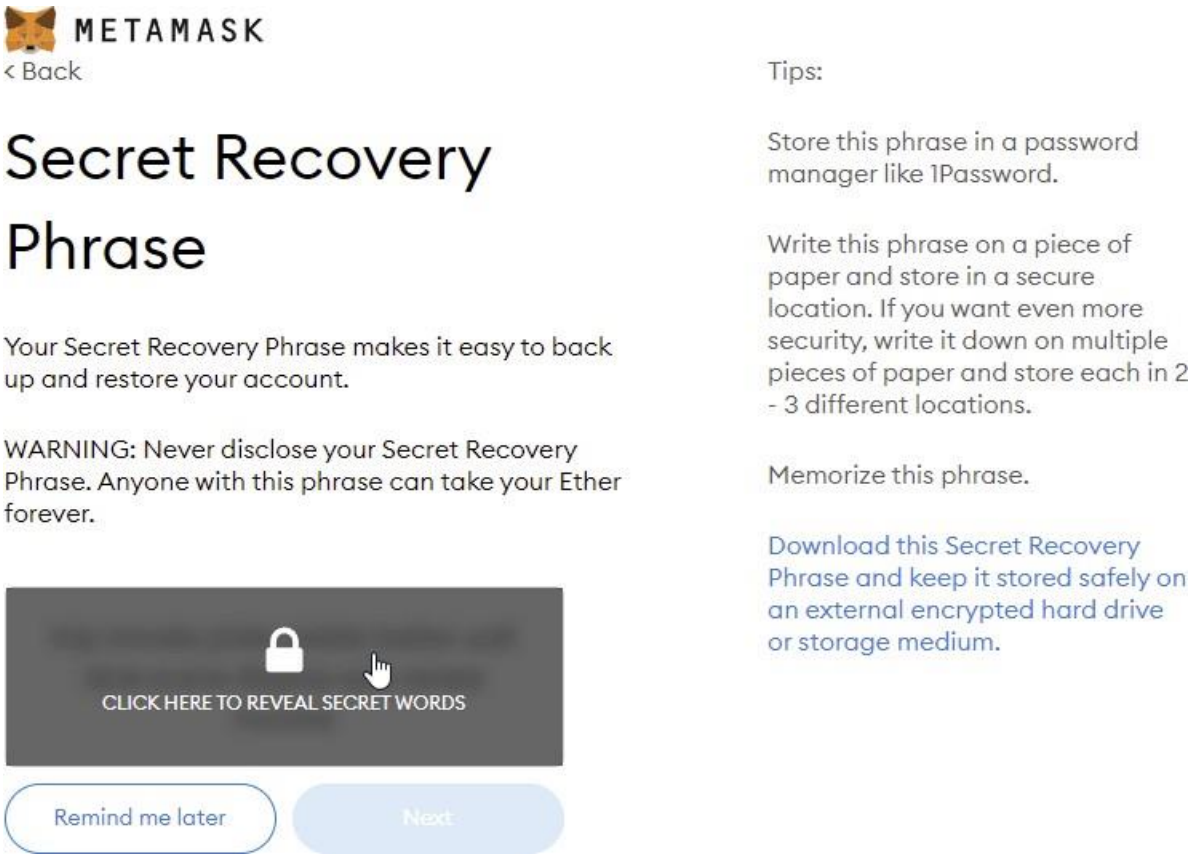


Create a password:
Save it somewhere, MetaMask will sometimes ask for it. DON”T LOSE THIS!

Click Next:


METAMASK
Secure your wallet
Before getting started, watch this short video to learn about your Secret Recovery Phrase and how to keep your wallet safe.
0:00 / 1:35
Next
What is a Secret Recovery Phrase?
Your Secret Recovery Phrase is a 12-word phrase that is the "master key" to your wallet and your funds
How do I save my Secret Recovery Phrase?
Save in a password manager
Store in a bank vault.
Store in a safe-deposit box.
Write down and store in multiple secret places.
Should I share my Secret Recovery Phrase?
Never, ever share your Secret Recovery Phrase, not even with MetaMask!
If someone asks for your recovery phrase they are likely trying to scam you and steal your wallet funds

Reveal your secret words:



These words are basically the key to your wallet.
They are your secret recovery phrase so the order is important.Write your phrase down, so you won't lose it.
Also never share it with anyone, ever!
When you have carefully written down your phrase, click Next:

METAMASK

< Back

Secret Recovery Phrase

Your Secret Recovery Phrase makes it easy to back up and restore your account.

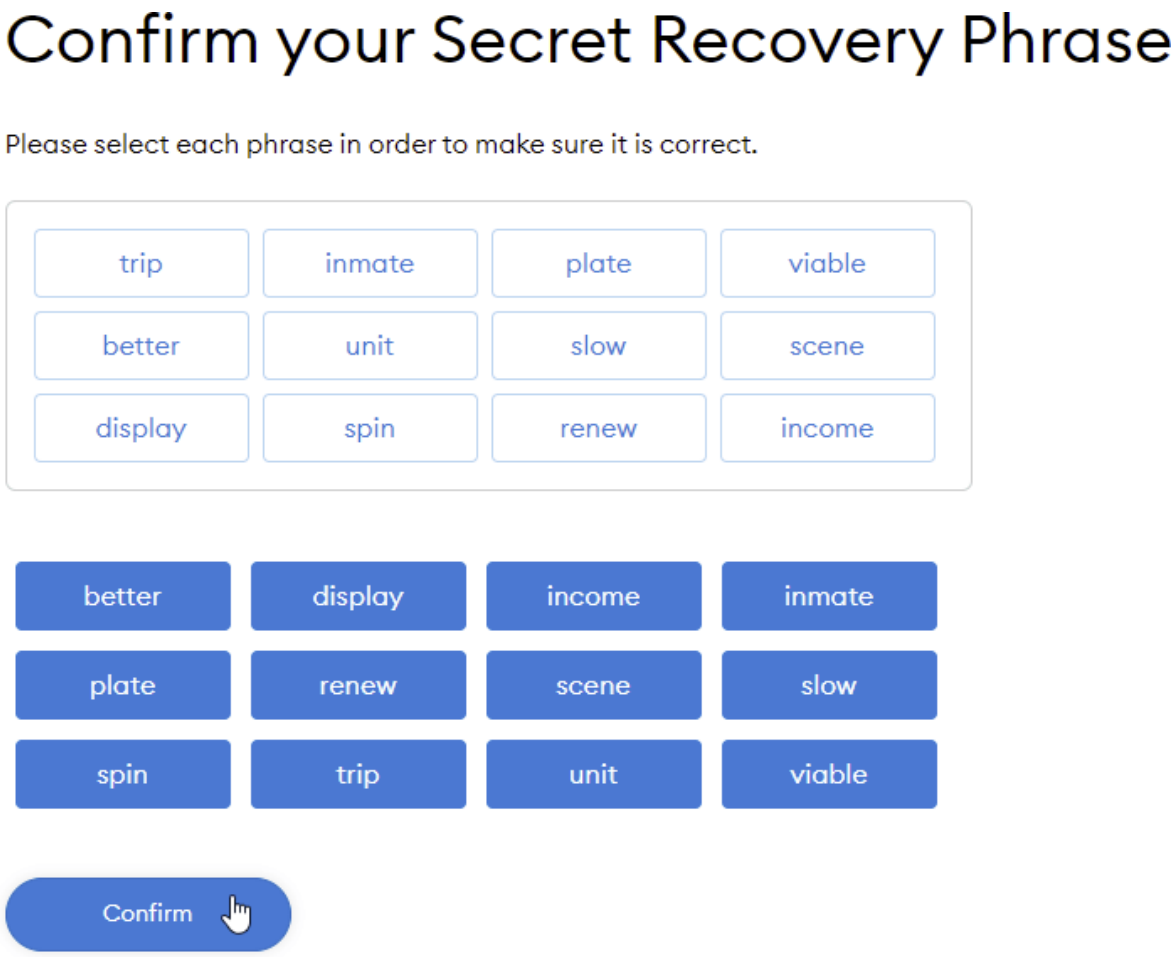
WARNING: Never disclose your Secret Recovery Phrase. Anyone with this phrase can take your Ether forever.

