IE343 TERM PROJECT REPORT

1- CODING APPROACH

I have split my solution into three parts; selecting tracks, sorting tracks and showing results.

- **Selecting Tracks**
  In the first part, I have created an album with the highest total popularity value by following the thirty minutes rule. If I have been used only track's popularity values or only durations, I would not have gotten higher total popularity values. The reason of this is, the tracks that has higher popularity values may have very long durations or the tracks that has shorter durations may have lower popularity values. Therefore, I have created a new value called "Real Value". Real Value means, popularity value divided by duration. Thanks to this value, I could be able to sort tracks by maximizing their popularity value while minimizing their durations. So, I have sorted the track list according to their Real Value in descending order. After that, I have added them into a new track list one by one starting from first track which has highest Real Value. While keep adding those tracks, on the other hand, I have calculated the total album duration. Then, I have tried to add every track until we reach the closest duration to thirty minutes.
- **Sorting Tracks**
  I have sorted the tracks according to needs of label in the second part. Firstly, I have assigned most valuable track as first element of final track list. Then assigned second most valuable track as last element of final track list. After that, I have compared the sequential values of first track with all selected tracks that is not included into the final track list. Then I add the highest sequential value with first track as the next element. In this way, I have kept adding most suitable track with previous track until I obtained every track from selected tracks list.
- **Showing Results**
  In the last part, I have printed selected track count, total album duration, cyclomatic complexity value, total album values and runtime durations of my program and exact solution.

2- EXPLANATION OF MY ALGORITHM

First of all, I have produced a new value in order to use in my calculations. I have named it as "Real Value" and it made me learn that how much a track worth choosing.

Track.java

```
public double getRealTrackValue() {
	return (double)getIndividualValue() / (double)getDuration;
}
```

The Real Value maximizes the individual value whilst minimizing the duration.

Then, I have created a function for reading the track properties from the .CSV files and writing them into a new track list.

ExcelReader.java

- Initialized the variables of CSV reader:

```
String valDataSheetFileName = "term_project_value_data";
String seqDataSheetFileName = "term_project_sequential_data";
Track[] trackList = null;
CSVReader valDataReader = null;
CSVReader seqDataReader = null;
```

- Read the .CSV files by using **OpenCSV** library:

```
valDataReader = new CSVReader(new FileReader(
                    System.getProperty("user.dir") + "\\" +
                    valDataSheetFileName + ".csv";
seqDataReader = new CSVReader(new FileReader(
                    System.getProperty("user.dir") + "\\" +
                    seqDataSheetFileName + ".csv";

List<String[]> valData = valDataReader.readAll;
List<String[]> seqData = seqDataReader.readAll;
```

- Initialized variables of new track array. I set the track list count as track count minus one because, valData.size also contains a row for column headers which we won't going to use.

```
int trackIndex = 0;
int trackCount = valData.size;
trackList = new Track[trackCount - 1];
```

- Created new track according to row values and added into the track list. I skipped the first element of seqValue because it belongs to ID column and we won't use it.

```
for (int rowIndex = 1; rowIndex < trackCount; rowIndex++) {
      double[] seqValue =
      ConvertStringListToDoubleArray(Arrays.asList(
                        seqData.get(rowIndex)
                        .stream()
                        .skip(1)
                        .collect(Collectors.toList());

      Track newTrack = new Track(
            Integer.parseInt(valData.get(rowIndex)[0],
            Integer.parseInt(valData.get(rowIndex)[5],
            Integer.parseInt(valData.get(rowIndex)[4],
                        seqValue);
            trackList[trackIndex++] = newTrack;
}
```

MySolution.java

- Initialized the constant variables given in the project documentation.

```
final int MAX_DURATION = 1800;
final double VAL_LOSS_PER_SEC = 0.02;
```

PART 1: SELECTING TRACKS

- Started the timer before running the algorithm for calculating the runtime in nanoseconds.

```
long startTime = System.nanoTime;
```

- Used my function above to read the tracks from .CSV file

```
Track[] trackList = ExcelReader.ReadTracksFromExcelSheets();
```

- Checked if track list is initialized properly

```
if (trackList == null)
	return;
```

- Used bubble sort to sort the track list in descending order by their real value

```
trackList = sortByRealValue(trackList);

private static Track[] sortByRealValue(Track[] trackArray) {
	Track[] newTrackArray = new Track trackArray.length;
	for (int trackIndex = 0; trackIndex < trackArray.length;
	trackIndex++) {
		newTrackArray[trackIndex] = trackArray[trackIndex];
	}
	for (int index1 = 0; index1 < newTrackArray.length; index1++) {
		for (int index2 = 0; index2 < newTrackArray.length;
		index2++) {
			if (newTrackArray[index1].getRealTrackValue() >
		newTrackArray[index2].getRealTrackValue()) {
			Track trackTemp = newTrackArray[index1];
			newTrackArray[index1] = newTrackArray[index2];
			newTrackArray[index2] = trackTemp;
		}
	return newTrackArray;
}
```

- Initialized the selected tracks list and other variables.

```
Track[] selectedTracks = new Track[trackList.length];

int selectedTrackCount = 0;
int totalDuration = 0;
int totalAlbumValue = 0;
```

