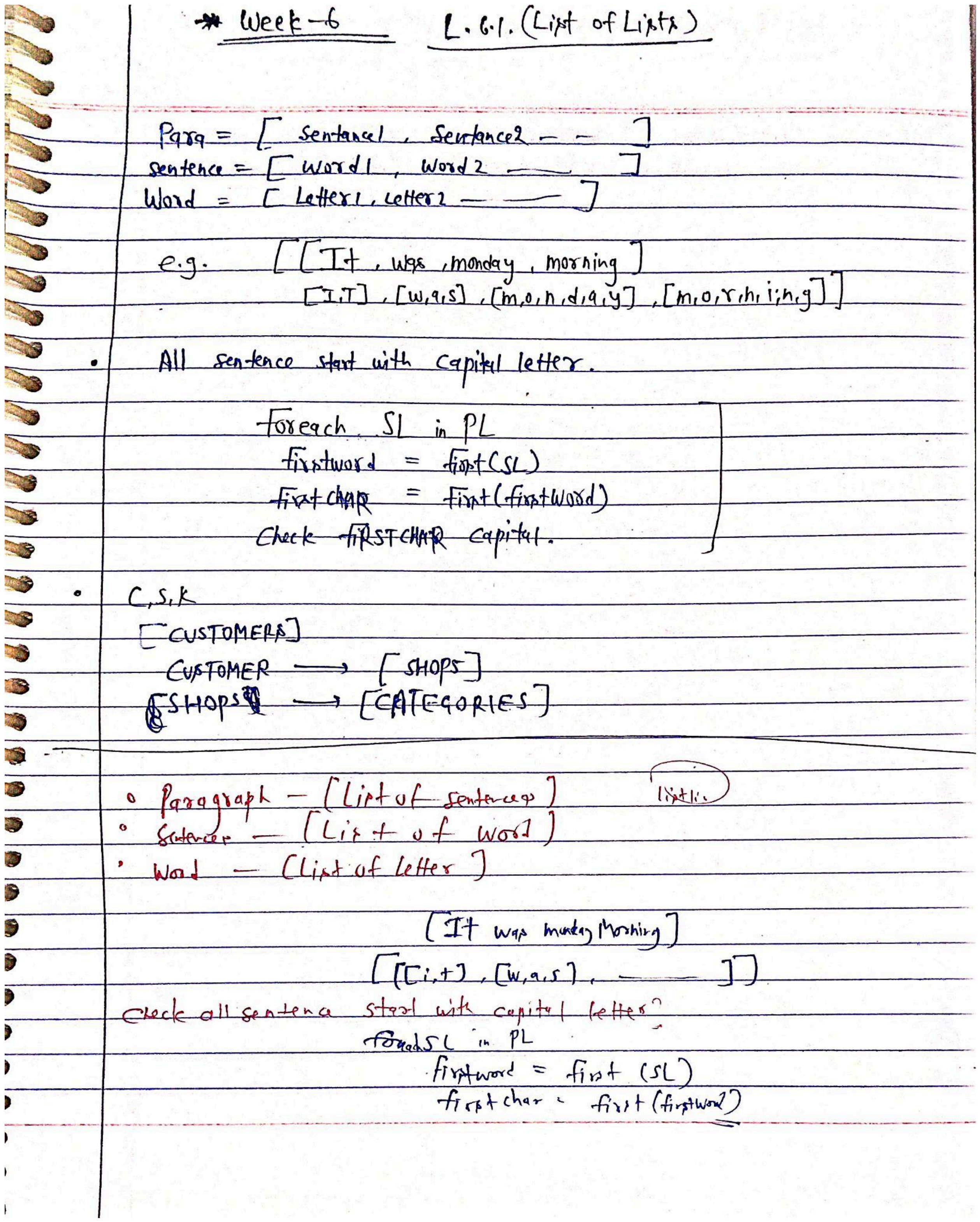
CT Week 6 : List and Insertion sort

Lecture 1 : Lists of List



* Week-6 L.6.1. (List of Lists)

Para = [Sentance1 , Sentance2 — —]
sentence = [word1 , word2 ——]
Word = [Letter1 , Letter2 —— ——]

e.g. [[It , was , monday , morning]
[I,T] , [w,a,s] , [m,o,n,d,a,y] , [m,o,r,n,i,n,g]]

- All sentence start with capital letter.

```
Foreach SL in PL
  firstword = first(SL)
  first CHAR = First(firstword)
Check FIRSTCHAR capital.
```

- C,S,K
  [CUSTOMERS]
  CUSTOMER ⟶ [SHOPS]
  [SHOPS] ⟶ [CATEGORIES]

- Paragraph — [List of sentences] list li
- Sentence — [List of word]
- Word — [List of letter]

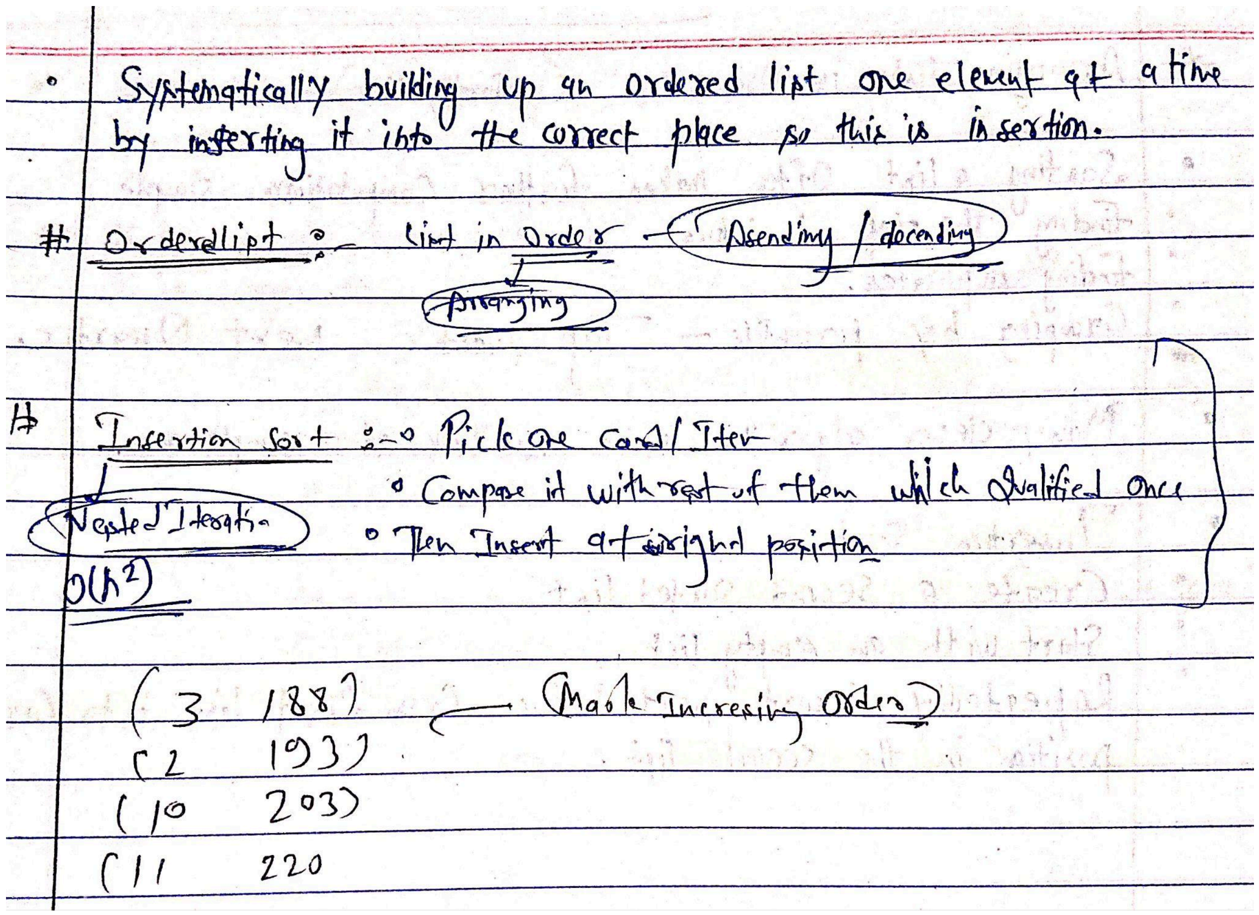
[It was monday Morning]
[[[i,t] , [w,a,s] , ———]]

