More List Processing

- list methods
- searching
- sorting

Searching for items in a list

```
def mymember(list,element):
    for item in list:
        if item == element:
            return True
    return False
```

How many times does the loop run in the worst case?

Searching for items in a list

```
def mymember(list,element):
    for item in list:
        if item == element:
            return True
    return False
```

Binary Search

- If the list is empty, the answer is False.
- Otherwise, check the midpoint of the list.
- If the desired item is found there, then you are done. The answer is True.
- If the desired item is greater than the midpoint, forget about the first half of the list, and repeat the process on the second half of the list.
- If the desired item is smaller than the midpoint, forget about the second half of the list, and repeat the process on the first half of the list.

A Visualization of Binary Search

http://www.dave-reed.com/book/source.html

Binary Search

```
def binsearch (element, list):
    if list==[]:
        answer = False
    else:
        mid = len(list)/2
        miditem = list[mid]
        if element == miditem:
            answer = True
        elif element < miditem:
            answer = binsearch(element, list[:mid-1])
        elif element > miditem:
            answer = binsearch(element, list[mid+1:])
    return answer
```

Binary Search

```
def binsearch (element, list):
    if list==[]:
        answer = False
    else:
        mid = len(list)/2
        miditem = list[mid]
        if element == miditem:
            answer = True
        elif element < miditem:
            answer = binsearch(element, list[:mid-1])
        elif element > miditem:
            answer = binsearch(element, list[mid+1:])
    return answer
```

If the list is empty, the answer is False.

Binary Search

```
def binsearch (element, list):
    if list==[]:
        answer = False
    else:
        mid = len(list)/2
        miditem = list[mid]
        if element == miditem:
            answer = True
        elif element < miditem:
            answer = binsearch(element, list[:mid-1])
        elif element > miditem:
            answer = binsearch(element, list[mid+1:])
    return answer
```

Otherwise, check the midpoint of the list.

Binary Search

```
def binsearch (element, list):
    if list==[]:
        answer = False
    else:
        mid = len(list)/2
        miditem = list[mid]
        if element == miditem:
            answer = True
        elif element < miditem:
            answer = binsearch(element, list[:mid-1])
        elif element > miditem:
            answer = binsearch(element, list[mid+1:])
    return answer
```

If the desired item is found there, you are done. The answer is True.

Binary Search

```
def binsearch (element, list):
    if list==[]:
        answer = False
    else:
        mid = len(list)/2
        miditem = list[mid]
        if element == miditem:
            answer = True
        elif element < miditem:
            answer = binsearch(element, list[:mid-1])
        elif element > miditem:
            answer = binsearch(element, list[mid+1:])
    return answer
```

If the desired item is **smaller** than the midpoint, forget about the second half of the list, and repeat the process with the first half of the list.

Binary Search

```
def binsearch (element, list):
    if list==[]:
        answer = False
    else:
        mid = len(list)/2
        miditem = list[mid]
        if element == miditem:
            answer = True
        elif element < miditem:
            answer = binsearch(e
        elif element > miditem:
            answer = binsearch(element, list[mid+1:])
    return answer
```

If the desired item is **greater** than the midpoint, forget about the second half of the list, and repeat the process with the first half of the list.

Binary Search

```
def binsearch (element, list):
    if list==[]:
        answer = False
    else:
        mid = len(list)/2
        miditem = list[mid]
        if eleme
            answer = True
        elif element < miditem:
            answer = binsearch(element, list[:mid-1])
        elif element > miditem:
            answer = binsearch(element, list[mid+1:])
    return answer
```

But how do we sort the list?

Bubble Sort

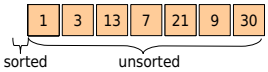
- Compare the first and the second element of the list and swap them if the first is greater than the second.
- Do the same for the second and third element, the third and forth element, and so on until you reach the end of the list.
- Repeat the whole process until you go through the list without making any swaps.

Selection Sort

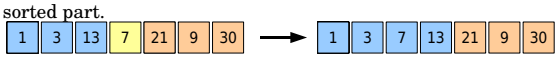
- If the list is empty or has length 1, there is nothing to sort. We are done.
- Otherwise, find the smallest element in the list and swap it with the first element.
- Then repeat the process with the rest of the list.

Insertion Sort

- Take the list to consist of a sorted part and an unsorted part.
- In the beginning, the sorted part is the slice from position 0 to position 0 and the unsorted part is the rest.

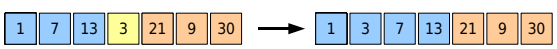


- Take the first element of the unsorted part and insert it into the sorted part.



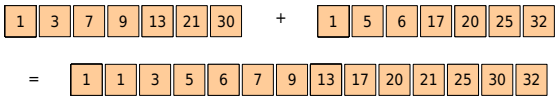
- Repeat this last step until the sorted part spans the whole list and the unsorted part is empty.

Insertion Sort – how to insert

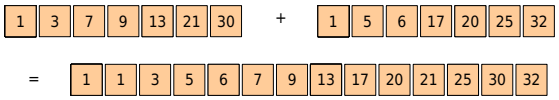
- 

- "Remember" the first element of the unsorted part. Let's call it `toInsert`.
- Compare `toInsert` to the elements of the sorted part one by one starting from the right.
- While the element of the sorted part is greater than `toInsert`, shift it one step to the right.
- When the element of the sorted part is smaller than `toInsert`, insert `toInsert` to the right of that element.

Merging two lists which are sorted



Merging two lists which are sorted



- Starting at position 0, compare an element of the first list to an element of the second list. Append the smaller element to the new list and move to the next position in that list.
- Keep doing this until you have completely traversed one of the lists.
- Take what's left of the other list and concatenate it to the end of the new list.