trip inmate plate viable better unit slow scene display spin renew income

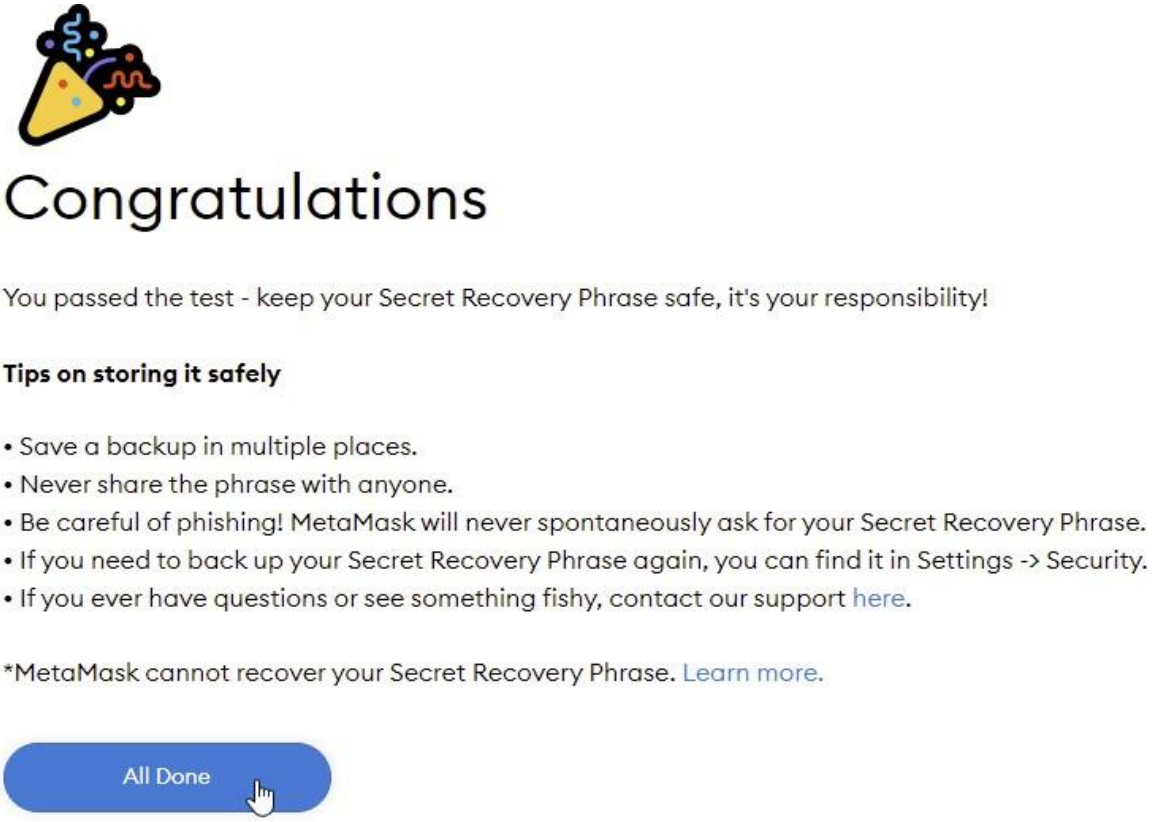
Remind me later

Next

We're sharing our phrase in this tutorial but never share yours with anyone!

