

Assignment No. 1

Depth First Search

Code -

```
#include <iostream>
#include <vector>
#include <stack> #include <omp.h> using
namespace
std;

const int MAX = 100000;
vector<int> graph[MAX]; bool visited[MAX];

void dfs(int node) { stack<int> s;
    s.push(node);

    while (!s.empty()) { int
        curr_node = s.top();

        if (!visited[curr_node]) { visited[curr_node] =
            true;

            s.pop(); cout<<curr_node<<" ";

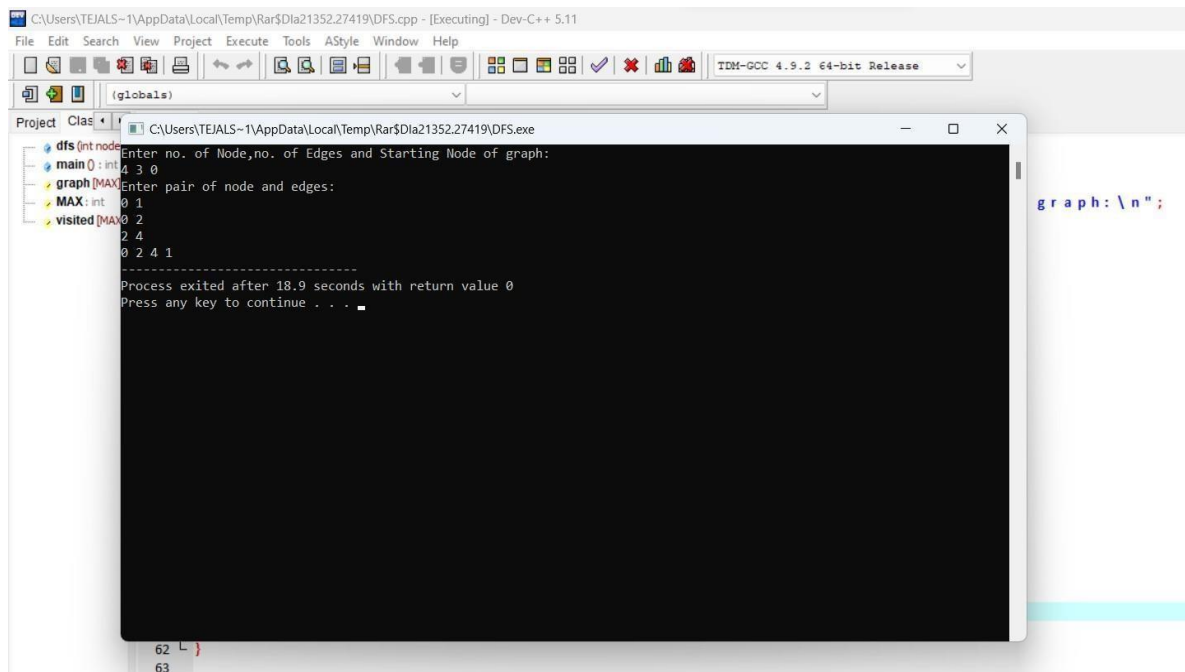
            #pragma omp parallel for for (int i = 0; i <
                graph[curr_node].size(); i++) { int adj_node =
                    graph[curr_node][i]; if (!visited[adj_node]) {
                        s.push(adj_node);
                    }
                }
            }
        }
    }

int main() { int n, m, start_node; cout<<"Enter no. of Node,no. of Edges and Starting Node
    of graph:\n"; cin >> n >> m >> start_node;
    //n: node,m:edges cout<<"Enter pair of node
    and edges:\n";
```

https://colab.research.google.com/drive/1UfSQTP04tbJa0foFJYC_lhRLnMxGMFKZ#printMode=true

```
for (int i = 0; i < m; i++) { int u, v;  
cin >> u >> v;  
  
//u and v: Pair of edges graph[u].push_back(v);  
graph[v].push_back(u);  
}  
  
#pragma omp parallel for for (int i  
= 0; i < n; i++) { visited[i] = false;  
}  
  
dfs(start_node);  
  
return 0;  
}
```

Output –



The screenshot shows a Dev-C++ IDE window titled "C:\Users\TEJALS~1\AppData\Local\Temp\Rar\$Dla21352.27419\DFS.cpp - [Executing] - Dev-C++ 5.11". The IDE is running a program named "DFS.exe". The console window shows the following output:

```
Enter no. of Node,no. of Edges and Starting Node of graph:  
4 3 0  
Enter pair of node and edges:  
0 1  
0 2  
2 4  
0 2 4 1  
-----  
Process exited after 18.9 seconds with return value 0  
Press any key to continue . . .
```

Breadth First Search

Code -

```
#include<iostream>
#include<stdlib.h> #include<queue>
using namespace
std;

class node { public: node *left,
*right; int data;

};

class Breadthfs { public:

node *insert(node *, int); void bfs(node
*);

};

node *insert(node *root, int data)
// inserts a node in tree
{

    if(!root)
    {

        root=new node; root->left=NULL;
        root->right=NULL; root->data=data;
        return root;
    }

    queue<node *> q;
    q.push(root); while(!q.empty())
    {

        node *temp=q.front(); q.pop();

        if(temp->left==NULL)
        {
```

```

        temp->left=new node;
        temp->left->left=NULL; temp->left-
        >right=NULL; temp->left-
        >data=data; return root;
    }
    else
    {

        q.push(temp->left);

    }

    if(temp->right==NULL)
    {

        temp->right=new node;
        temp->right->left=NULL; temp->right-
        >right=NULL; temp-
        >right->data=data; return root;
    }
    else
    {
        q.push(temp->right);

    }

}

}

```

```

void bfs(node *head)
{
    queue<node*> q;

    q.push(head); int qSize;
    while (!q.empty())
    {
        qSize = q.size(); #pragma
        omp parallel for
        //creates parallel threads for (int i = 0; i <
        qSize; i++)
    }
}

```

```

        {
            node* currNode;
            #pragma omp critical
            {
                currNode = q.front(); q.pop();
                cout<<"\t"<<currNode->data;

                }// prints parent node
                #pragma omp critical
                {
                    if(currNode->left)// push parent's left node in queue
                        q.push(currNode->left);
                    if(currNode->right)
                        q.push(currNode->right);
                }// push parent's right node in queue
            }
        }

    }

int main(){

    node *root=NULL; int
    data; char ans;

    do
    {
        cout<<"\n enter data=>"; cin>>data;

        root=insert(root,data);
        cout<<"do you want insert one more node?";

        cin>>ans; }while(ans=='y' || ans=='Y'); bfs(root);

    return 0;
}

```

Output –

```
enter data=> 5
do you want insert one more node? y

enter data=> 3
do you want insert one more node? y

enter data=> 2
do you want insert one more node? y

enter data=> 1
do you want insert one more node? y

enter data=> 7
do you want insert one more node? y

enter data=> 8
do you want insert one more node? n
      5      3      2      1      7      8
-----
Process exited after 50.64 seconds with return value 0
Press any key to continue . . .
```