- Selected tracks starting from the first track of sorted track list which means the track that has the highest real value and calculated raw total album value and total album duration

```
for (int trackIndex = 0; trackIndex < trackList.length; trackIndex++)
	int nextTotalDuration = totalDuration +
		millisecToSec(trackList[trackIndex].getDuration());
	if (nextTotalDuration <= MAX_DURATION) {
		selectedTracks[selectedTrackCount++] =
			trackList[trackIndex];
		totalAlbumValue +=
			trackList[trackIndex].getIndividualValue();
		totalDuration = nextTotalDuration;
	}
}
private static int millisecToSec(int millisecs) {
	return millisecs / 1000;
}
```

- Decreased total album value according to missing seconds

```
totalAlbumValue -= (MAX_DURATION - totalDuration) * VAL_LOSS_PER_SEC;
```

PART 2: SORTING TRACKS

- Initialized final tracks array that label wants

```
Track[] finalTracks = new Track[selectedTrackCount];
```

- Set first and last track of final track array when we use takeTrack function it returns the most valuable track in this case and makes it null, so when we call takeTrack function again, it returns the second most valuable song as a result.

```
finalTracks[0] = takeTrack(selectedTracks,
	getMostValuableTrackIndex(selectedTracks));
finalTracks[finalTracks.length - 1] = takeTrack(selectedTracks,
	getMostValuableTrackIndex(selectedTracks));
```

- Thi$s function returns the track index that has the highest individual value in the track array

```
private static int getMostValuableTrackIndex(Track[] trackArray) {
	int highestTrackValue = 0;
	int highestTrackIndex = 0;
	for (int trackIndex = 0; trackIndex < trackArray.length;
		trackIndex++)
		if (trackArray[trackIndex] != null)
			if (trackArray[trackIndex].getIndividualValue() >
					highestTrackValue) {
				highestTrackValue =
					trackArray[trackIndex].getIndividualV
				alue();
				highestTrackIndex = trackIndex;
			}
		}
	}
	return highestTrackIndex;
}
```

- Selected the remained tracks according to their most suitable track index, it decides the most suitable track among the unselected tracks

```
for (int trackIndex = 1; trackIndex < finalTracks.length - 1;
      trackIndex++) {
      finalTracks[trackIndex] = takeTrack(selectedTracks,
            getMostSuitableTrackIndex(finalTracks[trackIndex - 1],
            selectedTracks));
}
private static int getMostSuitableTrackIndex(Track track, Track[]
trackArray) {
      int mostSuitableTrackIndex = 0;
      double highestSeqValue = 0;

      for (int trackIndex = 0; trackIndex < trackArray.length;
trackIndex++) {
            if (trackArray[trackIndex] == null)
                  continue;
            if (track.getSeqValueOf(trackArray[trackIndex].getId())
> highestSeqValue) {
                  highestSeqValue =
                        track.getSeqValueOf(trackArray[trackIndex].
                        getId());
                  mostSuitableTrackIndex = trackIndex;
            }
      }
      return mostSuitableTrackIndex;
}
```

- In the takeTrack function, I aimed to take a track at specified index and set it as null in order to remove it from list like pop method in stacks.

```
private static Track takeTrack(Track[] trackArray, int trackIndex) {
      Track track = trackArray[trackIndex];
      trackArray[trackIndex] = null;
      return track;
```

- Lastly, I got the system time after executing the algorithm and calculated the difference between startTime. As the System.nanoTime() gives us current time in nanoseconds, I converted it into seconds by dividing it by 10^9

```
long endTime = System.nanoTime();
double runtimeDuration = (endTime - startTime) / 1000000000.0;
```

PART 3: SHOWING RESULTS

- Defined the results of exact solution with Gurobi.

```
int exactSolAlbumValue = 532;
double exactSolRuntimeDuration = 0.024909496307373047;
```

- Printed the results of my algorithm and compared it with the exact solution

```
System.out.println("SELECTED TRACK COUNT: " + selectedTrackCount);
System.out.println("\nTOTAL ALBUM DURATION: " + totalDuration + "
```

```
            secs (" + String.format("%.2f", totalDuration / 60.0) + "
            mins)");
        System.out.println("\nCOMPARISON");
        System.out.format("\nMY ALBUM VALUE: %d || EXACT SOLUTION ALBUM
            VALUE: %d", totalAlbumValue, exactSolAlbumValue);
        System.out.format("\nMY RUNTIME DURATION: %f || EXACT SOLUTION
            RUNTIME DURATION: %f", runtimeDuration,
        exactSolRuntimeDuration);
```

3- RESULTS

The results of my algorithm was very close to the exact solution, however it could not be able to beat the solution quality of Gurobi Optimizer. The total individual value of album was very close to the exact solution which is 529 while exact solution calculated as 532 points. On the other hand, the runtime was two times longer comparing to runtime of exact solution. The algorithm takes 0.049 second in average while exact solution got a runtime of 0.024 second.

Finally, the Cyclomatic Complexity of my algorithm was 20. Because, there are 20 independent paths through the code of my solution, including:

- The path that executes when `trackList == null`
- The path that executes when `trackList != null`
- The path that executes when `nextTotalDuration <= MAX_DURATION` in the `for` loop
- The path that executes when `nextTotalDuration > MAX_DURATION` in the `for` loop
- The path that executes in the `inner for loop`

There are also several branches in the code, including the `if statement` and the `for loops`, which contribute to the overall complexity of the code.