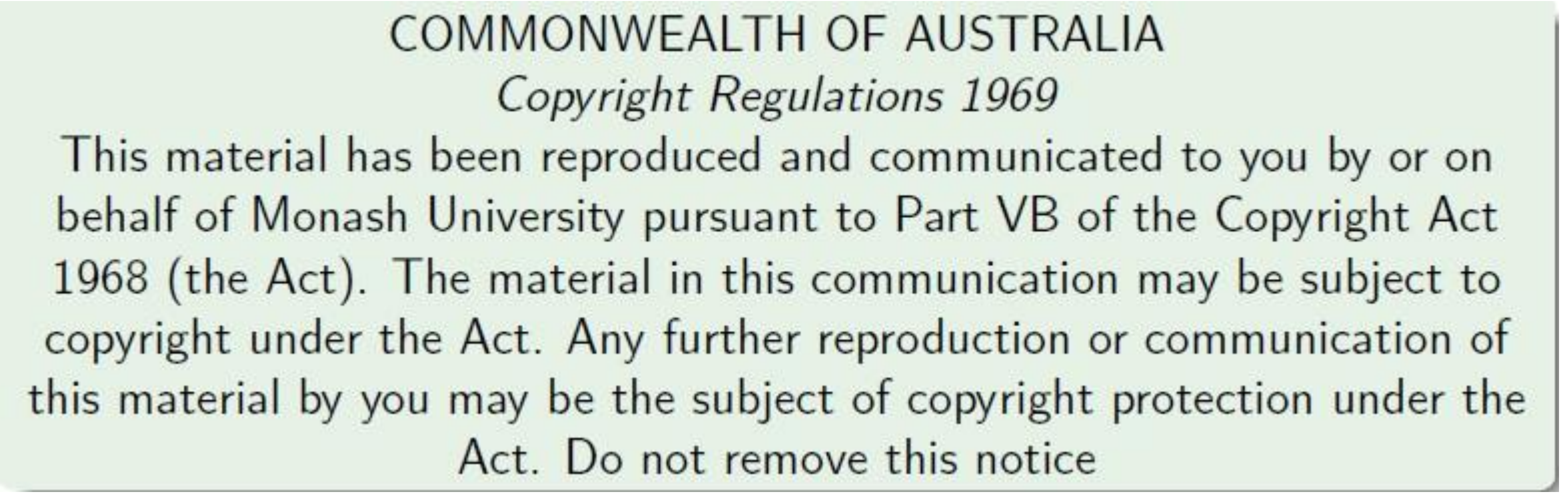
MONASH
INFORMATION
TECHNOLOGY

FIT2004
Algorithms and Data Structures

Ian Wern Han Lim
lim.wern.han@monash.edu

Referencing materials by
Nathan Companez, Aamir Cheema, Arun Konagurthu and Lloyd Allison

Faculty of Information Technology, Monash University



Ready?

Agenda

- Complexity Analysis
  - Time
  - Space
- Sorting Algorithms
  - Comparison based
    - Selection
    - Insertion
  - Non-comparison based (the IMBA ones)
    - Counting
    - Radix

MONASH University

Let us begin…

Algorithm Analysis

- Correctness
- Complexity

Algorithm Analysis

- Correctness
  - Loop invariant
  - Termination

  } Last lecture
- Complexity

Algorithm Analysis

- Correctness
  - Loop invariant
  - Termination
- Complexity
  - Time
  - Space

Algorithm Analysis

- Correctness
  - Loop invariant
  - Termination
- Complexity
  - Time
    - Best
    - Worst (big focus here)
    - Lower bound aka big Omega
    - Output sensitive
  - Space
    - Total
    - Auxiliary

Questions?

Complexity

Time

- Best
- Worst

Complexity
Time

- Best
- Worst

- You know what are they

Complexity

Time

- Best
- Worst
  - Focus!

- You know what are they

Complexity

Time

- Now let us have some recap with some functions

Complexity

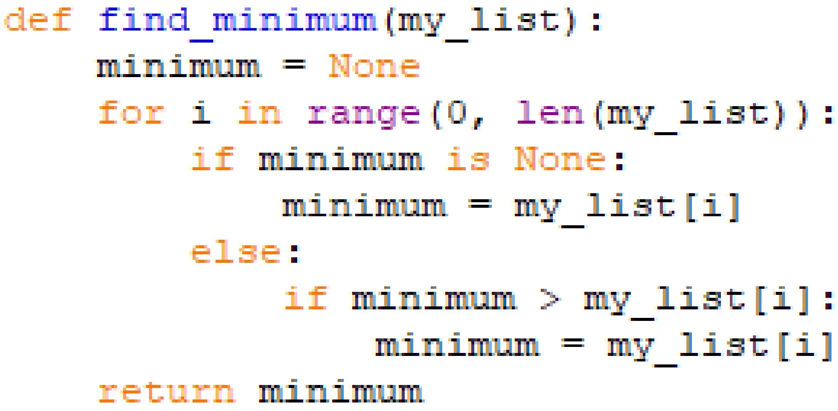
Time

- Now let us have some recap with some functions
  - Minimum
  - Binary search
  - Heap sort

Complexity

Minimum

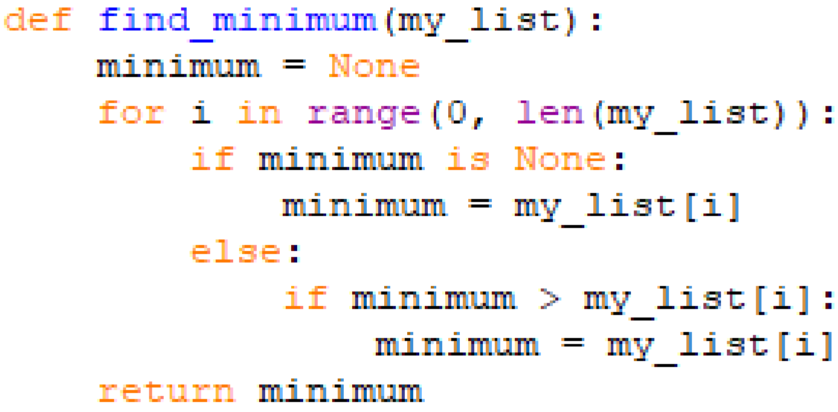
- Consider the code
- What is the time complexity?



```python
def find_minimum(my_list):
    minimum = None
    for i in range(0, len(my_list)):
        if minimum is None:
            minimum = my_list[i]
        else:
            if minimum > my_list[i]:
                minimum = my_list[i]
    return minimum
```

Complexity
Minimum

- Consider the code
- What is the time complexity?
  - Best
  - Worst

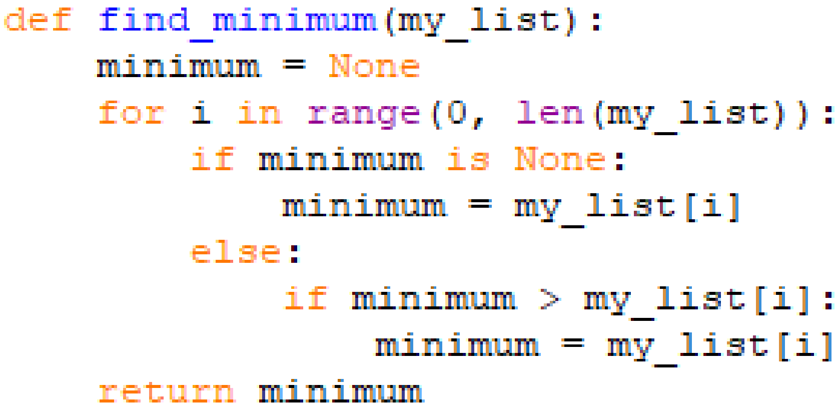


```python
def find_minimum(my_list):
    minimum = None
    for i in range(0, len(my_list)):
        if minimum is None:
            minimum = my_list[i]
        else:
            if minimum > my_list[i]:
                minimum = my_list[i]
    return minimum
```

Complexity

Minimum

- Consider the code
- What is the time complexity?
  - Best
  - Worst
  - Both are O(N) because…

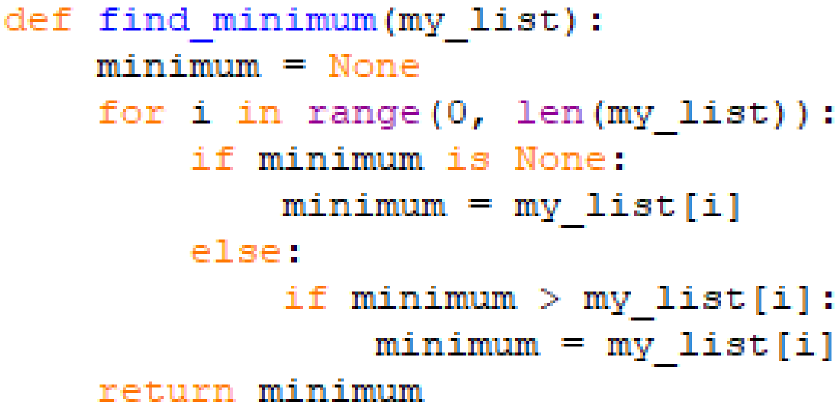


```
def find_minimum(my_list):
    minimum = None
    for i in range(0, len(my_list)):
        if minimum is None:
            minimum = my_list[i]
        else:
            if minimum > my_list[i]:
                minimum = my_list[i]
    return minimum
```

Complexity

Minimum

- Consider the code
- What is the time complexity?
  - Best
  - Worst
  - Both are O(N) because… need to go through entire list no matter what (can’t terminate earlier)

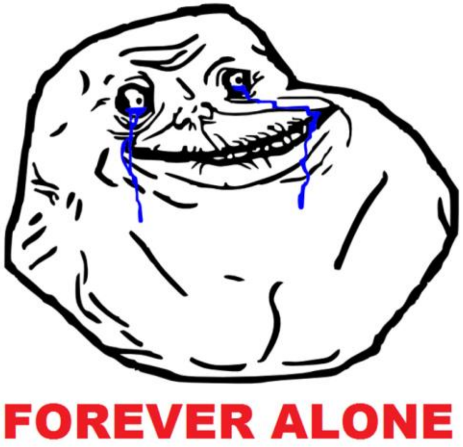


```
def find_minimum(my_list):
    minimum = None
    for i in range(0, len(my_list)):
        if minimum is None:
            minimum = my_list[i]
        else:
            if minimum > my_list[i]:
                minimum = my_list[i]
    return minimum
```

Complexity

Minimum

- Consider the code
- What is the time complexity?
  - Best
  - Worst
  - Both are O(N) because… need to go through entire list no matter what (can't terminate earlier)

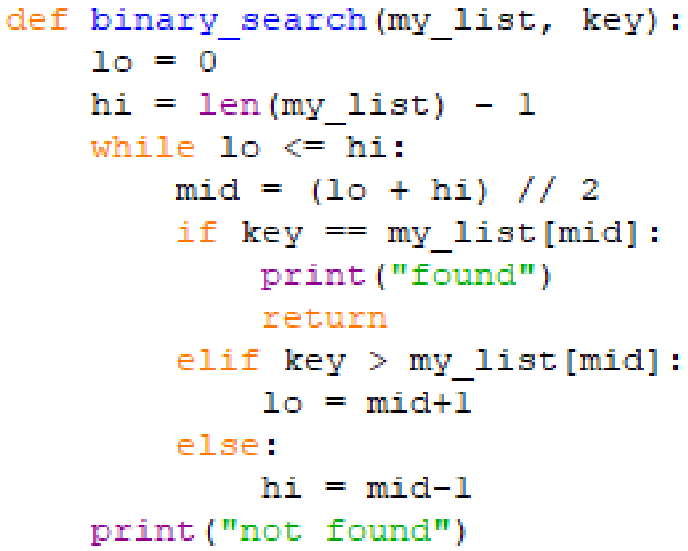


- Remember we can't say best O(1) when list have 1 item
  - Need to be for a list of size N

Complexity

Binary search

- Consider the code
- What is the time complexity?

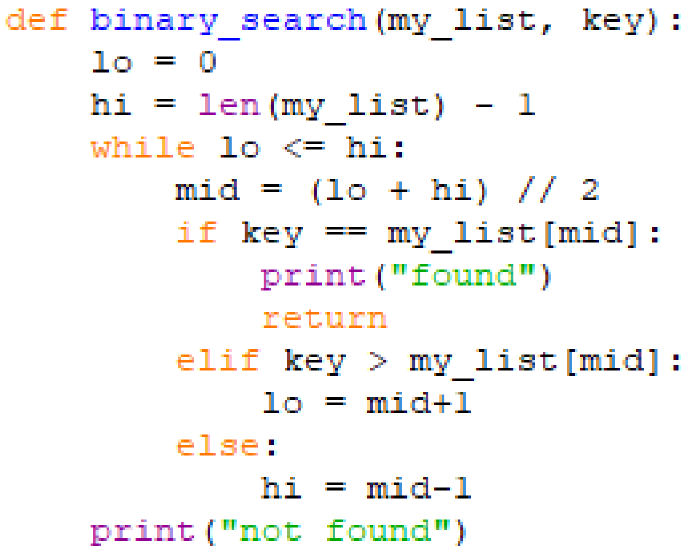


```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

Complexity

Binary search

- Consider the code
- What is the time complexity?
  - Best
  - Worst

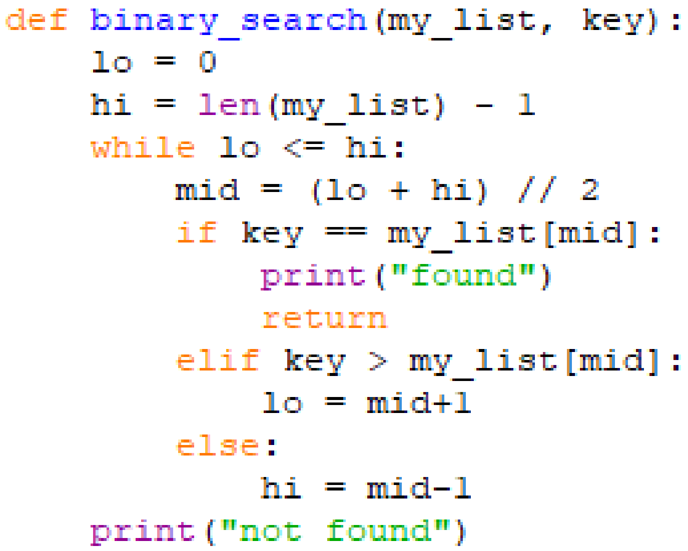


```python
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

Complexity

Binary search

- Consider the code
- What is the time complexity?
  - Best O(1)
  - Worst O(log N)

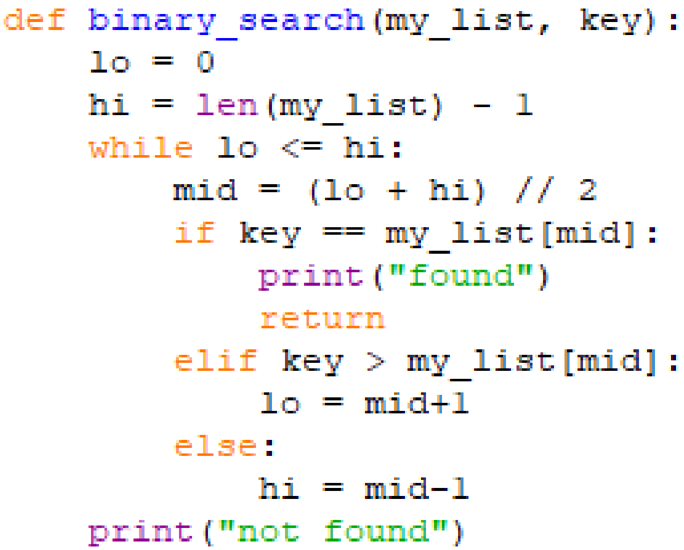


```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

Complexity

Binary search

- Consider the code
- What is the time complexity?
  - Best O(1)
  - Worst O(log N)
  - How can we show worst is O(log N)?



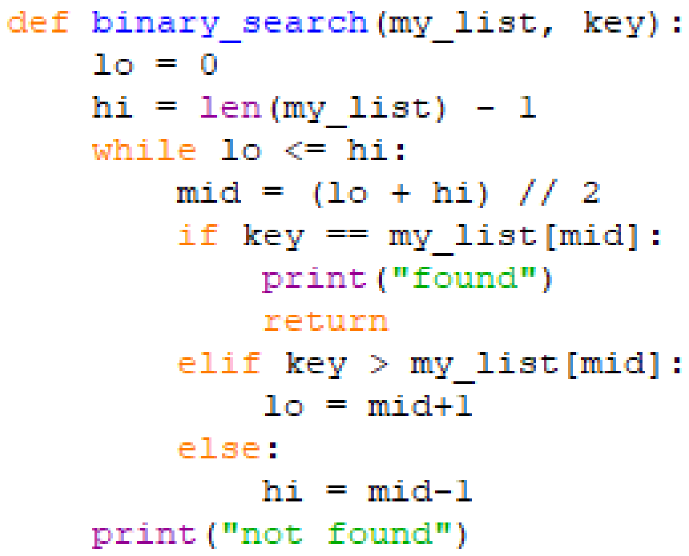
```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

Complexity

Binary search

- How can we show worst is O(log N)?

- Search space



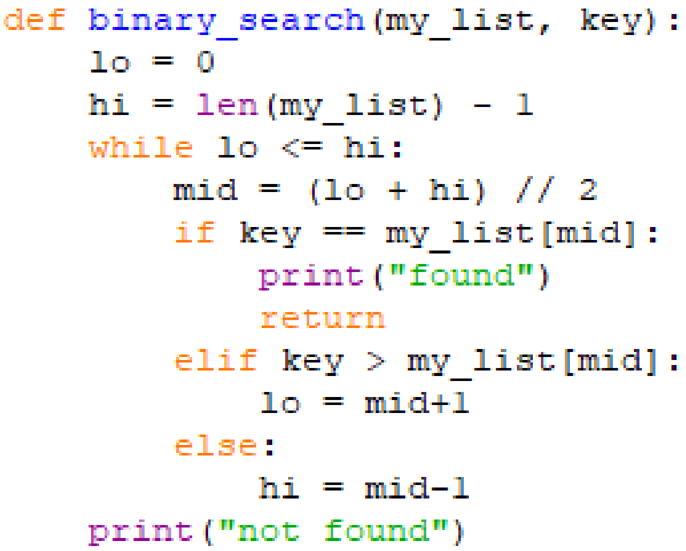
```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

Complexity

Binary search

- How can we show worst is O(log N)?

- Search space
  - Initially N

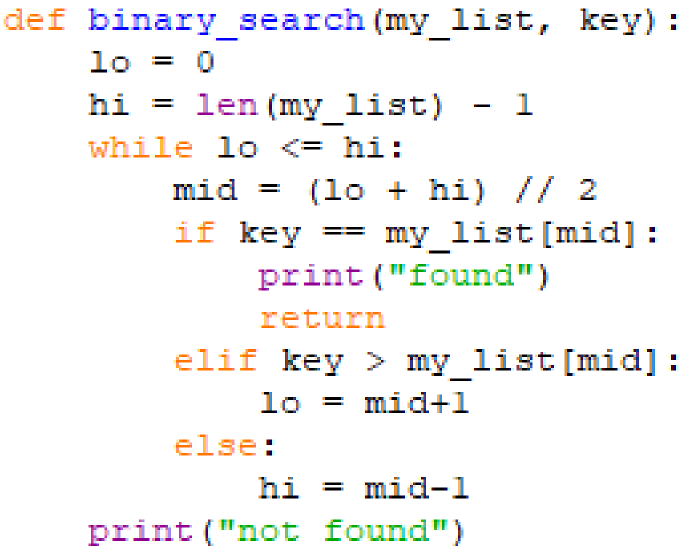


```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

Complexity

Binary search

- How can we show worst is O(log N)?
- Search space
  - Initially = N
  - 1st iteration = N/2
  - 2nd iteration = N/4



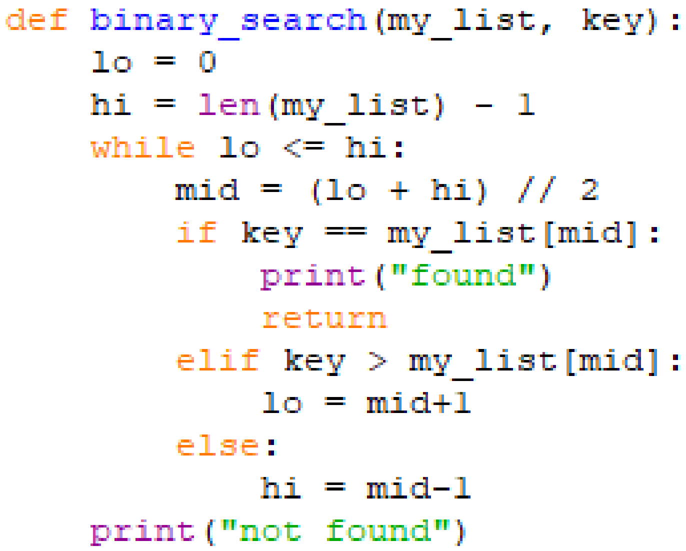
```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

Complexity

Binary search

- How can we show worst is O(log N)?

- Search space
  - Initially = N
  - 1st iteration = N/2
  - 2nd iteration = N/4
  - …
  - Last iteration = 1



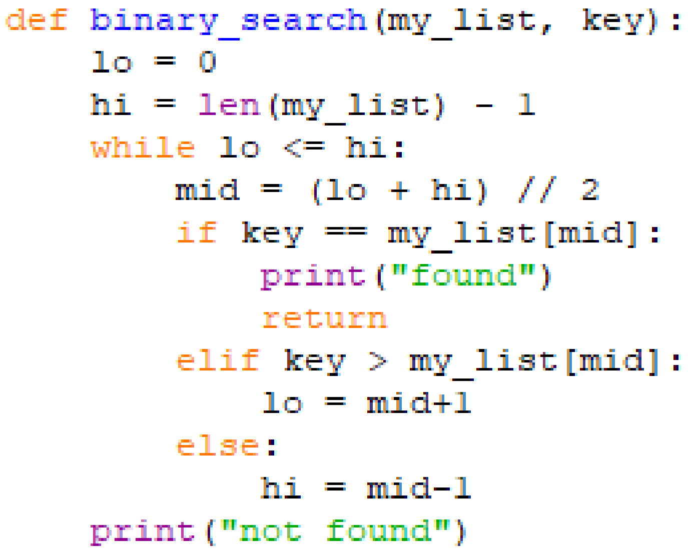
```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

Complexity

Binary search

- How can we show worst is O(log N)?

- Search space
  - Initially = N/2^0
  - 1st iteration = N/2^1
  - 2nd iteration = N/2^2
  - …
  - Last iteration = N/2^k = 1



```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

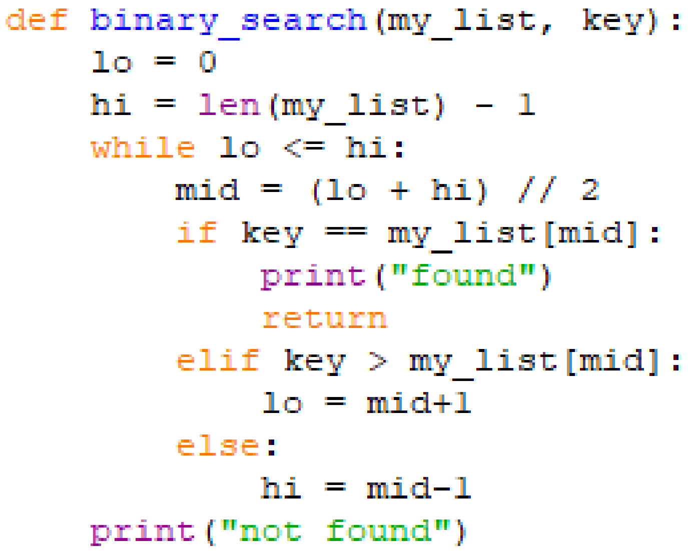
Complexity

Binary search

- How can we show worst is O(log N)?

- Search space
  - Initially = N/2^0
  - 1st iteration = N/2^1
  - 2nd iteration = N/2^2
  - …
  - Last iteration = N/2^k = 1

  - Thus N = 2^k
    - Which give us k = log N
    - Worst case is when we reach height k, which is log N



```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

Complexity

Time

- So we know time complexity pretty well now
  - Best case
  - Worst case

Complexity

Time

- So we know time complexity pretty well now
  - Best case
  - Worst case
- But we have more!

Complexity
Time

- So we know time complexity pretty well now
  - Best case
  - Worst case
- But we have more!
  - Lower bound (big omega)
  - Output-sensitive

Questions?

Complexity

Time – Lower Bound

- We know for a given problem, there can be a lot of solutions or algorithms….

Complexity

Time – Lower Bound

- We know for a given problem, there can be a lot of solutions or algorithms….

- The lower bound (aka big omega) is the best complexity we can achieve for a given problem irregardless of the solution or algorithm…

Complexity

Time – Lower Bound

- We know for a given problem, there can be a lot of solutions or algorithms….

- The lower bound (aka big omega) is the best complexity we can achieve for a given problem irregardless of the solution or algorithm…

- If we are to print items in a list, we don’t have a choice but to print through every item in the list. Thus $\Omega(N)$ for list printing.

Complexity

Time – Lower Bound

- We know for a given problem, there can be a lot of solutions or algorithms….
  - Known or unknown

- The lower bound (aka big-omega) is the best complexity we can achieve for a given problem irregardless of the solution or algorithm…
  - Opposite of big-O

- If we are to print items in a list, we don't have a choice but to print through every item in the list. Thus $\Omega(N)$ for list printing.

