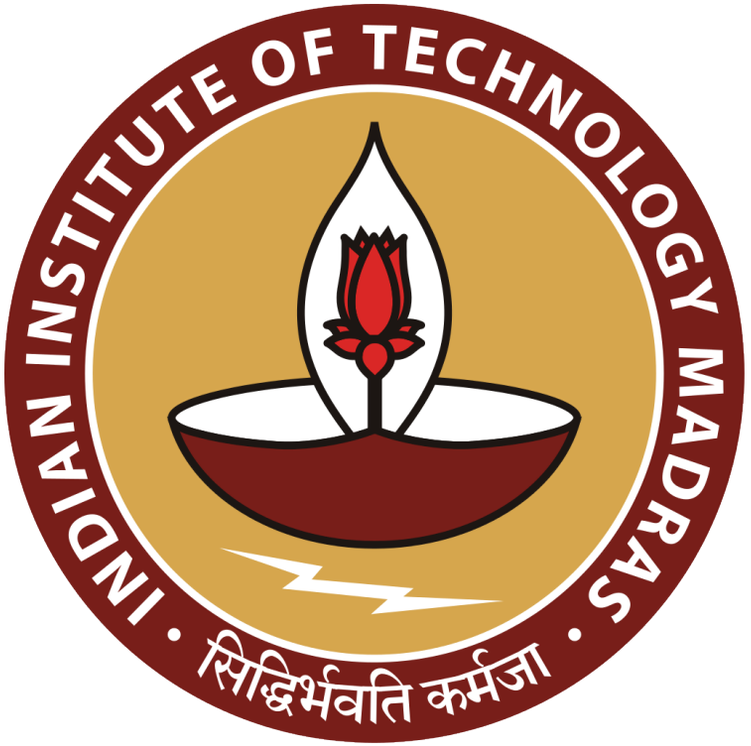

IIT Madras

ONLINE DEGREE

Pseudocode: Sorting lists

Arranging lists in order

- Sorting a list often makes further computations simple

Arranging lists in order

- Sorting a list often makes further computations simple
  - Finding the top k values

Arranging lists in order

- Sorting a list often makes further computations simple
  - Finding the top k values
  - Finding duplicates

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, ...

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

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

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

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`

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`

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`

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

Procedure InsertionSort(I)

Insertion sort

- Once we know how to insert, sorting is easy
- Create an empty list

Procedure InsertionSort(I)

```
sortedList = []
```

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

Procedure InsertionSort(l)

```
sortedList = []

foreach z in l {
  sortedList =
     SortedListInsert(sortedlList,z)
}
```

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

Procedure InsertionSort(l)

```
sortedList = []

foreach z in l {
  sortedList =
     SortedListInsert(sortedlList,z)
}

return(sortedList)
```

End InsertionSort

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

Summmary

- 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