**Implementation of external merge sort**

**Assignment No: 16**

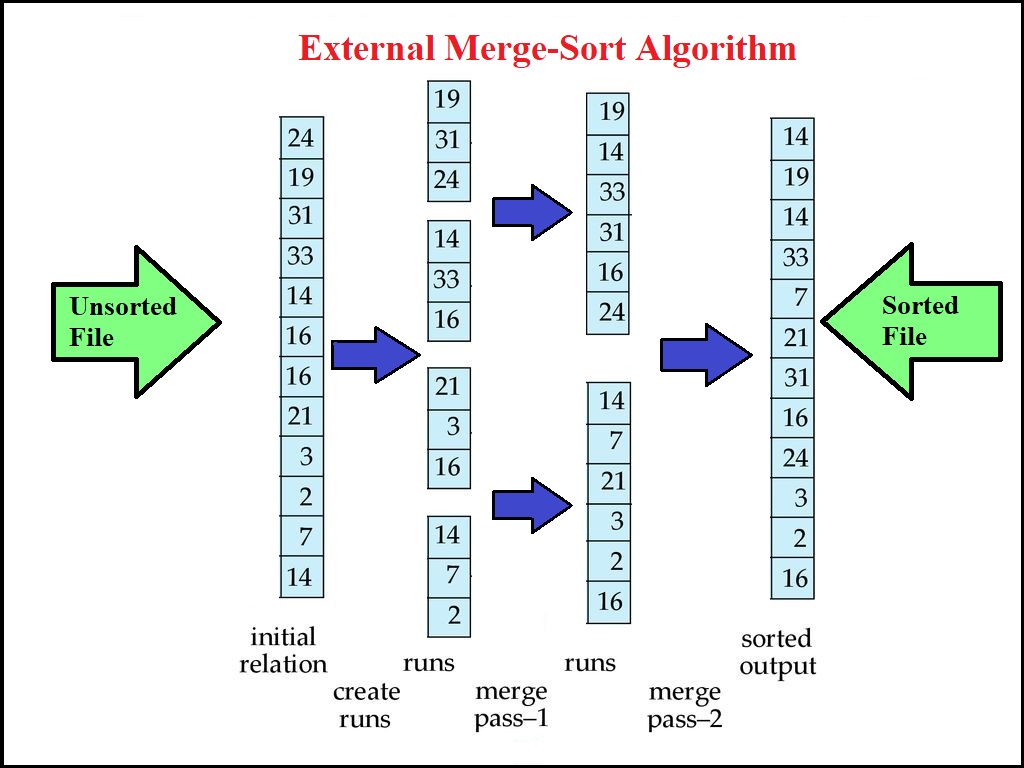
**Assignment Title:** Implementation of external merge sort

**Problem Statement:** Implementation of external merge sort

**Theoretical Concept:**

* **Definition Of External Merge Sort :**
* The external merge sort algorithm is used to efficiently sort massive amounts of data when the data being sorted cannot be fit into the main memory (usually RAM) and resides in the slower external memory (usually a HDD).
* External merge sort uses a hybrid sort-merge technique. The chunks of data small enough to fit in the RAM are read, sorted, and written out to a temporary file during the sorting phase. In the merge phase, the sorted sub-files are combined into a single larger file.
* External merge sort sorts the chunks of data that fits in the main memory, then merges the sorted chunks, i.e.,
* First, divide the file into runs such that the size of a run is small enough to fit into the main memory.
* Next, sort each run in main memory using the standard merge sort sorting algorithm.
* Finally, merge the resulting runs into successively bigger runs until the file is sorted.
* **Approach Of External Merge Sort :**
* The idea is very simple, All the elements cannot be sorted at once as the size is very large. So the data is divided into chunks and then sorted using merge sort. The sorted data is then dumped into files. As such huge amount of data cannot be handled altogether. Now After sorting the individual chunks. Sort the whole array by using the idea of merge k sorted arrays.
* **Algorithm:**

1. Read input file such that at most ‘run size’ elements are read at a time. Do following for every run read in an array.
2. Sort the run using Merge Sort.
3. Store the sorted array in a file. Let’s say ‘i’ for I’th file.
4. Merge the sorted files using the approach discussed merge k sorted arrays.



**Implementation Concept:**

* **Tools Used:**

1. Jupyter Notebook

* **Language Used:**

1. Python

* **Packages, Methods, Modules Used:**
* NumPy: Perform a wide variety of mathematical operations on arrays.
* shutil: The shutil module offers a number of high-level operations on files and collections of files. In particular, functions are provided which support file copying and removal.
* OS: The OS module in python provides functions for interacting with the operating system.
* pandas: Pandas is a software library written for the Python programming language for data manipulation and analysis.

**Program:**

import numpy as np

import shutil

import os

buffer\_size = 10000

total\_size = 55000

#Helper functions

def save\_array\_to\_file(file\_name, array\_to\_save):

np.savetxt(file\_name, array\_to\_save, fmt = '%d')

def sort\_and\_write(file\_name, array\_to\_sort):

mergeSort(array\_to\_sort, 0, len(array\_to\_sort)-1)

save\_array\_to\_file(file\_name, array\_to\_sort)

def read\_n\_int(file\_, numbers\_to\_read):

array\_ = []

if numbers\_to\_read <= 0 :

return array\_

num = file\_.readline()

while(num):

array\_.append(int(num))

if len(array\_) >= numbers\_to\_read:

break

num = file\_.readline()

return array\_

def create\_unsorted\_file(size\_, file\_name\_ = 'unsorted.csv'):

arr = np.arange(size\_)

np.random.shuffle(arr)

save\_array\_to\_file(file\_name\_, arr)

arr = None

#External Sort

#Merge Sort

def merge(arr, l, m, r):

n1 = m - l + 1

n2 = r - m

# create temp arrays

L = [0] \* (n1)