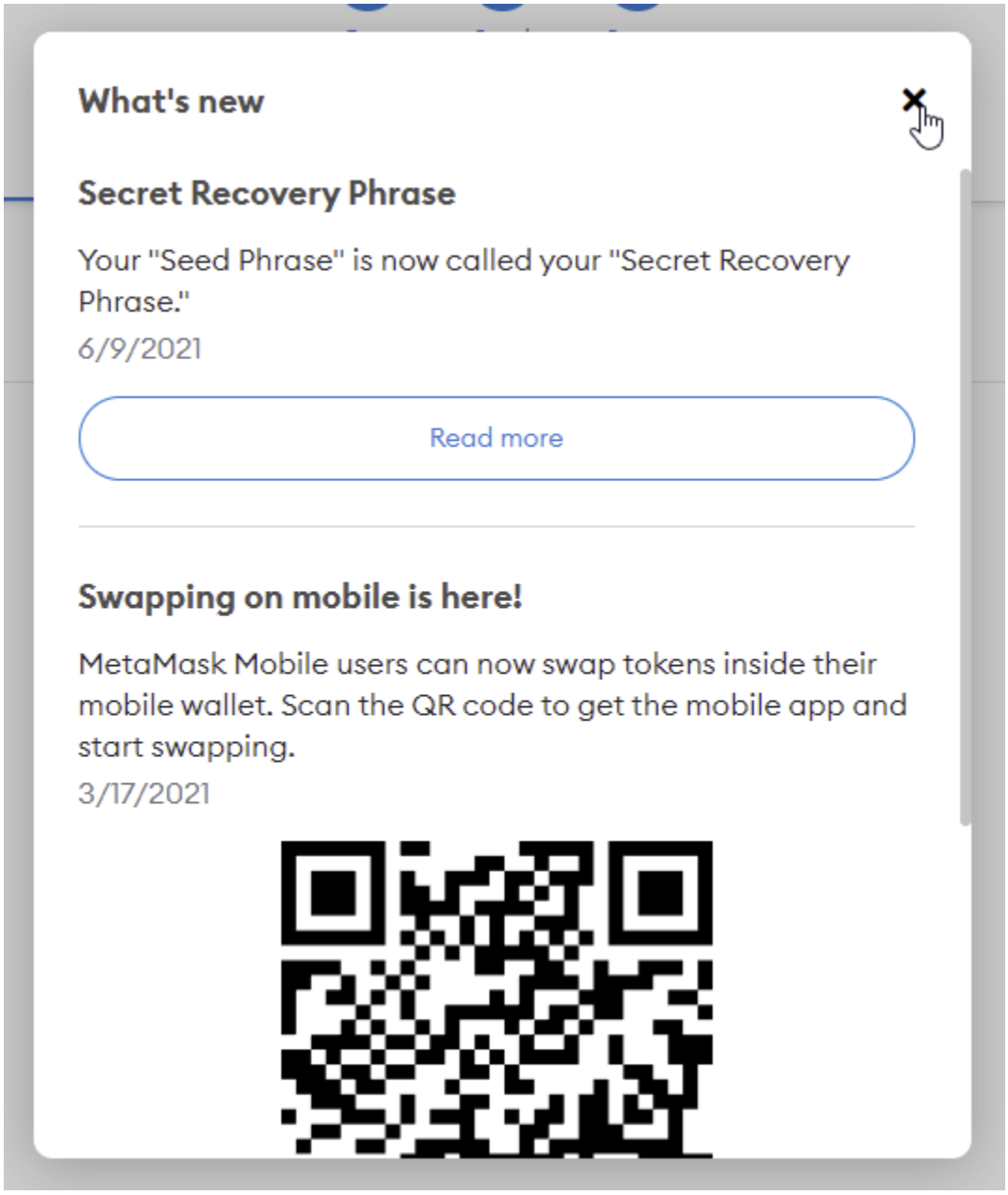


Click All Done:

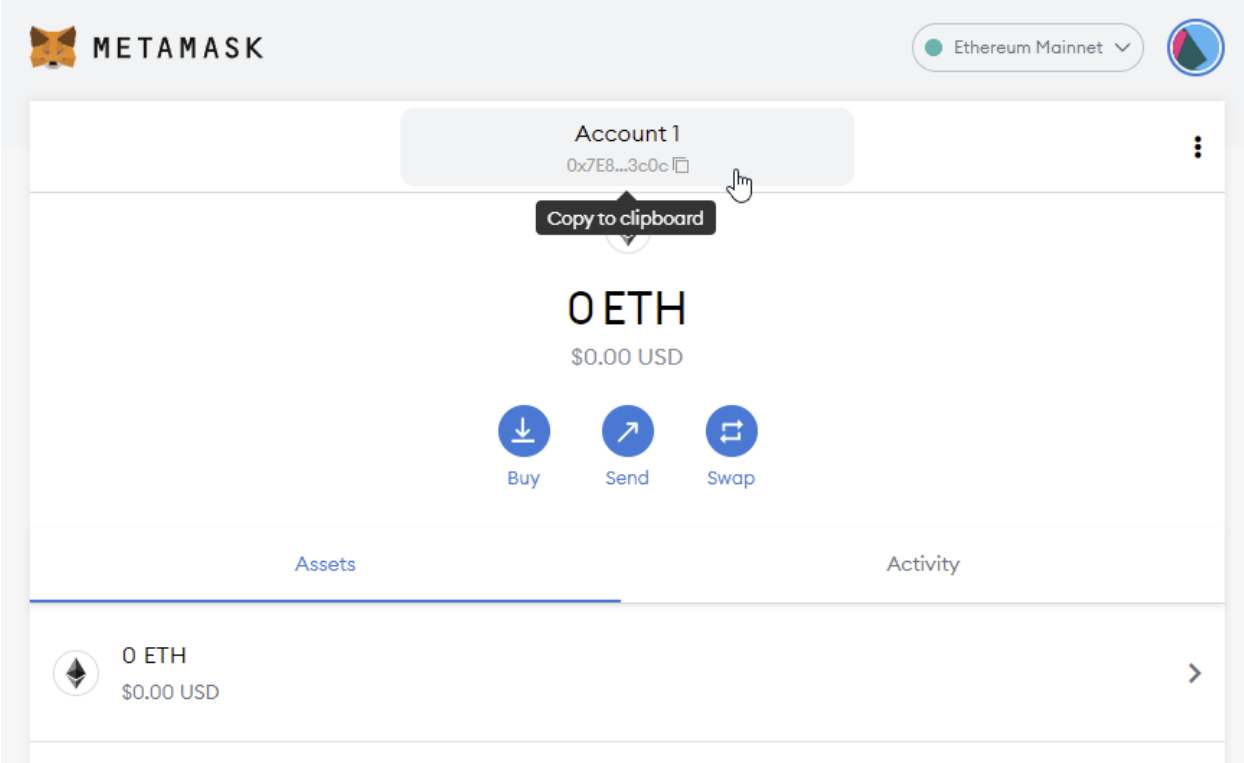


A popup will open.

Click [x] in the top right corner to close it:



Congratulations, you now have a MetaMask wallet!You can find your ETH address under Account 1:



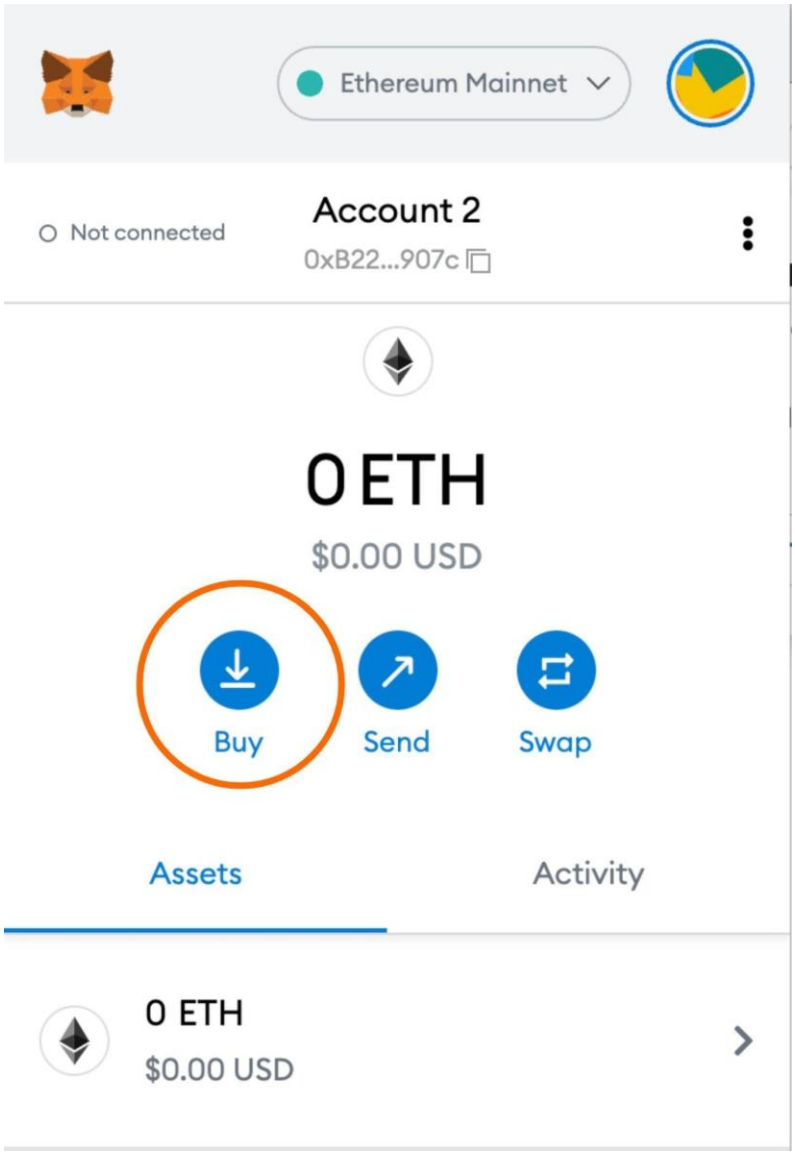
Our tutorial address is 0x7E8...3c0c.Yours will be different!

Buying ETH

There are multiple ways to buy Eth.

The simplest way to get Eth is to buy directly from MetaMask

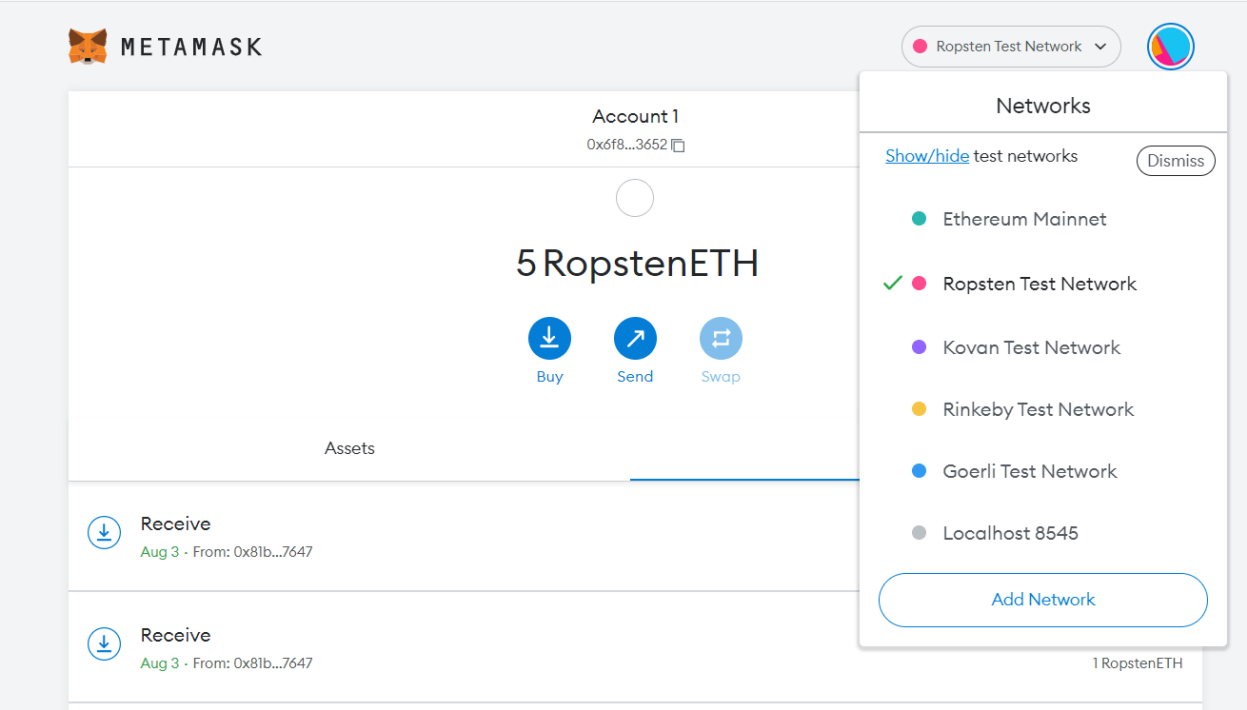
Simply go to your MetaMask, and select BUY:



For this Assignment you need to perform transactions any of the following test networks using metamask,

1) Rinkeby
2) Kovan
3) Ropsten
4) Goerli

You may get free test network ETH by following instructions on MetaMask.



Conclusion: This Assignments enables students to understand and apply blockchain on test cryptocurrency. Ethereum test networks supported by MetalMask is used to create a secure wallet.

Assignment No.	**C4**
Title	Study spending Ether per transaction.
PROBLEM STATEMENT/DEFINITION	Study spending Ether per transaction.
Objectives	Study spending Ether per transaction.
Software packages and hardware apparatus used	MetaMask, Browser
References	**https://metamask.io/** **https://ethereum.org/en/developers/docs** **https://ethereum.org/en/developers/docs/transactions/** **https://ethereum.org/en/developers/docs/gas/**
STEPS	1. Install Metamask on browser 2. Create at least two accounts on same machine or different machines 3. Perform series of transactions 4. Study and Analyze the transactions details
Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Test cases • Program Listing • Output • Conclusion

Theory:

ETHEREUM:

In the Ethereum universe, there is a single, canonical computer (called the Ethereum Virtual Machine, or EVM) whose state everyone on the Ethereum network agrees on. Everyone who participates in the Ethereum network (every Ethereum node) keeps a copy of the state of this computer. Additionally, any participant can broadcast a request for this computer to perform arbitrary computation. Whenever such a request is broadcast, other participants on the network verify, validate, and carry out ("execute") the computation. This execution causes a state change in the EVM, which is committed and propagated throughout the entire network.

