LINQ C# Summary

Basic Filtering

- Where : Filters elements based on a condition.

```
var highScorers = students.Where(s => s.Marks > 80);
```

- Distinct : Removes duplicate elements from a collection.

```
var uniqueScores = scores.Distinct();
```

Sorting

- OrderBy : Sorts elements in ascending order based on a key.

```
var sortedByMarks = students.OrderBy(s => s.Marks);
```

- OrderByDescending : Sorts elements in descending order.

```
var sortedByMarksDesc = students.OrderByDescending(s => s.Marks);
```

- ThenBy : Secondary sort in ascending order.

```
var sortedByMarksThenName = students.OrderBy(s => s.Marks).ThenBy(s => s.Name);
```

- ThenByDescending : Secondary sort in descending order.

```
var sortedByMarksThenNameDesc = students.OrderBy(s =>
s.Marks).ThenByDescending(s => s.Name);
```

- Reverse : Reverses the order of elements in a collection.

```
var reversedList = students.Reverse();
```

Quantifiers

- All : Checks if all elements satisfy a condition.

```
bool allPass = students.All(s => s.Marks > 50);
```

- Any : Checks if any elements satisfy a condition.

```
bool hasFailures = students.Any(s => s.Marks < 50);
```

- Contains : Checks if a collection contains a specific element.

```
bool hasStudent = students.Contains(specificStudent);
```

Projection

- Select : Projects each element into a new form.

```
var studentNames = students.Select(s => s.Name);
```

- SelectMany : Projects each element and flattens the resulting collections.

```
var allScores = students.SelectMany(s => s.Scores);
```

Aggregation

- Count : Counts elements in a collection.

```
int count = students.Count();
```

- Sum : Computes the sum of values in a collection.

```
int totalMarks = students.Sum(s => s.Marks);
```

- Average : Computes the average of values.

```
double averageMarks = students.Average(s => s.Marks);
```

- Min / Max : Finds the minimum or maximum value.

```
int minMarks = students.Min(s => s.Marks);
```

Element Operations

- First / FirstOrDefault : Gets the first element.

```
var topStudent = students.First(s => s.Marks > 90);
```

- Last / LastOrDefault : Gets the last element.

```
var lastHighScorer = students.Last(s => s.Marks > 80);
```

- Single / SingleOrDefault : Gets the single element that matches.

```
var uniqueStudent = students.Single(s => s.Id == 1);
```

- ElementAt : Gets the element at a specified index.

```
var thirdStudent = students.ElementAt(2);
```

Set Operations

- Union : Combines two collections, removing duplicates.

```
var allStudents = studentsA.Union(studentsB);
```

- Intersect : Returns common elements.

```
var commonStudents = studentsA.Intersect(studentsB);
```

- Except : Returns elements in one collection but not another.

```
var exclusiveStudents = studentsA.Except(studentsB);
```

Grouping

- GroupBy : Groups elements by a specified key.

```
var studentsByGrade = students.GroupBy(s => s.Grade);
```

Joining

- Join : Joins two collections based on keys.

```
var studentCourses = students.Join(courses, s => s.CourseId, c => c.Id, (s, c)
=> new { s.Name, c.CourseName });
```

- GroupJoin : Joins two collections and groups the results.

```
var groupedCourses = students.GroupJoin(courses, s => s.CourseId, c => c.Id, (s,
cs) => new { s.Name, Courses = cs });
```

Partitioning

- Take : Takes the first n elements.

```
var topThree = students.Take(3);
```

- Skip : Skips the first n elements.

```
var skipFirstTwo = students.Skip(2);
```

- TakeWhile : Takes elements as long as a condition is true.

```
var topScorers = students.TakeWhile(s => s.Marks > 80);
```

- SkipWhile : Skips elements while a condition is true.

```
var afterTopScorers = students.SkipWhile(s => s.Marks > 80);
```

Conversion

- ToList : Converts a collection to a List<T>.

```
var studentList = students.ToList();
```

- ToArray : Converts a collection to an array.

```
var studentArray = students.ToArray();
```

- ToDictionary : Converts a collection to a dictionary.

```
var studentDictionary = students.ToDictionary(s => s.Id, s => s.Name);
```