19. Sorting a List

Topics:
- Selection Sort
- Merge Sort

Our examples will highlight the interplay between functions and lists.

Sorting a List of Numbers

Before:

\[ x \rightarrow [50, 40, 10, 80, 20, 60] \]

After:

\[ x \rightarrow [10, 20, 40, 50, 60, 80] \]

We Will First Implement the Method of Selection Sort

At the Start:

\[ x \rightarrow [50, 40, 10, 80, 20, 60] \]

High-Level:

for \( k \) in range(len(x) - 1)

Swap \( x[k] \) with the smallest value in \( x[k:] \)

Selection Sort: How It Works

Before:

\[ x \rightarrow [50, 40, 10, 80, 20, 60] \]

Swap \( x[0] \) with the smallest value in \( x[0:] \)

After:

\[ x \rightarrow [10, 40, 50, 80, 20, 60] \]
Selection Sort: How It Works

Before:
\[ x \rightarrow 10 \ 40 \ 50 \ 80 \ 20 \ 60 \]
Swap x[1] with the smallest value in x[1:]

After:
\[ x \rightarrow 10 \ 20 \ 50 \ 80 \ 40 \ 60 \]

Selection Sort: How It Works

Before:
\[ x \rightarrow 10 \ 20 \ 50 \ 80 \ 40 \ 60 \]
Swap x[2] with the smallest value in x[2:]

After:
\[ x \rightarrow 10 \ 20 \ 40 \ 80 \ 50 \ 60 \]

Selection Sort: How It Works

Before:
\[ x \rightarrow 10 \ 20 \ 40 \ 80 \ 50 \ 60 \]
Swap x[3] with the smallest value in x[3:]

After:
\[ x \rightarrow 10 \ 20 \ 40 \ 80 \ 50 \ 60 \]
Selection Sort: How It Works

Before:

\[ x \rightarrow 10 \ 20 \ 40 \ 50 \ 80 \ 60 \]

Swap \( x[4] \) with the smallest value in \( x[4:] \)

After:

\[ x \rightarrow 10 \ 20 \ 40 \ 50 \ 80 \ 60 \]

Selection Sort: Recap

The Essential Helper Function:

\[
\text{def Select(x,i):}
\]

\[
\text{""" Swaps the smallest value in } x[i:] \text{ with } x[i] \text{"
}
\]

PreC: \( x \) is a list of integers and \( i \) is an index that satisfies \( 0 \leq i \leq \text{len}(x) \)

Does not return anything and it has a list argument

How Does it Work?

The calling program has a list. E.g.,

\[
a \rightarrow 0 \rightarrow 50
1 \rightarrow 40
2 \rightarrow 10
3 \rightarrow 80
4 \rightarrow 20
5 \rightarrow 60
\]

The calling program executes \( \text{Select}(a,0) \) and control passes to \( \text{Select} \)
How Does Select Work?

- Nothing new about the assignment of 0 to i.
- But there is no assignment of the list a to x.
- Instead x now refers to the same list as a.

```
<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
```

If inside Select we have
\[ t = x[0]; x[0] = x[2]; x[2] = t \]
it is as if we said
\[ t = a[0]; a[0] = a[2]; a[2] = t \]

```
<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
```

How Does Select Work?

It changes the list a in the calling program.
We say x and a are aliased. They refer to the same list

```
<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>
```

Let's Assume This Is Implemented

```
def Select(x,i):
    """ Swaps the smallest value in x[i:] with x[i]"
    t = x[i]; x[i] = x[2]; x[2] = t
    PreC: x is a list of integers and i is an in that satisfies 0<=i<len(x)""
```

In General We Have This

```
def SelectionSort(a):
    n = len(a)
    for k in range(n):
        Select(a,k)
```
Merging Two Sorted Lists into a Single Sorted List

Example

\[ x \to \begin{array}{c} 12 \, 33 \, 35 \, 45 \\ y \to \begin{array}{c} 15 \, 42 \, 55 \, 65 \, 75 \end{array} \end{array} \]

x and y are input
They are sorted
z is the output

\[ z \to \begin{array}{c} 12 \, 15 \, 33 \, 35 \, 42 \, 45 \, 55 \, 65 \, 75 \end{array} \]

Merging Two Sorted Lists

\[ x \to \begin{array}{c} 12 \, 33 \, 35 \, 45 \end{array} \]
\[ y \to \begin{array}{c} 15 \, 42 \, 55 \, 65 \, 75 \end{array} \]
\[ z \to \begin{array}{c} \end{array} \]

ix: 0
iy: 0
Do we pick from x? \( x[ix] \leq y[iy] \) ???