28 // inserts a node in tree
29 {
30

Assignment No. 2

Bubble Sort

Code -

```
#include<iostream>
#include<stdlib.h>
#include<omp.h> using
namespace std;

void bubble(int *, int); void swap(int
&, int &);

void bubble(int *a, int n)
{ int swapped; for( int i = 0; i < n;
  i++ )
  {
    int first = i % 2; swapped=0;
    #pragma omp parallel for shared(a,first) for( int j
    = first; j < n-1; j += 2 )
    { if( a[ j ] > a[ j+1 ]
      )
      {
        swap( a[ j ], a[ j+1 ] ); swapped=1;
      } }
    if(swapped==0) break;
  }
}

void swap(int &a, int &b)
{
  int test; test=a; a=b;
  b=test;
}

int main()
{
  int *a,n; cout<<"\n enter total no of  elements=>";
  cin>>n; a=new int[n]; cout<<"\n enter
  elements=>"; for(int i=0;i<n;i++)
  { cin>>a[i];
  }
}
```

```
double start_time = omp_get_wtime(); // start timer for sequential algorithm
bubble(a,n); double end_time = omp_get_wtime(); // end timer for sequential
algorithm
```

```
cout<<"\n sorted array is=>"; for(int
i=0;i<n;i++)
{
    cout<<a[i]<<endl;
}
```

```
cout << "Time taken by sequential algorithm: " << end_time - start_time << " seconds" << endl;
```

```
start_time = omp_get_wtime(); // start timer for parallel algorithm
bubble(a,n); end_time = omp_get_wtime(); // end timer for parallel
algorithm
```

```
cout<<"\n sorted array is=>"; for(int
i=0;i<n;i++)
{
    cout<<a[i]<<endl;
}
```

```
cout << "Time taken by parallel algorithm: " << end_time - start_time << " seconds" << endl;
```

```
return 0;
```

```
}
```

Output:

The screenshot shows the PyCharm IDE with a C++ file named `main.py`. The code implements a bubble sort algorithm with timing for both sequential and parallel versions. The sequential version is commented out, and the parallel version is active. The output window shows the sorted array: `[0, 0, 1, 2, 2, 4, 4, 5, 5, 5, 7, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 10, 10, 11, 11, 11, 12, 12, 12, 12, 12, 13, 13, 13, 14, 14, 16, 16, 16]` and the execution time: `Finished in 0.0 second(s)`. The status bar at the bottom indicates the file is in UTF-8 encoding with 4 spaces and is using Python 3.9.

Merge Sort

Code – #include<iostream>

```
#include<stdlib.h> #include<omp.h>
using namespace
std;
```

```
void mergesort(int a[],int i,int j); void merge(int a[],int i1,int
j1,int i2,int j2);
```

```
void mergesort(int a[],int i,int j)
{
    int mid;
    if(i<j)
    { mid=(i+j)/2;

        #pragma omp parallel sections
        {

            #pragma omp section
            {
                mergesort(a,i,mid);
            }

            #pragma omp section
            {
                mergesort(a,mid+1,j);
            }
        }

        merge(a,i,mid,mid+1,j);
    }
}
```

```
void merge(int a[],int i1,int j1,int i2,int j2)
{
    int temp[1000];
    int i,j,k; i=i1;
    j=i2; k=0;
    while(i<=j1 &&
j<=j2)
```

```

    { if(a[i]<a[j])
      {
        temp[k++]=a[i++];
      }
      else
      {
        temp[k++]=a[j++];
      }
    }

    while(i<=j1)
    {
      temp[k++]=a[i++];
    }

    while(j<=j2)
    {
      temp[k++]=a[j++];
    }

    for(i=i1,j=0;i<=j2;i++,j++)
    { a[i]=temp[j];
    }
}

```

```

int main()
{ int *a,n,i;
  double start_time, end_time, seq_time, par_time; cout<<"\n enter total no of
  elements=>"; cin>>n; a= new int[n];

  cout<<"\n enter elements=>"; for(i=0;i<n;i++)
  { cin>>a[i];
  }

  // Sequential algorithm start_time = omp_get_wtime();
  mergesort(a, 0, n-1); end_time = omp_get_wtime(); seq_time
  = end_time - start_time; cout << "\nSequential Time: " <<
  seq_time << endl;

  // Parallel algorithm start_time =
  omp_get_wtime();
  #pragma omp parallel
  {

```

```

#pragma omp single
{
    mergesort(a, 0, n-1);
}
}
end_time = omp_get_wtime(); par_time = end_time -
start_time; cout << "\nParallel Time: " << par_time
<< endl;

cout<<"\n sorted array is=>"; for(i=0;i<n;i++)
{
    cout<<"\n"<<a[i];
}

return 0;
}

```

Output:

The screenshot shows the PyCharm IDE with a C++ project named 'pythonProject'. The file 'main.py' is open, displaying a merge sort algorithm. The code defines a 'merge' function that recursively sorts an array. The main function calls 'mergesort' on an array of 12 elements: 12, 11, 13, 5, 6, 7. The output window at the bottom shows the execution results: 'Given array is 12 11 13 5 6 7', 'Sorted array is 5 6 7 11 12 13', and 'Process finished with exit code 0'.

```

def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m
    L = [0] * (n1)
    R = [0] * (n2)
    for i in range(0, n1):
        L[i] = arr[l + i]
    for j in range(0, n2):
        R[j] = arr[m + 1 + j]
    i = 0
    j = 0
    k = l
    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1
    return arr

```

Run: main

```

"C:\Users\Vaishnavi Nighot\PycharmProjects\pythonProject\venv\Scripts\python.exe" "C:/Users/Vaishnavi Nighot/PycharmProjects/pythonProject/main.py"
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13

Process finished with exit code 0