Requests for computation are called transaction requests; the record of all transactions and the EVM's present state gets stored on the blockchain, which in turn is stored and agreed upon by all nodes.

Cryptographic mechanisms ensure that once transactions are verified as valid and added to the blockchain, they can't be tampered with later. The same mechanisms also ensure that all transactions are signed and executed with appropriate "permissions" (no one should be able to send digital assets from Alice's account, except for Alice herself).

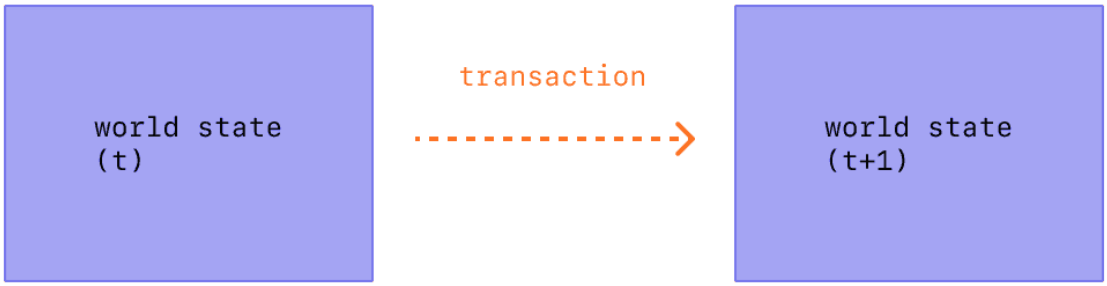
ETHER:
Ether (ETH) is the native cryptocurrency of Ethereum. The purpose of ether is to allow for a market for computation. Such a market provides an economic incentive for participants to verify and execute transaction requests and provide computational resources to the network.

Any participant who broadcasts a transaction request must also offer some amount of ether to the network as a bounty. This bounty will be awarded to whoever eventually does the work of verifying the transaction, executing it, committing it to the blockchain, and broadcasting it to the network.

The amount of ether paid corresponds to the time required to do the computation. These bounties also prevent malicious participants from intentionally clogging the network by requesting the execution of infinite computation or other resource-intensive scripts, as these participants must pay for computation time.

TRANSACTIONS:
Transactions are cryptographically signed instructions from accounts. An account will initiate a transaction to update the state of the Ethereum network. The simplest transaction is transferring ETH from one account to another.

An Ethereum transaction refers to an action initiated by an externally-owned account, in other words an account managed by a human, not a contract. For example, if Bob sends Alice 1 ETH, Bob's account must be debited and Alice's must be credited. This state-changing action takes place within a transaction.

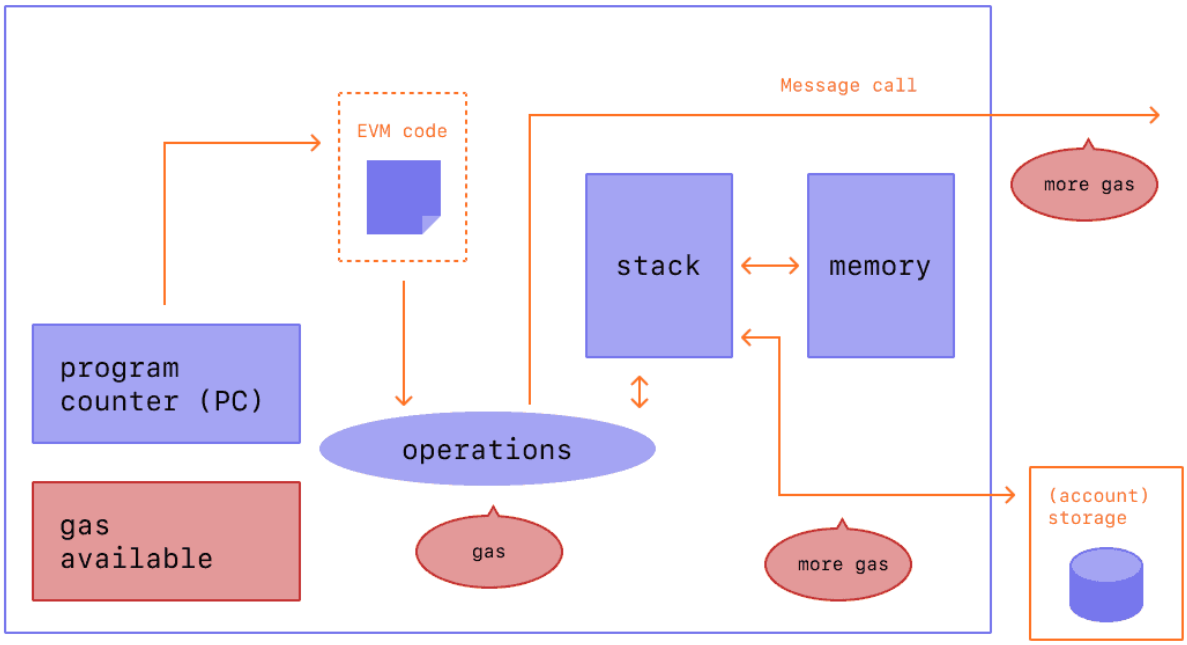


Transactions, which change the state of the EVM, need to be broadcast to the whole network. Any node can broadcast a request for a transaction to be executed on the EVM; after this happens, a miner will execute the transaction and propagate the resulting state change to the rest of the network.
Transactions require a fee and must be mined to become valid. To make this overview simpler we'll cover gas fees and mining elsewhere.
A submitted transaction includes the following information:

- recipient – the receiving address (if an externally-owned account, the transaction will transfer value. If a contract account, the transaction will execute the contract code)
- signature – the identifier of the sender. This is generated when the sender's private key signs the transaction and confirms the sender has authorized this transaction
- value – amount of ETH to transfer from sender to recipient (in WEI, a denomination of ETH)
- data – optional field to include arbitrary data
- gasLimit – the maximum amount of gas units that can be consumed by the transaction. Units of gas represent computational steps
- maxPriorityFeePerGas - the maximum amount of gas to be included as a tip to the miner
- maxFeePerGas - the maximum amount of gas willing to be paid for the transaction (inclusive of baseFeePerGas and maxPriorityFeePerGas)

Gas is a reference to the computation required to process the transaction by a miner. Users have to pay a fee for this computation. The gasLimit, and maxPriorityFeePerGas determine the maximum transaction fee paid to the miner.