Check all sentence start with capital letter?

```
Foreach SL in PL
  firstword = first (SL)
  firstchar = first (firstword)
```

QN : 1

Lecture 2 : Insertion sort and ordered list



- Systematically building up an ordered list one element at a time by inserting it into the correct place so this is insertion.

#Orderedlist :- List in Order ← (Asending / decending)

↓ (Arranging)

#Insertion sort :- • Pick one card / Item
• Compare it with rest of them which qualified once
• Then Insert at right position

↓ (Nested Iteration)

$O(n^2)$

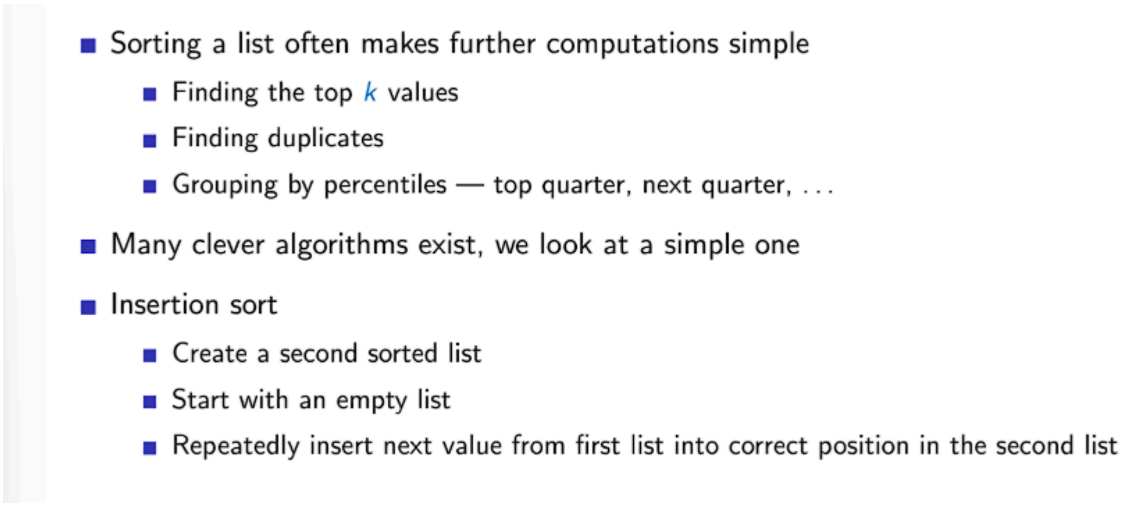
(3 188) ← (Make Increasing Order)
(2 193)
(10 203)
(11 220

QN : 1,2,3

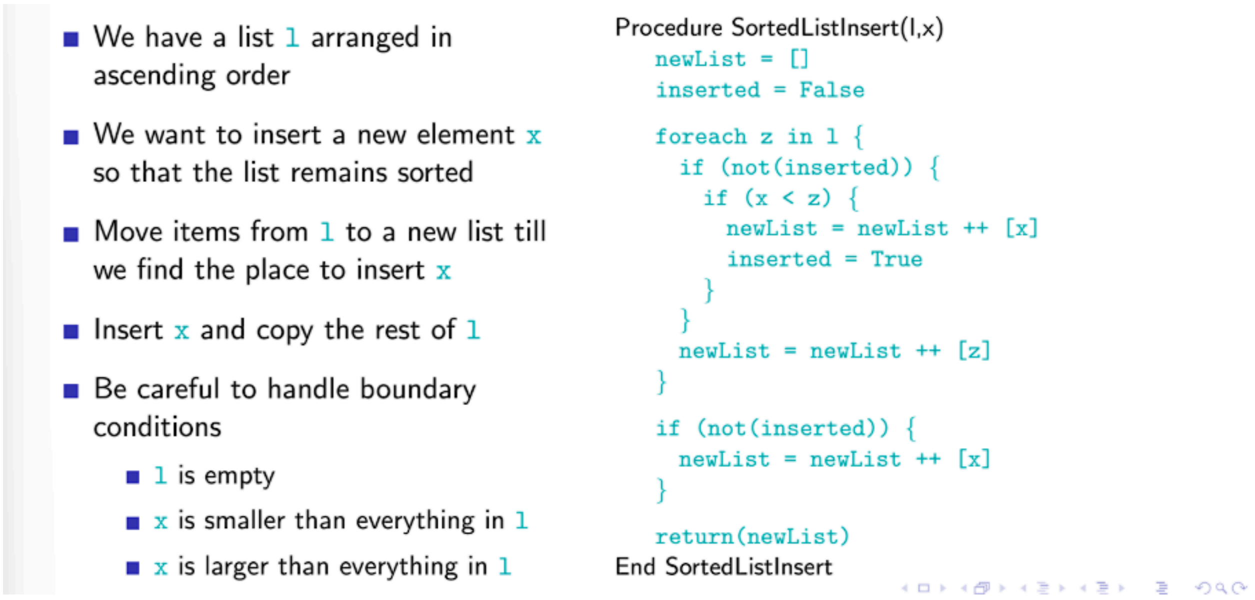
Lecture 3 : Pseudocode for insertion sort and ordered list