Complexity

Time – Lower Bound

- So… what is the lower bound for the sorting algorithms that we have learnt?
  - Bubble
  - Insertion
  - Selection
  - Quick
  - Merge

Complexity

Time – Lower Bound

- So… what is the lower bound for the sorting algorithms that we have learnt?
  - Bubble
  - Insertion
  - Selection
  - Quick
  - Merge

- These are all comparison based
- Ω(N log N)

Complexity

Time – Lower Bound

- So… what is the lower bound for the sorting algorithms that we have learnt?
  - Bubble
  - Insertion
  - Selection
  - Quick
  - Merge

- These are all comparison based
- Ω(N log N)
- We will see more of this later

Questions?

Complexity

Time – Output Sensitive

- What is it?

Complexity

Time – Output Sensitive

- What is it?
- The complexity depends on the output instead of the input!

Complexity

Time – Output Sensitive

- What is it?
- The complexity depends on the output instead of the input!

- Given a sorted array of unique numbers
- Given two values x and y
- Find all numbers greater than x but smaller than y

Complexity

Time – Output Sensitive

- What is it?
- The complexity depends on the output instead of the input!

- Given a sorted array of unique numbers
- Given two values x and y
- Find all numbers greater than x but smaller than y

- What is our complexity here?

Complexity

Time – Output Sensitive

- Given a sorted array of unique numbers
- Given two values x and y
- Find all numbers greater than x but smaller than y

- Approach 01
  - Loop through the entire list
  - If item > x and item < y, print item

Complexity

Time – Output Sensitive

- Given a sorted array of unique numbers
- Given two values x and y
- Find all numbers greater than x but smaller than y

- Approach 01
  - Loop through the entire list
  - If item > x and item < y, print item
  - This gives O(N) complexity
    - Looping through the list

Complexity

Time – Output Sensitive

- Given a sorted array of unique numbers
- Given two values x and y
- Find all numbers greater than x but smaller than y

- Approach 01
  - Loop through the entire list
  - If item > x and item < y, print item
  - This gives O(N) complexity
    - Looping through the list

  - This isn’t output sensitive, x and y value doesn’t matter

Complexity

Time – Output Sensitive

- Given a sorted array of unique numbers
- Given two values x and y
- Find all numbers greater than x but smaller than y

- Approach 02
  - Binary search to find smallest number greater than x
  - Linear search from x till reach a greater number or equal than y

Complexity

Time – Output Sensitive

- Given a sorted array of unique numbers
- Given two values x and y
- Find all numbers greater than x but smaller than y

- Approach 02
  - Binary search to find smallest number greater than x
  - Linear search from x till reach a greater number or equal than y
  - Complexity?

Complexity

Time – Output Sensitive

- Given a sorted array of unique numbers
- Given two values x and y
- Find all numbers greater than x but smaller than y

- Approach 02
  - Binary search to find smallest number greater than x
  - Linear search from x till reach a greater number or equal than y
  - Complexity?
    - O(log N) for binary search

Complexity

Time – Output Sensitive

- Given a sorted array of unique numbers
- Given two values x and y
- Find all numbers greater than x but smaller than y

- Approach 02
  - Binary search to find smallest number greater than x
  - Linear search from x till reach a greater number or equal than y
  - Complexity?
    - O(log N) for binary search
    - O(W) for printing the values where O(W) is O(y-x)

Complexity

Time – Output Sensitive

- Given a sorted array of unique numbers
- Given two values x and y
- Find all numbers greater than x but smaller than y

- Approach 02
  - Binary search to find smallest number greater than x
  - Linear search from x till reach a greater number or equal than y
  - Complexity? O(W + log N)
    - O(log N) for binary search
    - O(W) for printing the values where O(W) is O(y-x)

Complexity

Time – Output Sensitive

- Given a sorted array of unique numbers
- Given two values x and y
- Find all numbers greater than x but smaller than y

- Approach 02
  - Binary search to find smallest number greater than x
  - Linear search from x till reach a greater number or equal than y
  - Complexity? O(W + log N)
    - O(log N) for binary search
    - O(W) for printing the values where O(W) is O(y-x)
    - Why?

Complexity

Time – Output Sensitive

- Given a sorted array of unique numbers
- Given two values x and y
- Find all numbers greater than x but smaller than y

- Approach 02
  - Binary search to find smallest number greater than x
  - Linear search from x till reach a greater number or equal than y
  - Complexity? O(W + log N)
    - O(log N) for binary search
    - O(W) for printing the values where O(W) is O(y-x)
    - Why? W can be as big as N!

Complexity

Time – Output Sensitive

- Output-sensitive complexity is only relevant when the output-size may vary
  - Not sorting
  - Not finding minimum

Complexity

Time – Output Sensitive

- Output-sensitive complexity is only relevant when the output-size may vary
  - Not sorting
  - Not finding minimum

- If you look at your assignment, certain question have additional complexity – that is dependent on the output!

MONASH
University

Questions?

Complexity

Space

- What is it?

Complexity

Space

- How much space is used

Complexity

Space

- How much space is used

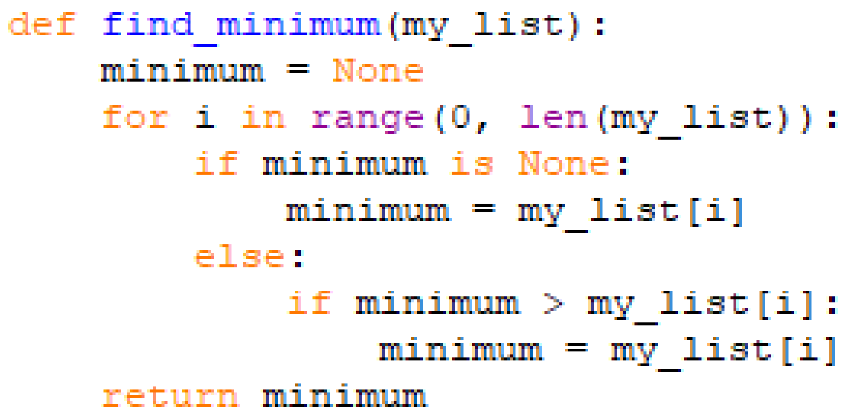
- Consider our functions earlier…

Complexity

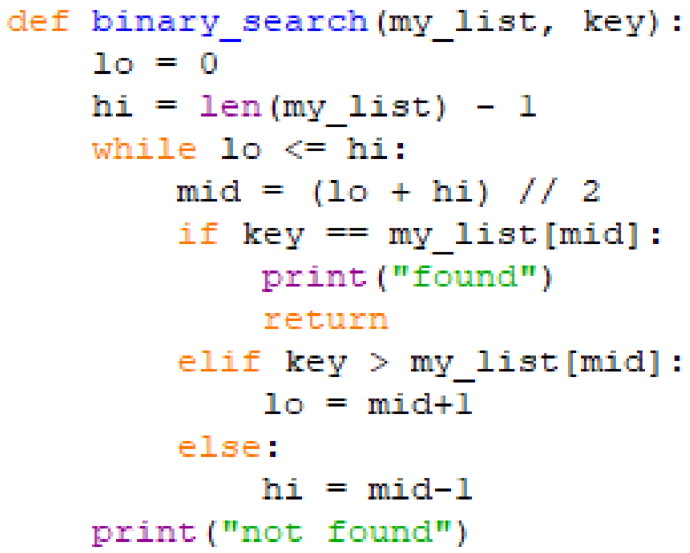
Space -- minimum

- How much space is used

- Consider our functions earlier…



```
def find_minimum(my_list):
    minimum = None
    for i in range(0, len(my_list)):
        if minimum is None:
            minimum = my_list[i]
        else:
            if minimum > my_list[i]:
                minimum = my_list[i]
    return minimum
```



```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

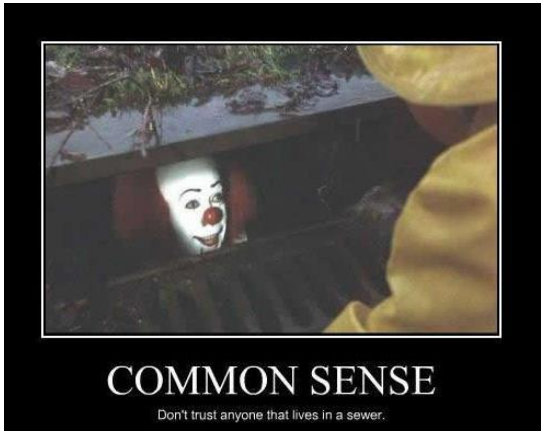
Complexity

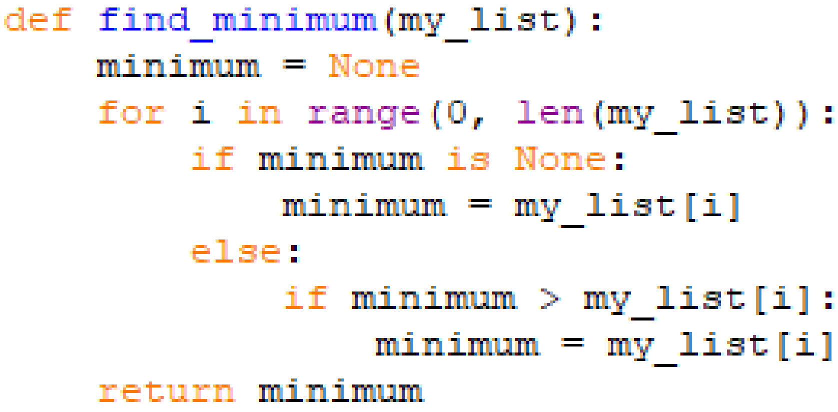
Space

- How much space is used

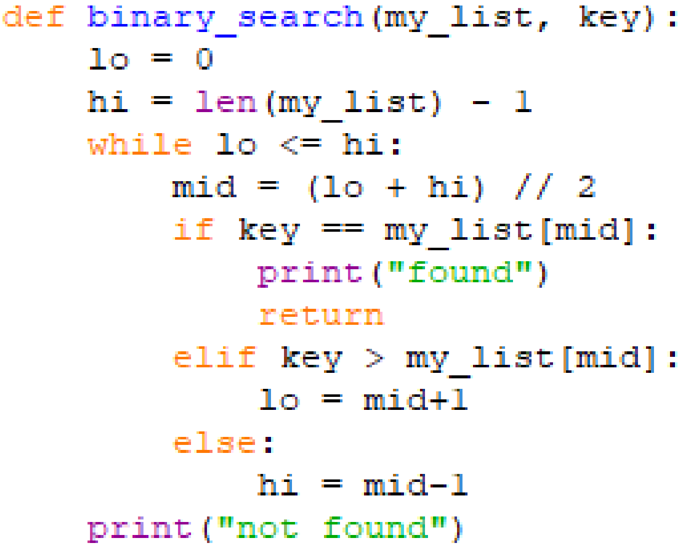
- Consider our functions earlier…

- We need O(N) space to for the input list





```python
def find_minimum(my_list):
    minimum = None
    for i in range(0, len(my_list)):
        if minimum is None:
            minimum = my_list[i]
        else:
            if minimum > my_list[i]:
                minimum = my_list[i]
    return minimum
```



```python
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

Questions?

Complexity

Auxiliary Space

- What is this now then?

Complexity

Auxiliary Space

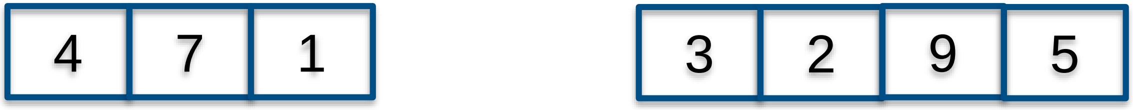
- What is this now then?
- Additional space required in addition to the input

Complexity

Auxiliary Space

- What is this now then?
- Additional space required in addition to the input

- Remember the merge sort’s merge operation?



4 7 1

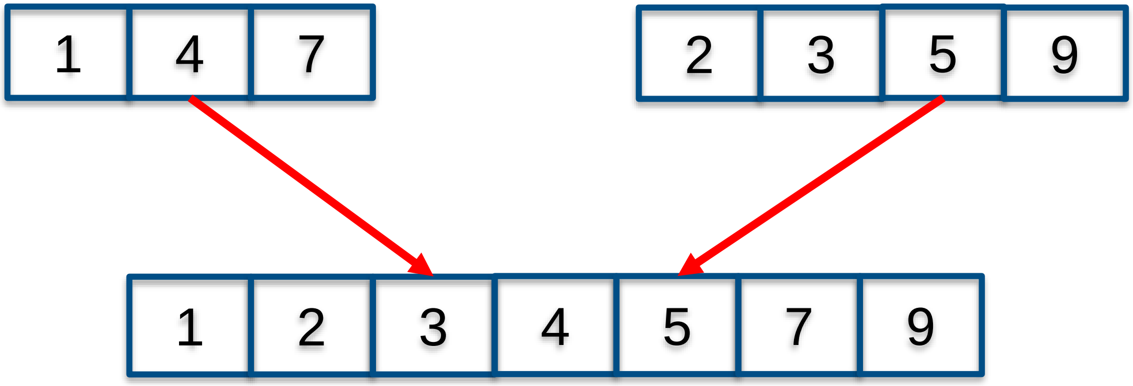
3 2 9 5

Complexity

Auxiliary Space

- What is this now then?
- Additional space required in addition to the input

- Remember the merge sort's merge operation?



1 4 7

2 3 5 9

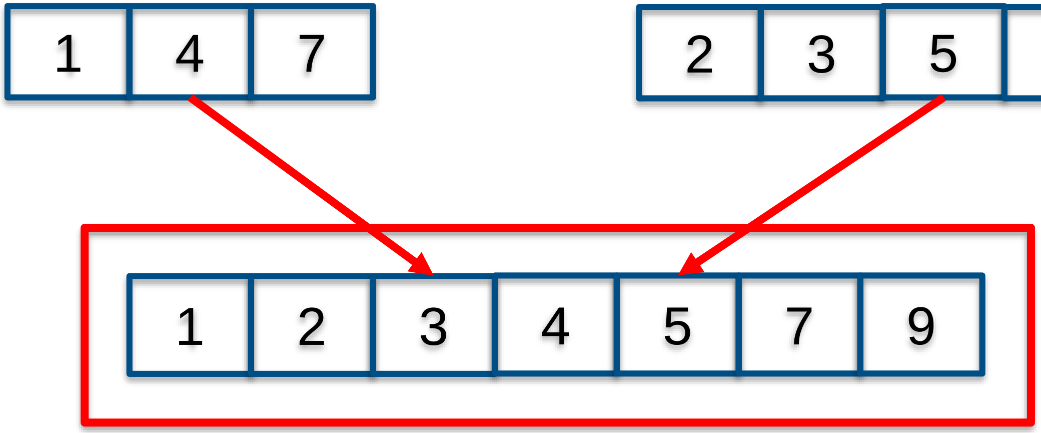
1 2 3 4 5 7 9

Complexity

Auxiliary Space

- What is this now then?
- Additional space required in addition to the input

- Remember the merge sort's merge operation?



Additional space

Complexity

Auxiliary Space

- What is this now then?
- Additional space required in addition to the input

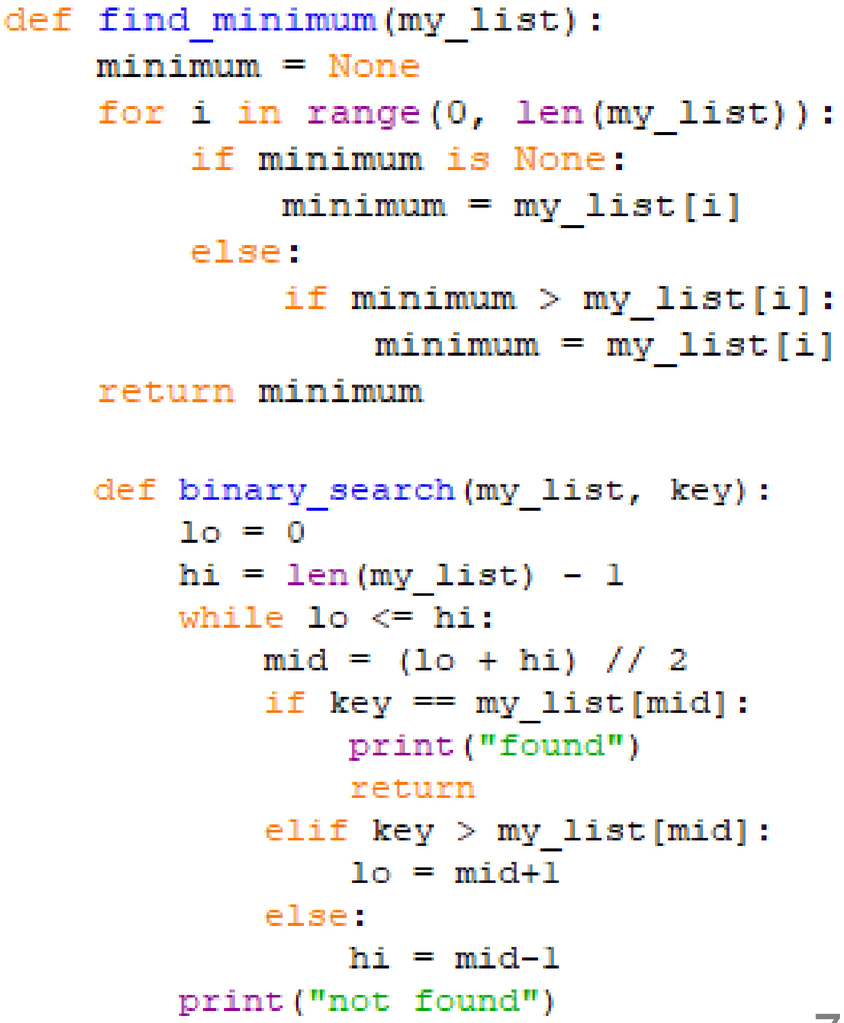
- Remember the merge sort’s merge operation?
  - Space complexity = 2N = O(N)
  - Auxiliary space = 2N – N = O(N)

Complexity

Auxiliary Space

MONASH University

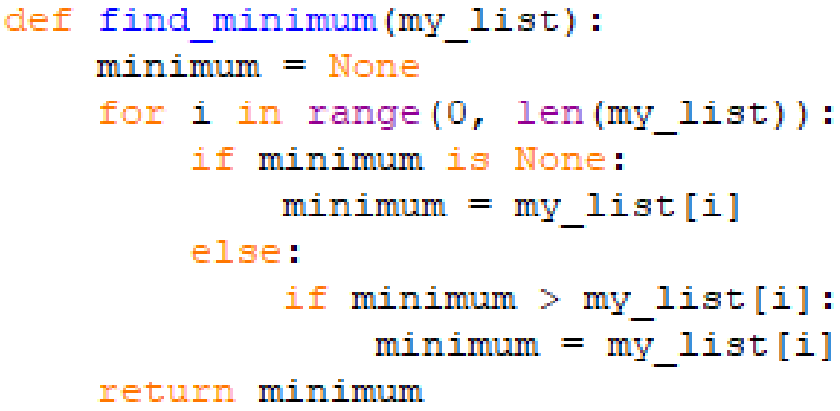
- So what is the auxiliary space complexity for these then?



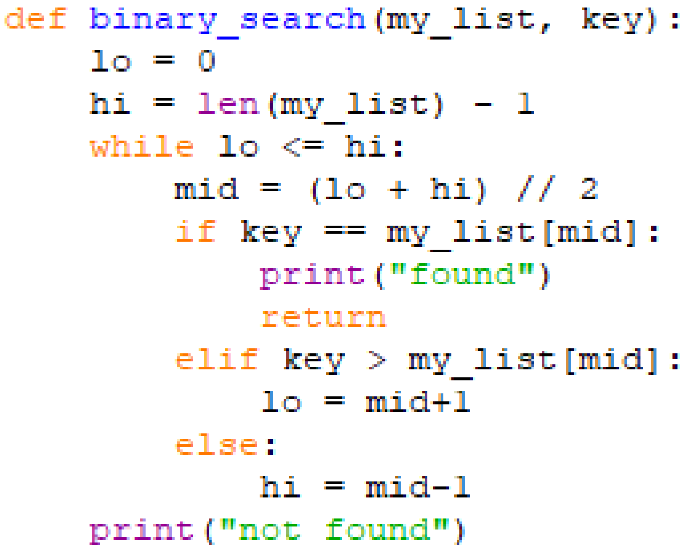
```
def find_minimum(my_list):
    minimum = None
    for i in range(0, len(my_list)):
        if minimum is None:
            minimum = my_list[i]
        else:
            if minimum > my_list[i]:
                minimum = my_list[i]
    return minimum

    def binary_search(my_list, key):
        lo = 0
        hi = len(my_list) - 1
        while lo <= hi:
            mid = (lo + hi) // 2
            if key == my_list[mid]:
                print("found")
                return
            elif key > my_list[mid]:
                lo = mid+1
            else:
                hi = mid-1
        print("not found")
```

Complexity
Auxiliary Space

- So what is the auxiliary space complexity for these then?
  - Both are O(1)
  - Do not require additional space



```
def find_minimum(my_list):
    minimum = None
    for i in range(0, len(my_list)):
        if minimum is None:
            minimum = my_list[i]
        else:
            if minimum > my_list[i]:
                minimum = my_list[i]
    return minimum
```

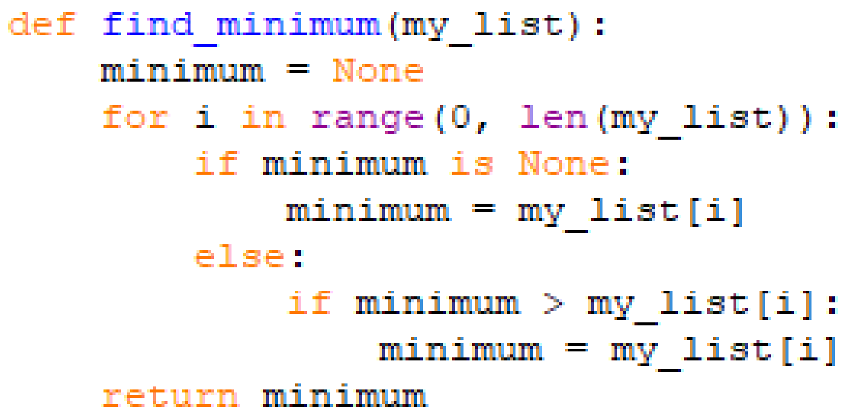


```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

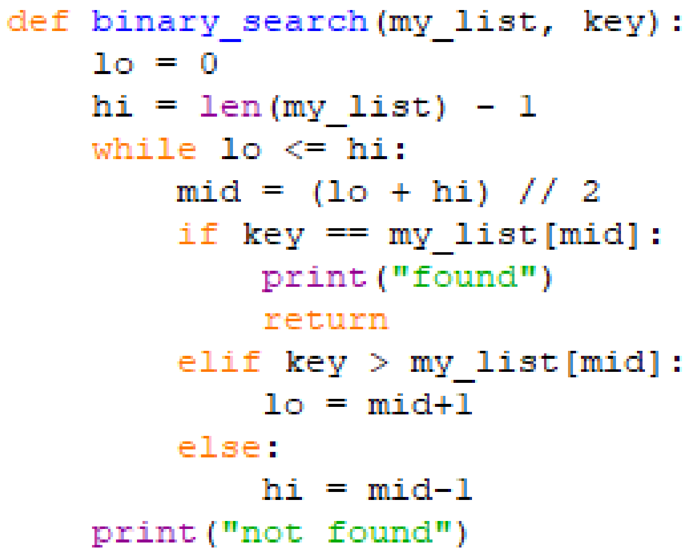
Complexity

Auxiliary Space

- So what is the auxiliary space complexity for these then?
  - Both are O(1)
  - Do not require additional space
- Known as in-place
  - Can process in the input itself!



```
def find_minimum(my_list):
    minimum = None
    for i in range(0, len(my_list)):
        if minimum is None:
            minimum = my_list[i]
        else:
            if minimum > my_list[i]:
                minimum = my_list[i]
    return minimum
```



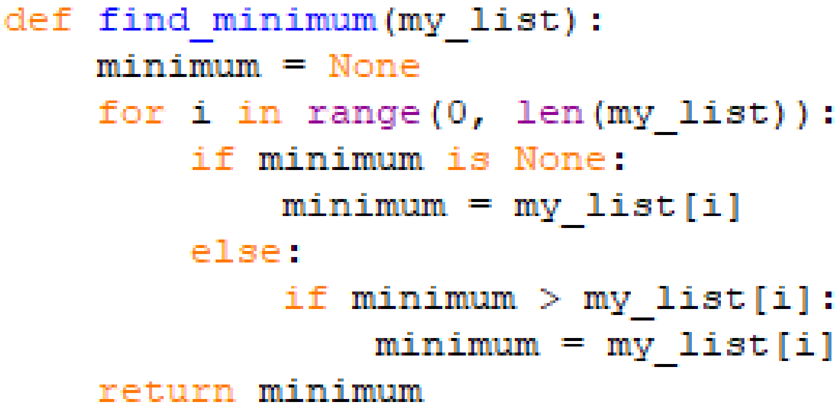
```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

Complexity

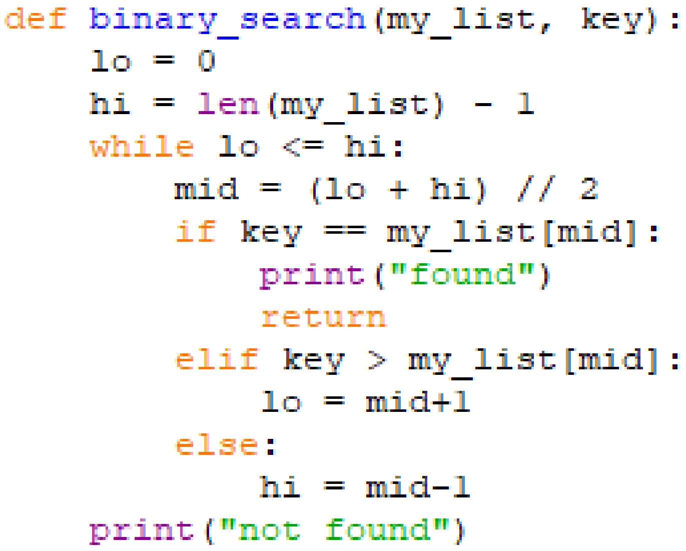
Auxiliary Space

- So what is the auxiliary space complexity for these then?
  - Both are O(1)
  - Do not require additional space

- Known as in-place
  - Can process in the input itself!
  - Auxiliary space of O(1)



```
def find_minimum(my_list):
    minimum = None
    for i in range(0, len(my_list)):
        if minimum is None:
            minimum = my_list[i]
        else:
            if minimum > my_list[i]:
                minimum = my_list[i]
    return minimum
```



```
def binary_search(my_list, key):
    lo = 0
    hi = len(my_list) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key == my_list[mid]:
            print("found")
            return
        elif key > my_list[mid]:
            lo = mid+1
        else:
            hi = mid-1
    print("not found")
```

MONASH University

Questions?

