Insertion Sort

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 2

Sorting a list

- You are the TA for a course
  - Instructor has a pile of evaluated exam papers
  - Papers in random order of marks
  - Your task is to arrange the papers in descending order of marks

Sorting a list

- You are the TA for a course
  - Instructor has a pile of evaluated exam papers
  - Papers in random order of marks
  - Your task is to arrange the papers in descending order of marks

Strategy 2

Sorting a list

- You are the TA for a course
  - Instructor has a pile of evaluated exam papers
  - Papers in random order of marks
  - Your task is to arrange the papers in descending order of marks

Strategy 2

- Move the first paper to a new pile

Sorting a list

- You are the TA for a course
  - Instructor has a pile of evaluated exam papers
  - Papers in random order of marks
  - Your task is to arrange the papers in descending order of marks

Strategy 2

- Move the first paper to a new pile
- Second paper
  - Lower marks than first paper? Place below first paper in new pile
  - Higher marks than first paper? Place above first paper in new pile

Sorting a list

- You are the TA for a course
    - Instructor has a pile of evaluated exam papers
    - Papers in random order of marks
    - Your task is to arrange the papers in descending order of marks

Strategy 2

- Move the first paper to a new pile
- Second paper
    - Lower marks than first paper? Place below first paper in new pile
    - Higher marks than first paper? Place above first paper in new pile
- Third paper
    - Insert into correct position with respect to first two

Sorting a list

- You are the TA for a course
  - Instructor has a pile of evaluated exam papers
  - Papers in random order of marks
  - Your task is to arrange the papers in descending order of marks

Strategy 2

- Move the first paper to a new pile
- Second paper
  - Lower marks than first paper? Place below first paper in new pile
  - Higher marks than first paper? Place above first paper in new pile
- Third paper
  - Insert into correct position with respect to first two
- Do this for the remaining papers
  - Insert each one into correct position in the second pile

Sorting a list

74	32	89	55	21	64

Sorting a list

~~74~~ 32 89 55 21 64

74

Sorting a list

~~74~~	~~32~~	89	55	21	64
32	74				

Sorting a list

~~74~~ ~~32~~ ~~89~~ 55 21 64

32 74 89

Sorting a list

~~74~~ ~~32~~ ~~89~~ ~~55~~ 21 64

32 55 74 89

Sorting a list

~~74~~ ~~32~~ ~~89~~ ~~55~~ ~~21~~ 64

21 32 55 74 89

Sorting a list

~~74~~	~~32~~	~~89~~	~~55~~	~~21~~	~~64~~

21	32	55	64	74	89

Insertion sort

- Start building a new sorted list

Insertion sort

- Start building a new sorted list
- Pick next element and insert it into the sorted list

Insertion sort

- Start building a new sorted list
- Pick next element and insert it into the sorted list
- An iterative formulation
  - Assume `L[:i]` is sorted
  - Insert `L[i]` in `L[:i]`

Insertion sort

- Start building a new sorted list
- Pick next element and insert it into the sorted list
- An iterative formulation
  - Assume `L[:i]` is sorted
  - Insert `L[i]` in `L[:i]`

```
def InsertionSort(L):
  n = len(L)
  if n < 1:
    return(L)
  for i in range(n):
    # Assume L[:i] is sorted
    # Move L[i] to correct position in L
    j = i
    while(j > 0 and L[j] < L[j-1]):
      (L[j],L[j-1]) = (L[j-1],L[j])
      j = j-1
    # Now L[:i+1] is sorted
  return(L)
```

Insertion sort

- Start building a new sorted list
- Pick next element and insert it into the sorted list
- An iterative formulation
  - Assume `L[:i]` is sorted
  - Insert `L[i]` in `L[:i]`
- A recursive formulation
  - Inductively sort `L[:i]`
  - Insert `L[i]` in `L[:i]`

```
def InsertionSort(L):
  n = len(L)
  if n < 1:
    return(L)
  for i in range(n):
    # Assume L[:i] is sorted
    # Move L[i] to correct position in L
    j = i
    while(j > 0 and L[j] < L[j-1]):
      (L[j],L[j-1]) = (L[j-1],L[j])
      j = j-1
    # Now L[:i+1] is sorted
  return(L)
```