Pseudocode: Sorting lists

Arranging lists in order

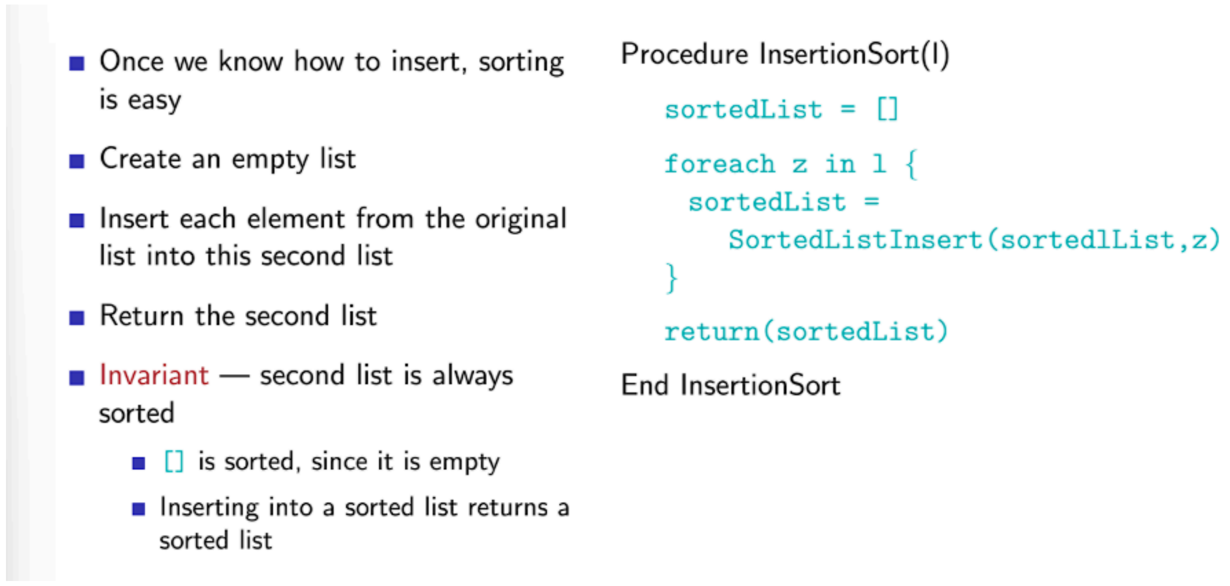

- Sorting a list often makes further computations simple
  - Finding the top k values
  - Finding duplicates
  - Grouping by percentiles — top quarter, next quarter, . . .
- Many clever algorithms exist, we look at a simple one
- Insertion sort
  - Create a second sorted list
  - Start with an empty list
  - Repeatedly insert next value from first list into correct position in the second list

Inserting into a sorted list


- We have a list `l` arranged in ascending order
- We want to insert a new element `x` so that the list remains sorted
- Move items from `l` to a new list till we find the place to insert `x`
- Insert `x` and copy the rest of `l`
- Be careful to handle boundary conditions
  - `l` is empty
  - `x` is smaller than everything in `l`
  - `x` is larger than everything in `l`

```
Procedure SortedListInsert(l,x)
   newList = []
   inserted = False

   foreach z in l {
     if (not(inserted)) {
       if (x < z) {
         newList = newList ++ [x]
         inserted = True
       }
     }
     newList = newList ++ [z]
   }

   if (not(inserted)) {
     newList = newList ++ [x]
   }

   return(newList)
End SortedListInsert
```

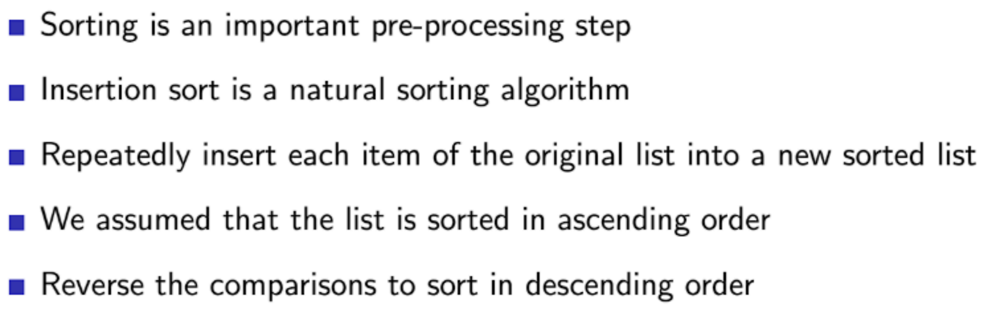
Insertion sort



- Once we know how to insert, sorting is easy
- Create an empty list
- Insert each element from the original list into this second list
- Return the second list
- Invariant — second list is always sorted
  - [] is sorted, since it is empty
  - Inserting into a sorted list returns a sorted list

```
Procedure InsertionSort(l)
   sortedList = []
   foreach z in l {
     sortedList =
        SortedListInsert(sortedlList,z)
   }
   return(sortedList)
End InsertionSort
```

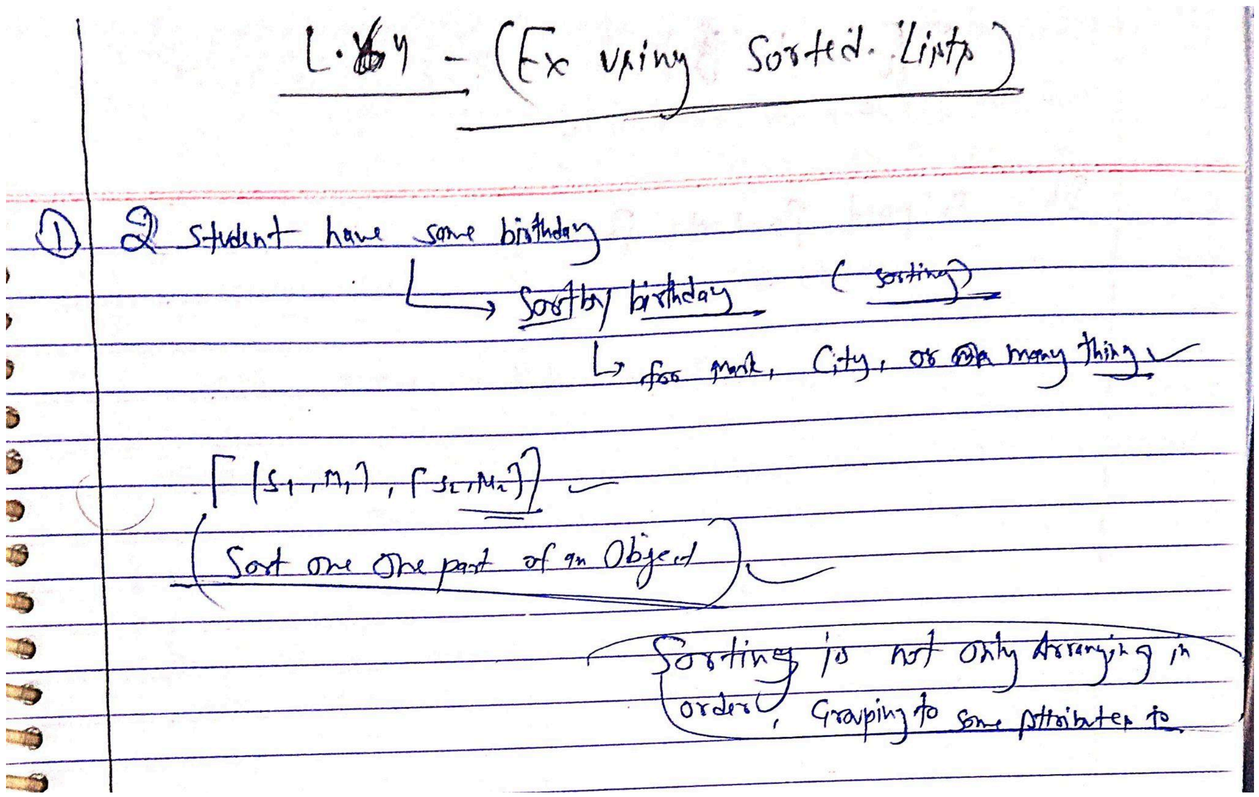
Summary



- Sorting is an important pre-processing step
- Insertion sort is a natural sorting algorithm
- Repeatedly insert each item of the original list into a new sorted list
- We assumed that the list is sorted in ascending order
- Reverse the comparisons to sort in descending order

QN : 2,3

Lecture 4 : Examples using sorted lists



L·~~[illegible]~~4 – (Ex using Sorted Lists)