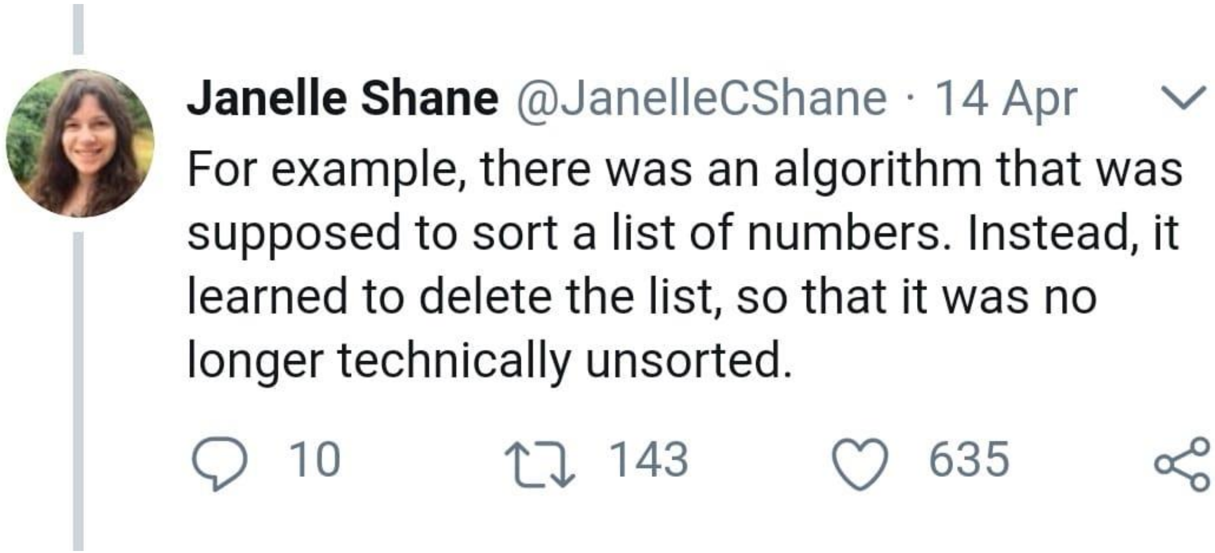
Thank You

Sorting

- We are back to sorting!
  - Bubble
  - Insertion
  - Selection
  - Merge
  - Quick

Sorting

- We are back to sorting!
  - Bubble
  - Insertion
  - Selection
  - Merge
  - Quick



Janelle Shane @JanelleCShane · 14 Apr

For example, there was an algorithm that was supposed to sort a list of numbers. Instead, it learned to delete the list, so that it was no longer technically unsorted.

Sorting

- We are back to sorting!
  - Bubble
  - Insertion
  - Selection
  - Merge
  - Quick

- All of these are known as comparison based sorting. Why?

Sorting

- We are back to sorting!
  - Bubble
  - Insertion
  - Selection
  - Merge
  - Quick

- All of these are known as comparison based sorting. Why? Because we compare between items to know if a < b or b > a

Sorting

- We are back to sorting!
  - Bubble
  - Insertion
  - Selection
  - Merge
  - Quick

- All of these are known as comparison based sorting. Why? Because we compare between items to know if
  a < b or b > a
- Now let us analyze them based on what we have learnt!

Questions?

Sorting

Selection Sort

- Correctness
- Complexity

Sorting

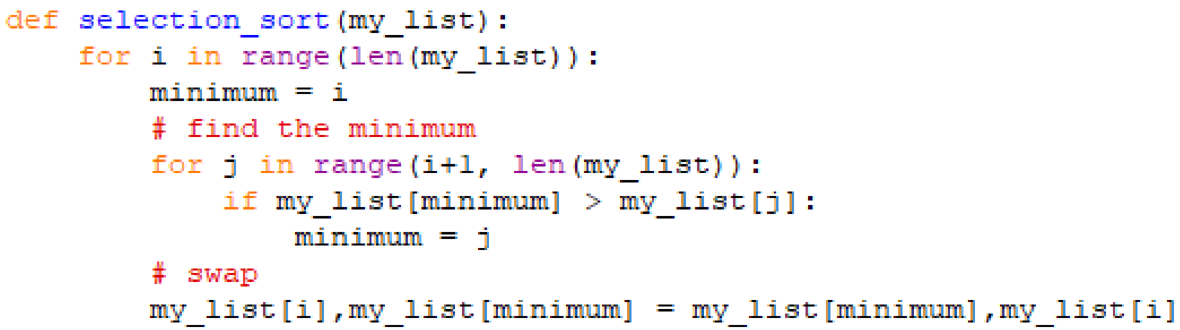
Selection Sort

- Correctness
  - Loop invariant
  - Termination
- Complexity
  - Time
  - Space

Sorting

Selection Sort

- Correctness
  - Loop invariant
  - Termination
- Complexity
  - Time
  - Space

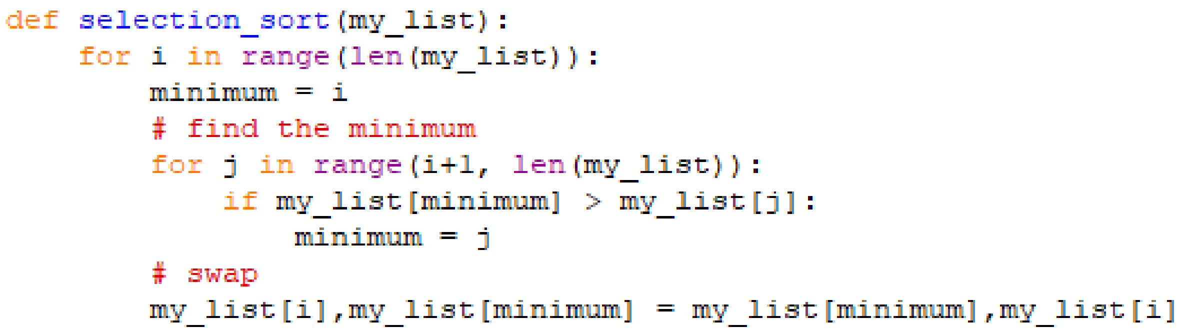


```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

- Correctness
  - Loop invariant
  - Termination

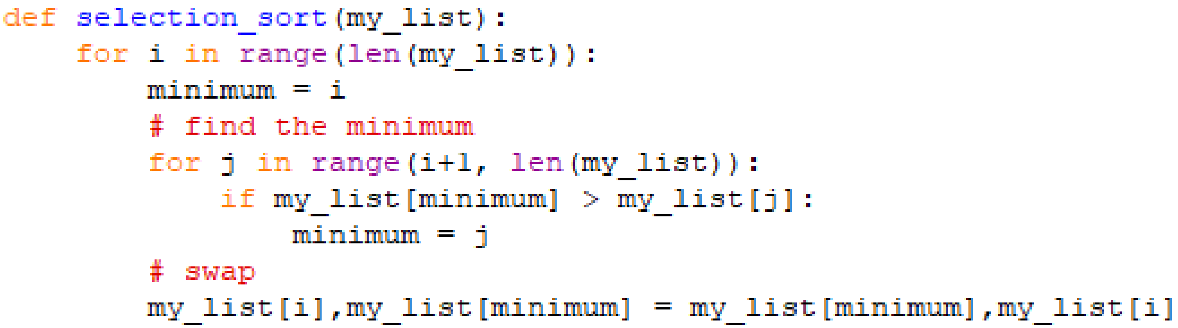


```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

- Correctness
  - Loop invariant
  - Termination

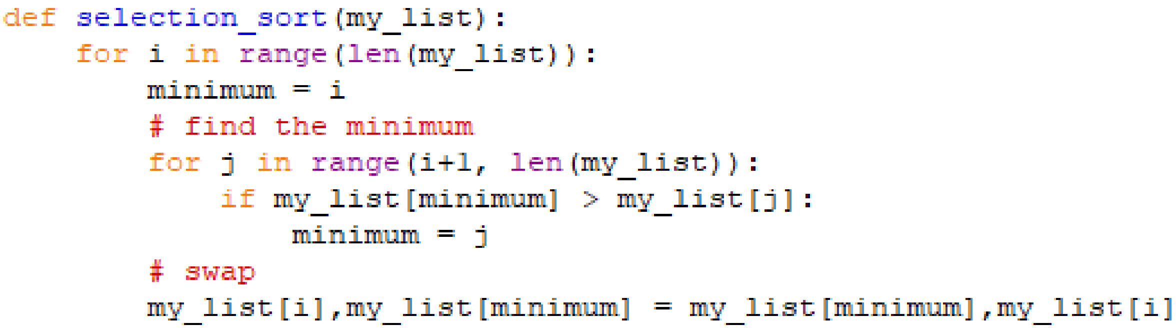


```python
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

- Correctness
  - Loop invariant
    - my_list[0…i-1] is sorted
    - my_list[0…i-1] <= my_list[i…N]
  - Termination

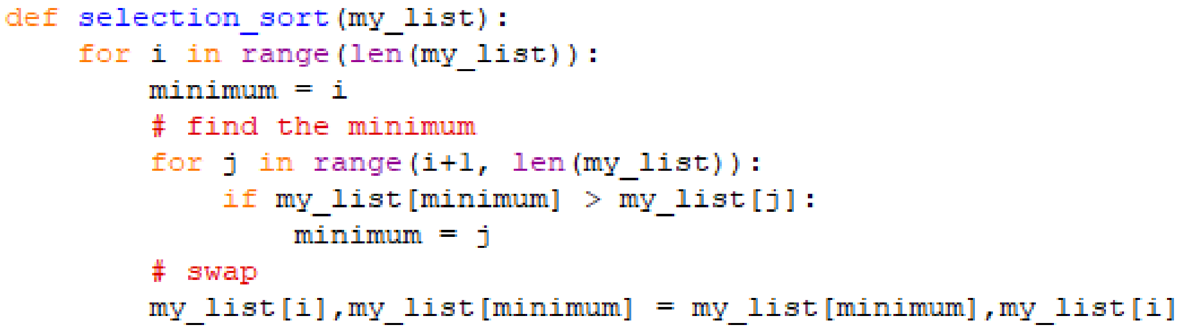


```python
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

- Correctness
  - Loop invariant
    - my_list[0…i-1] is sorted
    - my_list[0…i-1] <= my_list[i…N]
  - Termination

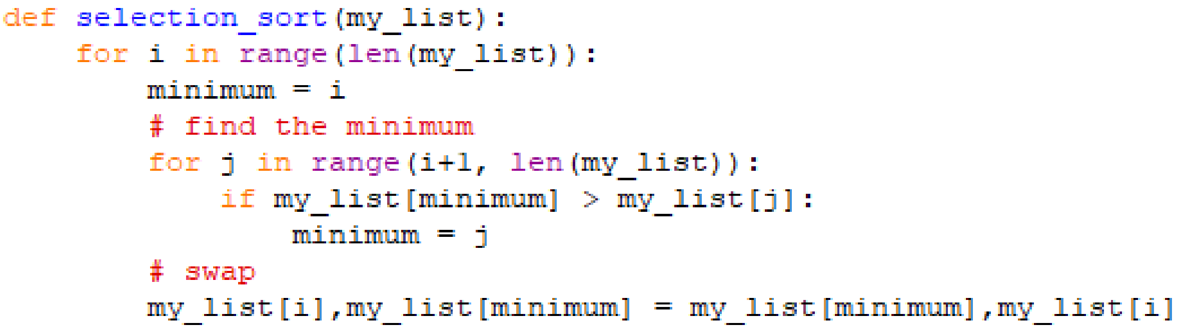


```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

- Correctness
  - Loop invariant
    - my_list[0…i-1] is sorted
    - my_list[0…i-1] <= my_list[i…N]
  - Termination
    - i and j always increment and both reach the end of the list

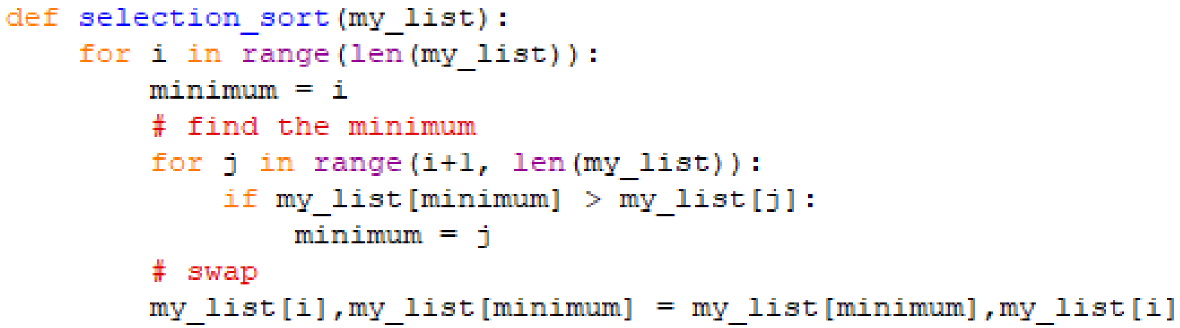


```python
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

- Correctness
  - Loop invariant
    - my_list[0…i-1] is sorted
    - my_list[0…i-1] <= my_list[i…N]
  - Termination
    - i and j always increment and both reach the end of the list
  - So why is it working then?

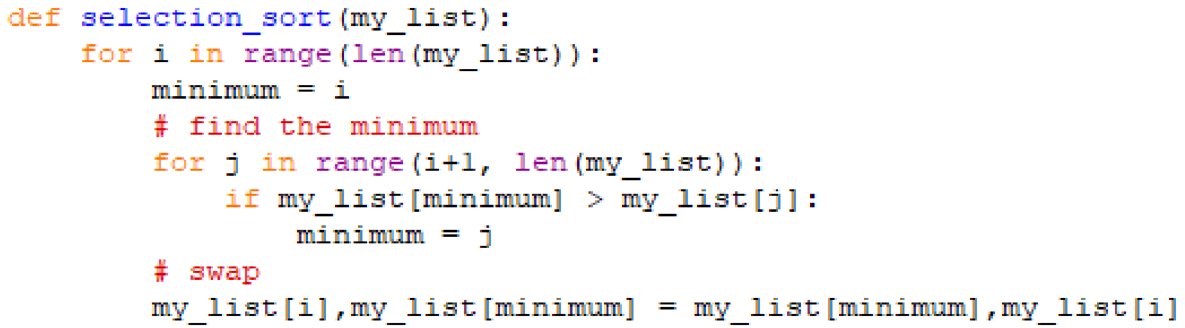


```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

- Correctness
  - Loop invariant
    - my_list[0…i-1] is sorted
    - my_list[0…i-1] <= my_list[i…N]
  - Termination
    - i and j always increment and both reach the end of the list
  - So why is it working then?
    - i keep increment till n and we know from invariant 0…i-1 is sorted, thus we will sort the entire list!

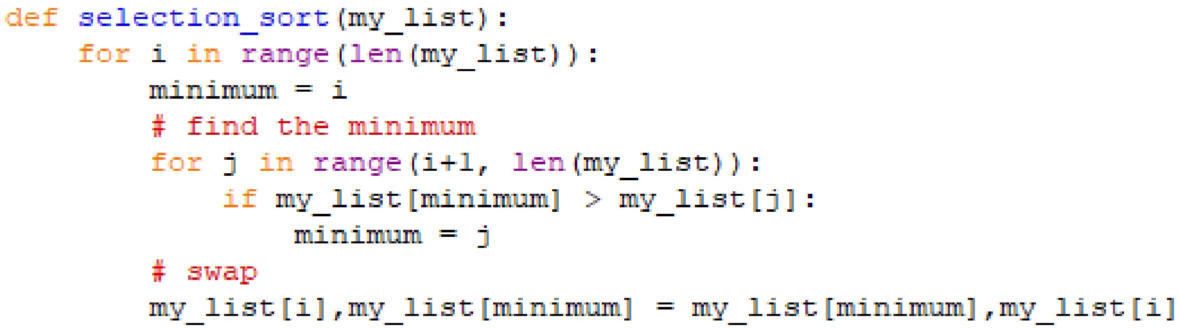


```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

- Correctness
- Complexity
  - Time
  - Space

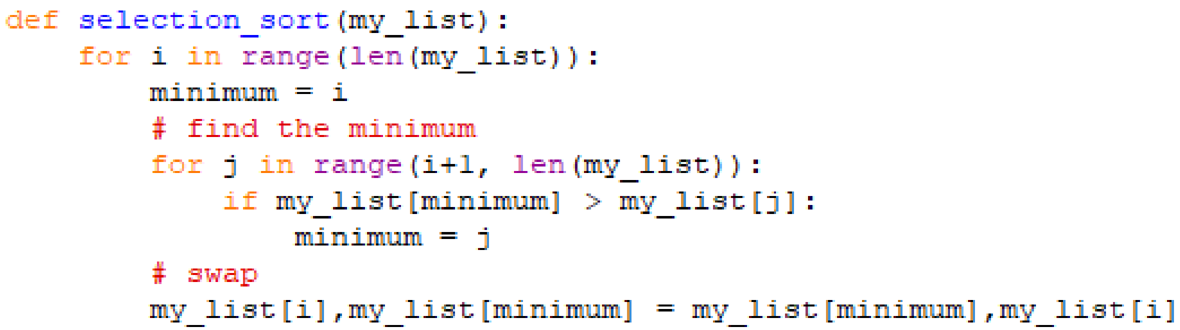


```python
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

- Correctness
- Complexity
  - Time
  - Space

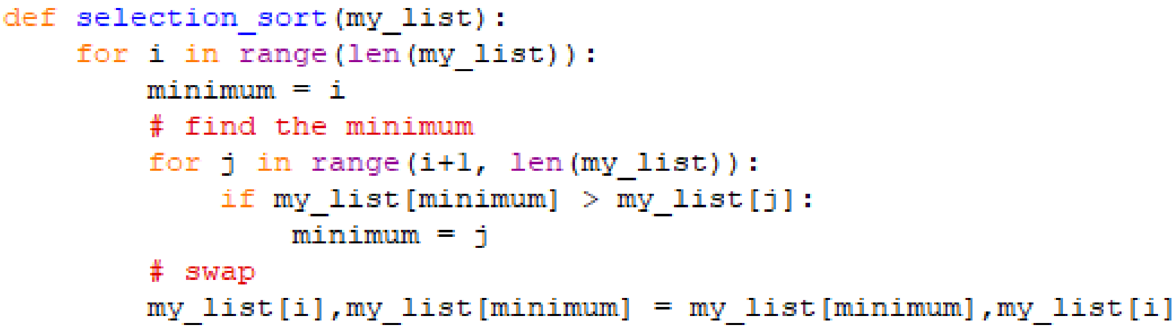


```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

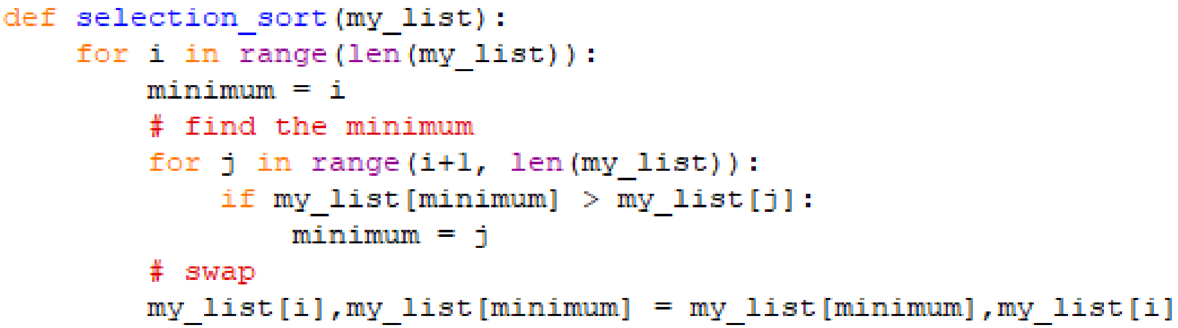
- Correctness
- Complexity
  - Time
    - Best = O(N^2)
    - Worst = O(N^2)
  - Space



```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting
Selection Sort

- Correctness
- Complexity
  - Time
    - Best = O(N^2) because no matter what we have to find the minimum and cant terminate earlier!
    - Worst = O(N^2)
  - Space

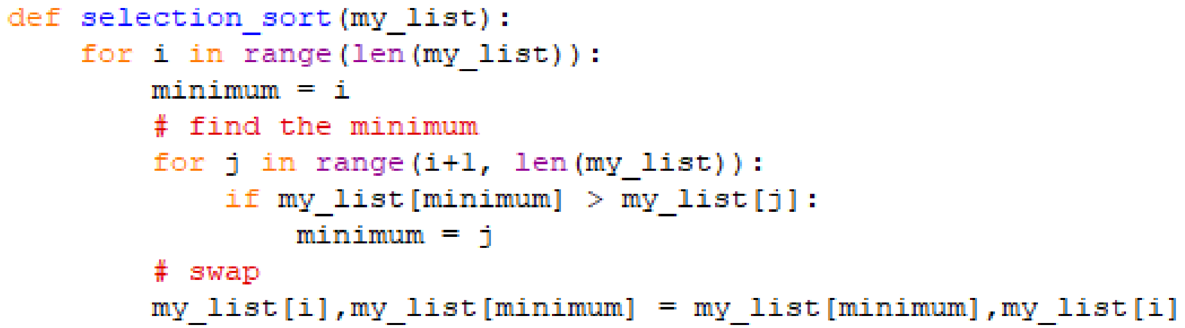


```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

- Correctness
- Complexity
  - Time
    - Best = O(N^2) because no matter what we have to find the minimum and cant terminate earlier!
    - Worst = O(N^2)
  - Space

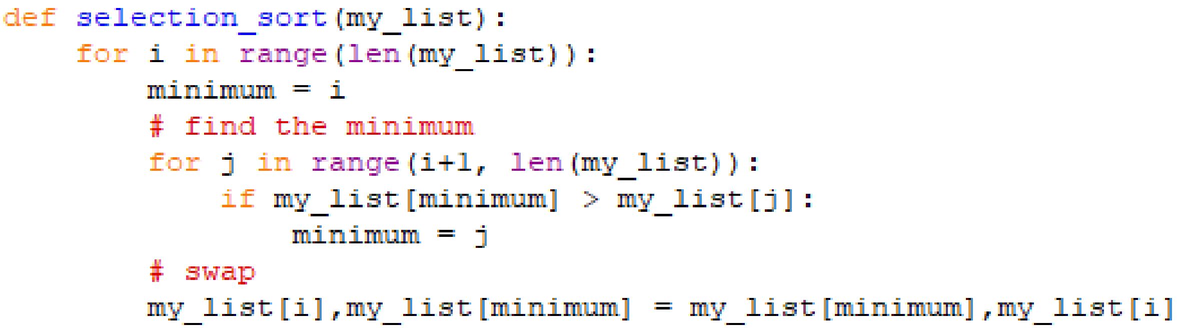


```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

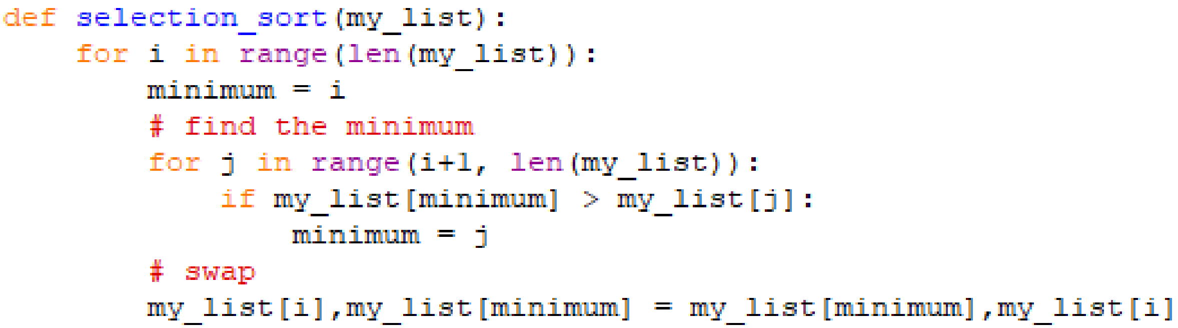
- Correctness
- Complexity
  - Time
    - Best = O(N^2) because no matter what we have to find the minimum and cant terminate earlier!
    - Worst = O(N^2)
  - Space
    - O(N) for the input list
    - Auxiliary?