Insertion sort

- Start building a new sorted list
- Pick next element and insert it into the sorted list
- An iterative formulation
  - Assume `L[:i]` is sorted
  - Insert `L[i]` in `L[:i]`
- A recursive formulation
  - Inductively sort `L[:i]`
  - Insert `L[i]` in `L[:i]`

```
def Insert(L,v):
    n = len(L)
    if n == 0:
      return([v])
    if v >= L[-1]:
      return(L+[v])
    else:
      return(Insert(L[:-1],v)+L[-1:])

def ISort(L):
    n = len(L)
    if n < 1:
       return(L)
    L = Insert(ISort(L[:-1]),L[-1])
    return(L)
```

Analysis of iterative insertion sort

- Correctness follows from the invariant

```
def InsertionSort(L):
    n = len(L)
    if n < 1:
        return(L)
    for i in range(n):
        # Assume L[:i] is sorted
        # Move L[i] to correct position in L
        j = i
        while(L[j] < L[j-1]):
            (L[j],L[j-1]) = (L[j-1],L[j])
            j = j-1
        # Now L[:i+1] is sorted
    return(L)
```

Analysis of iterative insertion sort

- Correctness follows from the invariant
- Efficiency

```
def InsertionSort(L):
    n = len(L)
    if n < 1:
        return(L)
    for i in range(n):
        # Assume L[:i] is sorted
        # Move L[i] to correct position in L
        j = i
        while(L[j] < L[j-1]):
          (L[j],L[j-1]) = (L[j-1],L[j])
          j = j-1
        # Now L[:i+1] is sorted
    return(L)
```

Analysis of iterative insertion sort

- Correctness follows from the invariant
- Efficiency
  - Outer loop iterates n times

```
def InsertionSort(L):
  n = len(L)
  if n < 1:
    return(L)
  for i in range(n):
    # Assume L[:i] is sorted
    # Move L[i] to correct position in L
    j = i
    while(L[j] < L[j-1]):
      (L[j],L[j-1]) = (L[j-1],L[j])
      j = j-1
    # Now L[:i+1] is sorted
  return(L)
```

Analysis of iterative insertion sort

- Correctness follows from the invariant
- Efficiency
  - Outer loop iterates n times
  - Inner loop: i steps to insert `L[i]` in `L[:i]`

```
def InsertionSort(L):
  n = len(L)
  if n < 1:
    return(L)
  for i in range(n):
    # Assume L[:i] is sorted
    # Move L[i] to correct position in L
    j = i
    while(L[j] < L[j-1]):
      (L[j],L[j-1]) = (L[j-1],L[j])
      j = j-1
    # Now L[:i+1] is sorted
  return(L)
```

Analysis of iterative insertion sort

- Correctness follows from the invariant
- Efficiency
  - Outer loop iterates n times
  - Inner loop: i steps to insert `L[i]` in `L[:i]`
  - $T(n) = 0 + 1 + \cdots + (n-1)$

```
def InsertionSort(L):
    n = len(L)
    if n < 1:
        return(L)
    for i in range(n):
        # Assume L[:i] is sorted
        # Move L[i] to correct position in L
        j = i
        while(L[j] < L[j-1]):
            (L[j],L[j-1]) = (L[j-1],L[j])
            j = j-1
        # Now L[:i+1] is sorted
    return(L)
```

Analysis of iterative insertion sort

- Correctness follows from the invariant
- Efficiency
  - Outer loop iterates n times
  - Inner loop: i steps to insert `L[i]` in `L[:i]`
  - $T(n) = 0 + 1 + \cdots + (n-1)$
  - $T(n) = n(n-1)/2$

```
def InsertionSort(L):
  n = len(L)
  if n < 1:
    return(L)
  for i in range(n):
    # Assume L[:i] is sorted
    # Move L[i] to correct position in L
    j = i
    while(L[j] < L[j-1]):
      (L[j],L[j-1]) = (L[j-1],L[j])
      j = j-1
    # Now L[:i+1] is sorted
  return(L)
```

Analysis of iterative insertion sort

- Correctness follows from the invariant
- Efficiency
  - Outer loop iterates n times
  - Inner loop: i steps to insert `L[i]` in `L[:i]`
  - $T(n) = 0 + 1 + \cdots + (n-1)$
  - $T(n) = n(n-1)/2$
- $T(n)$ is $O(n^2)$

```
def InsertionSort(L):
    n = len(L)
    if n < 1:
        return(L)
    for i in range(n):
        # Assume L[:i] is sorted
        # Move L[i] to correct position in L
        j = i
        while(L[j] < L[j-1]):
            (L[j],L[j-1]) = (L[j-1],L[j])
            j = j-1
        # Now L[:i+1] is sorted
    return(L)
```