```

Assignment No. 3

Min, Max, Sum and Average operations using Parallel Reduction

Code -

```
#include <iostream>
#include <vector>
#include <omp.h> #include <climits>

using namespace
std;

void min_reduction(vector<int>& arr) {
    int min_value = INT_MAX;
    #pragma omp parallel for reduction(min: min_value) for (int i
    = 0; i < arr.size(); i++) { if
    (arr[i] < min_value) {
        min_value = arr[i];
    }
    }
    cout << "Minimum value: " << min_value << endl;
}

void max_reduction(vector<int>& arr) {
    int max_value = INT_MIN;
    #pragma omp parallel for reduction(max: max_value) for (int i
    = 0; i < arr.size(); i++) { if
    (arr[i] > max_value) {
        max_value = arr[i];
    }
    }
    cout << "Maximum value: " << max_value << endl;
}

void sum_reduction(vector<int>& arr) {
    int sum = 0;
    #pragma omp parallel for reduction(+: sum) for (int i
    = 0; i < arr.size(); i++) { sum
    += arr[i];
    }
    cout << "Sum: " << sum << endl;
}

void average_reduction(vector<int>& arr) { int sum
    = 0;
    #pragma omp parallel for reduction(+: sum) for (int i
    = 0; i < arr.size(); i++) { sum
```

```

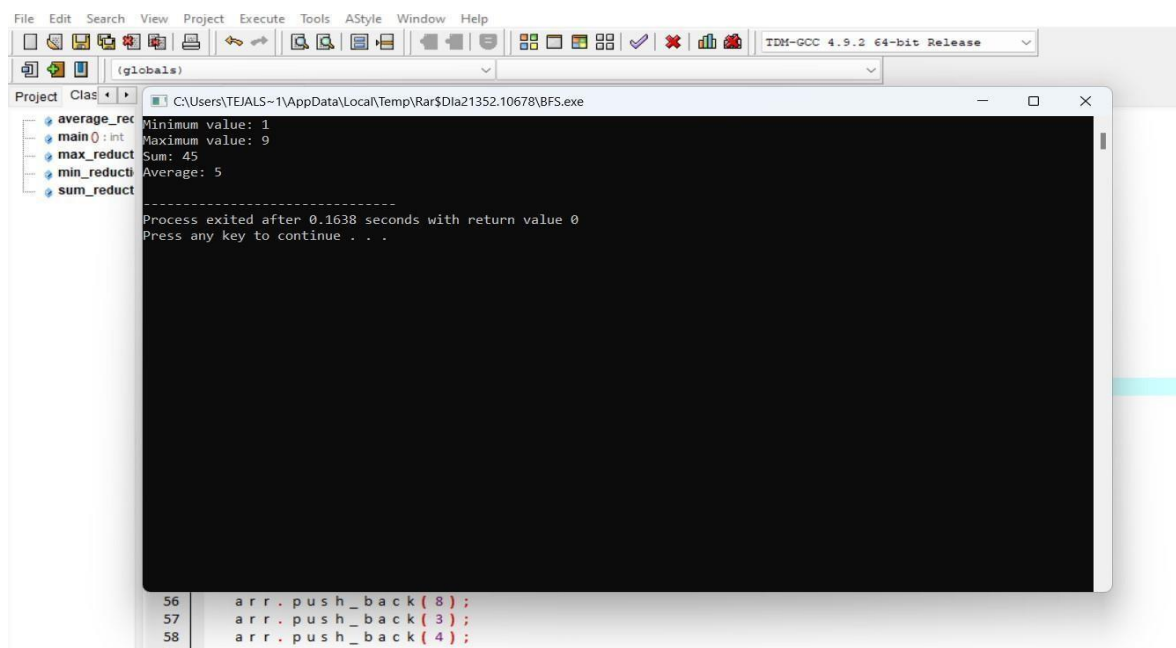
    += arr[i];
}
cout << "Average: " << (double)sum / arr.size() << endl;
}

int main() {
    vector<int> arr; arr.push_back(5); arr.push_back(2);
    arr.push_back(9); arr.push_back(1);
    arr.push_back(7); arr.push_back(6);
    arr.push_back(8); arr.push_back(3);
    arr.push_back(4);

    min_reduction(arr); max_reduction(arr);
    sum_reduction(arr); average_reduction(arr);
}

```

Output –



The screenshot shows a C++ IDE with the following components:

- Menu Bar:** File, Edit, Search, View, Project, Execute, Tools, AStyle, Window, Help.
- Toolbar:** Standard IDE icons for file operations, execution, and debugging.
- Project Explorer:** Shows a project named 'globals' with a file 'main0.cpp'.
- Output Window:** Displays the program's output:


```

Minimum value: 1
Maximum value: 9
Sum: 45
Average: 5
-----
Process exited after 0.1638 seconds with return value 0
Press any key to continue . . .
      
```
- Code Editor:** Shows the source code with lines 56, 57, and 58 highlighted:


```

56     arr.push_back(8);
57     arr.push_back(3);
58     arr.push_back(4);
      
```

Assignment No 6

```
#Importing the pandas for data processing and numpy for
numerical computing import numpy as np import pandas as pd
# Importing the Boston Housing dataset from the sklearn from
sklearn.datasets import load_boston boston = load_boston()
#Converting the data into pandas dataframe data
= pd.DataFrame(boston.data)
#First look at the data data.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

```
#Adding the feature names to the dataframe
data.columns = boston.feature_names #Adding
the target variable to the dataset data['PRICE']
= boston.target
#Looking at the data with names and target variable data.head(n=10)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21	28.7
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43	22.9
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15	27.1
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93	16.5
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10	18.9

```
print(data.shape)
```

```
#Checking the null values in the  
dataset data.isnull().sum()
```

```
data.describe()
```

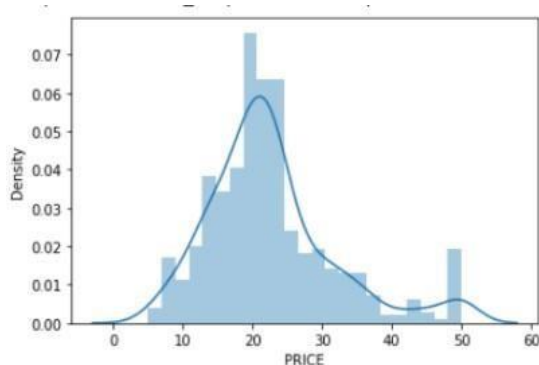
```
# This is sometimes very useful, for example if you look at the CRIM the max is  
88.97 and 75% of the value is below 3.677083 and
```

```
# mean is 3.613524 so it means the max values is actually an outlier or there  
are outliers present in the column import seaborn as sns
```

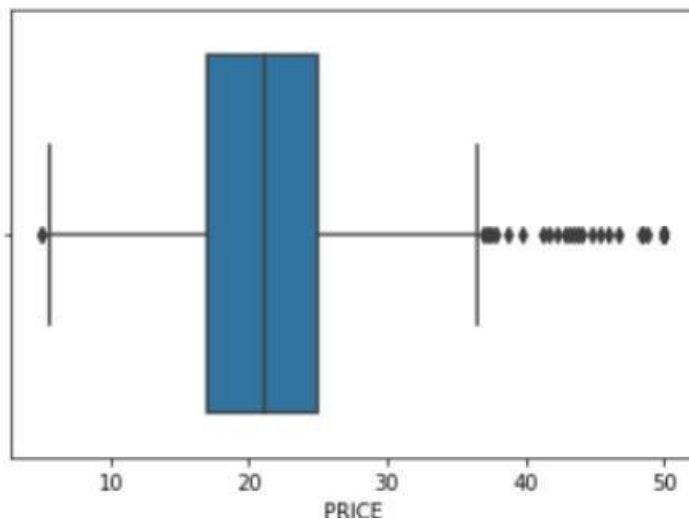
```
sns.distplot(data.PRICE)
```

```
#The distribution seems normal, has not be the data normal we would have perform  
log transformation or took to square root of the data to make the data normal. #
```

```
Normal distribution is need for the machine learning for better predictblity of the model
```



```
sns.boxplot(data.PRICE)
```



```
#checking Correlation of the data correlation
```

```
= data.corr() correlation.loc['PRICE']
```

```
CRIM -0.388305
```

```
ZN 0.360445
```

```
INDUS -0.483725
```

```
CHAS 0.175260
```

```
NOX -0.427321
```