```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting
Selection Sort

- Correctness
- Complexity
  - Time
    - Best = O(N^2) because no matter what we have to find the minimum and cant terminate earlier!
    - Worst = O(N^2)
  - Space
    - O(N) for the input list
    - Auxiliary? O(1)

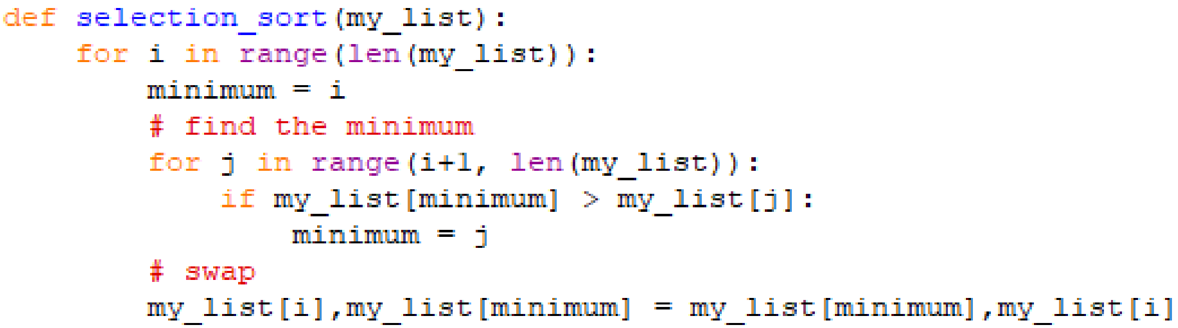


```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

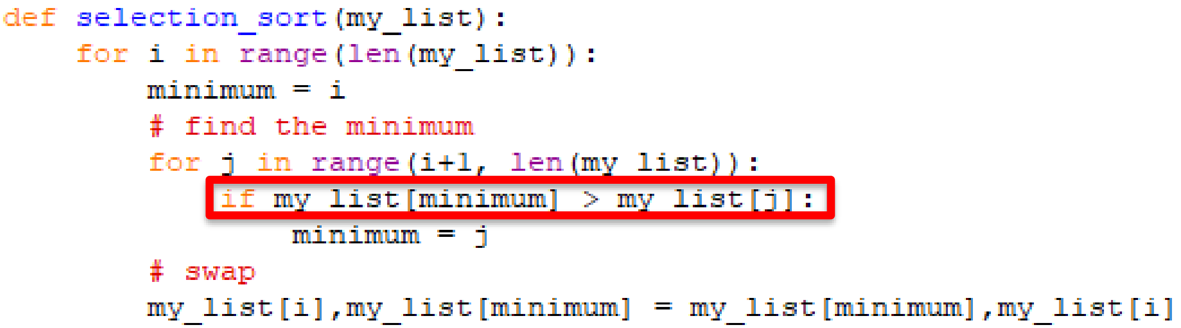
- Correctness
- Complexity
  - Time
    - Best = O(N^2) because no matter what we have to find the minimum and cant terminate earlier!
    - Worst = O(N^2)
  - Space
    - O(N) for the input list
    - Auxiliary? O(1) in place



```python
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting
Selection Sort

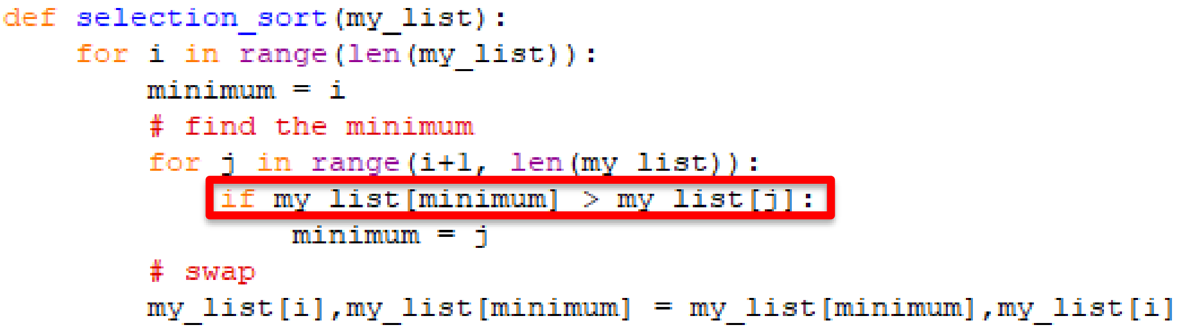
- Correctness
- Complexity
  - Time
    - Best = O(N^2) because no matter what we have to find the minimum and cant terminate earlier!
    - Worst = O(N^2)
    - But what if I tell you comparing the items have a cost of O(k)
      - Like comparing between words, you need to compare the alphabets



```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting
Selection Sort

- Correctness
- Complexity
  - Time
    - Best = O(N^2) because no matter what we have to find the minimum and cant terminate earlier!
    - Worst = O(N^2)
    - But what if I tell you comparing the items have a cost of O(k)
      - Like comparing between words, you need to compare the alphabets
      - We know complexity is based on comparison O(N^2) comparisons…



```
def selection_sort(my_list):
    for i in range(len(my_list)):
        minimum = i
        # find the minimum
        for j in range(i+1, len(my_list)):
            if my_list[minimum] > my_list[j]:
                minimum = j
        # swap
        my_list[i],my_list[minimum] = my_list[minimum],my_list[i]
```

Sorting

Selection Sort

- Correctness
- Complexity
  - Time
    - Best = O(N^2) because no matter what we have to find the minimum and cant terminate earlier!
    - Worst = O(N^2)
    - But what if I tell you comparing the items have a cost of O(k)
      - Like comparing between words, you need to compare the alphabets
      - We know complexity is based on comparison O(N^2) comparisons…
      - So our final complexity is O(kN^2)

Sorting
Selection Sort

- Correctness
- Complexity
- Stable?

Sorting

Selection Sort

- Correctness
- Complexity
- Stable?
  - Relative ordering doesn’t change

Sorting

Selection Sort

- Correctness
- Complexity
- Stable?
  - Relative ordering doesn’t change
  - Is it stable?

Sorting

Selection Sort

- Correctness
- Complexity
- Stable?
  - Relative ordering doesn't change
  - Is it stable? No! but why?

Sorting

Selection Sort

- Correctness
- Complexity
- Stable?
  - Relative ordering doesn’t change
  - Is it stable? No! but why?
  - [4a, 2, 3, 4b, 1]

Sorting
Selection Sort

- Correctness
- Complexity
- Stable?
  - Relative ordering doesn’t change
  - Is it stable? No! but why?
  - [4a, 2, 3, 4b, 1]
  - Minimum is 1, so we swap

Sorting

Selection Sort

- Correctness
- Complexity
- Stable?
  - Relative ordering doesn’t change
  - Is it stable? No! but why?
  - [4a, 2, 3, 4b, 1]
  - Minimum is 1, so we swap
  - [1, 2, 3, 4b, 4a]

Sorting
Selection Sort

- Correctness
- Complexity
- Stable?
  - Relative ordering doesn’t change
  - Is it stable? No! but why?
  - [4a, 2, 3, 4b, 1]
  - Minimum is 1, so we swap
  - [1, 2, 3, 4b, 4a]
  - Now we see that 4a is behind 4b!

MONASH University

Questions?

Sorting

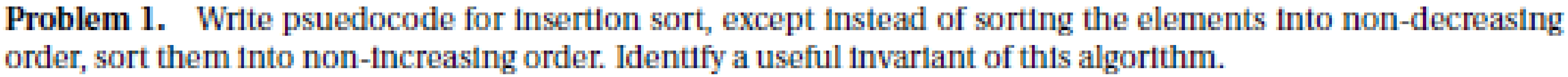
Insertion Sort

- Correctness
- Complexity

Sorting

Insertion Sort

- Correctness
- Complexity

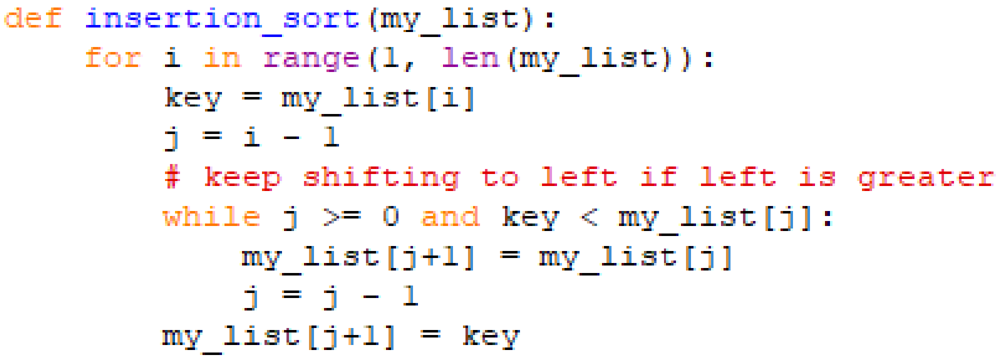


Problem 1. Write psuedocode for insertion sort, except instead of sorting the elements into non-decreasing order, sort them into non-increasing order. Identify a useful invariant of this algorithm.

Sorting

Insertion Sort

- Correctness
- Complexity

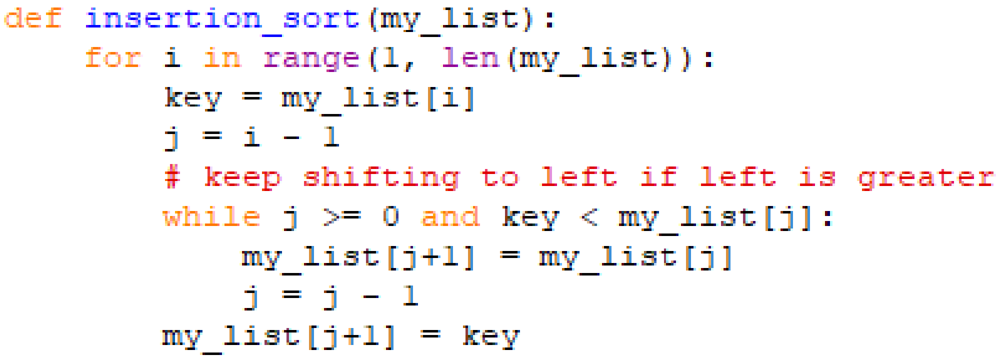


```
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting

Insertion Sort

- Correctness
  - Loop invariant
  - Termination
- Complexity

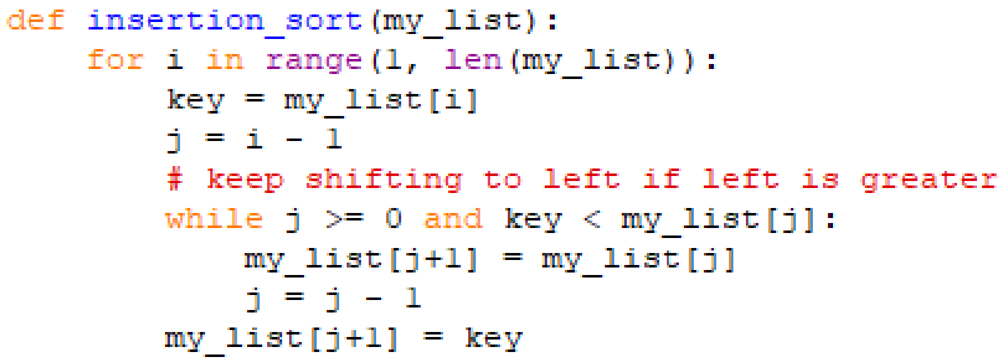


```
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting

Insertion Sort

- Correctness
  - Loop invariant
  - Termination
    - Simple, I skip this
- Complexity

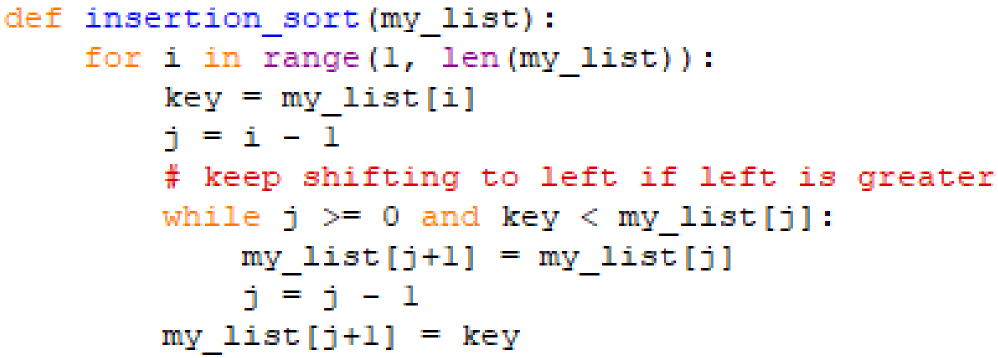


```python
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting

Insertion Sort

- Correctness
  - Loop invariant
    - my_list[0…i-1] sorted
  - Termination
    - Simple, I skip this
- Complexity

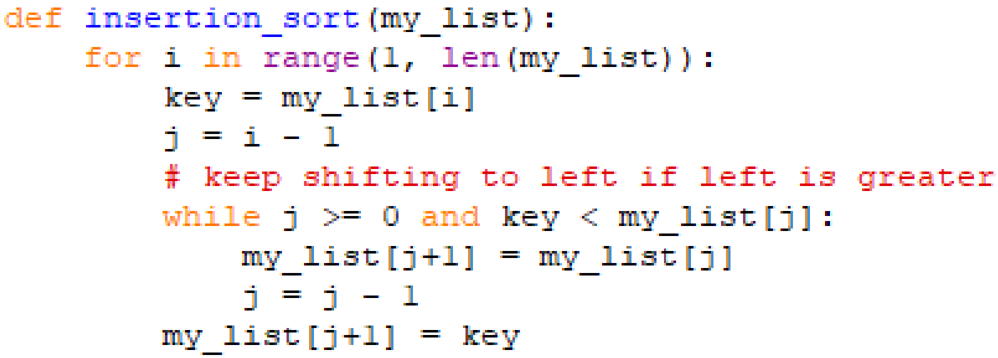


```
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting

Insertion Sort

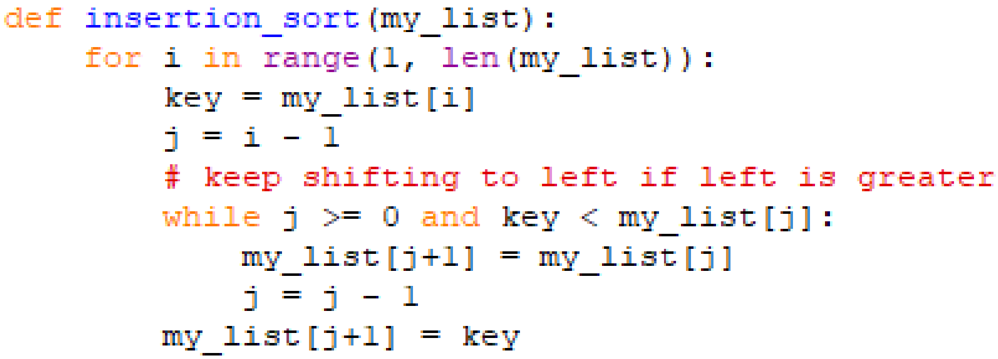
- Correctness
- Complexity
  - Best
  - Worst



```python
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting
Insertion Sort

- Correctness
- Complexity
  - Best O(N) comparison
    - Each loop only look and compare with left item once
  - Worst

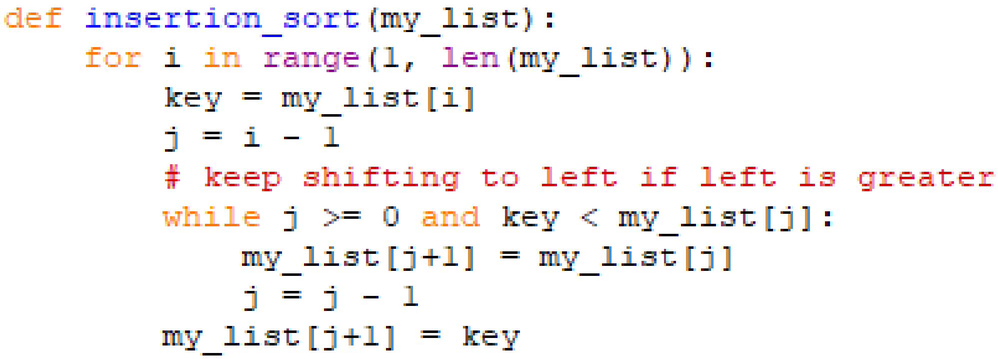


```
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting

Insertion Sort

- Correctness
- Complexity
  - Best O(N) comparison
    - Each loop only look and compare with left item once
  - Worst O(N^2)
    - Each loop keep look left, compare and swap till beginning of list

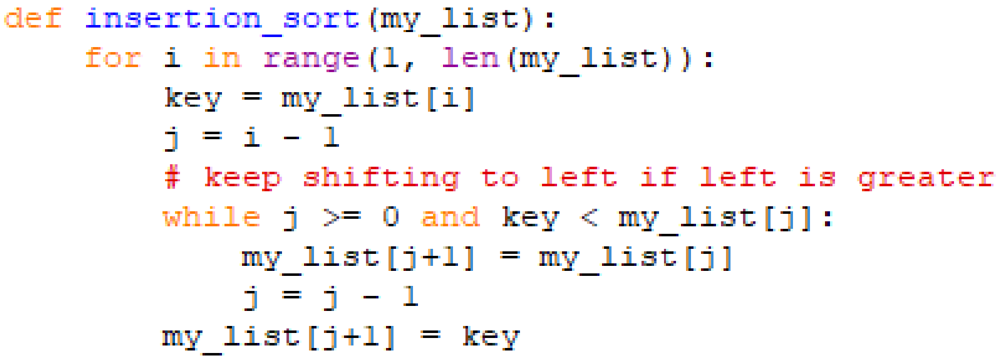


```
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting

Insertion Sort

- Correctness
- Complexity
  - Best O(N) comparison
    - Each loop only look and compare with left item once
  - Worst O(N^2)
    - Each loop keep look left, compare and swap till beginning of list
  - So if O(k) is the comparison cost, when we have O(kN^2) worst case!

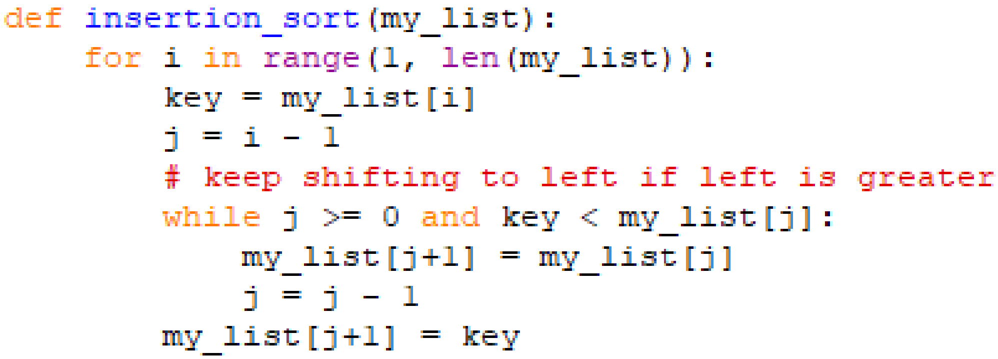


```python
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting

Insertion Sort

- Correctness
- Complexity
  - Best O(N) comparison
    - Each loop only look and compare with left item once
  - Worst O(N^2)
    - Each loop keep look left, compare and swap till beginning of list
  - So if O(k) is the comparison cost, when we have O(kN^2) worst case!
  - What about space?

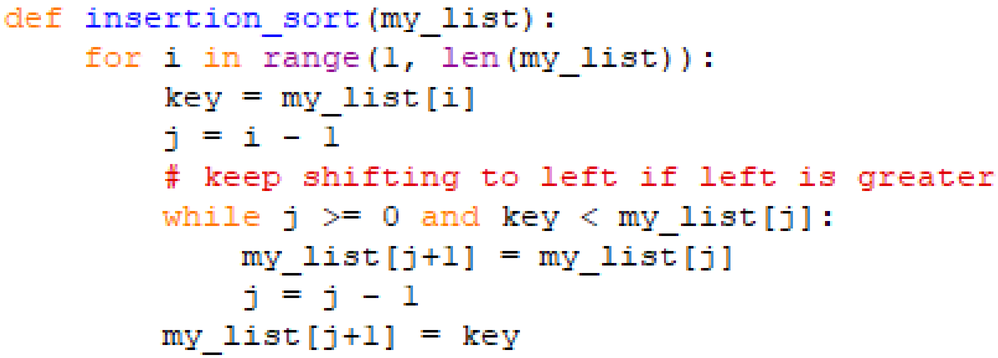


```
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting

Insertion Sort

- Correctness
- Complexity
  - Best O(N) comparison
    - Each loop only look and compare with left item once
  - Worst O(N^2)
    - Each loop keep look left, compare and swap till beginning of list
  - So if O(k) is the comparison cost, when we have O(kN^2) worst case!
  - What about space?
    - O(N) for the input list
    - O(1) auxiliary cause it is in-place

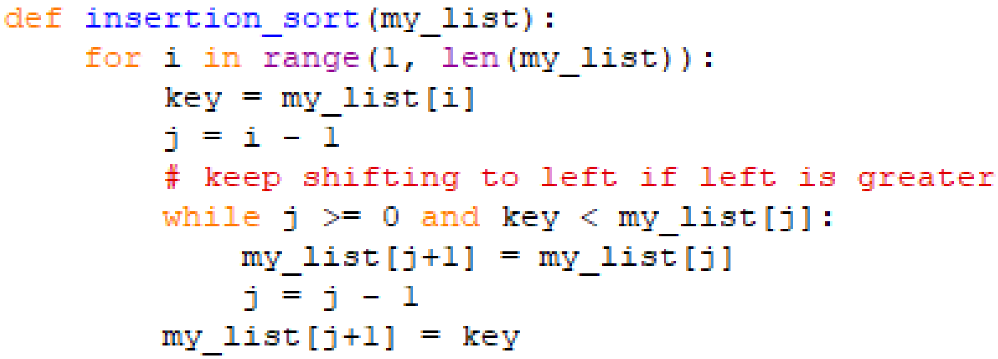


```python
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting

Insertion Sort

- Correctness
- Complexity
- Stability

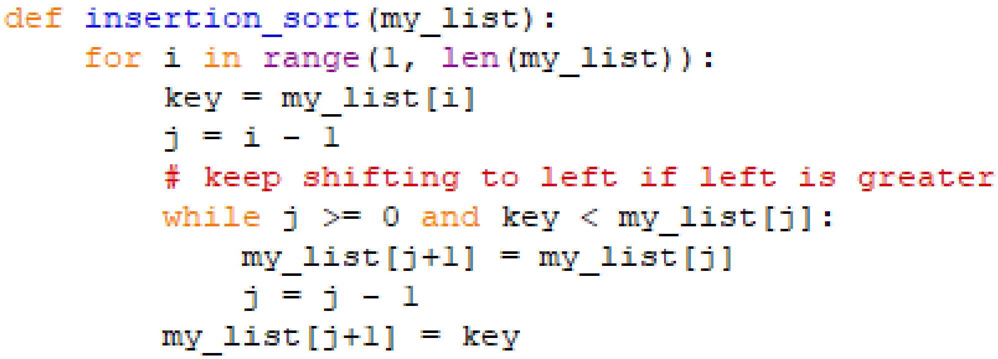


```
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting

Insertion Sort

- Correctness
- Complexity
- Stability
  - Yes

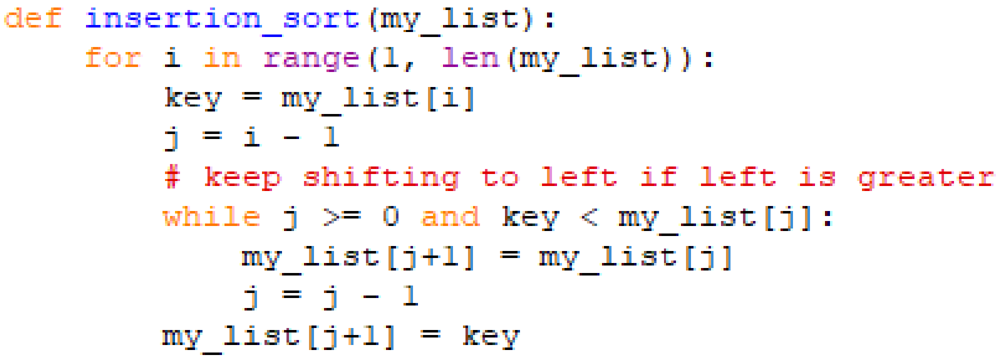


```python
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting

Insertion Sort

- Correctness
- Complexity
- Stability
  - Yes
  - Don’t swap if value is the same

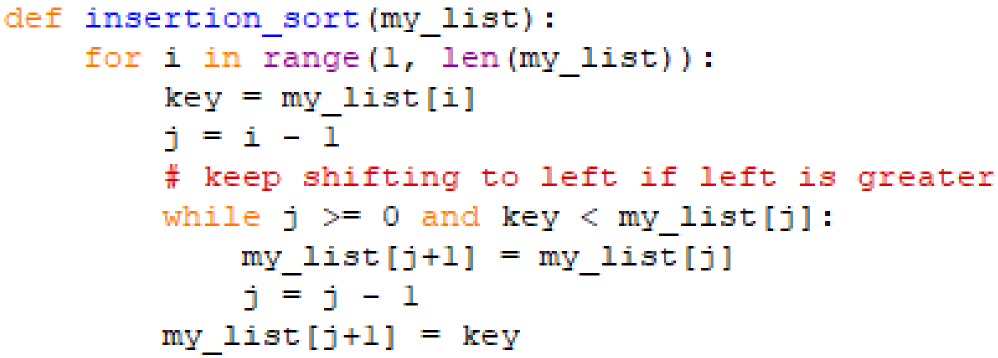


```python
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Sorting

Insertion Sort

- Correctness
- Complexity
- Stability
  - Yes
  - Don’t swap if value is the same
  - Most shifting will ensure stability



```python
def insertion_sort(my_list):
    for i in range(1, len(my_list)):
        key = my_list[i]
        j = i - 1
        # keep shifting to left if left is greater
        while j >= 0 and key < my_list[j]:
            my_list[j+1] = my_list[j]
            j = j - 1
        my_list[j+1] = key
```

Questions?