Merging two sorted lists in Python

```
def merge (l1,l2):
    pos1 = 0
    pos2 = 0
    lout = []

    while pos1 < len(l1) and pos2 < len(l2):
        if l1[pos1] < l2[pos2]:
            lout.append(l1[pos1])
            pos1 += 1
        else:
            lout.append(l2[pos2])
            pos2 += 1

    if pos1 < len(l1):
        lout.extend(l1[pos1:])
    elif pos2 < len(l2):
        lout.extend(l2[pos2:])

    return lout
```

Merging two sorted lists in Python

```
def merge (l1,l2):
    pos1 = 0
    pos2 = 0
    lout = []

    while pos1 < len(l1) and pos2 < len(l2):
        if l1[pos1] < l2[pos2]:
            lout.append(l1[pos1])
            pos1 += 1
        else:
            lout.append(l2[pos2])
            pos2 += 1

    if pos1 < len(l1):
        lout.extend(l1[pos1:])
    elif pos2 < len(l2):
        lout.extend(l2[pos2:])

    return lout
```

Starting at position 0 ...

Merging two sorted

... compare an element of the first list to an element of the second list. Append the smaller element to the new list and move to the next position in that list.

Keep doing this until you have completely traversed one of the lists.

```
def merge (l1,l2):
    pos1 = 0
    pos2 = 0
    lout = []

    while pos1 < len(l1) and pos2 < len(l2):
        if l1[pos1] < l2[pos2]:
            lout.append(l1[pos1])
            pos1 += 1
        else:
            lout.append(l2[pos2])
            pos2 += 1

    if pos1 < len(l1):
        lout.extend(l1[pos1:])
    elif pos2 < len(l2):
        lout.extend(l2[pos2:])

    return lout
```

Merging two sorted lists in Python

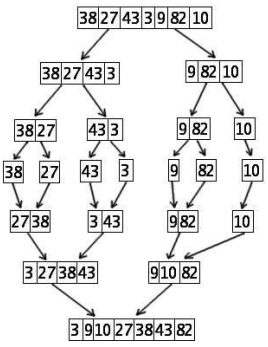
```
def merge (l1,l2):
    pos1 = 0
    pos2 = 0
    lout = []

    while pos1 < len(l1) and pos2 < len(l2):
        if l1[pos1] < l2[pos2]:
            lout.append(l1[pos1])
            pos1 += 1
        else:
            lout.append(l2[p
            pos2 += 1

    if pos1 < len(l1):
        lout.extend(l1[pos1:])
    elif pos2 < len(l2):
        lout.extend(l2[pos2:])

    return lout
```

Take what's left of the other list and concatenate it to the end of the new list.

Merge Sort

- If the list is empty or has length 1, there is nothing to sort. We are done.
- Otherwise, split the list in two halves.
- Sort each of the the halves.
- Merge the sorted halves back together.

Merge Sort



Merge Sort in Python

```
def mergesort (list):
    if len(list) < 2:
        return list
    else:
        mid = len(list)/2
        l1 = list[:mid]
        l2 = list[mid:]
        l1sorted = mergesort(l1)
        l2sorted = mergesort(l2)
        sorted = merge(l1sorted, l2sorted)
        return sorted
```

Merge Sort in Python

```
def mergesort (list):
    if len(list) < 2:
        return list
    else:
        mid = len(list)/2
        l1 = list[:mid]
        l2 = list[mid:]
        l1sorted = mergesort(l1)
        l2sorted = mergesort(l2)
        sorted = merge(l1sorted, l2sorted)
        return sorted
```

If the list is empty or has length 1, there is nothing to sort. We are done.

Merge Sort in Python

```
def mergesort (list):
    if len(list) < 2:
        return list
    else:
        mid = len(list)/2
        l1 = list[:mid]
        l2 = list[mid:]
        l1sorted = mergesort(l1)
        l2sorted = mergesort(l2)
        sorted = merge(l1sorted, l2sorted)
        return sorted
```

Otherwise, split the list in two halves.

Merge Sort in Python

```
def mergesort (list):
    if len(list) < 2:
        return list
    else:
        mid = len(list)/2
        l1 = list[:mid]
        l2 = list[mid:]
        l1sorted = mergesort(l1)
        l2sorted = mergesort(l2)
        sorted = merge(l1sorted, l2sorted)
        return sorted
```

Sort each of the halves.

Merge Sort in Python

```
def mergesort (list):
    if len(list) < 2:
        return list
    else:
        mid = len(list)/2
        l1 = list[:mid]
        l2 = list[mid:]
        l1sorted = mergesort(l1)
        l2sorted = mergesort(l2)
        sorted = merge(l1sorted, l2sorted)
        return sorted
```

Merge the sorted halves back together.

This is recursion

```
def binsearch (element, list):
    ...
    answer = binsearch(element, list[:mid-1])
    ...


def mergesort (list):
    ...
    l1sorted = mergesort(l1)
    ...
```

Visualization of Sorting Algorithms

http://math.hws.edu/TMCM/java/xSortLab/

Some Sorting Algorithms

- Bubble sort
- Heap sort
- Insertion sort
- Merge sort
- Quick sort
- Selection sort
- Shell sort

Selection Sort

```
def selection_sort(list):
    for i in range(0, len(list)):
        min = i
        for j in range(i + 1, len(list)):
            if list[j] < list[min]:
                min = j
        list[i], list[min] = list[min], list[i]
```

Insertion Sort

```
def insertion_sort(list):

    for i in range(1, len(list)):
        toInsert = list[i]
        j = i

        while j > 0 and list[j - 1] > toInsert:
            list[j] = list[j - 1]
            j -= 1

        list[j] = toInsert
```