RM 0.695360

AGE -0.376955 DIS

0.249929

RAD -0.381626

TAX -0.468536

PTRATIO -0.507787

B 0.333461

LSTAT -0.737663

PRICE 1.000000

plotting the heatmap import matplotlib.pyplot as plt

fig, axes = plt.subplots(figsize=(15,12))

sns.heatmap(correlation, square = True, annot = True)

By looking at the correlation plot LSTAT is negatively correlated with -0.75 and

RM is positively correlated to the price and PTRATIO is correlated negatively with

-0.51



plt.figure(figsize = (20,5)) features =

['LSTAT','RM','PTRATIO'] for i, col

in enumerate(features):


```
plt.subplot(1, len(features) , i+1) x =
data[col] y = data.PRICE
plt.scatter(x, y, marker='o')
plt.title("Variation in House prices")
plt.xlabel(col) plt.ylabel("House
prices in $1000")
```



Splitting the dependent feature and independent feature

```
#X = data[['LSTAT','RM','PTRATIO']]
```

```
X = data.iloc[:, :-1] y= data.PRICE
```

In order to provide a standardized input to our neural network, we need to perform the normalization of our dataset.

This can be seen as a step to reduce the differences in scale that may arise from the existent features.

We perform this normalization by subtracting the mean from our data and dividing it by the standard deviation.

```
mean = X_train.mean(axis=0) std =
```

```
X_train.std(axis=0)
```

```
X_train = (X_train - mean) / std
```

```
X_test = (X_test - mean) / std #Linear Regression from
```

```
sklearn.linear_model import LinearRegression regressor =
```

```
LinearRegression()
```

```
#Fitting the model regressor.fit(X_train,y_train) #
```

```
Model Evaluation #Prediction on the test dataset
```

```
y_pred = regressor.predict(X_test) # Predicting RMSE
```

```
the Test set results from sklearn.metrics import
```

```
mean_squared_error rmse =
```

```
(np.sqrt(mean_squared_error(y_test, y_pred))) print(rmse)
```

```
from sklearn.metrics import r2_score
```

```

r2 = r2_score(y_test, y_pred) print(r2)

# Neural Networks #Scaling the
dataset

from sklearn.preprocessing import StandardScaler sc =
StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Due to the small amount of presented data in this dataset, we must be careful to
not create an overly complex model,
# which could lead to overfitting our data. For this, we are going to adopt an architecture
based on two Dense layers,
# the first with 128 and the second with 64 neurons, both using a ReLU activation function.
#Creating the neural network model import
keras

from keras.layers import Dense, Activation,Dropout
from keras.models import Sequential model = Sequential()

model.add(Dense(128,activation = 'relu',input_dim =13))
model.add(Dense(64,activation
= 'relu'))

SNJB's Late Sau. K.B. Jain College Of Engineering

model.add(Dense(32,activation = 'relu'))

model.add(Dense(16,activation = 'relu')) model.add(Dense(1))

#model.compile(optimizer='adam', loss='mse', metrics=['mae']) model.compile(optimizer
= 'adam',loss ='mean_squared_error',metrics=['mae'])

!pip install ann_visualizer !pip install
graphviz from ann_visualizer.visualize

import ann_viz; #Build your model here

ann_viz(model, title="DEMO ANN");

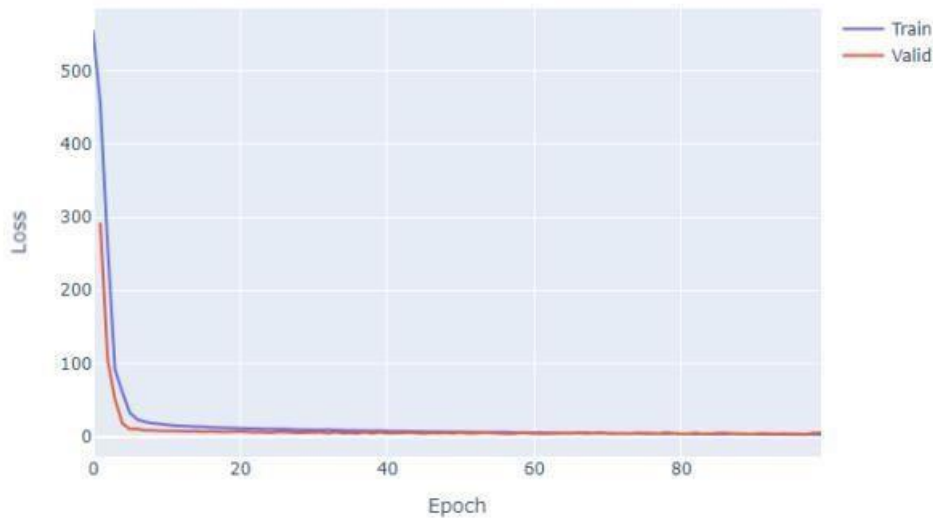
history = model.fit(X_train, y_train, epochs=100, validation_split=0.05) # By
plotting both loss and mean average error, we can see that our model was
capable of learning patterns in our data without overfitting taking place (as shown
by the validation set curves) from plotly.subplots import make_subplots import
plotly.graph_objects as go fig = go.Figure()

fig.add_trace(go.Scattergl(y=history.history['loss'], name='Train'))

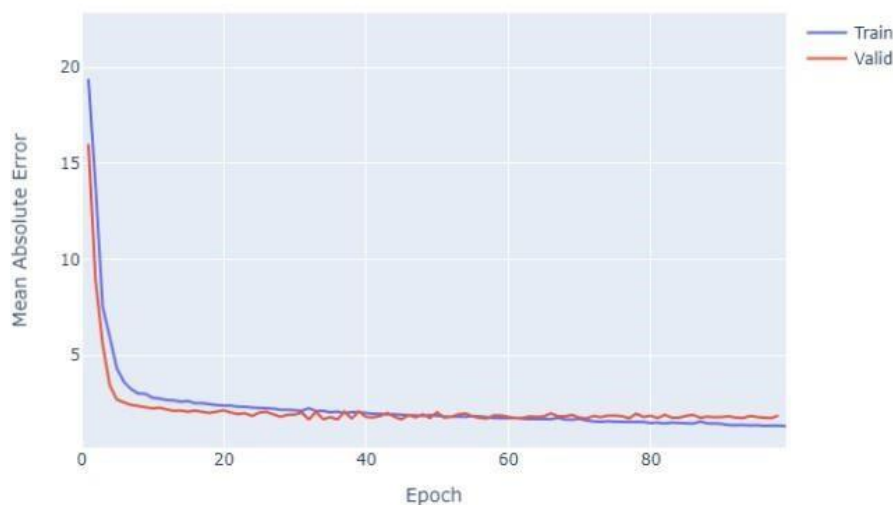
fig.add_trace(go.Scattergl(y=history.history['val_loss'], name='Valid'))