Summary

Sorting

	Best	Worst	Average	Stable?	In-place?
Selection Sort	$O(N^2)$	$O(N^2)$	$O(N^2)$	No	Yes
Insertion Sort	O(N)	$O(N^2)$	$O(N^2)$	Yes	Yes
Heap Sort	O(N log N)	O(N log N)	O(N log N)	No	Yes
Merge Sort	O(N log N)	O(N log N)	O(N log N)	Yes	No
Quick Sort	O(N log N)	$O(N^2)$ – can be made O(N log N)	O(N log N)	Depends	No

Summary

Sorting

	Best	Worst	Average		
Selection Sort	$O(N^2)$	$O(N^2)$	$O(N^2)$		
Insertion Sort	$O(N)$	$O(N^2)$	$O(N^2)$		
Heap Sort	O(N log N)	O(N log N)	O(N log N)		
Merge Sort	O(N log N)	O(N log N)	O(N log N)	Yes	No
Quick Sort	O(N log N)	$O(N^2)$ – can be made O(N log N)	O(N log N)	Depends	No

Summary

Auxiliary for Recursion

- The recursion stack takes up memory!!!

Summary

Auxiliary for Recursion

- The recursion stack takes up memory!!!
  - So that is why it isn’t in place!

Summary

Auxiliary for Recursion

- The recursion stack takes up memory!!!
  - So that is why it isn't in place!
  - If I have recursion log N times, then I take O(log N) space for the recursion alone!
  - If each recursion is k, then my total space is O(k log N)!!!

Summary

Auxiliary for Recursion

- The recursion stack takes up memory!!!
  - So that is why it isn't in-place!
    - Iterative is easier to get in-place
  - If I have recursion log N times, then I take O(log N) space for the recursion alone!
  - If each recursion is k, then my total space is O(k log N)!!!

Summary

Sorting

	Best	Worst	Average	Stable?	In-place?
Selection Sort	$O(N^2)$	$O(N^2)$	$O(N^2)$	No	Yes
Insertion Sort	$O(N)$	$O(N^2)$	$O(N^2)$	Yes	Yes
Heap Sort	$O(N \log N)$	$O(N \log N)$	$O(N \log N)$	No	Yes
Merge Sort	$O(N \log N)$	$O(N \log N)$	$O(N \log N)$	Yes	No
Quick Sort	$O(N \log N)$	$O(N^2)$ – can be made $O(N \log N)$	$O(N \log N)$	Depends	No

Complexity

Time – Lower Bound

- So… what is the lower bound for the sorting algorithms that we have learnt?
  - Bubble
  - Insertion
  - Selection
  - Quick
  - Merge

- These are all comparison based
- Ω(N log N)
- We will see more of this later

Questions?

Have a break!

Sorting

Non-Comparison

- We can sort without comparing elements in a list!

Sorting

Non-Comparison

- We can sort without comparing elements in a list!
  - Counting sort
  - Radix sort

Questions?

Counting Sort

- Very simple concept
- I am sure we all know this…

- Now let us begin with a list

4	2	1	3	1	4	5

Counting Sort

- Very simple concept
- I am sure we all know this…

- Now let us begin with a list

4	2	1	3	1	4	5

- What is the maximum number?

Counting Sort

- Very simple concept
- I am sure we all know this…

- Now let us begin with a list

4	2	1	3	1	4	5

- What is the maximum number?
  - 5 but how do we know?

Counting Sort

- Very simple concept
- I am sure we all know this…

- Now let us begin with a list

4	2	1	3	1	4	5

- What is the maximum number?
  - 5 but how do we know? Loop through the list in O(N)

Counting Sort

- Our input

4	2	1	3	1	4	5

- We know max is 5

Counting Sort

- Our input

4	2	1	3	1	4	5

Anyone noticed
the list is crooked?
#OCDtrigger

- We know max is 5

Counting Sort

- Our input

4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5

Counting Sort

- Our input

4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5

Counting Sort

- Out input

4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5
0	0	0	0	0	0

Counting Sort

- Our input

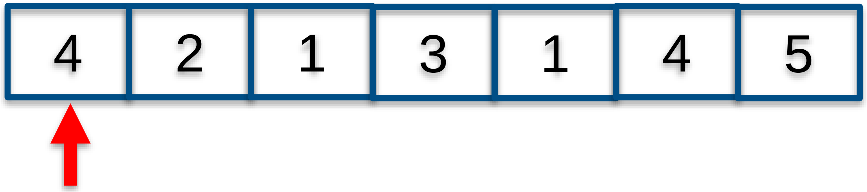
4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5
0	0	0	0	0	0

Counting Sort

- Our input



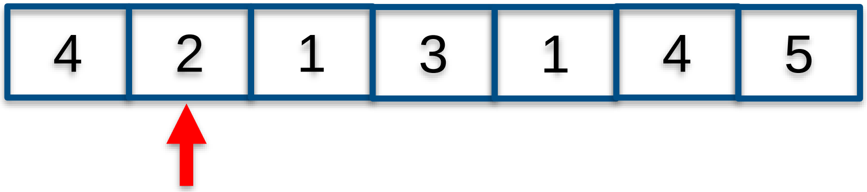
4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5
0	0	0	0	1	0

Counting Sort

- Our input



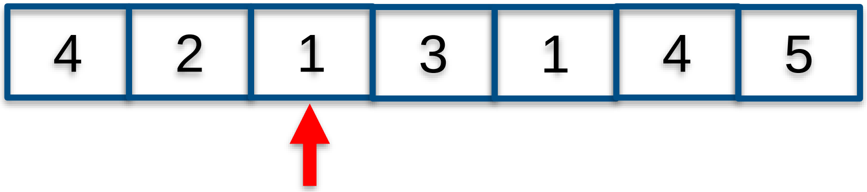
4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5
0	0	1	0	1	0

Counting Sort

- Out input



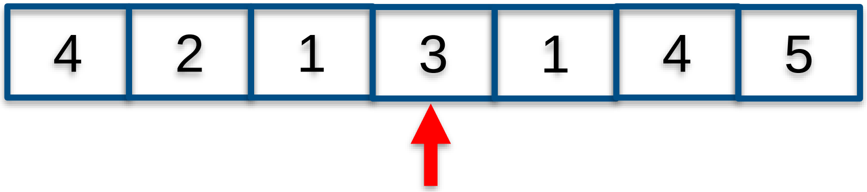
4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5
0	1	1	0	1	0

Counting Sort

- Our input



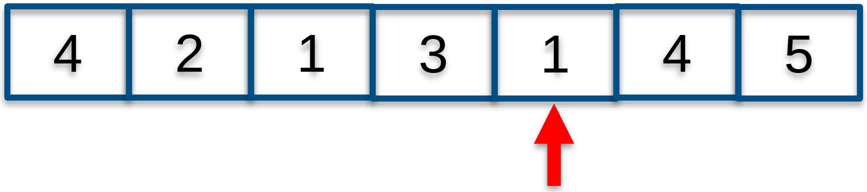
4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5
0	1	1	1	1	0

Counting Sort

- Our input



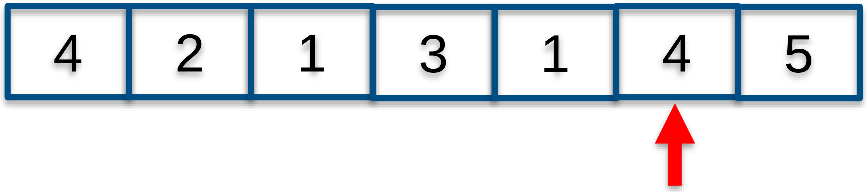
4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5
0	2	1	1	1	0

Counting Sort

- Our input



4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5
0	2	1	1	2	0

Counting Sort

- Our input

4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5
0	2	1	1	2	1

Counting Sort

- Our input

4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency

Counting Sort

- Our input

4	2	1	3	1	4	5

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency

- So how do we sort it now then?

Counting Sort

- Our input

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency

- So how do we sort it now then?

Counting Sort

- Our input

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency

- So how do we sort it now then?

Counting Sort

- Our input

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency

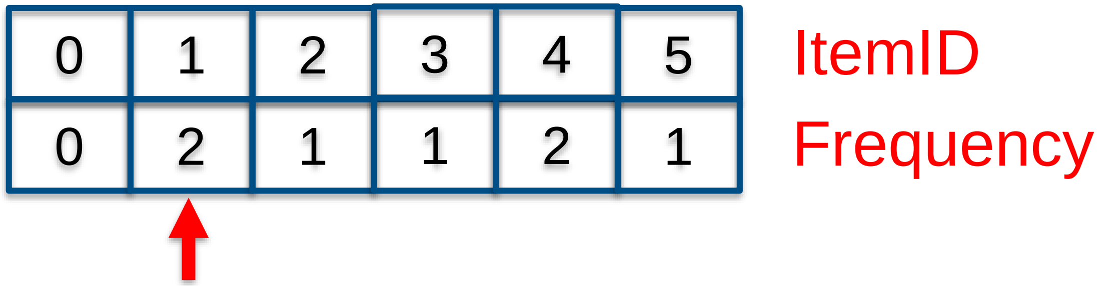
- So how do we sort it now then?

Counting Sort

- Our input

1	1					

- We know max is 5



0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency

- So how do we sort it now then?

Counting Sort

- Our input

1	1	2				

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency

- So how do we sort it now then?

Counting Sort

- Our input

1	1	2	3			

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency

- So how do we sort it now then?

Counting Sort

- Our input

1	1	2	3	4	4	

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency

- So how do we sort it now then?

Counting Sort

- Our input

1	1	2	3	4	4	5

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency

- So how do we sort it now then?

Counting Sort

- Our input

1	1	2	3	4	4	5

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency

- So how do we sort it now then?

Counting Sort
Complexity

- Time?

Counting Sort

Complexity

- Time?
  - Find the maximum O(N)

Counting Sort

Complexity

- Time?
  - Find the maximum O(N)
  - Build the count-array O(M) where M is the max

Counting Sort

Complexity

- Time?
  - Find the maximum O(N)
  - Build the count-array O(M) where M is the max
  - Go through input list and update the count-array

Counting Sort

Complexity

- Time?
  - Find the maximum O(N)
  - Build the count-array O(M) where M is the max
  - Go through input list and update the count-array
    - How to make it fast?

Counting Sort

Complexity

- Time?
  - Find the maximum O(N)
  - Build the count-array O(M) where M is the max
  - Go through input list and update the count-array
    - How to make it fast?

0	1	2	3	4	5	Index
0	2	1	1	2	1	Frequency

Counting Sort

Complexity

- Time?
  - Find the maximum O(N)
  - Build the count-array O(M) where M is the max
  - Go through input list and update the count-array
    - How to make it fast?
    - Therefore this is O(N) since we can have O(1) access to the count-array

Counting Sort

Complexity

- Time?
  - Find the maximum O(N)
  - Build the count-array O(M) where M is the max
  - Go through input list and update the count-array
    - How to make it fast?
    - Therefore this is O(N) since we can have O(1) access to the count-array
  - Loop through count-array to rebuild the original list O(M)

Counting Sort

Complexity

- Time?
  - Find the maximum O(N)
  - Build the count-array O(M) where M is the max
  - Go through input list and update the count-array
    - How to make it fast?
    - Therefore this is O(N) since we can have O(1) access to the count-array
  - Loop through count-array to rebuild the original list O(M)
  - Total = O(N + M + N + M) = O(N+M)

Counting Sort

Complexity

- Time?
  - Find the maximum O(N)
  - Build the count-array O(M) where M is the max
  - Go through input list and update the count-array
    - How to make it fast?
    - Therefore this is O(N) since we can have O(1) access to the count-array
  - Loop through count-array to rebuild the original list O(M)
  - Total = O(N + M + N + M) = O(N+M)

  - So we want M << N for this to be good
    - Else even N log N < M

Counting Sort
Complexity

- Time?
  - Find the maximum O(N)
  - Build the count-array O(M) where M is the max
  - Go through input list and update the count-array
    - How to make it fast?
    - Therefore this is O(N) since we can have O(1) access to the count-array
  - Loop through count-array to rebuild the original list O(M)
  - Total = O(N + M + N + M) = O(N+M)

  - So we want M << N for this to be good
  - If we are doing alphabets only, then the M = 26 for the 26 character (after ascii conversion + maths)

MONASH University

Questions?

Counting Sort

Complexity

- Space?

Counting Sort

Complexity

- Space?
  - Input list O(N)
  - Count-array O(M)

Counting Sort

Complexity

- Space?
  - Input list O(N)
  - Count-array O(M)

  - Total = O(N + M)
  - Auxiliary = O(M)

Questions?

Counting Sort

- Live programming session
- Let us try to code this since it is simple…

Counting Sort

- Live programming session
- Let us try to code this since it is simple…

- I will start writing the first part
  - You try to add in your own codes and compare at each step

Questions?

Counting Sort

Issue…

- Now imagine the following:

200	151	291	981	369	421	671

Counting Sort

Issue…

- Now imagine the following:

200	151	291	981	369	421	671

  – What is my complexity?

Counting Sort

Issue…

- Now imagine the following:

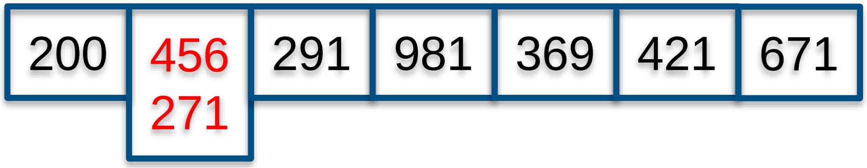
200	151	291	981	369	421	671

  - What is my complexity?
    - Time…
    - Space…

Counting Sort

Issue…

- Now imagine the following:



200	456 271	291	981	369	421	671

  - What is my complexity?
    - Time…
    - Space…

  - What if one of the value is LARGE

Counting Sort

Issue…

- Now imagine the following:

200	456 271	291	981	369	421	671

  - What is my complexity?
    - Time…
    - Space…

  - What if one of the value is LARGE

M is large!!!

Counting Sort

Issue…

- Now imagine the following:

200	151	291	981	369	421	671

  – What is my complexity?
    - Time…
    - Space…

  – Let us leave it at it is first…

Questions?

Counting Sort

- Stable?

Counting Sort

- Stable?
  - No
  - We only remember the frequency

Counting Sort

- Stable?
  - No
  - We only remember the frequency

- But can we make it stable?

Counting Sort

- Stable?
  - No
  - We only remember the frequency

- But can we make it stable?
  - Yes but at the cost of memory

Counting Sort

4a	2	1a	3	1b	4b	5

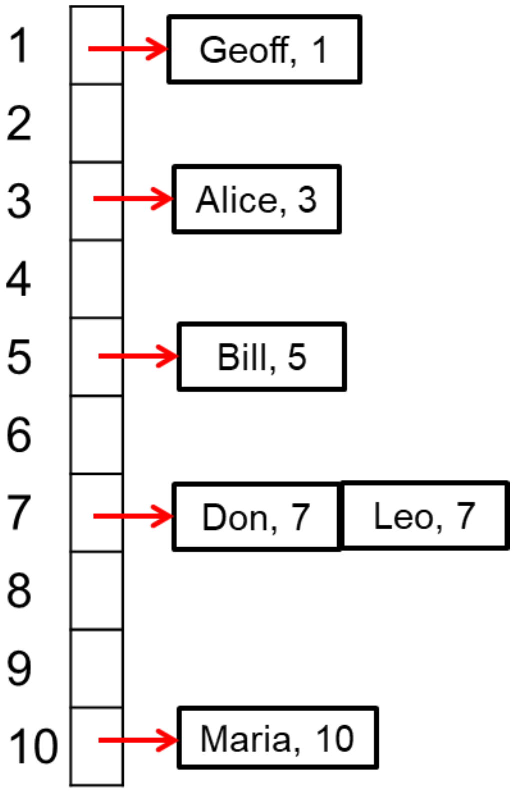
0	1	2	3	4	5	Index
	1a	2	3	4a	5	Frequency
	1b			4b		

Counting Sort

- Stable?
  - No
  - We only remember the frequency
- But can we make it stable?
  - Yes but at the cost of memory
  - Similar to separate chaining

Counting Sort

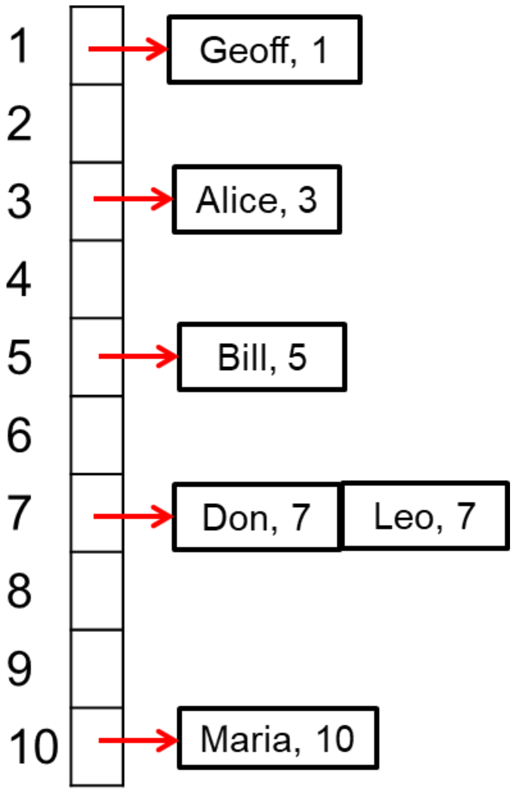
- Stable?
  - No
  - We only remember the frequency
- But can we make it stable?
  - Yes but at the cost of memory
  - Similar to separate chaining



Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

Counting Sort

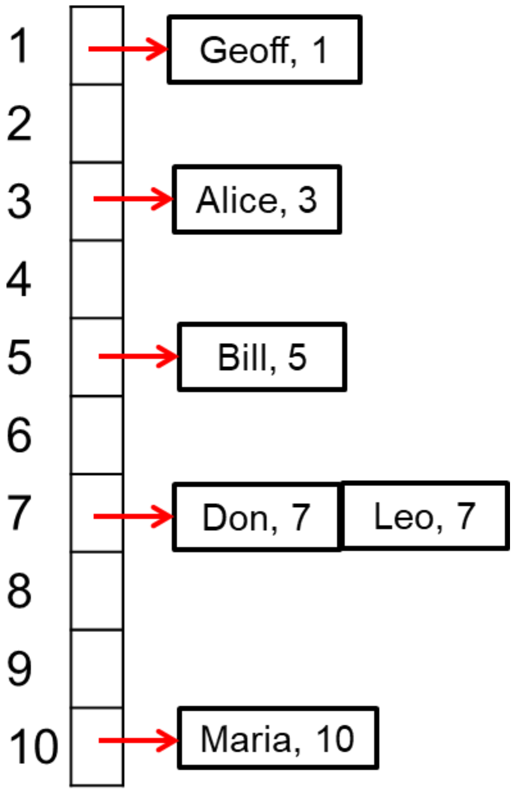
- Stable?
  - No
  - We only remember the frequency
- But can we make it stable?
  - Yes but at the cost of memory
  - Similar to separate chaining
  - At most we have N items only anyways
    - So it is O(M + N) space still



Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

Counting Sort

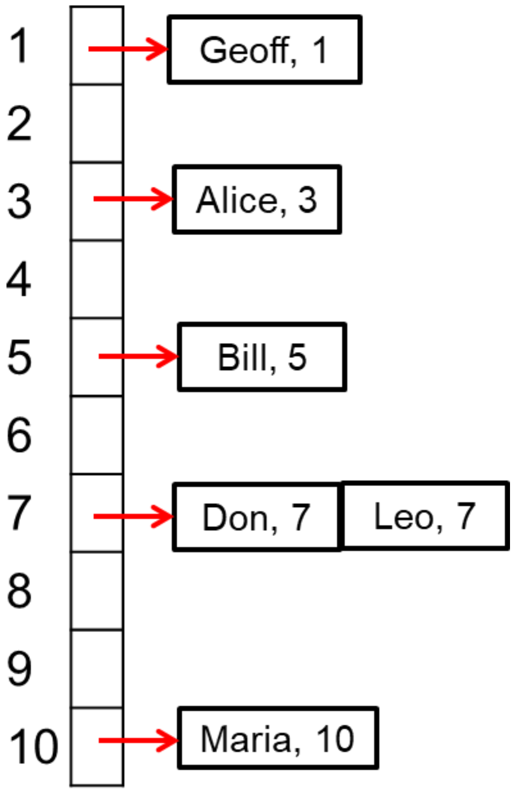
- Stable?
  - No
  - We only remember the frequency
- But can we make it stable?
  - Yes but at the cost of memory
  - Similar to separate chaining
  - At most we have N items only anyways
    - So it is O(M + N) space still
    - Can you see why?



Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

Counting Sort

- Stable?
  - No
  - We only remember the frequency
- But can we make it stable?
  - Yes but at the cost of memory
  - Similar to separate chaining
  - At most we have N items only anyways
    - So it is O(M + N) space still
    - Can you see why?



1 → Geoff, 1

2

3 → Alice, 3

4

5 → Bill, 5

6

7 → Don, 7 → Leo, 7

8

9

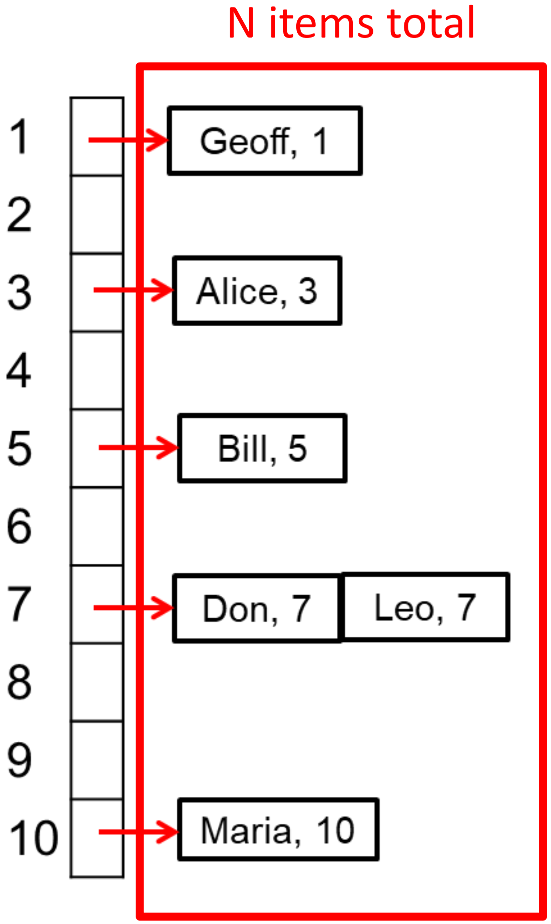
10 → Maria, 10

Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

N items

Counting Sort

- Stable?
  - No
  - We only remember the frequency
- But can we make it stable?
  - Yes but at the cost of memory
  - Similar to separate chaining
  - At most we have N items only anyways
    - So it is O(M + N) space still
    - Can you see why?



Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

N items

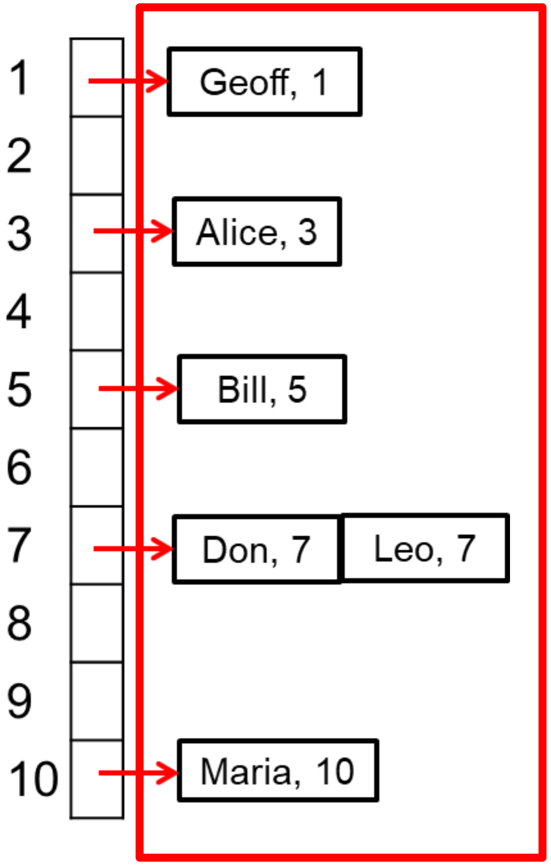
Counting Sort

- Stable?
  - No
  - We only remember the frequency

- But can we make it stable?
  - Yes but at the cost of memory
  - Similar to separate chaining
  - At most we have N items only anyways
    - So it is O(M + N) space still
    - Can you see why?