WHAT IS GAS?
Gas refers to the unit that measures the amount of computational effort required to execute specific operations on the Ethereum network.
Since each Ethereum transaction requires computational resources to execute, each transaction requires a fee. Gas refers to the fee required to conduct a transaction on Ethereum successfully.



Gas fees are paid in Ethereum's native currency, ether (ETH). Gas prices are denoted in gwei, which itself is a denomination of ETH - each gwei is equal to 0.000000001 ETH (10^{-9} ETH). For example, instead of saying that your gas costs 0.000000001 ether, you can say your gas costs 1 gwei. The word 'gwei' itself means 'giga-wei', and it is equal to 1,000,000,000 wei. Wei itself (named after Wei Dai) is the smallest unit of ETH.

TYPES OF TRANSACTIONS

On Ethereum there are a few different types of transactions:

- Regular transactions: a transaction from one account to another.
- Contract deployment transactions: a transaction without a 'to' address, where the data field is used for the contract code.
- Execution of a contract: a transaction that interacts with a deployed smart contract. In this case, 'to' address is the smart contract address.

The transaction object will look a little like this:

```
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
  gasLimit: "21000",
  maxFeePerGas: "300",
  maxPriorityFeePerGas: "10",
  nonce: "0",
  value: "10000000000"
}
```

But a transaction object needs to be signed using the sender's private key. This proves that the transaction

could only have come from the sender and was not sent fraudulently.
An Ethereum client like Geth will handle this signing process.

ETHEREUM MINING:
Mining is the process of creating a block of transactions to be added to the Ethereum blockchain.
The word mining originates in the context of the gold analogy for crypto currencies. Gold or precious metals are scarce, so are digital tokens, and the only way to increase the total volume is through mining. This is appropriate to the extent that in Ethereum too, the only mode of issuance post launch is via mining. Unlike these examples however, mining is also the way to secure the network by creating, verifying, publishing and propagating blocks in the blockchain.
Mining ether = Securing the Network
Ethereum, like Bitcoin, currently uses a proof-of-work (PoW) consensus mechanism. Mining is the lifeblood of proof-of-work. Ethereum miners - computers running software - using their time and computation power to process transactions and produce blocks.

WHY DO MINERS EXIST?
In decentralized systems like Ethereum, we need to ensure that everyone agrees on the order of transactions. Miners help this happen by solving computationally difficult puzzles to produce blocks, securing the network from attacks.

WHO CAN BECOME A MINER ON ETHEREUM?

Technically, anyone can mine on the Ethereum network using their computer. However, not everyone can mine ether (ETH) profitably. In most cases, miners must purchase dedicated computer hardware to mine profitably. While it is true anyone can run the mining software on their computer, it is unlikely that the average computer would earn enough block rewards to cover the associated costs of mining.

Cost of mining

- Potential costs of the hardware necessary to build and maintain a mining rig
- Electrical cost of powering the mining rig
- If you are mining in a pool, mining pools typically charge a flat % fee of each block generated by the pool
- Potential cost of equipment to support mining rig (ventilation, energy monitoring, electrical wiring, etc.)

To further explore mining profitability, use a mining calculator, such as the one Etherscan provides.

HOW ETHEREUM TRANSACTIONS ARE MINED

1. A user writes and signs a transaction request with the private key of some account.
2. The user broadcasts the transaction request to the entire Ethereum network from some node.
3. Upon hearing about the new transaction request, each node in the Ethereum network adds the request to their local mempool, a list of all transaction requests they've heard about that have not yet been committed to the blockchain in a block.
4. At some point, a mining node aggregates several dozen or hundred transaction requests into a potential block, in a way that maximizes the transaction fees they earn while still staying under the block gas limit. The mining node then:
    1. Verifies the validity of each transaction request (i.e. no one is trying to transfer ether out of an account they haven't produced a signature for, the request is not malformed, etc.), and then executes the code of the request, altering the state of their local copy of the EVM. The miner awards the transaction fee for each such transaction request to their own account.
    2. Begins the process of producing the proof-of-work "certificate of legitimacy" for the potential block, once all transaction requests in the block have been verified and executed on the local EVM copy.
5. Eventually, a miner will finish producing a certificate for a block which includes our specific

transaction request. The miner then broadcasts the completed block, which includes the certificate and a checksum of the claimed new EVM state.

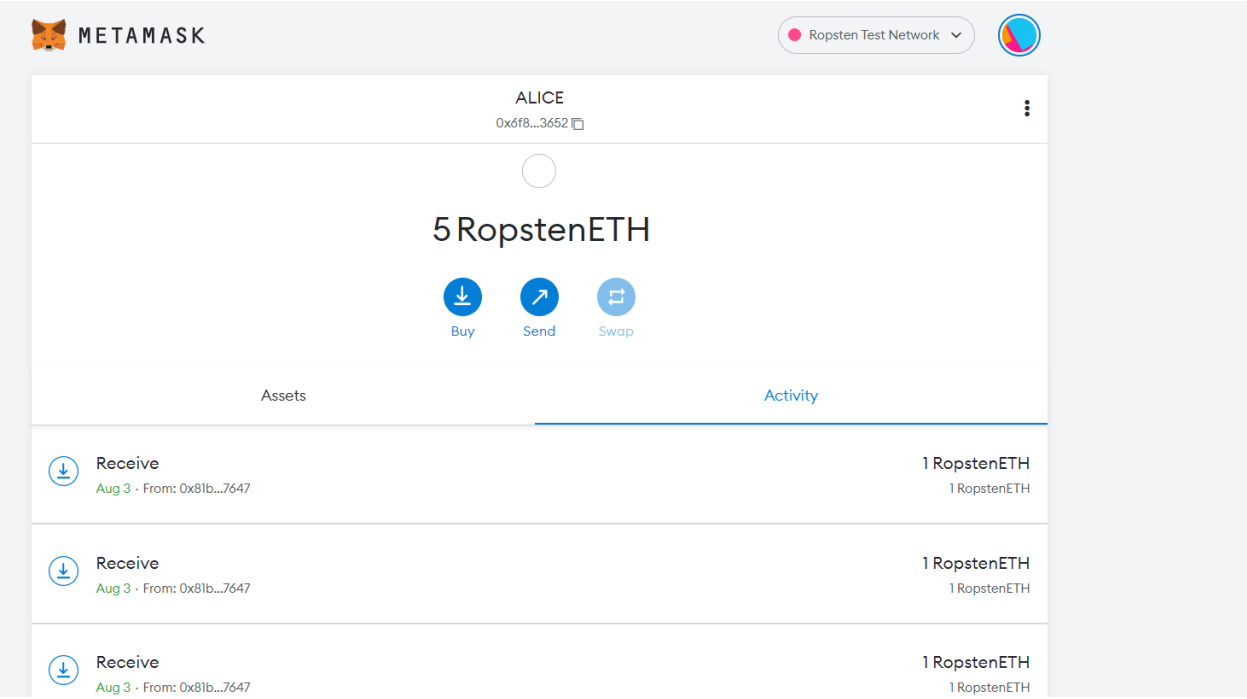
6. Other nodes hear about the new block. They verify the certificate, execute all transactions on the block themselves (including the transaction originally broadcasted by our user), and verify that the checksum of their new EVM state after the execution of all transactions matches the checksum of the state claimed by the miner's block. Only then do these nodes append this block to the tail of their blockchain, and accept the new EVM state as the canonical state.
7. Each node removes all transactions in the new block from their local mempool of unfulfilled transaction requests.
8. New nodes joining the network download all blocks in sequence, including the block containing our transaction of interest. They initialize a local EVM copy (which starts as a blank-state EVM), and then go through the process of executing every transaction in every block on top of their local EVM copy, verifying state checksums at each block along the way.