```

```
fig.update_layout(height=500, width=700,
axis_title='Epoch', yaxis_title='Loss') fig.show()
```



```
fig.add_trace(go.Scattergl(y=history.history['mae'], name='Train'))
fig.add_trace(go.Scattergl(y=history.history['val_mae'], name='Valid')) fig.update_layout(height=500,
width=700,
axis_title='Epoch', yaxis_title='Mean
Absolute Error') fig.show()
```



```
#Evaluation of the model y_pred =
model.predict(X_test) mse_nn, mae_nn =
model.evaluate(X_test, y_test) print('Mean
squared error on test data: ', mse_nn) print('Mean
absolute error on test data: ', mae_nn)
```

4/4 [=====] - 0s 4ms/step - loss: 10.5717 - mae: 2.2670

Mean squared error on test data: 10.571733474731445

Mean absolute error on test data: 2.2669904232025146

```

#Comparison with traditional approaches

#First let's try with a simple algorithm, the Linear Regression:

from sklearn.metrics import mean_absolute_error

lr_model = LinearRegression() lr_model.fit(X_train,
y_train) y_pred_lr = lr_model.predict(X_test)

mse_lr = mean_squared_error(y_test, y_pred_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)
print('Mean squared error on test data: ', mse_lr)
print('Mean absolute error on test data: ', mae_lr)

from sklearn.metrics import r2_score r2 =
r2_score(y_test, y_pred) print(r2)

0.8812832788381159

# Predicting RMSE the Test set results from

sklearn.metrics import mean_squared_error rmse =
(np.sqrt(mean_squared_error(y_test, y_pred))) print(rmse)

3.320768607496587 # Make
predictions on new data import sklearn

new_data = sklearn.preprocessing.StandardScaler().fit_transform([[0.1, 10.0,
5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]])

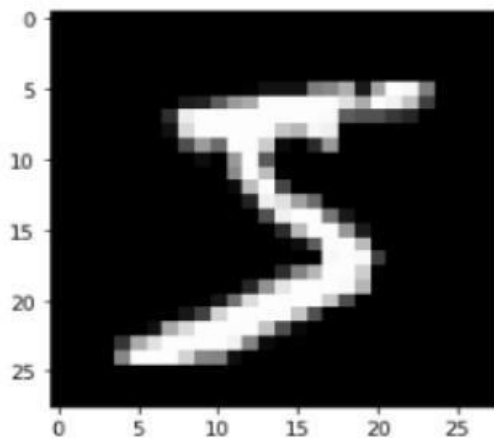
prediction = model.predict(new_data) print("Predicted
house price:", prediction)

Predicted house price: [[11.104753]

```

Assignment No : 7

```
import numpy as np from tensorflow.keras.models
import Sequential from tensorflow.keras.layers import
Dense, Dropout from tensorflow.keras.optimizers
import RMSprop from tensorflow.keras.datasets
import mnist import matplotlib.pyplot as plt from
sklearn import metrics
# Load the OCR dataset
# The training set consists of 60,000 images, while the test set has 10,000 images.
(x_train, y_train), (x_test, y_test) = mnist.load_data() plt.imshow(x_train[0],
cmap='gray') # imshow() function which simply displays an image. plt.show()
print(x_train[0])
```

[illegible]

```
# reformat our X_train array and our X_test array because they do not have the correct shape.
# Reshape the data to fit the model

print("X_train shape", x_train.shape)

print("y_train shape", y_train.shape)

print("X_test shape", x_test.shape) print("y_test
shape", y_test.shape) X_train shape (60000,
28, 28) y_train shape (60000,)
X_test shape (10000, 28, 28) y_test shape
(10000,)
```

```

# X: Training data of shape (n_samples, n_features)
# y: Training label values of shape (n_samples, n_labels)
# Whereas X_test has 10,000 elements, each with each with 784 total pixels so will become shape
(10000, 784). x_train = x_train.reshape(60000, 784) x_test
= x_test.reshape(10000, 784) x_train = x_train.astype('float32') # use 32-bit precision when training a
neural network, so at one point the training data will have to be converted to 32 bit floats. Since the
dataset fits easily in RAM, we might as well convert to float immediately. x_test = x_test.astype('float32')
x_train /= 255 # Each image has Intensity from 0 to 255 x_test /=
255
# Convert class vectors to binary class matrices num_classes
= 10 y_train = np.eye(num_classes)[y_train] # Return a 2-D array with ones on the diagonal and
zeros
elsewhere. y_test = np.eye(num_classes)[y_test] # f your particular categories is present then it mark
as 1 else 0 in remain row
# Define the model architecture model
= Sequential() model.add(Dense(512, activation='relu', input_shape=(784,))) # Input consist of 784
Neuron ie 784 input,
512 in the hidden layer
model.add(Dropout(0.2)) # DROP OUT RATIO 20%
model.add(Dense(512, activation='relu')) #returns a sequence of another vectors of dimension 512
model.add(Dropout(0.2)) model.add(Dense(num_classes, activation='softmax')) # 10 neurons ie
output node in the output layer.
# Compile the model
model.compile(loss='categorical_crossentropy', # for a multi-class classification problem
optimizer=RMSprop(), metrics=['accuracy'])
# Train the model batch_size = 128 # batch_size argument is passed to the layer to define a batch
size for the inputs.
epochs = 20
history = model.fit(x_train, y_train,
batch_size=batch_size, epochs=epochs,
verbose=1, # verbose=1 will show you
an animated progress bar eg.
[=====]

```

```
validation_data=(x_test, y_test)) # Using validation_data means you are providing the training
set and validation set yourself,
# 60000image/128=469 batch each
# Evaluate the model
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0]) print('Test accuracy:', score[1])
```

Test loss: 0.08541901409626007

Test accuracy: 0.9851999878883362

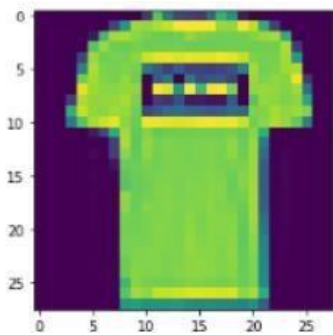
Assignment No : 8

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras
import numpy as np

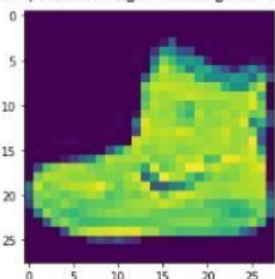
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

# There are 10 image classes in this dataset and each class has a mapping corresponding to the following labels:
#0 T-shirt/top
#1 Trouser
#2 pullover
#3 Dress
#4 Coat
#5 sandals
#6 shirt
#7 sneaker #8 bag
#9 ankle boot

plt.imshow(x_train[1]
)
```



```
plt.imshow(x_train[0])
<matplotlib.image.AxesImage at 0x7f8584b93d00>
```



```
# Next, we will preprocess the data by scaling the pixel values to
be between 0 and 1, and then reshaping the images to be 28x28
pixels. x_train = x_train.astype('float32') / 255.0 x_test =
x_test.astype('float32') / 255.0 x_train =
x_train.reshape(-1, 28, 28, 1) x_test = x_test.reshape(-1,
28, 28, 1)
```