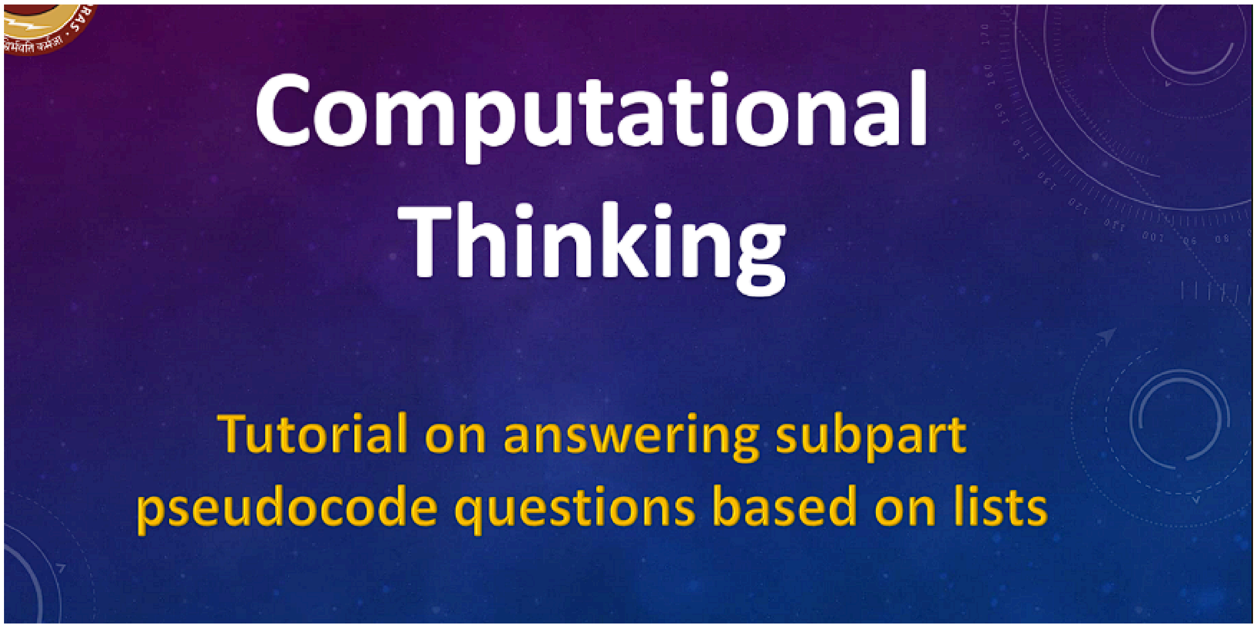
① 2 student have same birthday
→ Sort by birthday (sorting)
→ for mark, City, or ~~[illegible]~~ many thing

[[S_1, M_1], [S_2, M_2]]

(Sort one the part of an Object)

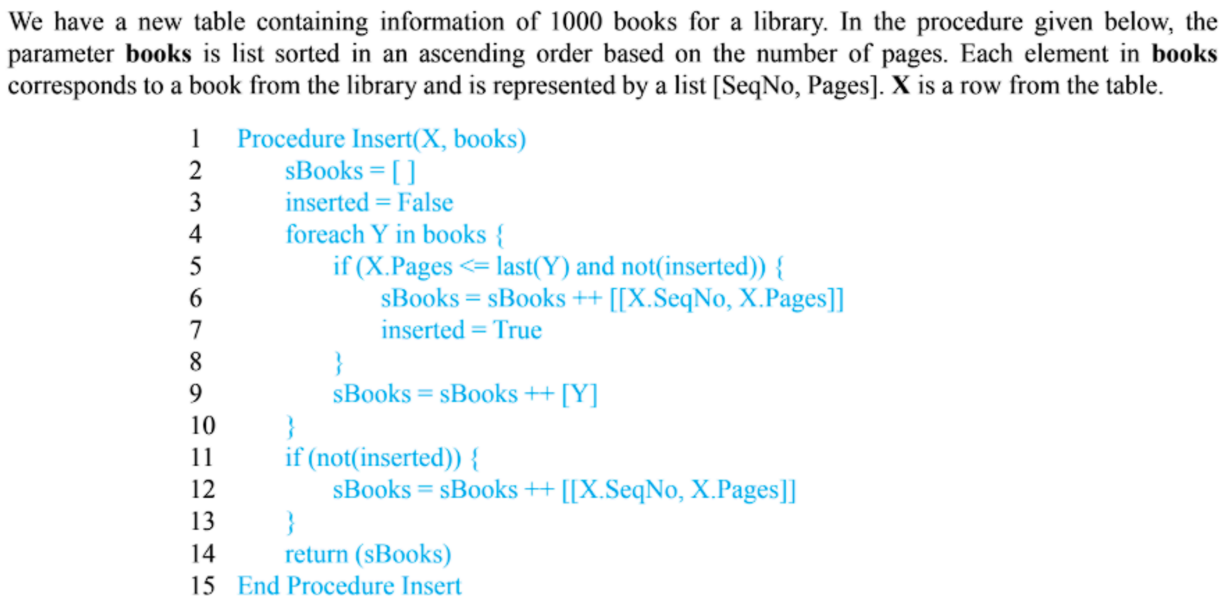
Sorting is not only Arranging in order, Grouping to some attributes to

Tutorial 6.1 on answering subpart pseudocode questions based on lists



Computational Thinking

Tutorial on answering subpart pseudocode questions based on lists



We have a new table containing information of 1000 books for a library. In the procedure given below, the parameter **books** is list sorted in an ascending order based on the number of pages. Each element in **books** corresponds to a book from the library and is represented by a list [SeqNo, Pages]. **X** is a row from the table.

```
Procedure Insert(X, books)
    sBooks = [ ]
    inserted = False
    foreach Y in books {
        if (X.Pages <= last(Y) and not(inserted)) {
            sBooks = sBooks ++ [[X.SeqNo, X.Pages]]
            inserted = True
        }
        sBooks = sBooks ++ [Y]
    }
    if (not(inserted)) {
        sBooks = sBooks ++ [[X.SeqNo, X.Pages]]
    }
    return (sBooks)
End Procedure Insert
```

Q1: **Z** is some arbitrary value containing a book's details. Consider the following code:

```
someBooks = [ ]
someBooks = Insert(Z, someBooks)
```

Which of the following lines in the procedure **Insert** will be executed during the above call? It is a Multiple Select Question (MSQ).

- ☐ Line 5
- ☐ Line 6
- ☐ Line 7
- ☐ Line 9
- ✓ Line 12
- ☐ No lines. An empty list cannot be passed as a parameter to the procedure.

```
1   Procedure Insert(X, books)
2       sBooks = [ ]
3       inserted = False
4       foreach Y in books {
5           if (X.Pages <= last(Y) and not(inserted)) {
6               sBooks = sBooks ++ [[X.SeqNo, X.Pages]]
7               inserted = True
8           }
9           sBooks = sBooks ++ [Y]
10      }
11      if (not(inserted)) {
12          sBooks = sBooks ++ [[X.SeqNo, X.Pages]]
13      }
14      return (sBooks)
15  End Procedure Insert
```

Q2: **Z** is a row in the table with the following data: **Z**.SeqNo is 12 and **Z**.Pages is 350.
What will be the contents of the list **someBooks** at the end of execution of the following code?

```
someBooks = [ [5, 220], [10, 350], [15, 350], [20, 400] ]
someBooks = Insert(Z, someBooks)
```

- ○ [[5, 220], [10, 350], [15, 350], [20, 400]]
- ✓ [[5, 220], [12, 350], [10, 350], [15, 350], [20, 400]]
- ○ [[5, 220], [10, 350], [12, 350], [15, 350], [20, 400]]
- ○ [[5, 220], [10, 350], [15, 350], [12, 350], [20, 400]]