Not O(N*M)

N items total



Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

N items

Counting Sort

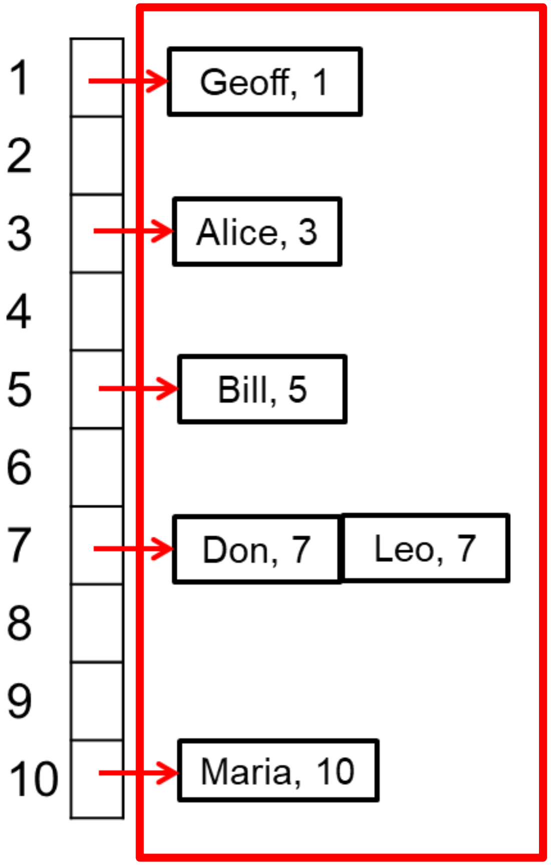
- Stable?
  - No
  - We only remember the frequency

VERY COMMON MISCONCEPTION

- But can we make it stable?
  - Yes but at the cost of memory
  - Similar to separate chaining
  - At most we have N items only anyways
    - So it is O(M + N) space still
    - Can you see why?

Not O(N*M)

N items total



Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

N items

Questions?

Counting Sort

- Stable?
  - No
  - We only remember the frequency

- But can we make it stable?
  - Yes but at the cost of memory
  - Similar to separate chaining
  - There is another way, refer to Nathan's amazing slide

Stable Counting Sort (Method 1)

Input

(3,a)	**(1,p)**	**(3,c)**	**(7,f)**	**(5,g)**	**(3,b)**	**(7,d)**	**(8,w)**

Construct count:

- For each key in input,
- count[key] += 1

count

1	1
2	0
3	3
4	0
5	1
6	0
7	2
8	1

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	**(1,p)**	**(3,c)**	**(7,f)**	**(5,g)**	**(3,b)**	**(7,d)**	**(8,w)**

Construct count:
- For each key in input,
- count[key] += 1

Construct position:
- Initialise first position as a 1

count

1	1
2	0
3	3
4	0
5	1
6	0
7	2
8	1

position

1	1
2	0
3	0
4	0
5	0
6	0
7	0
8	0

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	**(1,p)**	**(3,c)**	**(7,f)**	**(5,g)**	**(3,b)**	**(7,d)**	**(8,w)**

Construct count:
- For each key in input,
- count[key] += 1

Construct position:
- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

count

1	1
2	0
3	3
4	0
5	1
6	0
7	2
8	1

position

1	1
2	2
3	0
4	0
5	0
6	0
7	0
8	0

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

count

1	1
2	0
3	3
4	0
5	1
6	0
7	2
8	1

position

1	1
2	2
3	2
4	0
5	0
6	0
7	0
8	0

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	**(1,p)**	**(3,c)**	**(7,f)**	**(5,g)**	**(3,b)**	**(7,d)**	**(8,w)**

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

count

1	1
2	0
3	3
4	0
5	1
6	0
7	2
8	1

position

1	1
2	2
3	2
4	5
5	0
6	0
7	0
8	0

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	**(1,p)**	**(3,c)**	**(7,f)**	**(5,g)**	**(3,b)**	**(7,d)**	**(8,w)**

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

	count	position
1	1	1
2	0	2
3	3	2
4	0	5
5	1	5
6	0	0
7	2	0
8	1	0

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	**(1,p)**	**(3,c)**	**(7,f)**	**(5,g)**	**(3,b)**	**(7,d)**	**(8,w)**

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

count

1	1
2	0
3	3
4	0
5	1
6	0
7	2
8	1

position

1	1
2	2
3	2
4	5
5	5
6	6
7	0
8	0

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

	count	position
1	1	1
2	0	2
3	3	2
4	0	5
5	1	5
6	0	6
7	2	6
8	1	0

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	**(1,p)**	**(3,c)**	**(7,f)**	**(5,g)**	**(3,b)**	**(7,d)**	**(8,w)**

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

	count
1	1
2	0
3	3
4	0
5	1
6	0
7	2
8	1

	position
1	1
2	2
3	2
4	5
5	5
6	6
7	6
8	8

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	**(1,p)**	**(3,c)**	**(7,f)**	**(5,g)**	**(3,b)**	**(7,d)**	**(8,w)**

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

- Go through input, looking at each (key, val)
- Set output[position[key]] to the (key, val) pair from input
- Increment position[key]

count

1	1
2	0
3	3
4	0
5	1
6	0
7	2
8	1

position

1	1
2	2
3	2
4	5
5	5
6	6
7	6
8	8

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)

Construct count:
- For each key in input,
- count[key] += 1

Construct position:
- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output
- Go through input, looking at each (key, val)
- Set output[position[key]] to the (key, val) pair from input
- Increment position[key]

	count		position
1	1	1	1
2	0	2	2
3	3	3	2
4	0	4	5
5	1	5	5
6	0	6	6
7	2	7	6
8	1	8	8

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

- Go through input, looking at each (key, val)
- Set output[position[key]] to the (key, val) pair from input
- Increment position[key]

	count	position
1	1	1
2	0	2
3	3	3
4	0	5
5	1	5
6	0	6
7	2	6
8	1	8

Output

	(3,a)						
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	**(1,p)**	**(3,c)**	**(7,f)**	**(5,g)**	**(3,b)**	**(7,d)**	**(8,w)**

Construct count:
- For each key in input,
- count[key] += 1

Construct position:
- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output
- Go through input, looking at each (key, val)
- Set output[position[key]] to the (key, val) pair from input
- Increment position[key]

	count		position
1	1	1	2
2	0	2	2
3	3	3	3
4	0	4	5
5	1	5	5
6	0	6	6
7	2	7	6
8	1	8	8

Output

(1,p)	**(3,a)**						
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

- Go through input, looking at each (key, val)
- Set output[position[key]] to the (key, val) pair from input
- Increment position[key]

count

1	1
2	0
3	3
4	0
5	1
6	0
7	2
8	1

position

1	2
2	2
3	4
4	5
5	5
6	6
7	6
8	8

Output

(1,p)	(3,a)	(3,c)					
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	**(1,p)**	**(3,c)**	**(7,f)**	**(5,g)**	**(3,b)**	**(7,d)**	**(8,w)**

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

- Go through input, looking at each (key, val)
- Set output[position[key]] to the (key, val) pair from input
- Increment position[key]

count

1	1
2	0
3	3
4	0
5	1
6	0
7	2
8	1

position

1	2
2	2
3	4
4	5
5	5
6	7
7	6
8	8

Output

(1,p)	**(3,a)**	**(3,c)**			**(7,f)**		
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

- Go through input, looking at each (key, val)
- Set output[position[key]] to the (key, val) pair from input
- Increment position[key]

count

1	1
2	0
3	3
4	0
5	1
6	0
7	2
8	1

position

1	2
2	2
3	4
4	5
5	6
6	7
7	6
8	8

Output

(1,p)	(3,a)	(3,c)		(5,g)	(7,f)		
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	**(1,p)**	**(3,c)**	**(7,f)**	**(5,g)**	**(3,b)**	**(7,d)**	**(8,w)**

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

- Go through input, looking at each (key, val)
- Set output[position[key]] to the (key, val) pair from input
- Increment position[key]

count

1	1
2	0
3	3
4	0
5	1
6	0
7	2
8	1

position

1	2
2	2
3	5
4	5
5	6
6	7
7	6
8	8

Output

(1,p)	**(3,a)**	**(3,c)**	**(3,b)**	**(5,g)**	**(7,f)**		
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	**(1,p)**	**(3,c)**	**(7,f)**	**(5,g)**	**(3,b)**	**(7,d)**	**(8,w)**

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

- Go through input, looking at each (key, val)
- Set output[position[key]] to the (key, val) pair from input
- Increment position[key]

	count	position
1	1	2
2	0	2
3	3	5
4	0	5
5	1	6
6	0	7
7	2	7
8	1	8

Output

(1,p)	**(3,a)**	**(3,c)**	**(3,b)**	**(5,g)**	**(7,f)**	**(7,d)**	
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)

Construct count:

- For each key in input,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

- Go through input, looking at each (key, val)
- Set output[position[key]] to the (key, val) pair from input
- Increment position[key]

	count		position
1	1	1	2
2	0	2	2
3	3	3	5
4	0	4	5
5	1	5	6
6	0	6	7
7	2	7	7
8	1	8	9

Output

(1,p)	(3,a)	(3,c)	(3,b)	(5,g)	(7,f)	(7,d)	(8,w)
1	2	3	4	5	6	7	8

Questions?

Counting Sort

- Stable?
  - No
  - We only remember the frequency

- But can we make it stable?
  - Yes but at the cost of memory
  - Similar to separate chaining
  - There is another way, refer to Nathan’s amazing slide
  - Are the complexity the same?

Questions?

Have a break again!

Counting Sort

Remember this issue…

- Now imagine the following:

200	151	291	981	369	421	671

  - What is my complexity?
    - Time…
    - Space…
  - Let us leave it at it is first…

Counting Sort

Remember this issue…

- Now imagine the following:

200	151	291	981	369	421	671

  - What is my complexity?
    - Time…
    - Space…
  - Let us leave it at it is first… We shall resolve this now…

Counting Sort

Remember this issue…

- Now imagine the following:

200	151	291	981	369	421	671

  - What is my complexity?
    - Time…
    - Space…
  - Let us leave it at it is first… We shall resolve this now…

Questions?

Radix Sort

A different outlook…

- With this input…

200	151	291	981	369	421	671

Radix Sort

A different outlook…

- With this input…

200	151	291	981	369	421	671

  - What if we view it differently?

Radix Sort

A different outlook…

- With this input…

200	151	291	981	369	421	671

  - What if we view it differently?

200
151
291
981
369
421

Radix Sort

A different outlook…

- With this input…

200	151	291	981	369	421	671

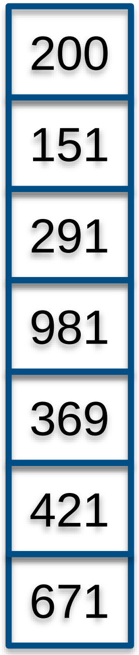
  - What if we view it differently? How would we sort it?

200
151
291
981
369
421

Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?



200

151

291

981

369

421

671

Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant

200
151
291
981
369
421
671

Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant

200
151
291
981
369
421
671

Radix Sort

A different outlook…

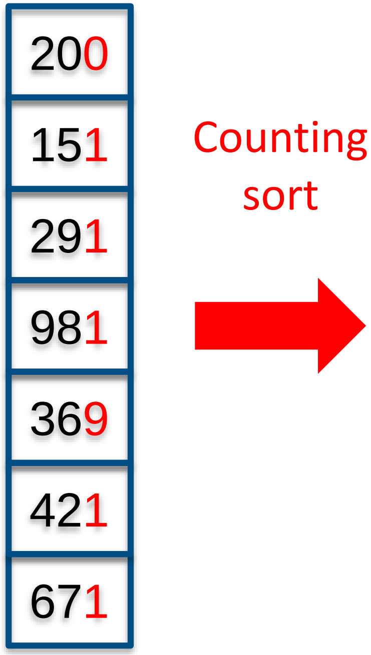
- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant

200
151
291
981
369
421
671

Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant



200

151

291

981

369

421

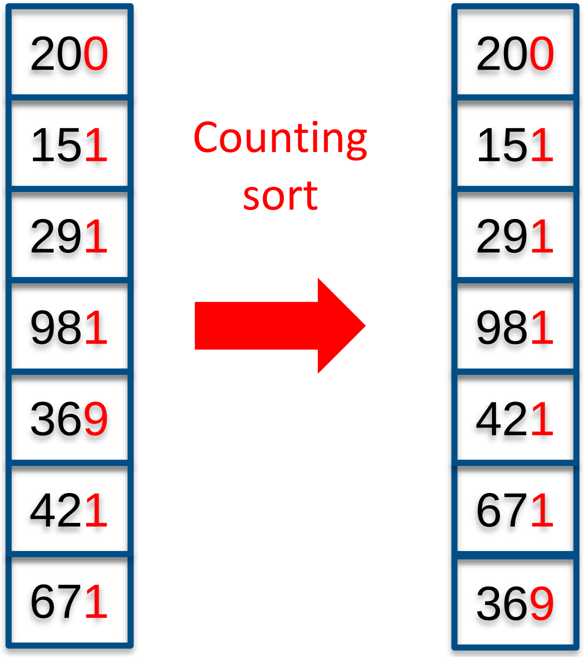
671

Counting sort

Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant



200
151
291
981
369
421
671

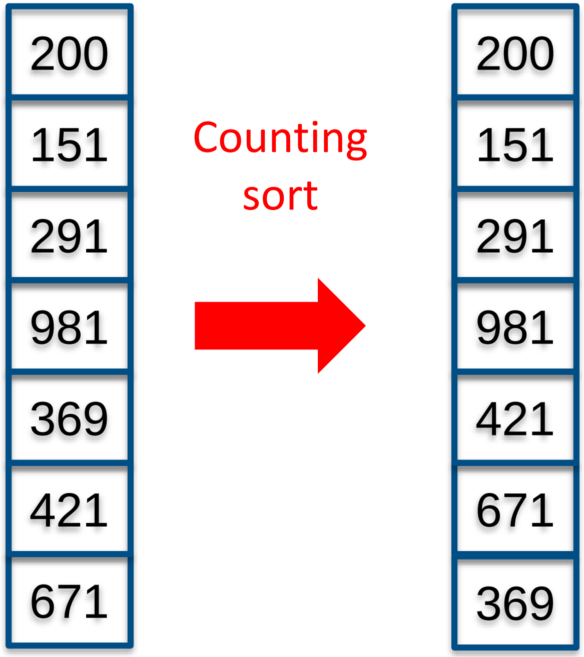
Counting sort

200
151
291
981
421
671
369

Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant



200
151
291
981
369
421
671

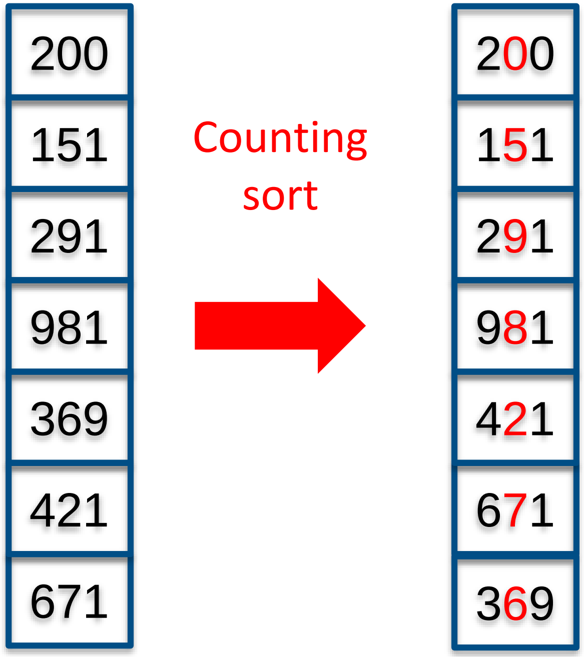
Counting sort

200
151
291
981
421
671
369

Radix Sort

A different outlook…

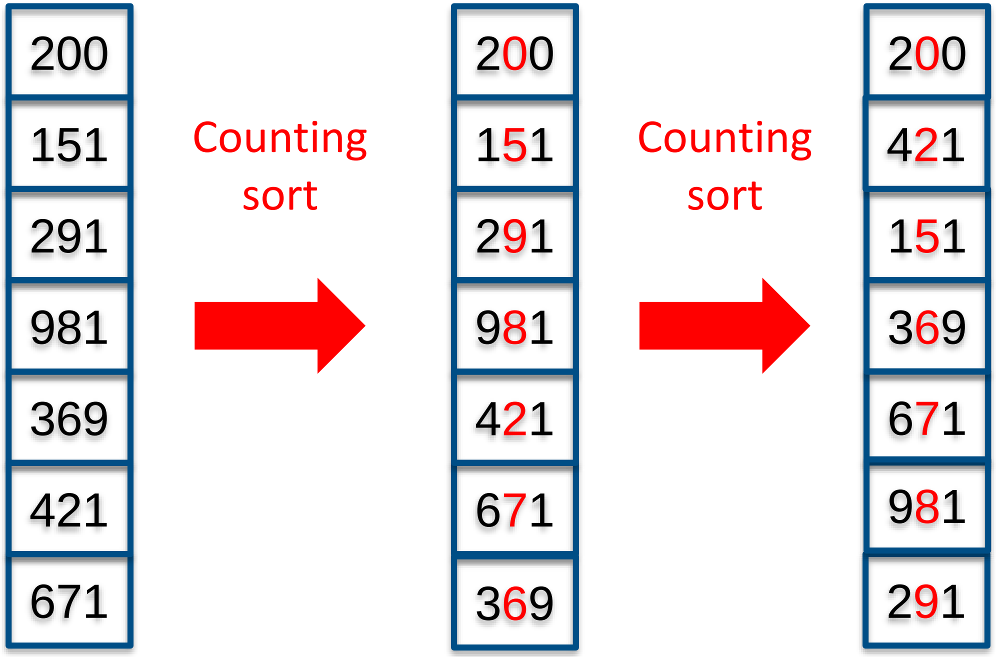
- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant



Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant

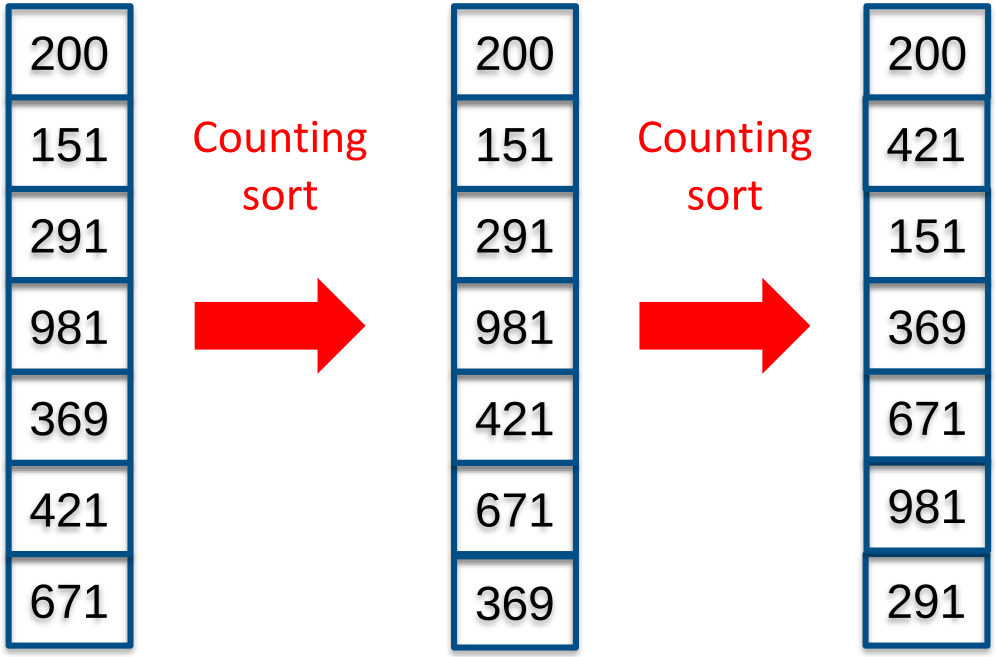


200	Counting sort	200	Counting sort	200
151		151		421
291		291		151
981		981		369
369		421		671
421		671		981
671		369		291

Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant



200
151
291
981
369
421
671

Counting sort

200
151
291
981
421
671
369

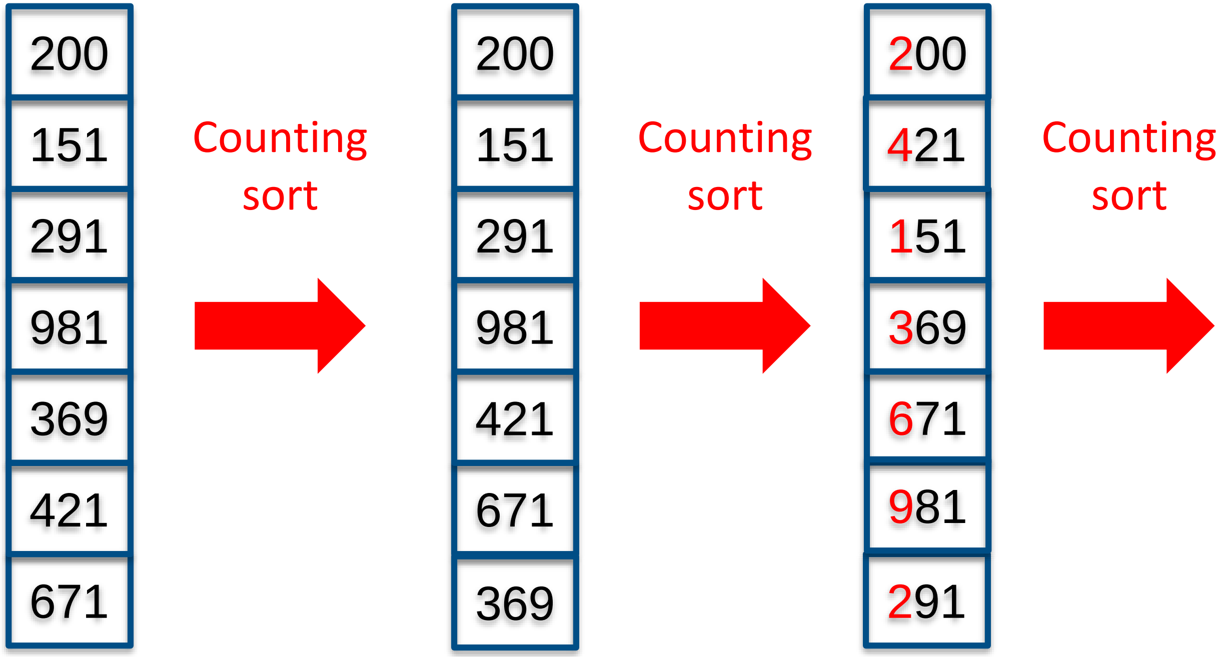
Counting sort

200
421
151
369
671
981
291

Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant

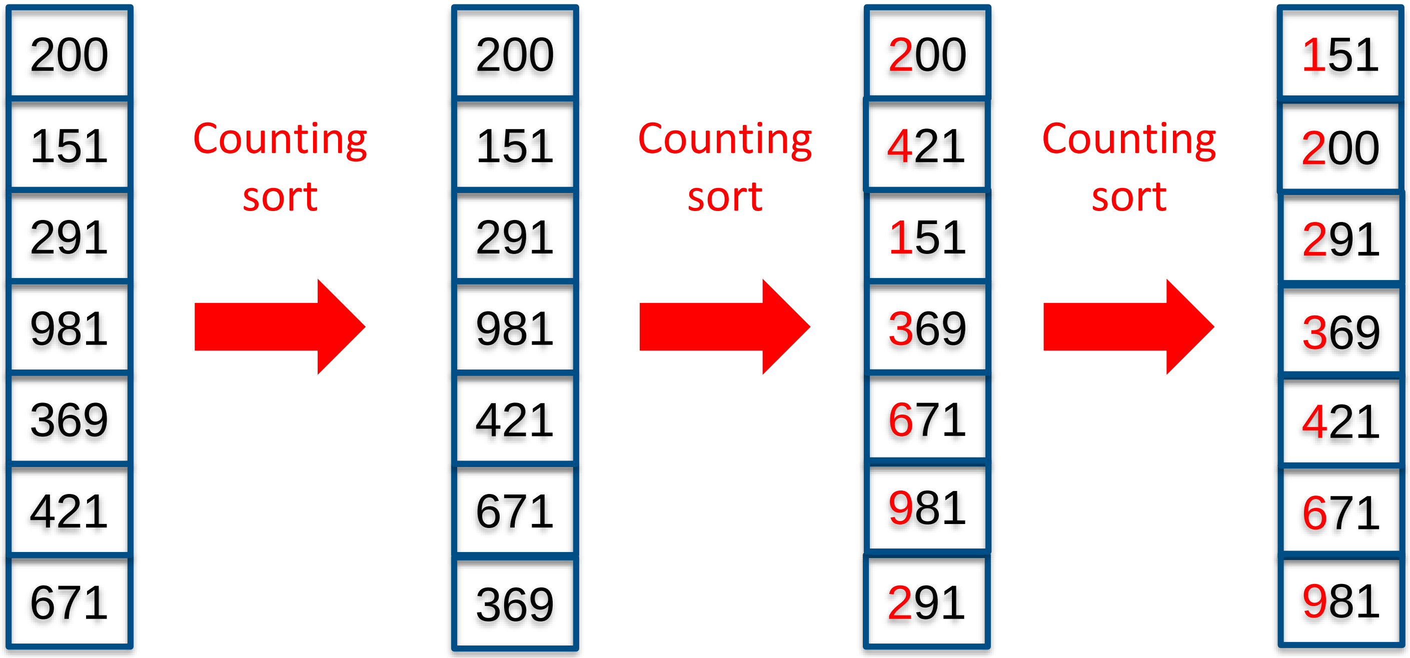


200	Counting sort	200	Counting sort	200	Counting sort
151		151		421	
291		291		151	
981		981		369	
369		421		671	
421		671		981	
671		369		291	

Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant

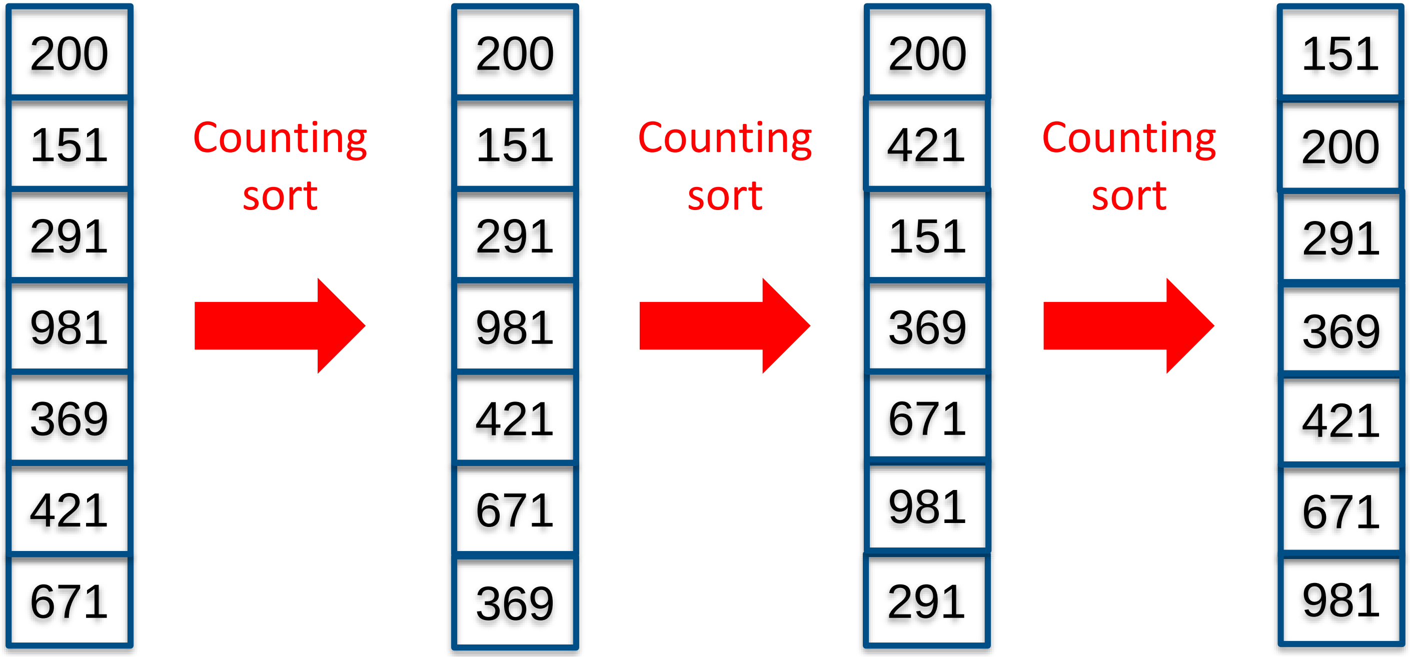


200		200		200		151
151	Counting sort	151	Counting sort	421	Counting sort	200
291		291		151		291
981		981		369		369
369		421		671		421
421		671		981		671
671		369		291		981

Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant



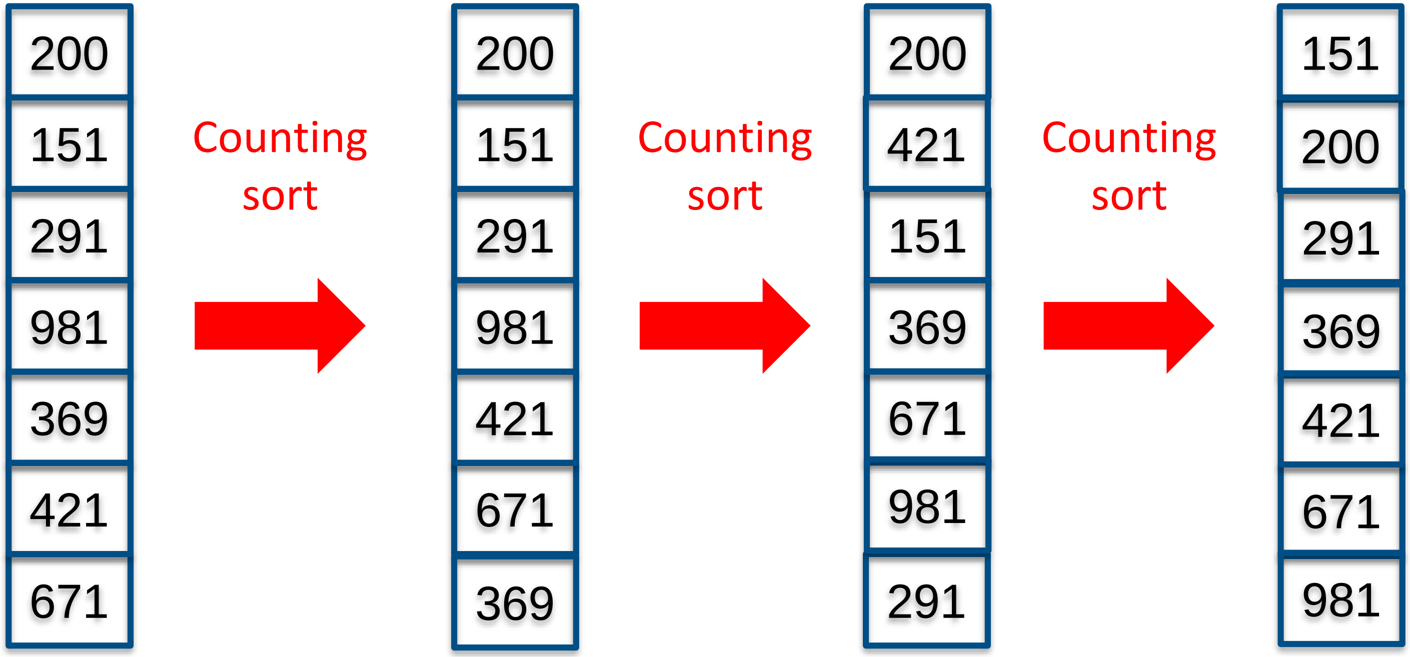
200		200		200		151
151	Counting sort	151	Counting sort	421	Counting sort	200
291		291		151		291
981		981		369		369
369		421		671		421
421		671		981		671
671		369		291		981

Questions?

Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?
  - But the sorting need to be stable



200
151
291
981
369
421
671

Counting sort

200
151
291
981
421
671
369

Counting sort

200
421
151
369
671
981
291

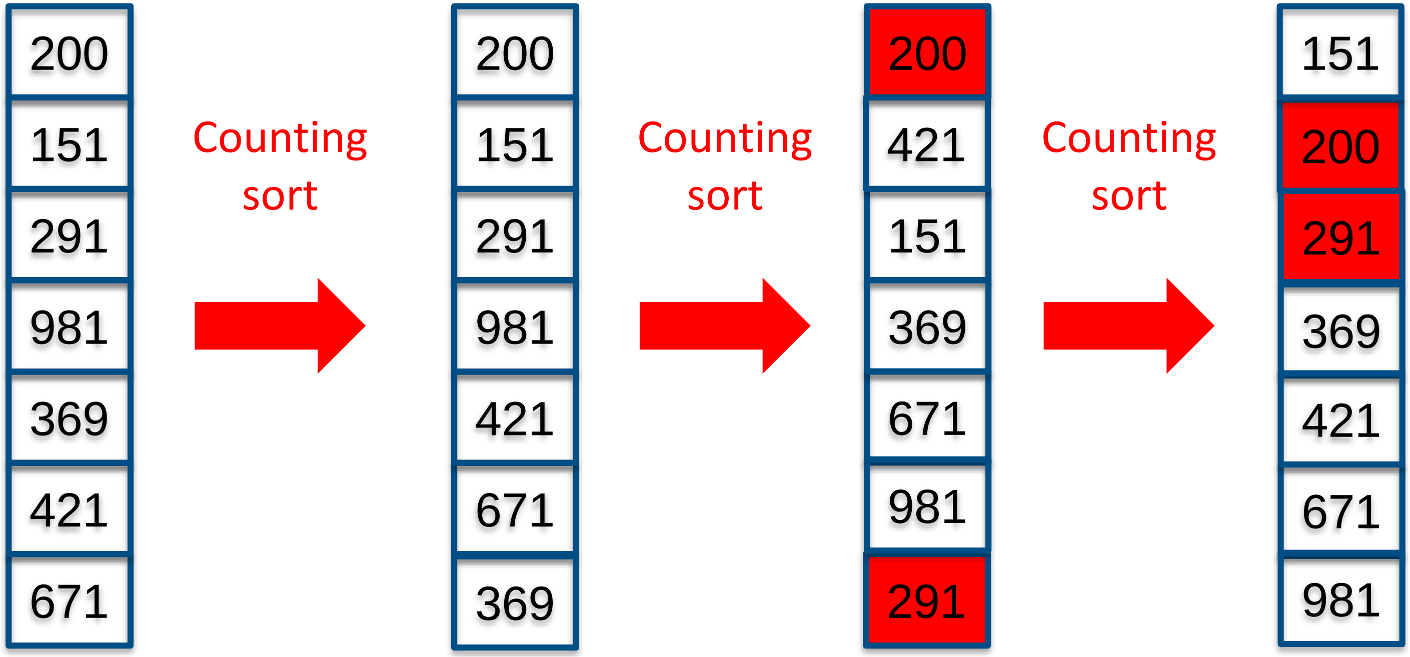
Counting sort

151
200
291
369
421
671
981

Radix Sort

A different outlook…

- With this input…
  - What if we view it differently? How would we sort it?
  - But the sorting need to be stable



200	200	200	151
151	151	421	200
291	291	151	291
981	981	369	369
369	421	671	421
421	671	981	671
671	369	291	981

Counting sort

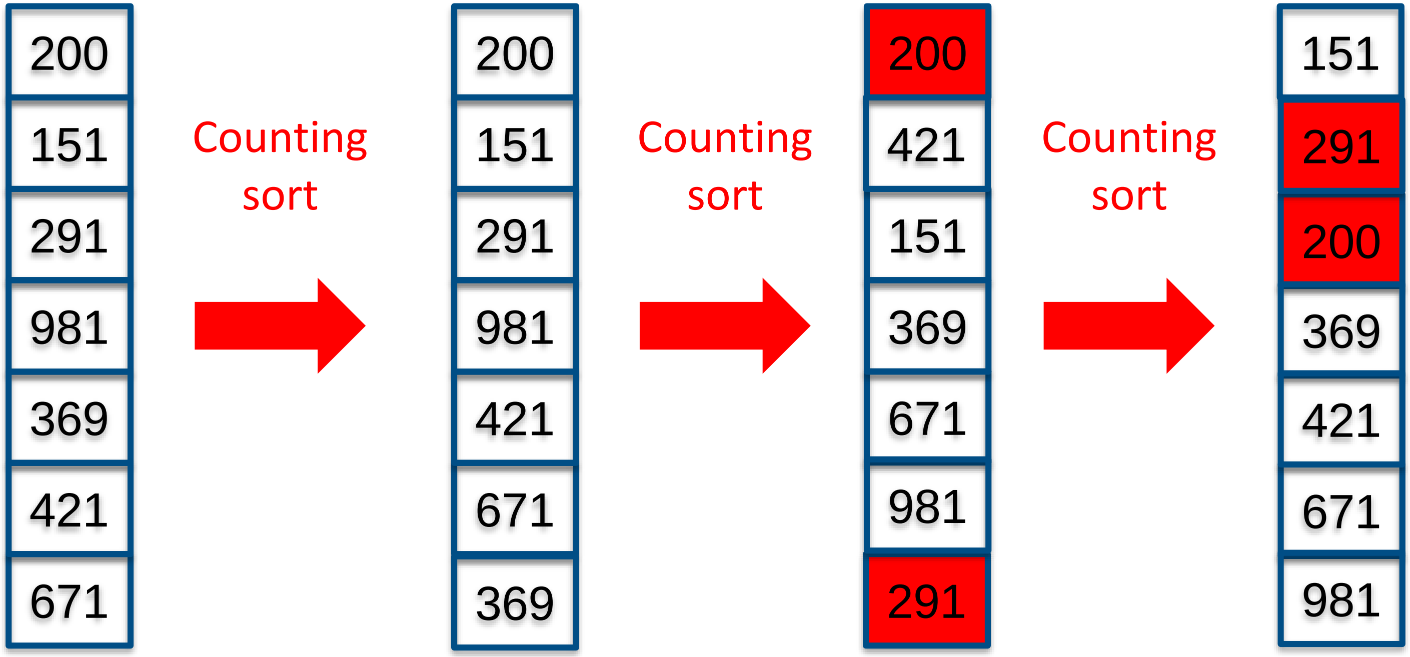
Counting sort

Counting sort

Radix Sort

A different outlook…

- With this input…
  – What if we view it differently? How would we sort it?
  – But the sorting need to be stable, if not…



200
151
291
981
369
421
671

Counting sort

200
151
291
981
421
671
369

Counting sort

200
421
151
369
671
981
291

Counting sort

151
291
200
369
421
671
981

It’s a disastah!

Questions?

Radix Sort

Complexity

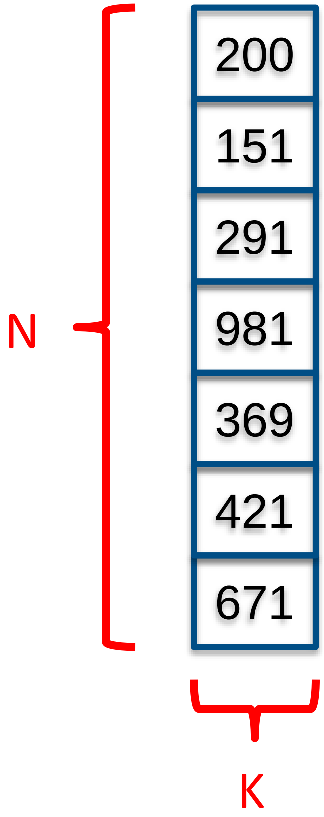
- What is the complexity?
  - Time
  - Space

200
151
291
981
369
421
671

Radix Sort

Complexity

- What is the complexity?
  - Time
  - Space



200

151

291

981

369

421

671

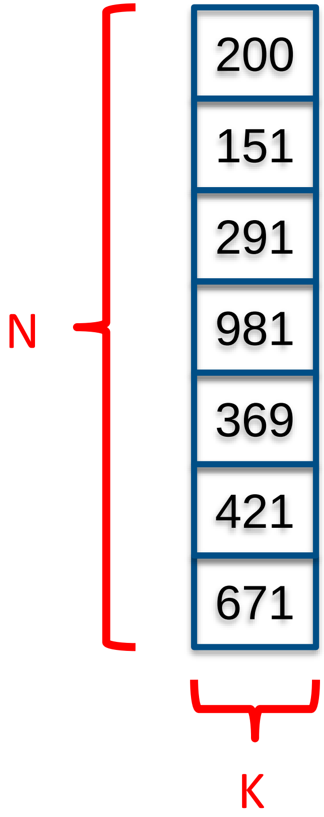
N

K

Radix Sort

Complexity

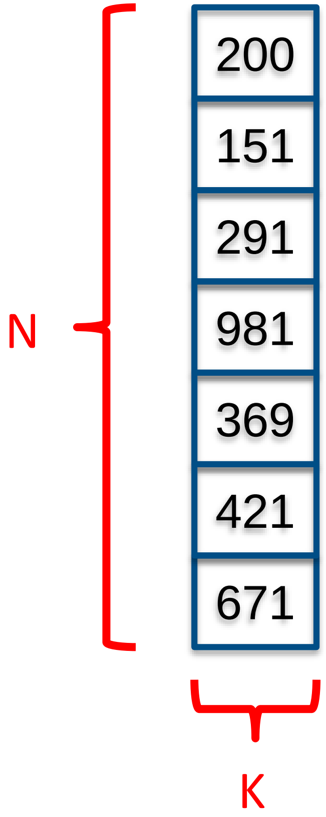
- What is the complexity?
  - Time
    - O(KN)?
  - Space



Radix Sort

Complexity

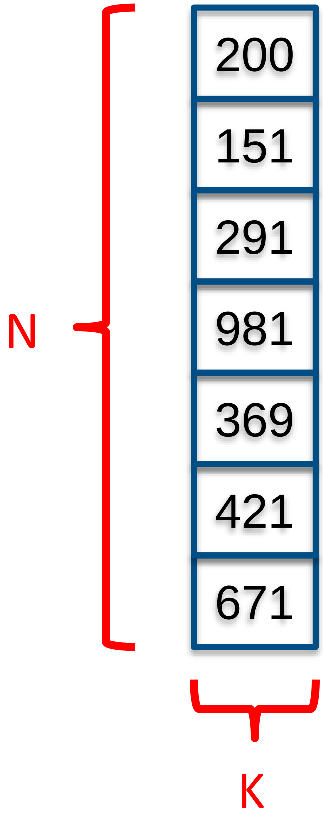
- What is the complexity?
  - Time
    - O(KN) + O(KM)
      where M is the number of unique characters
  - Space



Radix Sort

Complexity

- What is the complexity?
  - Time
    - O(KN) + O(KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max
  - Space

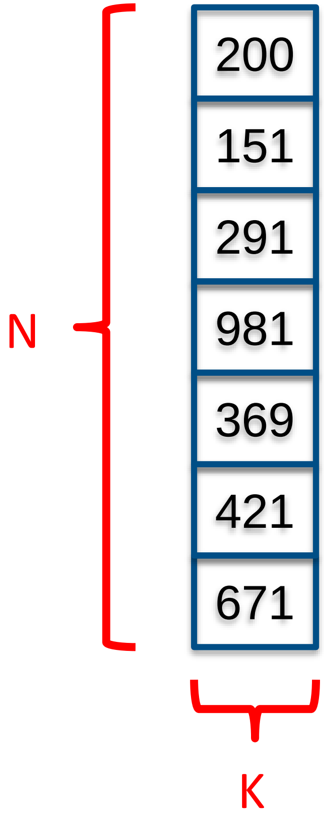


200
151
291
981
369
421
671

Radix Sort

Complexity

- What is the complexity?
  - Time
    - O(KN) + O(KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
  - Space

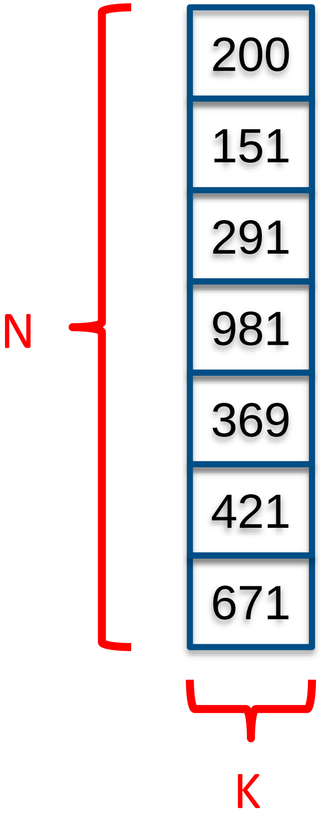


200
151
291
981
369
421
671

Radix Sort

Complexity

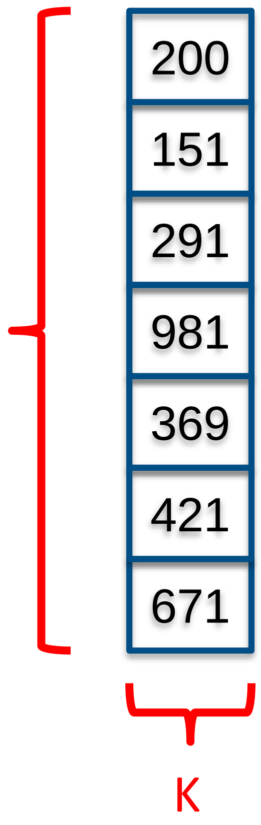
- What is the complexity?
  - Time
    - O(KN) + O(KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
  - Space



Radix Sort

Complexity

- What is the complexity?
  - Time
    - O(KN) + O(KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space



200

151

291

981

369

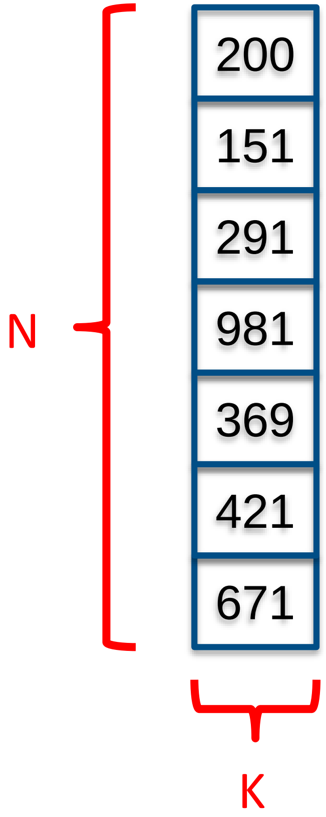
421

671

Radix Sort

Complexity

- What is the complexity?
  - Time
    - O(KN + KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space

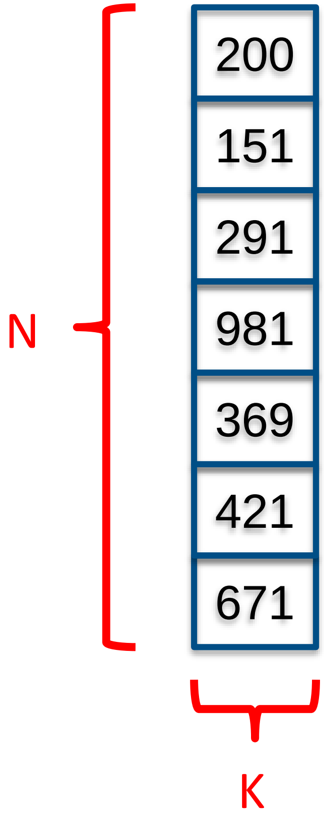


200
151
291
981
369
421
671

Radix Sort

Complexity

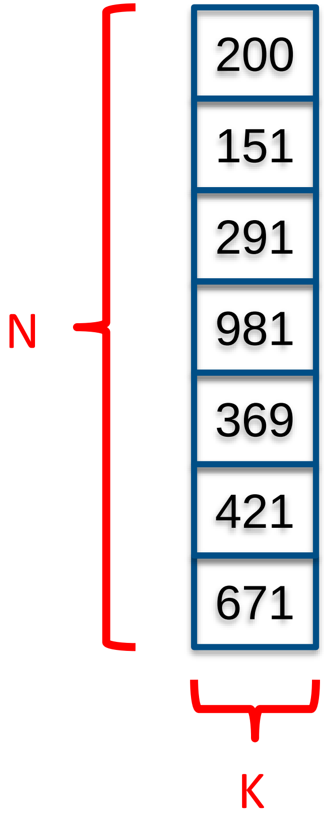
- What is the complexity?
  - Time
    - O(KN + KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space
    - Input is O(KN)
    - Each counting sort needs O(M+N)



Radix Sort

Complexity

- What is the complexity?
  - Time
    - O(KN + KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space
    - Input is O(KN)
    - Each counting sort needs O(M+N)
    - Total is O(KN + M + N)



N

200

151

291

981

369

421

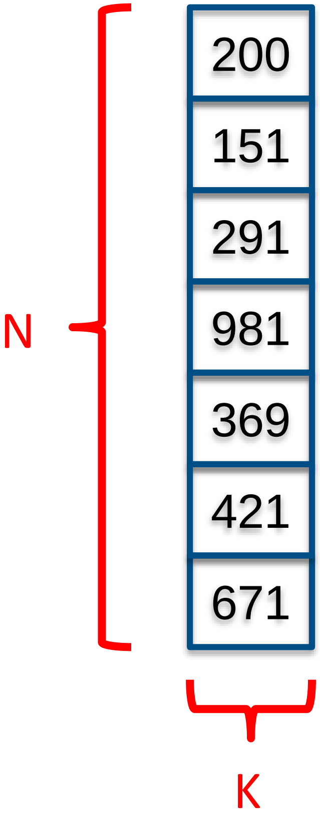
671

K

Radix Sort

Complexity

- What is the complexity?
  - But we know M = 10 for 0, 1, …, 9
  - Time
    - O(KN + KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns giving us O(K) * O(N+M)
  - Space
    - Input is O(KN)
    - Each counting sort needs O(M+N)
    - Total is O(KN + M + N)