```

# 28, 28 comes from width, height, 1 comes from the number of channels # -1
means that the length in that dimension is inferred.
# This is done based on the constraint that the number of elements in an ndarray or Tensor when reshaped must
remain the same.
# each image is a row vector (784 elements) and there are lots of such rows (let it be n, so there are 784n
elements). So TensorFlow can infer that -1 is n.
# converting the training_images array to 4 dimensional array with sizes 60000, 28, 28, 1 for 0th to
3rd dimension. x_train.shape (60000, 28, 28) x_test.shape (10000, 28, 28, 1) y_train.shape (60000,)
y_test.shape (10000,)
# We will use a convolutional neural network (CNN) to classify the fashion
items. # The CNN will consist of multiple convolutional layers followed by max
pooling, # dropout, and dense layers. Here is the code for the model: model =
keras.Sequential([
keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
# 32 filters (default), randomly initialized
# 3*3 is Size of Filter
# 28,28,1 size of Input Image
# No zero-padding: every output 2 pixels less in every dimension
# in Paramter shwon 320 is value of weights: (3x3 filter weights + 32 bias) * 32 filters
# 32*3*3=288(Total)+32(bias)= 320 keras.layers.MaxPooling2D((2,2)),
# It shown 13 * 13 size image with 32 channel or filter or depth. keras.layers.Dropout(0.25),
# Reduce Overfitting of Training sample drop out 25% Neuron
keras.layers.Conv2D(64, (3,3), activation='relu'),
# Deeper layers use 64 filters
# 3*3 is Size of Filter
# Observe how the input image on 28x28x1 is transformed to a 3x3x64 feature map
# 13(Size)-3(Filter Size )+1(bias)=11 Size for Width and Height with 64 Depth or filter or channel
# in Paramter shwon 18496 is value of weights: (3x3 filter weights + 64 bias) * 64 filters
# 64*3*3=576+1=577*32 + 32(bias)=18496 keras.layers.MaxPooling2D((2,2)),
# It shown 5 * 5 size image with 64 channel or filter or depth. keras.layers.Dropout(0.25),
keras.layers.Conv2D(128, (3,3), activation='relu'),
# Deeper layers use 128 filters
# 3*3 is Size of Filter
# Observe how the input image on 28x28x1 is transformed to a 3x3x128 feature map # It
show 5(Size)-3(Filter Size )+1(bias)=3 Size for Width and Height with 64 Depth or filter or
channel
# 128*3*3=1152+1=1153*64 + 64(bias)= 73856
# To classify the images, we still need a Dense and Softmax layer. #
We need to flatten the 3x3x128 feature map to a vector of size 1152
keras.layers.Flatten(), keras.layers.Dense(128, activation='relu'),
# 128 Size of Node in Dense Layer
# 1152*128 = 147584 keras.layers.Dropout(0.25), keras.layers.Dense(10,
activation='softmax')
# 10 Size of Node another Dense Layer
# 128*10+10 bias= 1290
])

```

```
model.summary()
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
dropout (Dropout)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_1 (Dropout)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 128)	73856
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 128)	147584
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290

Total params: 241,546

Trainable params: 241,546

Non-trainable params: 0

Compile and Train the Model

After defining the model, we will compile it and train it on the training data.

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test)) # 1875 is a number of batches. By default batches contain 32 samples. 60000 / 32 = 1875 # Finally, we will evaluate the performance of the model on the test data. test_loss, test_acc =
```

```
model.evaluate(x_test, y_test)
```

```
print('Test accuracy:',
```

```
test_acc)
```

```
313/313 [=====] - 3s 10ms/step - loss: 0.2606 - accuracy: 0.9031
```

```
Test accuracy: 0.9031000137329102
```

Mini Project on gender and age detection

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2

# Define constants
img_height = 128
img_width = 128
batch_size = 32
epochs = 32
num_classes = 10

# Load the "UTKFace" dataset
df = pd.read_csv('UTKFace.csv')
df['age'] = df['age'].apply(lambda x: min(x, 100)) # limit age to 100
df = df.sample(frac=1).reset_index(drop=True) # shuffle the dataset
df['image_path'] = 'UTKFace/' + df['image_path']
df_train = df[:int(len(df)*0.8)] # 80% for training
df_val = df[int(len(df)*0.8):int(len(df)*0.9)] # 10% for validation
df_test = df[int(len(df)*0.9):] # 10% for testing

# Define data generators for training, validation, and testing sets
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_dataframe(
    dataframe=df_train,
    x_col='image_path', y_col=['male', 'age'],
    target_size=(img_height, img_width),
    batch_size=batch_size, class_mode='raw')
val_generator = val_datagen.flow_from_dataframe(
    dataframe=df_val, x_col='image_path', y_col=['male', 'age'],
    target_size=(img_height, img_width),
    batch_size=batch_size, class_mode='raw')
test_generator = test_datagen.flow_from_dataframe(
    dataframe=df_test, x_col='image_path', y_col=['male', 'age'],
    target_size=(img_height, img_width), batch_size=batch_size,
    class_mode='raw')

# Define the neural network model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(img_height, img_width, 3)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
```

```

MaxPooling2D((2,2)),
Conv2D(128, (3,3), activation='relu'),
MaxPooling2D((2,2)),
Conv2D(128, (3,3), activation='relu'),
MaxPooling2D((2,2)),
Flatten(),
Dropout(0.5),
Dense(512, activation='relu'),
Dense(2)
])

# Compile the model model.compile(optimizer='adam', loss={'dense_1':
'binary_crossentropy', 'dense_2': 'mse'}, metrics={'dense_1': 'accuracy',
'dense_2': 'mae'})

# Train the model history =
model.fit(train_generator,
epochs=epochs,
validation_data=val_generator)

# Evaluate the model on the test set loss, accuracy, mae =
model.evaluate(test_generator) print("Test accuracy:", accuracy)
print("Test MAE:", mae)

# Predict the gender and age of a sample image
img = cv2.imread('sample_image.jpg') img

```

Mini Project
On
**PARALLEL IMPLEMENTATION AND EVALUATION OF QUICK
SORT USING OPENMP**

1. PROPOSED ALGORITHM

In general, the overall algorithm used here to perform QuickSort with MPI works as followed:

- i. Start and initialize MPI.
- ii. Under the root process MASTER, get inputs:
 - a. Read the list of numbers L from an input file.
 - b. Initialize the main array globaldata with L.
 - c. Start the timer.
- iii. Divide the input size SIZE by the number of participating processes npes to get each chunk size localsize.
- iv. Distribute globaldata proportionally to all processes:
 - a. From MASTER scatter globaldata to all processes.
 - b. Each process receives in a sub data localdata.
- v. Each process locally sorts its localdata of size localsize.
- vi. Master gathers all sorted localdata by other processes in globaldata.
 - a. Gather each sorted localdata.
 - b. Free localdata.
- vii. Under MASTER perform a final sort of globaldata.
 - a. Final sort of globaldata.
 - b. Stop the timer.
 - c. Write the output to file.
 - d. Sequentially check that globaldata is properly and correctly sorted.
 - e. Free globaldata.
- viii. Finalize MPI.