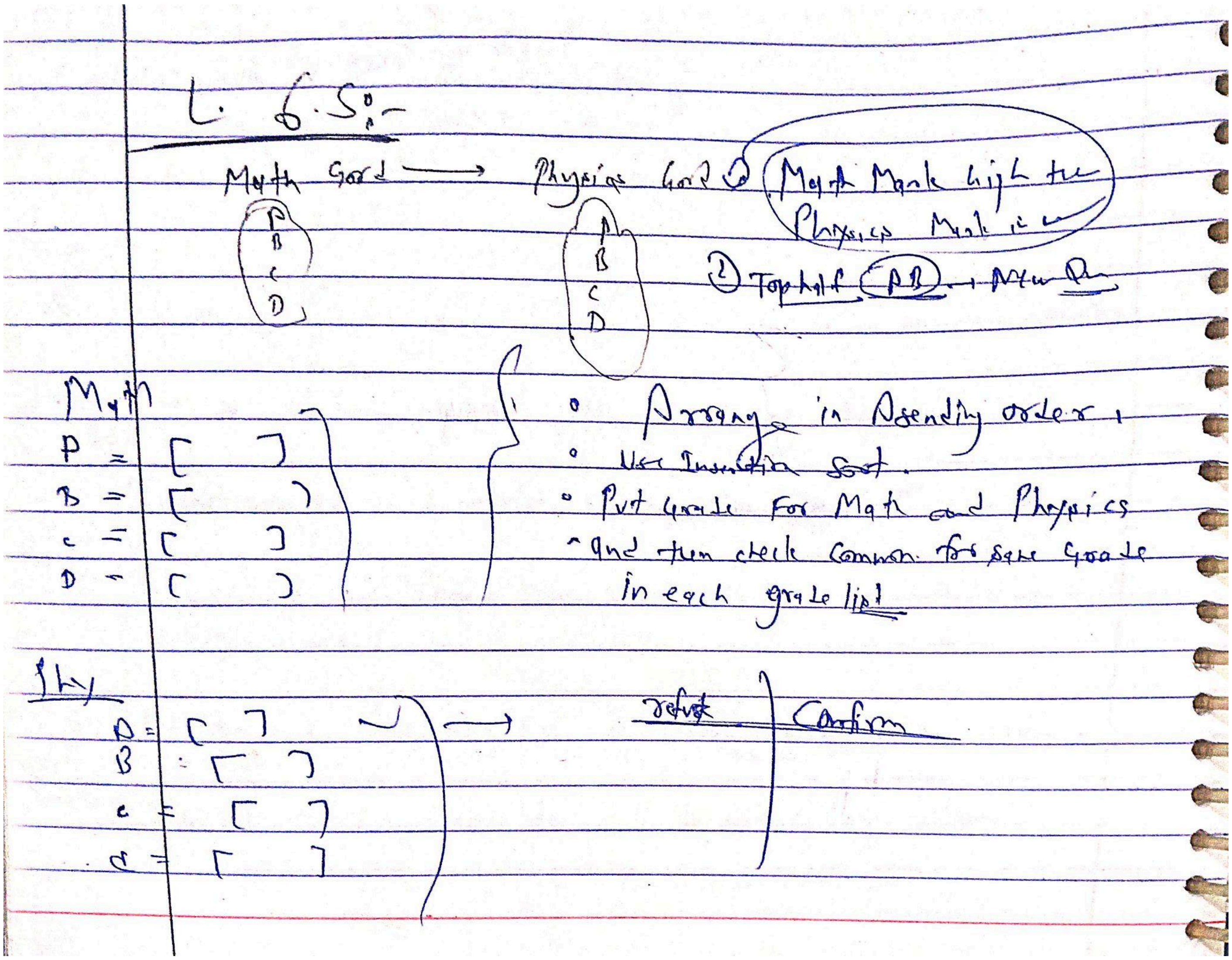
```
1   Procedure Insert(X, books)
2       sBooks = [ ]
3       inserted = False
4       foreach Y in books {
5           if (X.Pages <= last(Y) and not(inserted)) {
6               sBooks = sBooks ++ [[X.SeqNo, X.Pages]]
7               inserted = True
8           }
9           sBooks = sBooks ++ [Y]
10      }
11      if (not(inserted)) {
12          sBooks = sBooks ++ [[X.SeqNo, X.Pages]]
13      }
14      return (sBooks)
15  End Procedure Insert
```

Q3: Execute the following pseudocode on the "Library" table. Which of the following statements are true after execution? It is a Multiple Select Question (MSQ).

```
books = [ ]
while(Table 1 has more rows) {
    Read top row X from Table 1
    books = Insert(X, books)
    Move X to Table 2
}
```

- ✓ first(books) corresponds to a book having the least number of pages in the library.
- ❑ first(books) corresponds to a book having the most number of pages in the library.
- ✓ last(last(books)) is the most number of pages among all the books in the library.
- ❑ first(last(books)) is the least number of pages among all the books in the library.
- ❑ last(first(books)) is the most number of pages among all the books in the library.
- ✓ last(first(books)) is the least number of pages among all the books in the library.

Lecture 5 : Systematic process of hypothesis verification to find relation between Mathematics and Physics marks using lists



L. G. S:-

Math Gard ⟶ Physics Gard ① Math Mark high the Physics Mark is [illegible]

② Top half (AB) → [illegible]

Math Gard: A, B, C, D

Physics Gard: A, B, C, D

Math

P = [ ]

B = [ ]

c = [ ]

D = [ ]

- Arrange in Asending order.
- Use Insertion sort.
- Put grade For Math and Physics
- And then check common for same Grade in each grade list

Phy

A = [ ]

B = [ ]

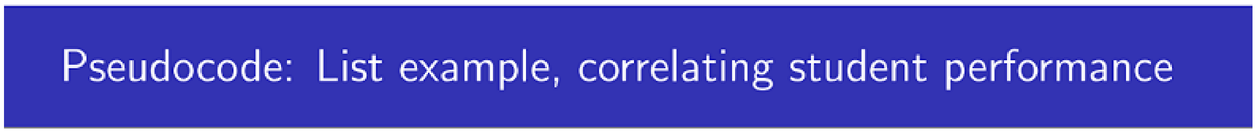
c = [ ]

d = [ ]

⟶ refute | Confirm

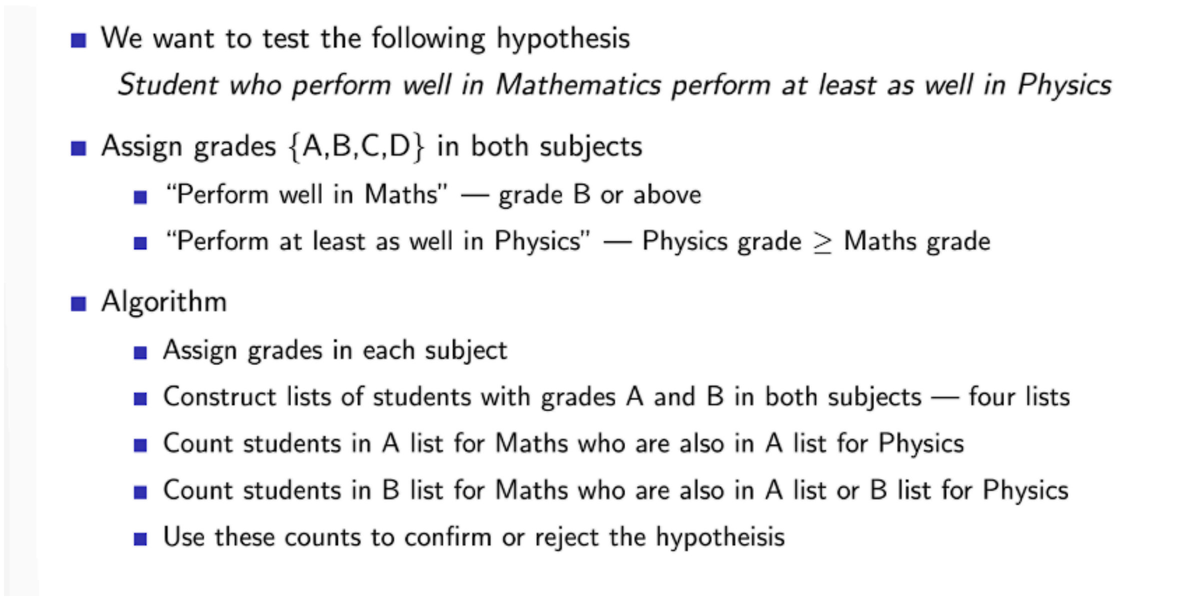
QN : 4

Lecture 6 : Pseudocode for systematic process of hypothesis verification to find relation between Mathematics and Physics marks using lists