Every transaction is mined (included in a new block and propagated for the first time) once, but executed and verified by every participant in the process of advancing the canonical EVM state.

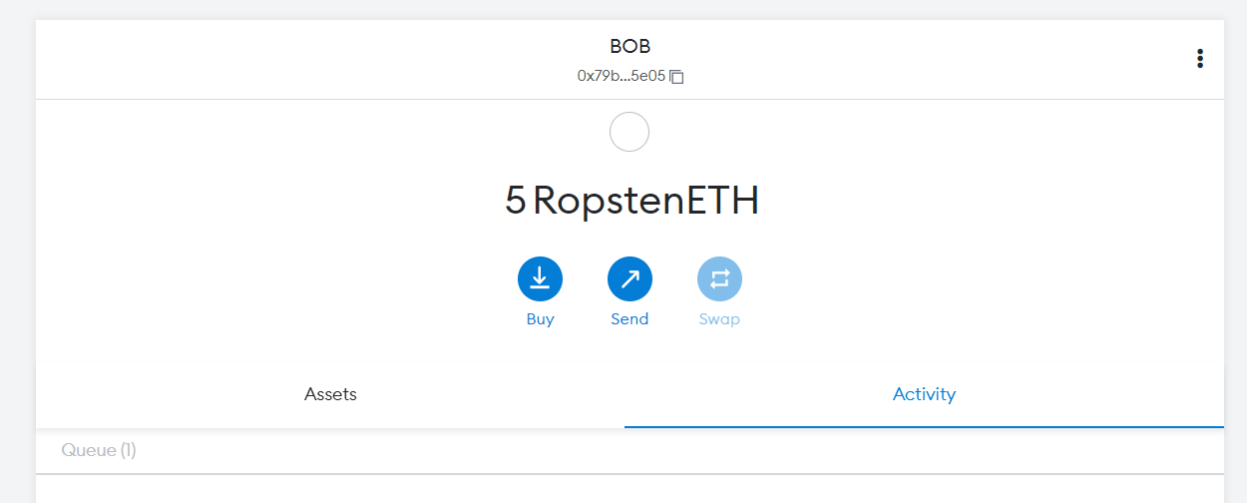
Transaction on Metamask:

In order to perform transaction on MetaMask create at least two sample account for example Alice and bob ash shown in the image below. And buy/ Deposit ETH coin on any test network.

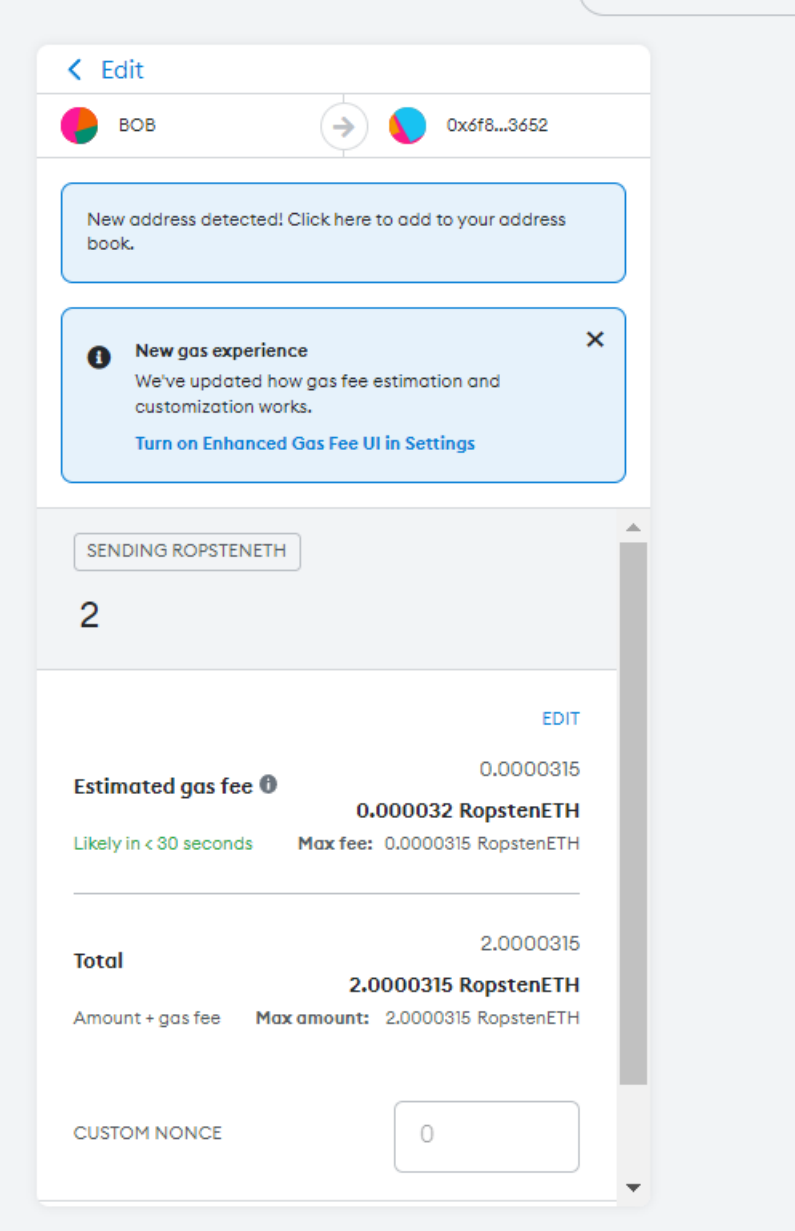
Alice Account:



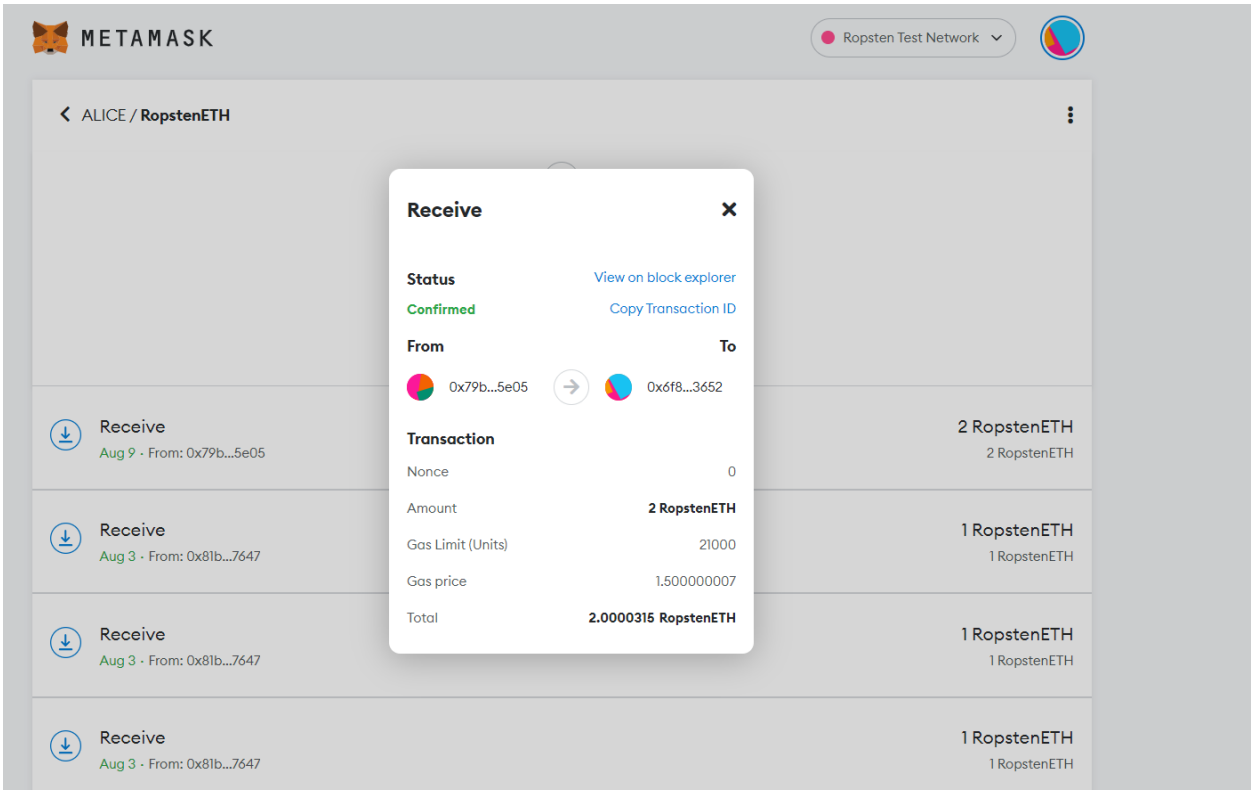
Bob's account



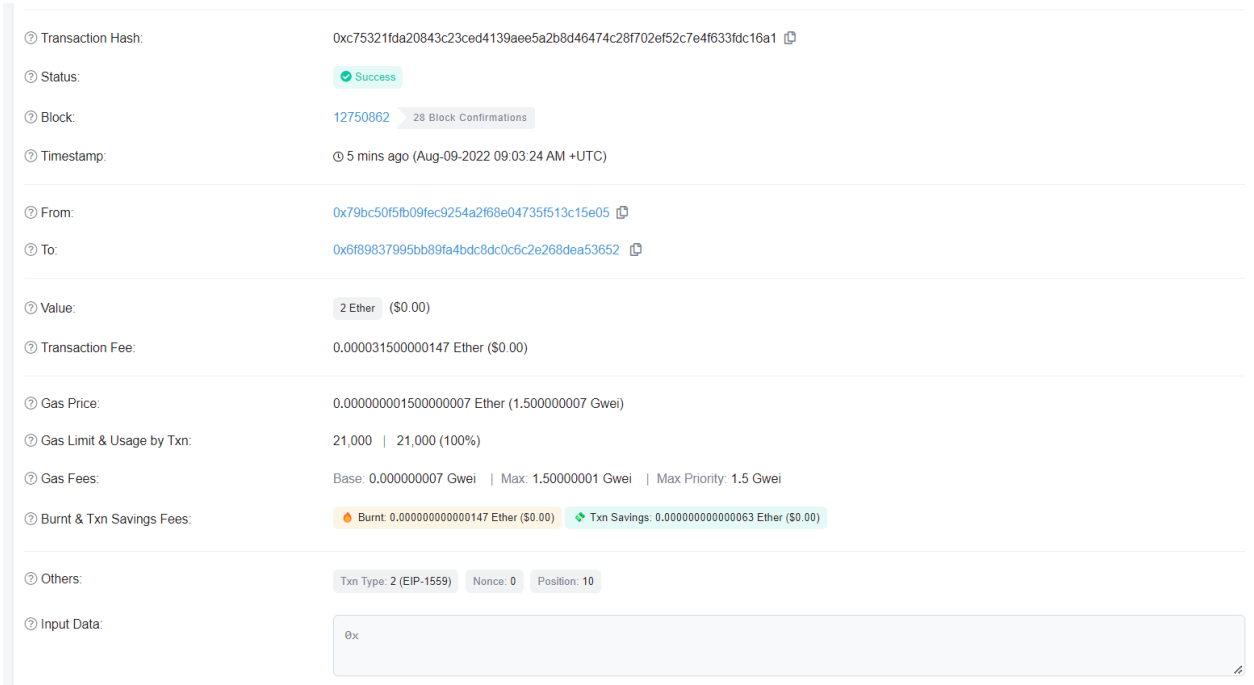
Transfer of 2 Ropsten ETH coin on a test network by Bob to Alice



Ropsten ETH coin received by Alice from BOB



Transaction Details on Etherscan



Conclusion: This assignment enables students to understand blockchain transactions on Ethereum network.