Analysis of recursive insertion sort

- For input of size n, let
  - $TI(n)$ be the time taken by `Insert`
  - $TS(n)$ be the time taken by `ISort`

```
def Insert(L,v):
   n = len(L)
   if n == 0:
     return([v])
   if v >= L[-1]:
     return(L+[v])
   else
     return(Insert(L[:-1],v)+l[-1:])

def ISort(L):
   n = len(L)
   if n < 1:
      return(L)
   L = Insert(ISort(L[:-1]),L[-1])
   return(L)
```

Analysis of recursive insertion sort

- For input of size n, let
  - $TI(n)$ be the time taken by `Insert`
  - $TS(n)$ be the time taken by `ISort`
- First calculate $TI(n)$ for `Insert`
  - $TI(0) = 1$
  - $TI(n) = TI(n-1) + 1$

```
def Insert(L,v):
   n = len(L)
   if n == 0:
     return([v])
   if v >= L[-1]:
     return(L+[v])
   else
     return(Insert(L[:-1],v)+L[-1:])

def ISort(L):
   n = len(L)
   if n < 1:
      return(L)
   L = Insert(ISort(L[:-1]),L[-1])
   return(L)
```

Analysis of recursive insertion sort

- For input of size n, let
  - $TI(n)$ be the time taken by `Insert`
  - $TS(n)$ be the time taken by `ISort`
- First calculate $TI(n)$ for `Insert`
  - $TI(0) = 1$
  - $TI(n) = TI(n-1) + 1$
  - Unwind to get $TI(n) = n$

```
def Insert(L,v):
  n = len(L)
  if n == 0:
    return([v])
  if v >= L[-1]:
    return(L+[v])
  else
    return(Insert(L[:-1],v)+L[-1:])

def ISort(L):
  n = len(L)
  if n < 1:
     return(L)
  L = Insert(ISort(L[:-1]),L[-1])
  return(L)
```

Analysis of recursive insertion sort

- For input of size n, let
  - $TI(n)$ be the time taken by `Insert`
  - $TS(n)$ be the time taken by `ISort`
- First calculate $TI(n)$ for `Insert`
  - $TI(0) = 1$
  - $TI(n) = TI(n-1) + 1$
  - Unwind to get $TI(n) = n$
- Set up a recurrence for $TS(n)$
  - $TS(0) = 1$
  - $TS(n) = TS(n-1) + TI(n-1)$

```
def Insert(L,v):
   n = len(L)
   if n == 0:
     return([v])
   if v >= L[-1]:
     return(L+[v])
   else
     return(Insert(L[:-1],v)+l[-1:])

def ISort(L):
   n = len(L)
   if n < 1:
      return(L)
   L = Insert(ISort(L[:-1]),L[-1])
   return(L)
```

Analysis of recursive insertion sort

- For input of size n, let
  - $TI(n)$ be the time taken by `Insert`
  - $TS(n)$ be the time taken by `ISort`
- First calculate $TI(n)$ for `Insert`
  - $TI(0) = 1$
  - $TI(n) = TI(n-1) + 1$
  - Unwind to get $TI(n) = n$
- Set up a recurrence for $TS(n)$
  - $TS(0) = 1$
  - $TS(n) = TS(n-1) + TI(n-1)$
- Unwind to get $1 + 2 + \cdots + n - 1$

```
def Insert(L,v):
   n = len(L)
   if n == 0:
     return([v])
   if v >= L[-1]:
     return(L+[v])
   else
     return(Insert(L[:-1],v)+l[-1:])

def ISort(L):
   n = len(L)
   if n < 1:
      return(L)
   L = Insert(ISort(L[:-1]),L[-1])
   return(L)
```

Summary

- Insertion sort is another intuitive algorithm to sort a list

Summary

- Insertion sort is another intuitive algorithm to sort a list
- Create a new sorted list
- Repeatedly insert elements into the sorted list

Summary

- Insertion sort is another intuitive algorithm to sort a list
- Create a new sorted list
- Repeatedly insert elements into the sorted list
- Worst case complexity is $O(n^2)$
  - Unlike selection sort, not all cases take time n^2
  - If list is already sorted, `Insert` stops in 1 step
  - Overall time can be close to $O(n)$