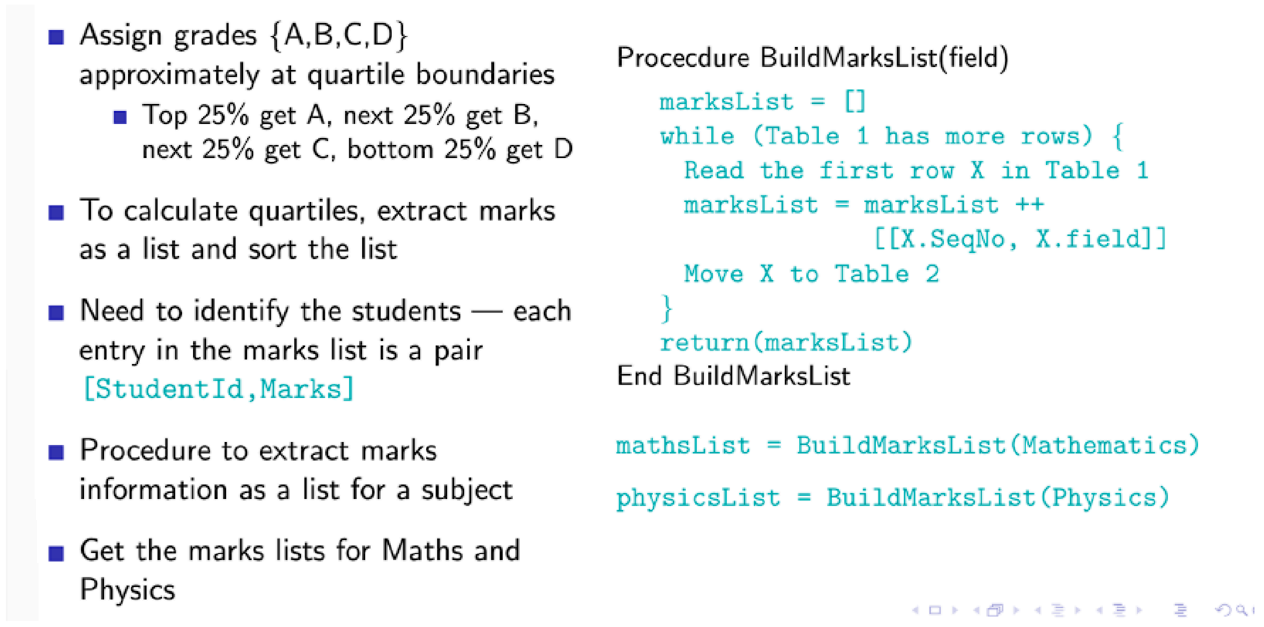
Pseudocode: List example, correlating student performance

Correlating marks in Maths and Physics



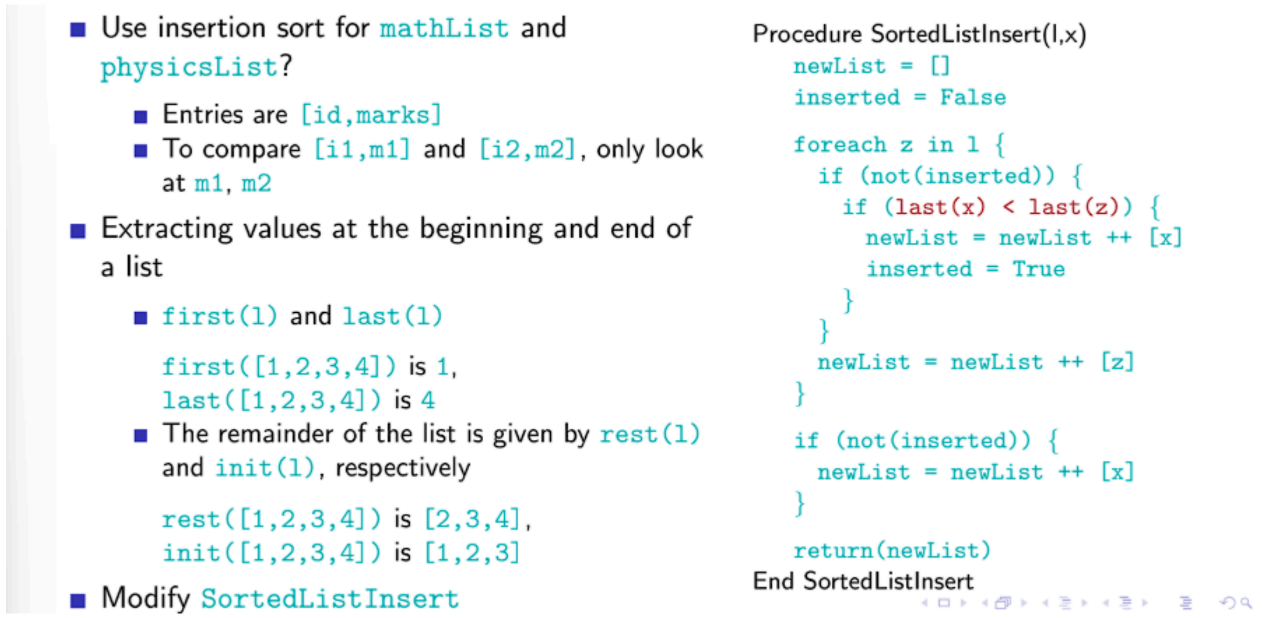
- We want to test the following hypothesis
  Student who perform well in Mathematics perform at least as well in Physics
- Assign grades {A,B,C,D} in both subjects
  - "Perform well in Maths" — grade B or above
  - "Perform at least as well in Physics" — Physics grade $\geq$ Maths grade
- Algorithm
  - Assign grades in each subject
  - Construct lists of students with grades A and B in both subjects — four lists
  - Count students in A list for Maths who are also in A list for Physics
  - Count students in B list for Maths who are also in A list or B list for Physics
  - Use these counts to confirm or reject the hypotheisis

Assigning grades



- Assign grades {A,B,C,D} approximately at quartile boundaries
  - Top 25% get A, next 25% get B, next 25% get C, bottom 25% get D
- To calculate quartiles, extract marks as a list and sort the list
- Need to identify the students — each entry in the marks list is a pair `[StudentId,Marks]`
- Procedure to extract marks information as a list for a subject
- Get the marks lists for Maths and Physics

```
Procecdure BuildMarksList(field)
  marksList = []
  while (Table 1 has more rows) {
    Read the first row X in Table 1
    marksList = marksList ++
                [[X.SeqNo, X.field]]
    Move X to Table 2
  }
  return(marksList)
End BuildMarksList

mathsList = BuildMarksList(Mathematics)

physicsList = BuildMarksList(Physics)
```