Merging Two Sorted Lists

\[ x \to \begin{array}{c} 12 \, 33 \, 35 \, 45 \end{array} \]
\[ y \to \begin{array}{c} 15 \, 42 \, 55 \, 65 \, 75 \end{array} \]
\[ z \to \begin{array}{c} \end{array} \]

ix: 0
iy: 0
Do we pick from x? \( x[ix] \leq y[iy] \) ???

Merge

\[ x \to \begin{array}{c} 12 \, 33 \, 35 \, 45 \end{array} \]
\[ y \to \begin{array}{c} 15 \, 42 \, 55 \, 65 \, 75 \end{array} \]
\[ z \to \begin{array}{c} 12 \end{array} \]

ix: 0
iy: 0
Yes. So update ix

Merge

\[ x \to \begin{array}{c} 12 \, 33 \, 35 \, 45 \end{array} \]
\[ y \to \begin{array}{c} 15 \, 42 \, 55 \, 65 \, 75 \end{array} \]
\[ z \to \begin{array}{c} 12 \end{array} \]

ix: 1
iy: 0
Do we pick from x? \( x[ix] \leq y[iy] \) ???
Merge

x-> 12 33 35 45  
 y-> 15 42 55 65 75  
 z-> 12 15  

No. So update iy

ix: 1  
iy: 0  
iz:

Merge

x-> 12 33 35 45  
 y-> 15 42 55 65 75  
 z-> 12 15  

Do we pick from x?  x[ix] <= y[iy]  ???

Merge

x-> 12 33 35 45  
 y-> 15 42 55 65 75  
 z-> 12 15 33  

Yes. So update ix

ix: 1  
iy: 1  
iz:

Merge

x-> 12 33 35 45  
 y-> 15 42 55 65 75  
 z-> 12 15 33  

Do we pick from x?  x[ix] <= y[iy]  ???

Merge

x-> 12 33 35 45  
 y-> 15 42 55 65 75  
 z-> 12 15 33  

Yes. So update ix

ix: 2  
iy: 1  
iz:

Merge

x-> 12 33 35 45  
 y-> 15 42 55 65 75  
 z-> 12 15 33 35  

Do we pick from x?  x[ix] <= y[iy]  ???
Merge

\[ x \rightarrow 12 \quad 33 \quad 35 \quad 45 \]
\[ y \rightarrow 15 \quad 42 \quad 55 \quad 65 \quad 75 \]
\[ z \rightarrow 12 \quad 15 \quad 33 \quad 35 \quad 42 \]

No. So update iy...

Do we pick from x? \[ x[ix] \leq y[iy] \] ???

Merge

\[ x \rightarrow 12 \quad 33 \quad 35 \quad 45 \]
\[ y \rightarrow 15 \quad 42 \quad 55 \quad 65 \quad 75 \]
\[ z \rightarrow 12 \quad 15 \quad 33 \quad 35 \quad 42 \]

Yes. So update ix.

Done with x. Pick from y

Merge

\[ x \rightarrow 12 \quad 33 \quad 35 \quad 45 \]
\[ y \rightarrow 15 \quad 42 \quad 55 \quad 65 \quad 75 \]
\[ z \rightarrow 12 \quad 15 \quad 33 \quad 35 \quad 42 \quad 45 \]

Yes. So update ix.

Done with x. Pick from y

Merge

\[ x \rightarrow 12 \quad 33 \quad 35 \quad 45 \]
\[ y \rightarrow 15 \quad 42 \quad 55 \quad 65 \quad 75 \]
\[ z \rightarrow 12 \quad 15 \quad 33 \quad 35 \quad 42 \quad 45 \quad 55 \]

So update iy

Done with x. Pick from y

Merge

\[ x \rightarrow 12 \quad 33 \quad 35 \quad 45 \]
\[ y \rightarrow 15 \quad 42 \quad 55 \quad 65 \quad 75 \]
\[ z \rightarrow 12 \quad 15 \quad 33 \quad 35 \quad 42 \quad 45 \quad 55 \]

Done with x. Pick from y
The Python Implementation...

```python
def Merge(x, y):
    n = len(x); m = len(y);
    ix = 0; iy = 0; z = []
    for iz in range(n+m):
        if ix>=n:
            z.append(y[iy]); iy+=1
        elif iy>=m:
            z.append(x[ix]); ix+=1
        elif x[ix] <= y[iy]:
            z.append(x[ix]); ix+=1
        else:
            z.append(y[iy]); iy+=1
    return z
```

x-list exhausted  y-list exhausted  x-value smaller  y-value smaller
def Merge(x,y):
    n = len(x); m = len(y);
    ix = 0; iy = 0; z = []
    for iz in range(n+m):
        if ix>=n:
            z.append(y[iy]); iy+=1
        elif iy>=m:
            z.append(x[ix]); ix+=1
        elif x[ix] <= y[iy]:
            z.append(x[ix]); ix+=1
        else:
            z.append(y[iy]); iy+=1
    return z