200

151

291

981

369

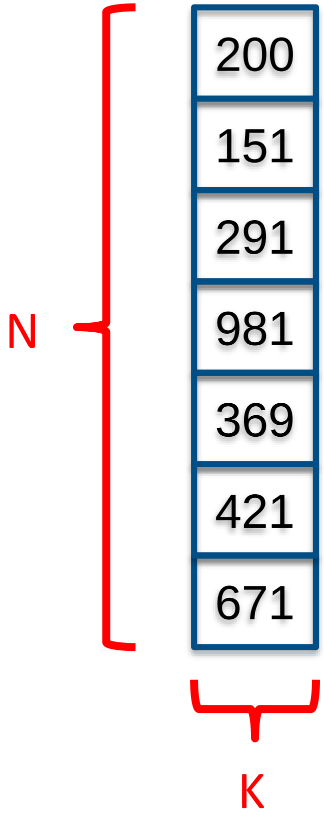
421

671

Radix Sort

Complexity

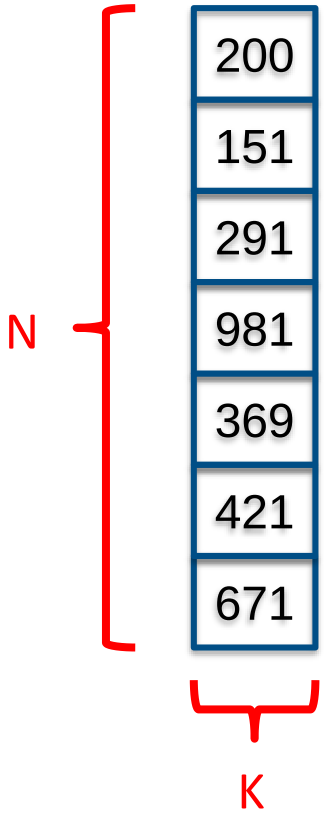
- What is the complexity?
  - But we know M = 10 for 0, 1, …, 9
  - Time
    - O(KN + KM) ≈ O(KN)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space
    - Input is O(KN)
    - Each counting sort needs O(M+N)
    - Total is O(KN + M + N) ≈ O(KN)
    - Auxiliary is O(M + N) ≈ O(N)



Radix Sort

Complexity

- What is the complexity?
  - Better than merge sort O(k N log N)!
  - But we know M = 10 for 0, 1, …, 9
  - Time
    - O(KN + KM) ≈ O(KN)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space
    - Input is O(KN)
    - Each counting sort needs O(M+N)
    - Total is O(KN + M + N) ≈ O(KN)
    - Auxiliary is O(M + N) ≈ O(N)



N

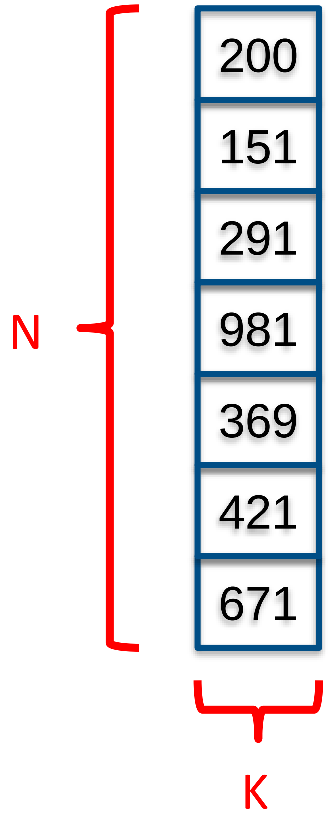
200
151
291
981
369
421
671

K

Radix Sort

Complexity

- What is the complexity?
  - Better than merge sort O(k N log N)!
  - But we know M = 10 for 0, 1, …, 9
  - Time
    - O(KN + KM) ≈ O(KN)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space
    - Input is O(KN)
    - Each counting sort needs O(M+N)
    - Total is O(KN + M + N) ≈ O(KN)
    - Auxiliary is O(M + N) ≈ O(N) <- why no K? Come ask me if interested…



200

151

291

981

369

421

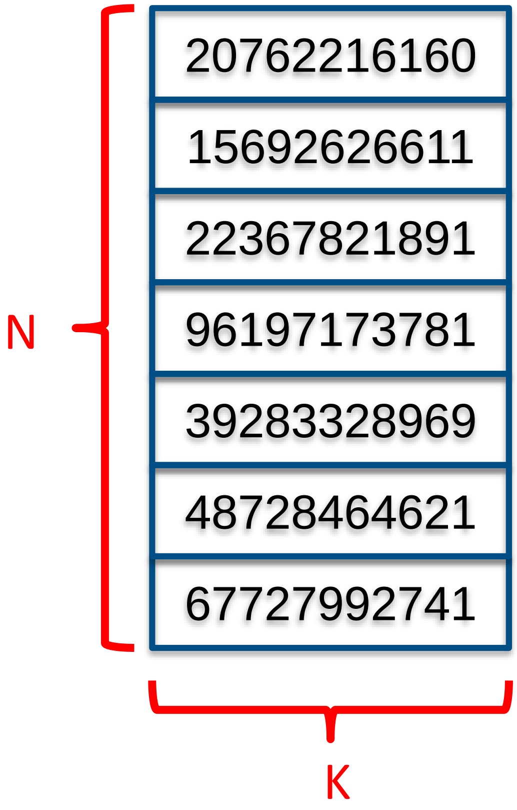
671

Questions?

Radix Sort

Complexity

- What is the complexity?
  - What if k is bigger?
  - But we know M = 10 for 0, 1, …, 9
  - Time
    - O(KN + KM) ≈ O(KN)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space
    - Input is O(KN)
    - Each counting sort needs O(M+N)
    - Total is O(KN + M + N) ≈ O(KN)
    - Auxiliary is O(M + N) ≈ O(N)



20762216160

15692626611

22367821891

96197173781

39283328969

48728464621

67727992741

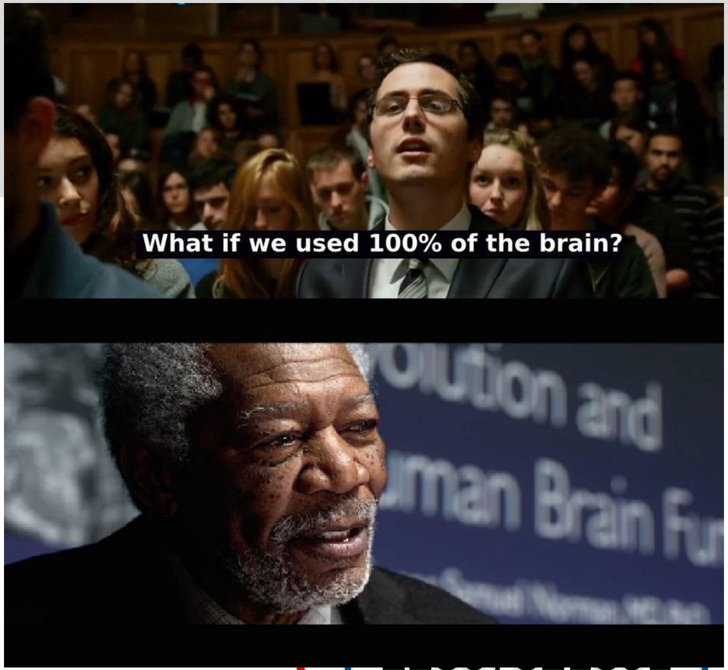
N

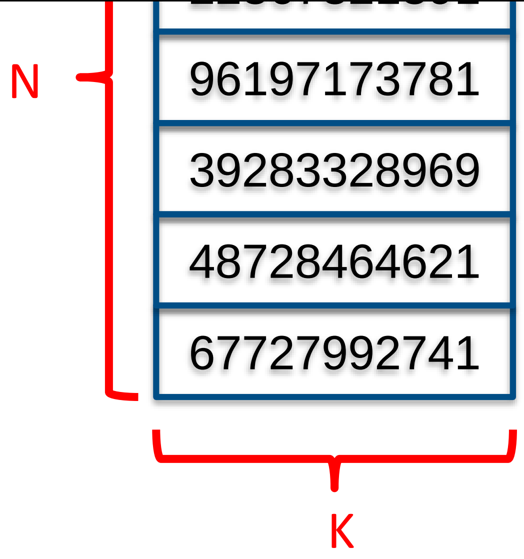
K

Radix Sort

Complexity

- What is the complexity?
  - What if k is bigger?
  - But we know M = 10 for 0, 1, …, 9
  - Time
    - O(KN + KM) ≈ O(KN)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space
    - Input is O(KN)
    - Each counting sort needs O(M+N)
    - Total is O(KN + M + N) ≈ O(KN)
    - Auxiliary is O(M + N) ≈ O(N)





96197173781

39283328969

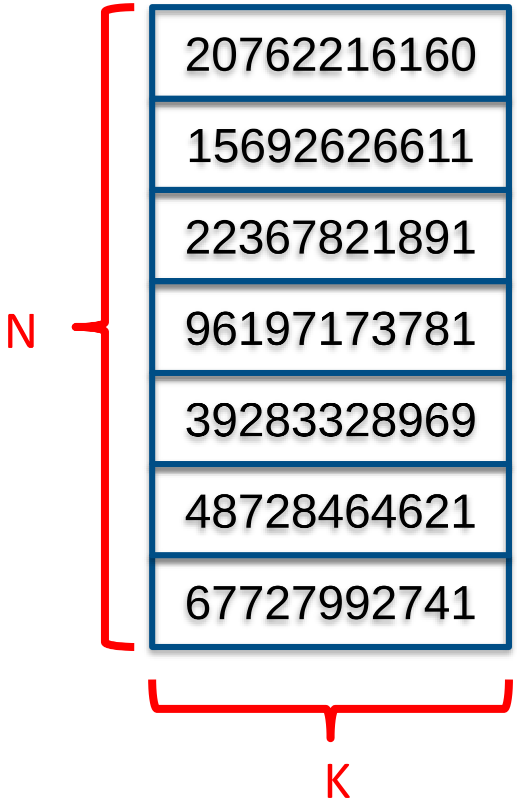
48728464621

67727992741

Radix Sort

Complexity

- What is the complexity?
  - What if k is bigger?
  - We increase M = 100 for 0, 1, …, 99
  - Time
    - O(KN + KM) ≈ O(KN)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space
    - Input is O(KN)
    - Each counting sort needs O(M+N)
    - Total is O(KN + M + N) ≈ O(KN)
    - Auxiliary is O(M + N) ≈ O(N)



20762216160
15692626611
22367821891
96197173781
39283328969
48728464621
67727992741

Radix Sort

Complexity

- Time complexity is O(KN +KM)
- Space complexity is O(KN + M + N)

Radix Sort

Complexity

- Time complexity is O(KN +KM)
- Space complexity is O(KN + M + N)

- M is the base

Radix Sort

Complexity

- Time complexity is O(KN +KM)
- Space complexity is O(KN + M + N)

- M is the base
  - For decimal numbers, it is 10 from 0 to 10

Radix Sort

Complexity

- Time complexity is O(KN +KM)
- Space complexity is O(KN + M + N)

- M is the base
  - For decimal numbers, it is 10 from 0 to 10
  - For binary numbers,

Radix Sort

Complexity

- Time complexity is O(KN +KM)
- Space complexity is O(KN + M + N)

- M is the base
  - For decimal numbers, it is 10 from 0 to 10
  - For binary numbers, it is 2 from 0 to 1

Radix Sort

Complexity

- Time complexity is O(KN +KM)
- Space complexity is O(KN + M + N)

- M is the base
  - For decimal numbers, it is 10 from 0 to 10
  - For binary numbers, it is 2 from 0 to 1
  - We can increase the M, to reduce the K?

Radix Sort

Complexity

- Time complexity is O(KN +KM)
- Space complexity is O(KN + M + N)

- M is the base
  - For decimal numbers, it is 10 from 0 to 10
  - For binary numbers, it is 2 from 0 to 1
  - We can increase the M, to reduce the K?

  - If we deal with the English alphabet, this would be 26 from a to z

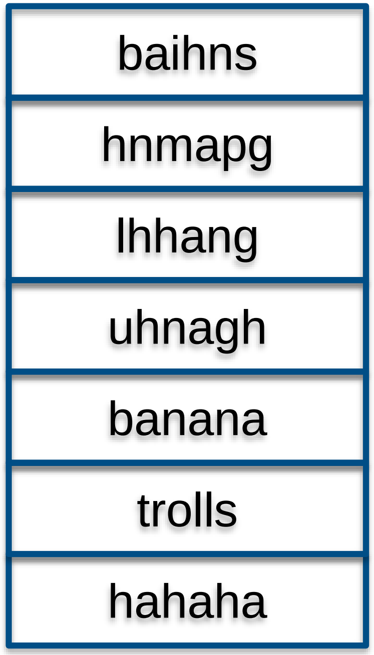
baihns
hnmapg
lhhang
uhnagh
banana
trolls
hahaha

Radix Sort

Complexity

- Time complexity is O(KN +KM)
- Space complexity is O(KN + M + N)

- M is the base
  - For decimal numbers, it is 10 from 0 to 10
  - For binary numbers, it is 2 from 0 to 1
  - We can increase the M, to reduce the K?



baihns

hnmapg

lhhang

uhnagh

banana

trolls

hahaha

  - If we deal with the English alphabet, this would be 26 from a to z

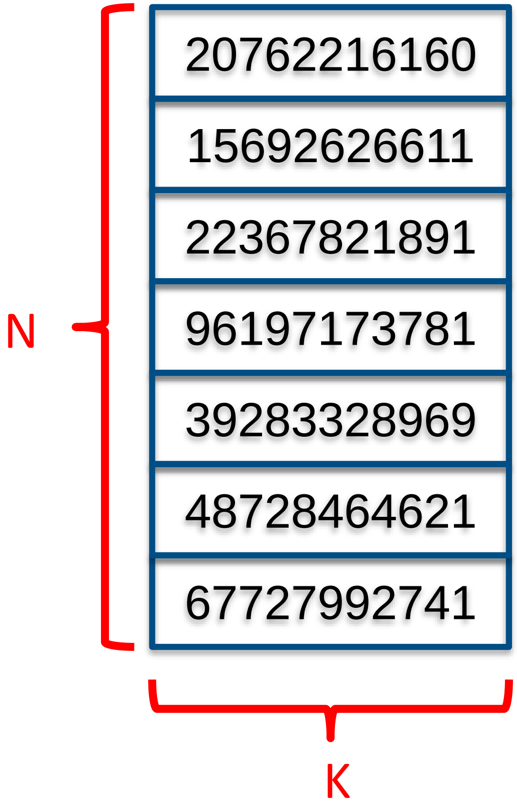
  - Nathan did a good analysis on it

Questions?

Radix Sort

TL;DR

- So you know radix sort
- What have you notice?



N

20762216160

15692626611

22367821891

96197173781

39283328969

48728464621

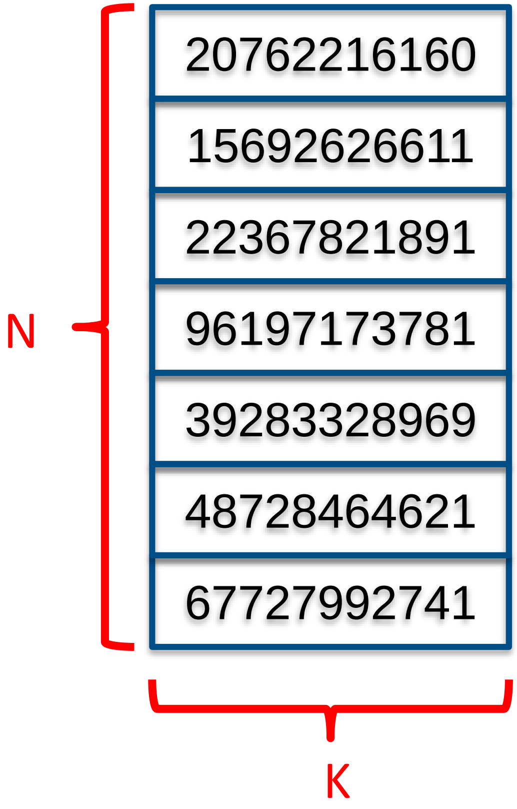
67727992741

K

Radix Sort

TL;DR

- So you know radix sort
- What have you notice?
  - It is counting sort really, done multiple times



20762216160

15692626611

22367821891

96197173781

39283328969

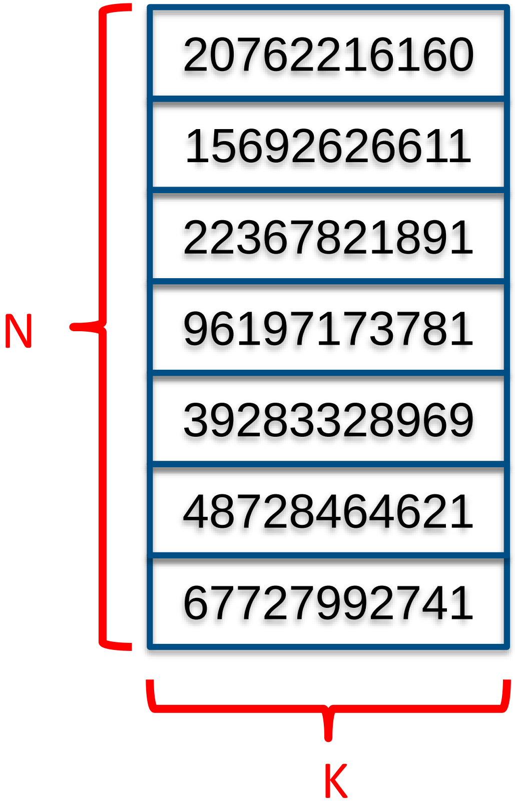
48728464621

67727992741

Radix Sort

TL;DR

- So you know radix sort
- What have you notice?
  - It is counting sort really, done multiple times
  - Usually least significant (right) to most significant (left)

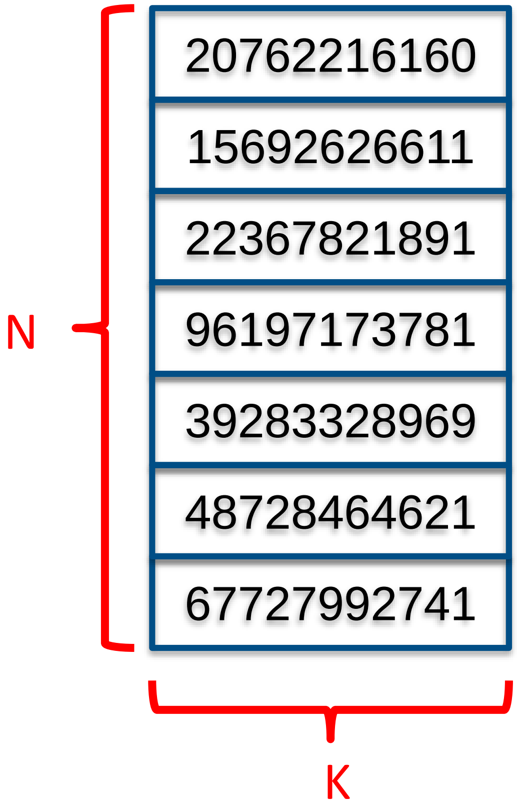


20762216160
15692626611
22367821891
96197173781
39283328969
48728464621
67727992741

Radix Sort

TL;DR

- So you know radix sort
- What have you notice?
  - It is counting sort really, done multiple times
    - We can reduce this by increasing the base
  - Usually least significant (right) to most significant (left)



20762216160

15692626611

22367821891

96197173781

39283328969

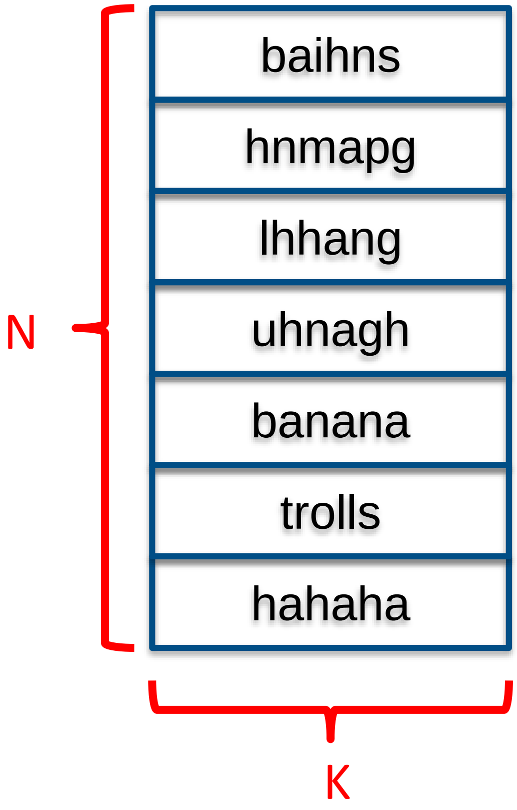
48728464621

67727992741

Radix Sort

TL;DR

- So you know radix sort
- What have you notice?
  - It is counting sort really, done multiple times
    - We can reduce this by increasing the base
    - Works well for characters as well
  - Usually least significant (right) to most significant (left)



baihns

hnmapg

lhhang

uhnagh

banana

trolls

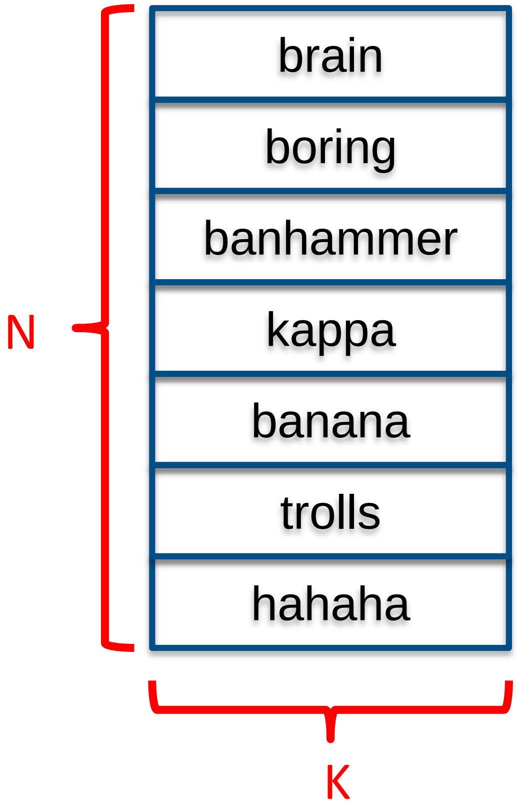
hahaha

Radix Sort

TL;DR

- So you know radix sort
- What have you notice?
  - It is counting sort really, done multiple times
    - We can reduce this by increasing the base
    - Works well for characters as well
  - Usually least significant (right) to most significant (left)

- But what if they are not the same length?

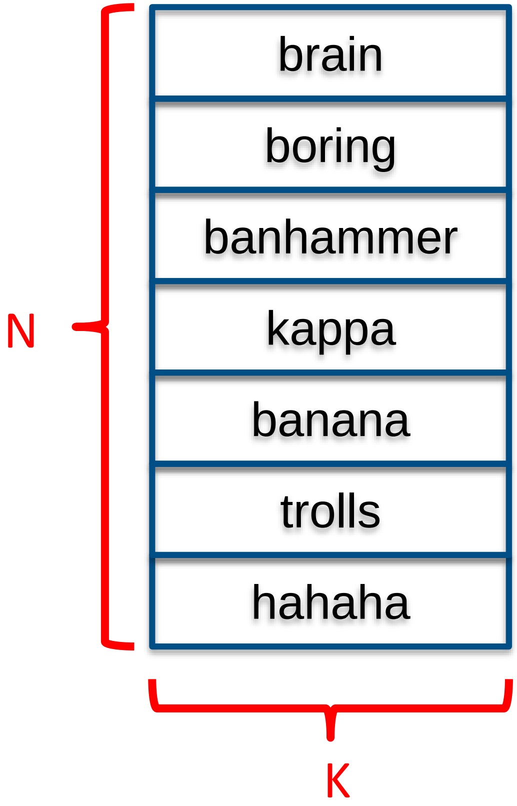


Radix Sort

TL;DR

- So you know radix sort
- What have you notice?
  - It is counting sort really, done multiple times
    - We can reduce this by increasing the base
    - Works well for characters as well
  - Usually least significant (right) to most significant (left)

- But what if they are not the same length?
  - Left-aligned?
  - Right-aligned?



brain

boring

banhammer

kappa

banana

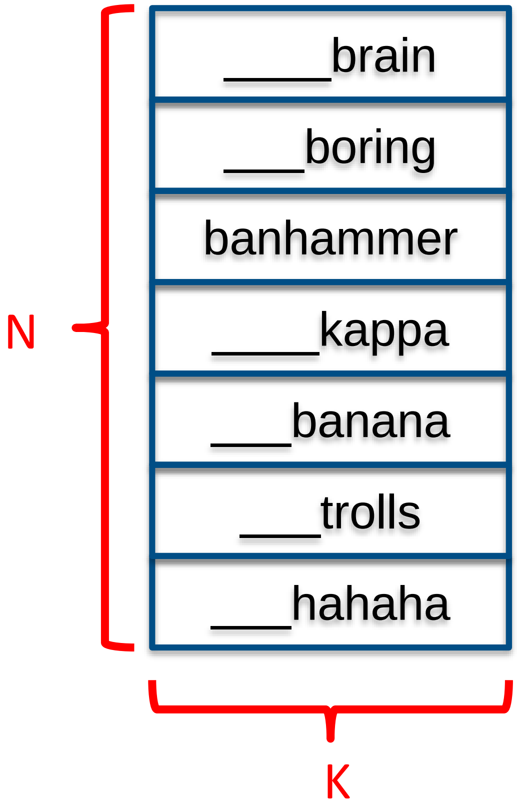
trolls

hahaha

Radix Sort

TL;DR

- So you know radix sort
- What have you notice?
  - It is counting sort really, done multiple times
    - We can reduce this by increasing the base
    - Works well for characters as well
  - Usually least significant (right) to most significant (left)
- But what if they are not the same length? Add spaces!
  - Left-aligned?
  - Right-aligned?



____brain

___boring

banhammer

____kappa

___banana

___trolls

___hahaha

N

K

Questions?

Thank You