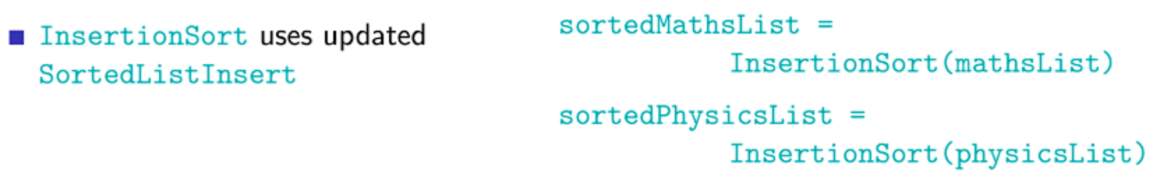


- Use insertion sort for `mathList` and `physicsList`?
  - Entries are `[id,marks]`
  - To compare `[i1,m1]` and `[i2,m2]`, only look at `m1`, `m2`
- Extracting values at the beginning and end of a list
  - `first(l)` and `last(l)`

    `first([1,2,3,4])` is `1`,
    `last([1,2,3,4])` is `4`
  - The remainder of the list is given by `rest(l)` and `init(l)`, respectively

    `rest([1,2,3,4])` is `[2,3,4]`,
    `init([1,2,3,4])` is `[1,2,3]`
- Modify `SortedListInsert`

```
Procedure SortedListInsert(l,x)
  newList = []
  inserted = False

  foreach z in l {
    if (not(inserted)) {
      if (last(x) < last(z)) {
        newList = newList ++ [x]
        inserted = True
      }
    }
    newList = newList ++ [z]
  }

  if (not(inserted)) {
    newList = newList ++ [x]
  }

  return(newList)
End SortedListInsert
```



- `InsertionSort` uses updated `SortedListInsert`

```
sortedMathsList =
        InsertionSort(mathsList)

sortedPhysicsList =
        InsertionSort(physicsList)
```

- Assign grades to a sorted list by quartile
  - `length(l)` returns number of elements in `l`
  - Compute quartile boundaries based on class size

```
Procedure SimpleGradeAssignment(l)
   classSize = length(l)
   q4 = classSize/4
   q3 = classSize/2
   q2 = 3*classSize/4
```

- Assign grades to a sorted list by quartile
  - `length(l)` returns number of elements in `l`
  - Compute quartile boundaries based on class size
  - Initialize list for each grade

```
Procedure SimpleGradeAssignment(l)
   q4 = ..., q3 = ..., q2 = ...
   gradeA = []
   gradeB = []
   gradeC = []
   gradeD = []
```

- Assign grades to a sorted list by quartile
  - `length(l)` returns number of elements in `l`
  - Compute quartile boundaries based on class size
  - Initialize list for each grade
  - Assign grades based on the position in the list
- `SimpleGradeAssignment` returns a list containing four lists, for the four grades

```
Procedure SimpleGradeAssignment(l)
   q4 = ..., q3 = ..., q2 = ...
   gradeA = [], ..., gradeD = []
   position = 0
   foreach x in l {
     if (position > q2) {
       gradeA = gradeA ++ [first(x)]
     }
     if (position > q3 and position <= q2) {
       gradeB = gradeB ++ [first(x)]
     }
     if (position > q4 and position <= q3) {
       gradeC = gradeC ++ [first(x)]
     }
     if (position <= q4) {
       gradeD = gradeD ++ [first(x)]
     }
     position = position + 1
   }
   return([gradeA,gradeB,gradeC,gradeD])
End SimpleGradeAssignment
```

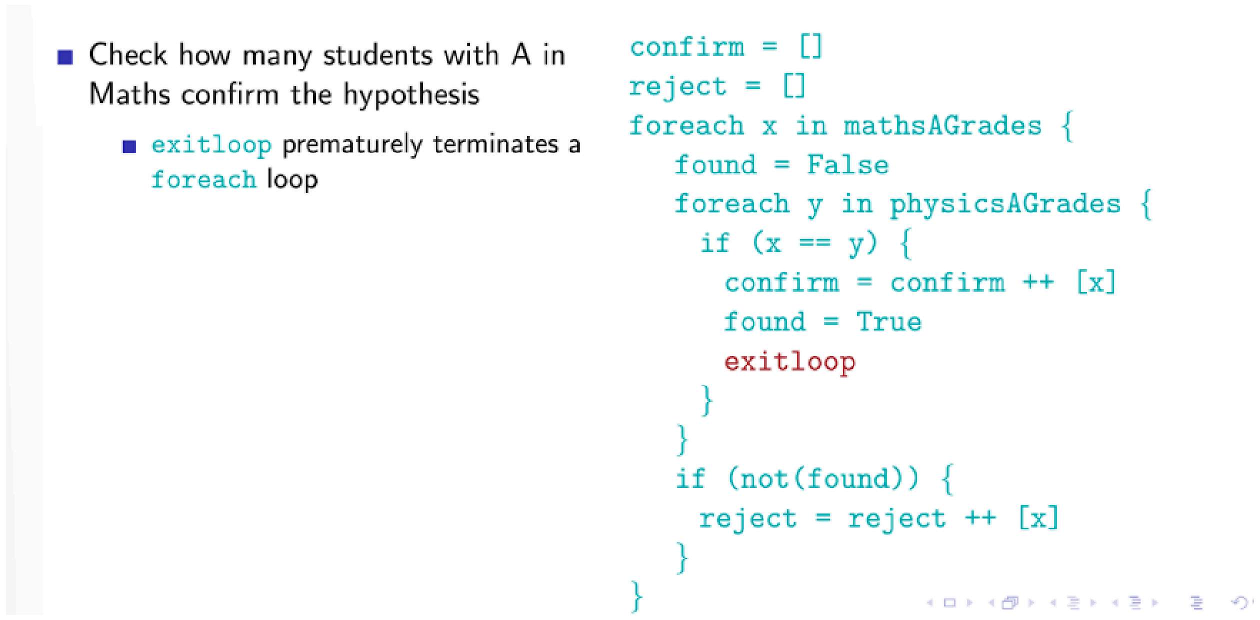
- Assign grades corresponding to Maths and Physics marks

```
mathsGrades =
    SimpleGradeAssignment(sortedMathsList)
physicsGrades =
    SimpleGradeAssignment(sortedPhysicsList)
```