Implementation Using Pop

def Merge(x,y):
    u = list(x)
    v = list(y)
    z = []
    while len(u)>0 and len(v)>0 :
        if u[0]<= v[0]:
            g = u.pop(0)
        else:
            g = v.pop(0)
        z.append(g)
    z.extend(u)
    z.extend(v)
    return z

Implementation Using Pop

def Merge(x,y):
    u = list(x)
    v = list(y)
    z = []
    while len(u)>0 and len(v)>0 :
        if u[0]<= v[0]:
            g = u.pop(0)
        else:
            g = v.pop(0)
        z.append(g)
    z.extend(u)
    z.extend(v)
    return z

Implementation Using Pop

def Merge(x,y):
    u = list(x)
    v = list(y)
    z = []
    while len(u)>0 and len(v)>0 :
        if u[0]<= v[0]:
            g = u.pop(0)
        else:
            g = v.pop(0)
        z.append(g)
    z.extend(u)
    z.extend(v)
    return z

Implementation Using Pop

def Merge(x,y):
    u = list(x)
    v = list(y)
    z = []
    while len(u)>0 and len(v)>0 :
        if u[0]<= v[0]:
            g = u.pop(0)
        else:
            g = v.pop(0)
        z.append(g)
    z.extend(u)
    z.extend(v)
    return z

Implementation Using Pop

def Merge(x,y):
    u = list(x)
    v = list(y)
    z = []
    while len(u)>0 and len(v)>0 :
        if u[0]<= v[0]:
            g = u.pop(0)
        else:
            g = v.pop(0)
        z.append(g)
    z.extend(u)
    z.extend(v)
    return z

Implementation Using Pop
Implementation Using Pop

```python
def Merge(x,y):
    u = list(x)
    v = list(y)
    z = []
    while len(u) > 0 and len(v) > 0:
        if u[0] <= v[0]:
            g = u.pop(0)
        else:
            g = v.pop(0)
        z.append(g)
    z.extend(u)
    z.extend(v)
    return z
```

MergeSort

Binary Search is an example of a "divide and conquer" approach to problem solving.

A method for sorting a list that features this strategy is MergeSort.

Motivation

You are asked to sort a list but you have two "helpers": H1 and H2.

Idea:

1. Split the list in half and have each helper sort one of the halves.
2. Then merge the two sorted lists into a single larger list.

This idea can be repeated if H1 has two helpers and H2 has two helpers.

Subdivide the Sorting Task

And Again
And One Last Time

Now Merge

And Merge Again

And Again

And One Last Time

Done!
Let's write a function to do this making use of

```python
def Merge(x, y):
    """ Returns a float list that is the merge of sorted lists x and y.
    PreC: x and y are lists of floats that are sorted from small to big.
    """
```

**Handcoding the n =16 case**

- \(A_0 = \text{Merge}(a[0], a[1])\)
- \(A_1 = \text{Merge}(a[2], a[3])\)
- \(A_2 = \text{Merge}(a[4], a[5])\)
- \(A_3 = \text{Merge}(a[6], a[7])\)
- \(A_4 = \text{Merge}(a[8], a[9])\)
- \(A_5 = \text{Merge}(a[10], a[11])\)
- \(A_6 = \text{Merge}(a[12], a[13])\)
- \(A_7 = \text{Merge}(a[14], a[15])\)

**8 Merges Producing length-2 lists**

**4 Merges Producing length-4 lists**

**2 Merges Producing length-8 lists**
Handcoding the n = 16 case

C_0 = Merge(B_0, B_1)
C_1 = Merge(B_2, B_3)

All Done!

D_0 = Merge(C_0, C_1)

For general n, it can be handled using recursion.

Back To Merge Sort

1 Merge Producing a Length-16 List

Recursive Merge Sort

```
def MergeSort(a):
    n = length(a)
    if n==1:
        return a
    else:
        m = n/2
        u_0 = list(a[:m])
        u_1 = list(a[m:])
        y_0 = MergeSort(u_0)
        y_1 = MergeSort(u_1)
        return Merge(y_0, y_1)
```

Recursive Merge Sort

```
def MergeSort(a):
    n = length(a)
    if n==1:
        return a
    else:
        m = n/2
        u_0 = list(a[:m])
        u_1 = list(a[m:])
        y_0 = MergeSort(u_0)
        y_1 = MergeSort(u_1)
        return Merge(y_0, y_1)
```
A Sorted List is produced at each "." Let's look at the order in which lists are sorted.
A Sorted List is produced at each "::" Let's look at the order in which lists are sorted.
Some Conclusions

Infinite recursion (like infinite loops) can happen so careful reasoning is required.

Will we reach the "base case"?

In `MergeSort`, a recursive call always involves a list that is shorter than the input list. So eventually we reach the `len(a)==1` base case.