2. IMPLEMENTATION

In order to implement the above algorithm using C programming, we have made use of a few MPI collective routine operations. Therefore after initializing MPI with MPI_Init, the size and rank are obtained respectively using MPI_Comm_size and MPI_Comm_rank. The beginning wall time is received from MPI_Wtime and the array containing inputs is distributed proportionally to the size to all participating processes with MPI_Scatter by the root process which collect them again after they are sorted using MPI_Gather. Finally the ending wall time is retrieved again from MPI_Wtime and the MPI terminate calling MPI_Finalize. In the following part, each section of our proposed algorithm is illustrated with a code snippet taken from the implementation source code.

- i. Start and initialize MPI.

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &npes);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

- ii. Under the root process MASTER, get inputs:

```
if (rank == MASTER) {
    printf("SIZE IS %lld", SIZE);
    globaldata = malloc (SIZE * sizeof(long long) );
    if (globaldata == NULL) {
        printf ("\n\n globaldata Memory Allocation Failed ! \n\n ");
        exit(EXIT_FAILURE);
    }
    inp = fopen (argv[1], "r");          /* Open file for reading */
    if (inp == NULL) {
        printf ("\n\n inp Memory Allocation Failed ! \n\n ");
        exit(EXIT_FAILURE);
    }
    printf ("\n\nInput Data \n\n ");
    for (i=0; i<SIZE; i++) {
        retscan = fscanf (inp, "%lld \t", &tmp);
        globaldata[i] = tmp;
    }
}
```

Starting the timer.

```
if (rank == MASTER)
{
    t_start = MPI_Wtime ();
}
```

- i. Divide the input size SIZE by the number of participating processes npes to get each chunk size localsize.

```
/*Getting the size to be used by each process */
if (SIZE < npes) {
    printf ("\n\n SIZE is less than the number of process!  \n\n ");
    exit (EXIT_FAILURE);
}
localsize = SIZE/npes;
```

- ii. Distribute globaldata proportionally to all processes:

```
/*Scatter the integers to each number of processes (npes) */
MPI_Scatter (globaldata, localsize, MPI_LONG_LONG, localdata,
            localsize, MPI_LONG_LONG, MASTER, MPI_COMM_WORLD);
```

- i. Each process locally sorts its localdata of size localsize.

```
/* Perform local sort on each sub data by each process */
quickSortRecursive (localdata,0, localsize-1);
```

- ii. Master gathers all sorted localdata by other processes in globaldata.

```
/* Merge locally sorted data of each process by MASTER to globaldata */
MPI_Gather (localdata, localsize, MPI_LONG_LONG , globaldata,
            localsize, MPI_LONG_LONG, MASTER, MPI_COMM_WORLD);
free (localdata);
```

- iii. Under MASTER perform a final sort of globaldata.

```
if (rank == MASTER) {
    /* Final sorting */
    quickSortRecursive (globaldata, 0, SIZE-1);
}
```

Stop the timer

```
/* End wall time */
t_end = MPI_Wtime ();
```

Write information to in the output file

```
/* Opening output file to write sorted data */
out = fopen (argv[2], "w");
if (out == NULL) {
    printf ("\n\n out Memory Allocation Failed ! \n\n ");
    exit (EXIT_FAILURE);
}
/* Write information to output file */
fprintf (out, "Recursively Sorted Data : ");
fprintf (out, "\n\nInput size : %lld\t", SIZE);
fprintf (out, "\n\nNber processes : %d\t", npes);
fprintf (out, "\n\nWall time : %7.4f\t", t_end - t_start);
printf ("\n\nWall time : %7.4f\t", t_end - t_start);
fprintf (out, "\n\n");
for (i = 0; i<SIZE; i++) {
    fprintf (out, " %lld \t", globaldata[i]);
}
fclose (out); /* closing the file */
```

Checking that the final globaldata array is properly sorted.

```
/* checking if the final globaldata content is properly sorted */
sortCheckers ( SIZE, globaldata );
```

Free the allocated memory

```
if (rank == MASTER) {
    free (globaldata); /* free the allocated memory */
}
```

ix. Finalize MPI.

```
/* MPI_Finalize Terminates MPI execution environment */
MPI_Finalize ();
```

The two versions of QuickSort algorithm have been implemented; however even though they have almost the same implementation using similar functions, the recursion in the recursive part has been replaced by a stack in order to make it iterative. Their function signatures are presented in the following part:

- Recursive QuickSort


```

void quickSortRecursive (long long [], long long, long long);
long long partition (long long [], long long , long long );
void swap (long long [], long long , long long );
void sortCheckers (long long, long long []);
long long getSize ( char str[] );

```

- Iterative QuickSort

```

void quickSortIterative (long long [], long long, long long);
long long partition (long long [], long long , long long );
void swap (long long [], long long , long long );
void sortCheckers (long long, long long []);
long long getSize ( char str[] );

```

The input file necessary to run this program can be generated using an input generator where we specify the size and the filename (input_generator.c), then compile and run with the following instructions:

3. RESULTS

The following table presents the different recorded data. In the first column we have the experiment number (No.); the second column is the number of participating processes (# process), the third column is the input data size applied to QuickSort. Finally the last two columns represent respective the execution wall time of the iterative and recursive version of parallel QuickSort.

No.	# process	Input size	Iterative wall time	Recursive wall time
1	1	20	0.0000	0.0000
	2		0.0001	0.0000
	5		0.0002	0.0004
	10		0.0004	0.0005
	20		0.0013	0.0032
2	1	100	0.0000	0.0000
	2		0.0001	0.0001
	5		0.0002	0.0004

	10		0.0003	0.0005
	20		0.0016	0.0020
3	1	1000	0.0011	0.0012
	2		0.0003	0.0003
	5		0.0004	0.0004
	10		0.0007	0.0008
	20		0.0014	0.0016
4	1	10000	0.0849	0.0860
	2		0.0030	0.0030
	5		0.0031	0.0030
	10		0.0038	0.0035
	20		0.0035	0.0043
5	1	100000	8.2165	8.5484
	2		0.0393	0.0383
	5		0.0333	0.0325
	10		0.0418	0.0488
	20		0.0446	0.0475
6	1	1000000	835.8316	2098.7

	2		0.4786	0.4471
	5		0.3718	0.3590
	10		0.3646	0.3445
	20		0.4104	0.3751

4. VISUALISATION

Different charts comparing the iterative QuickSort and the recursive QuickSort for different number of processes and various input sizes are presented in this section. The X-axis represents the number of processes and the Y-axis represents the execution wall time in seconds.