R = [0] \* (n2)

# Copy data to temp arrays L[] and R[]

for i in range(0, n1):

L[i] = arr[l + i]

for j in range(0, n2):

R[j] = arr[m + 1 + j]

# Merge the temp arrays back into arr[l..r]

i = 0 # Initial index of first subarray

j = 0 # Initial index of second subarray

k = l # Initial index of merged subarray

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

# Copy the remaining elements of L[], if there

# are any

while i < n1:

arr[k] = L[i]

i += 1

k += 1

# Copy the remaining elements of R[], if there

# are any

while j < n2:

arr[k] = R[j]

j += 1

k += 1

# l is for left index and r is right index of the

# sub-array of arr to be sorted

def mergeSort(arr, l, r):

if l < r:

# Same as (l+r)//2, but avoids overflow for

# large l and h

m = l+(r-l)//2

# Sort first and second halves

mergeSort(arr, l, m)

mergeSort(arr, m+1, r)

merge(arr, l, m, r)

import heapq

def sort\_slices(file\_name, buffer\_size\_):

read\_arr = []

chunk = 1

f = open(file\_name)

if os.path.exists('./tmp/'):

shutil.rmtree('./tmp/')

os.mkdir('./tmp/')

read\_arr = read\_n\_int(f, buffer\_size\_)

while (len(read\_arr) > 0):

sort\_and\_write('./tmp/sorted\_' + str(chunk), read\_arr)

read\_arr = read\_n\_int(f, buffer\_size\_)

chunk = chunk + 1

f.close()

def min\_heap\_sort(output\_file):

sorted\_file = open(output\_file, 'w+')

min\_heap = []

heapq.heapify(min\_heap)

open\_files = []

for f in os.listdir('./tmp/'):

if os.path.isfile('./tmp/' + f):

file\_ = open('./tmp/' + f)

open\_files.append(file\_)

val = file\_.readline()

heapq.heappush(min\_heap, (int(val), file\_))

while(len(min\_heap) > 0):

min\_element = heapq.heappop(min\_heap)

sorted\_file.write(str(min\_element[0]) + '\n')

next\_str = min\_element[1].readline()

if next\_str:

heapq.heappush(min\_heap, (int(next\_str), min\_element[1]))

else:

min\_element[1].close()

sorted\_file.close()

def external\_sort(input\_file, output\_file, buffer\_size\_ = 10000):

sort\_slices(input\_file, buffer\_size\_)

min\_heap\_sort(output\_file)

print('Sorted values are written to' , str(output\_file))

create\_unsorted\_file(total\_size, file\_name\_ = 'unsorted.csv')

import pandas as pd

unsorted=pd.read\_csv("/content/unsorted.csv")

unsorted.head(10)

#This function performs external sort, just call it with input and output file name and it will do the magic

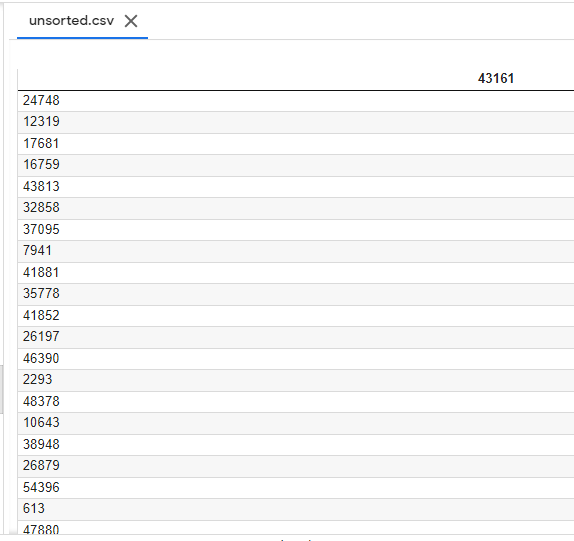
external\_sort(input\_file= 'unsorted.csv', output\_file='sorted\_external.csv', buffer\_size\_ = buffer\_size)

sorted=pd.read\_csv("/content/sorted\_external.csv")

sorted.head(10)

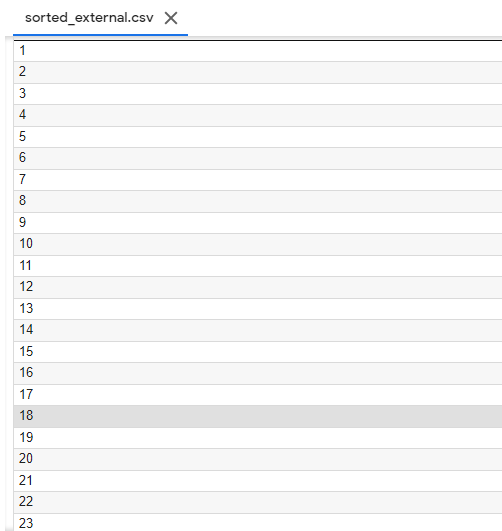
**Input:**

* Input changes randomly, according to standard numpy random function.



**Output:**

* Output of the program is sorted file.



**Conclusion:**

* In this Assignment, we have seen how the external merge sort exactly works, how it sorts the records which fits into the main memory in detail. We have implemented the external merge sort algorithm and also studied the Concept of external merge Sort Theoretically As well as Practically.