- Unpack the four lists into four separate lists

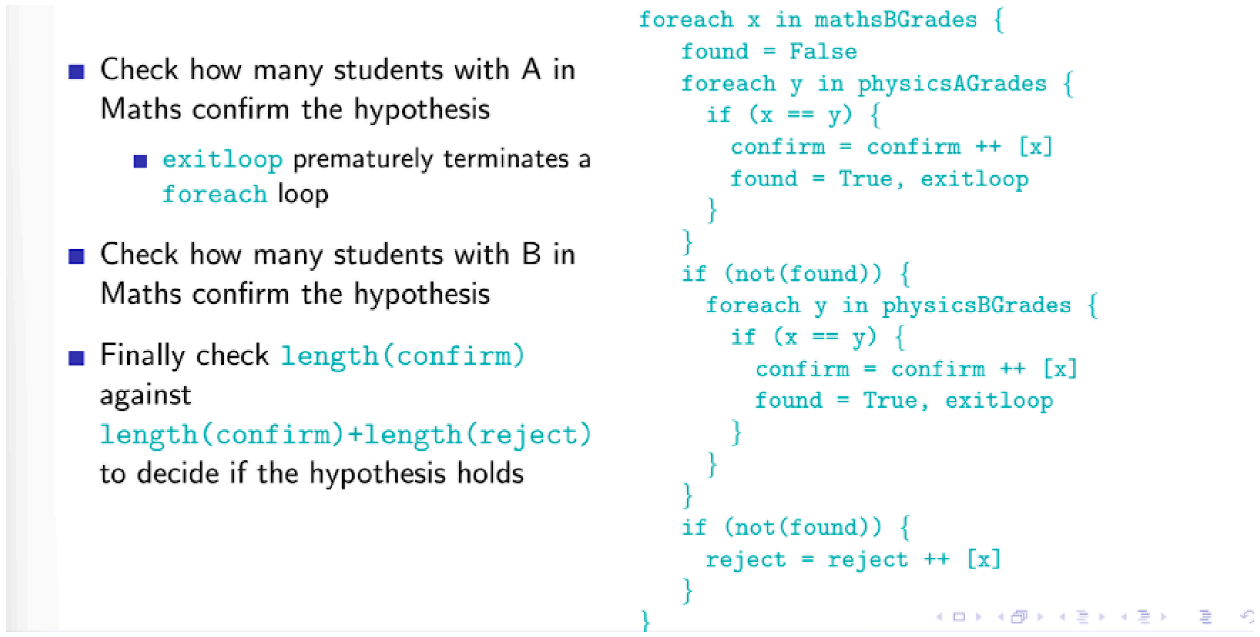
```
mathsAGrades = first(mathsGrades)
mathsBGrades = first(rest(mathsGrades))
mathsCGrades = last(init(mathsGrades))
mathsDGrades = last(mathsGrades)

physicsAGrades = first(physicsGrades)
physicsBGrades = first(rest(physicsGrades))
physicsCGrades = last(init(physicsGrades))
physicsDGrades = last(physicsGrades)
```

Test the hypothesis



- Check how many students with A in Maths confirm the hypothesis
  - `exitloop` prematurely terminates a `foreach` loop

```
confirm = []
reject = []
foreach x in mathsAGrades {
   found = False
   foreach y in physicsAGrades {
     if (x == y) {
       confirm = confirm ++ [x]
       found = True
       exitloop
     }
   }
   if (not(found)) {
     reject = reject ++ [x]
   }
}
```



- Check how many students with A in Maths confirm the hypothesis
  - `exitloop` prematurely terminates a `foreach` loop
- Check how many students with B in Maths confirm the hypothesis
- Finally check `length(confirm)` against `length(confirm)+length(reject)` to decide if the hypothesis holds

```
foreach x in mathsBGrades {
   found = False
   foreach y in physicsAGrades {
     if (x == y) {
       confirm = confirm ++ [x]
       found = True, exitloop
     }
   }
   if (not(found)) {
     foreach y in physicsBGrades {
       if (x == y) {
         confirm = confirm ++ [x]
         found = True, exitloop
       }
     }
   }
   if (not(found)) {
     reject = reject ++ [x]
   }
}
```

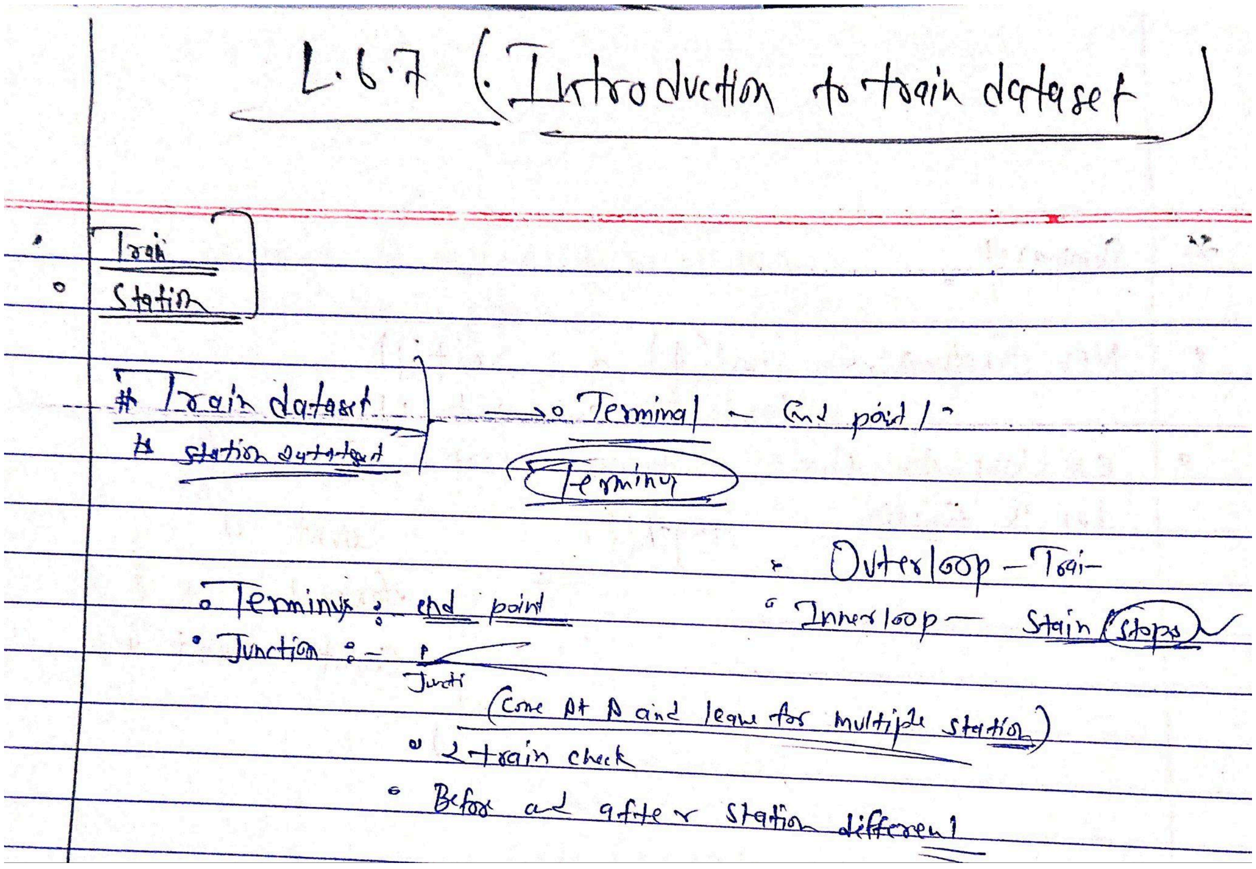
Summary

- Sorting was used to identify quartiles for grade assignment
- Need to modify the comparison function based on the items in the list
- `length(l)` returns number of elements in `l`
- New functions to extract first and last items of a list
  - `first(l)` and `rest(l)`
  - `last(l)` and `init(l)`
- `exitloop` to abort a `foreach` loop

QN : 11

Lecture 7 : Introduction to train dataset

Dataset in 1st week PDF



L.6.7 (Introduction to train dataset)

- Train
- Station

Train dataset
station dataset

→ Terminal – End point /
Terminus

- Outerloop – Train
- Innerloop – Stain (stops)

- Terminus : end point
- Junction :– Juncti
  (Come At A and leave for multiple station)
- 2 train check
- Before and after station different

QN : 2,5,6

PA : 3,4,5,6,8,9,10
GA: 3,4,7