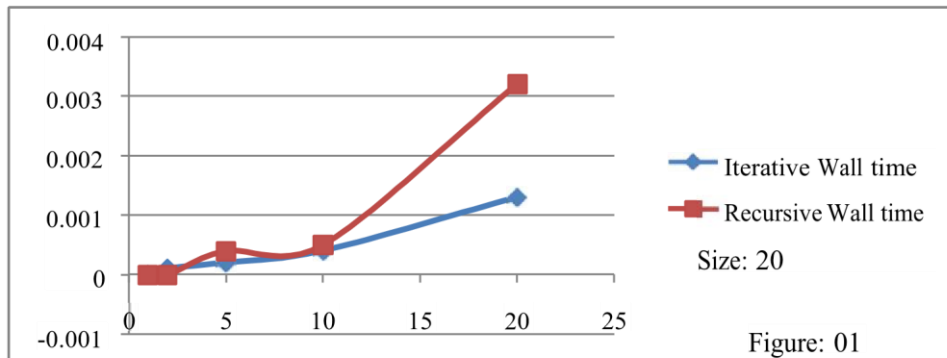


Figure: 01

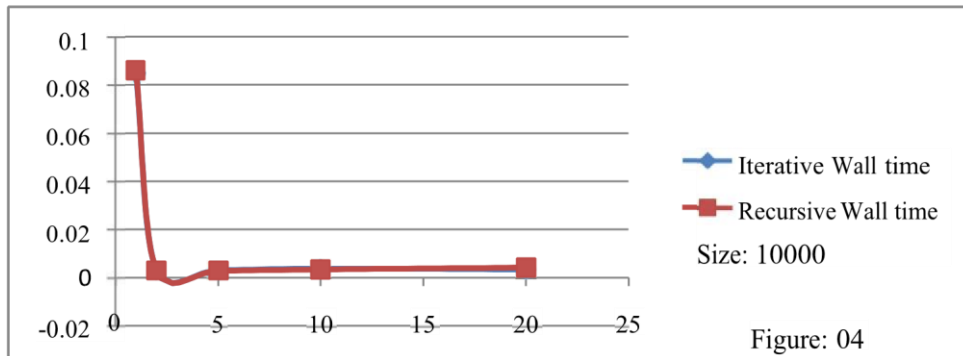


Figure: 04

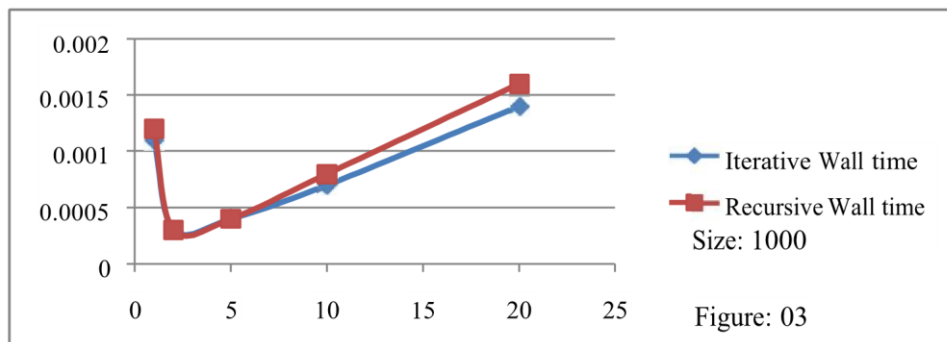


Figure: 03

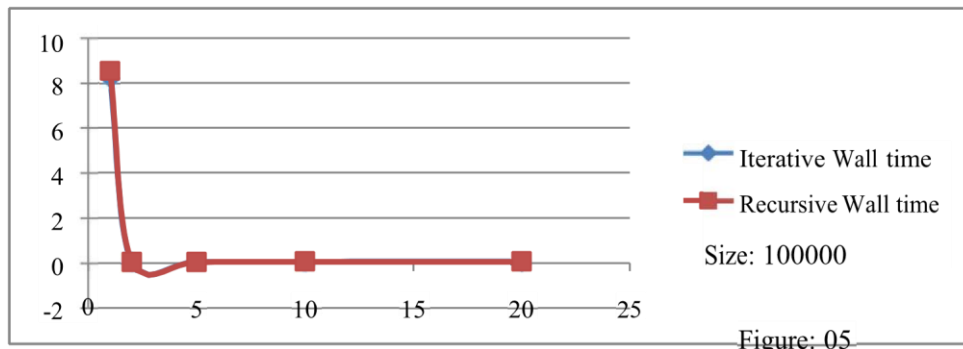


Figure: 05

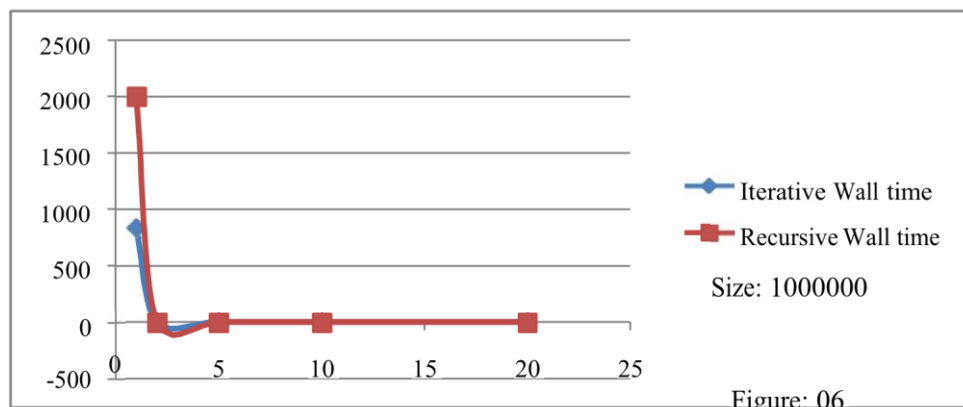


Figure: 06

Here is a sample output file format we have used along this experiment.

```
Iteratively Sorted Data :
Input size : 100
Nber processes : 10
Wall time      : 0.0003
2  3  5  5  8  11  11  12  13  13
```

5. ANALYSIS

In Figure 01 and Figure 02, the sorting is faster with a single process and keeps slowing as long as the number of processes increases. However in Figure 03 the execution time drastically dropped from a single process to its fastest execution time where the two processes are running in parallel, then it starts slowing again when we increase the number of processes. Finally in Figure 04 and Figure 05 the iterative and recursive implementations have almost the same execution time, the sorting becomes faster with more processes, with less variations. Figure 06 having one million numbers as input size. Here on single process, the sorting is the slowest of this experiment and even stop on recursion. The sorting becomes again faster with the number of process increases. On these different charts we can clearly observe that in general the iterative QuickSort is faster than the recursion version. On sequential execution with a single process, the sorting is faster with small input and slows down as long as the input size goes up. However on parallel execution with more than one process, the execution time decreases when the input size and the number of process increases. We noticed sometimes an unusual behavior of MPI execution time that keeps changing after each execution. We have taken only the first execution time to minimize this variation and obtain a more consistent execution time.

6. CONCLUSION

To conclude this project, we have successfully implemented the sequential QuickSort algorithm both the recursive and iterative version using the C programming language. Then we have done a parallel implementation with the help of Open MPI library. Through this project we have explored different aspects of MPI and the QuickSort algorithm as well as their respective limitations. This study has revealed that in general the sequential QuickSort is faster on small inputs while the parallel QuickSort excels on large inputs. The study also shows that in general the iterative version of QuickSort perform slightly better than the recursive version. The study reveals some unusual behavior of the execution time that unexpectedly varies. However due to the limited time we could not overcome all the difficulties and limitations, the next section opens an eventual scope for future improvements.