MONASH University

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

GROUP OF EIGHT AUSTRALIA

Faculty of Information Technology, Monash University

Ready?

Agenda

- Complexity Analysis
  - Time
  - Space

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

- Now let us have some recap with some functions

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

- Consider the code
- What is the time complexity?

```
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

```
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…

```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
```

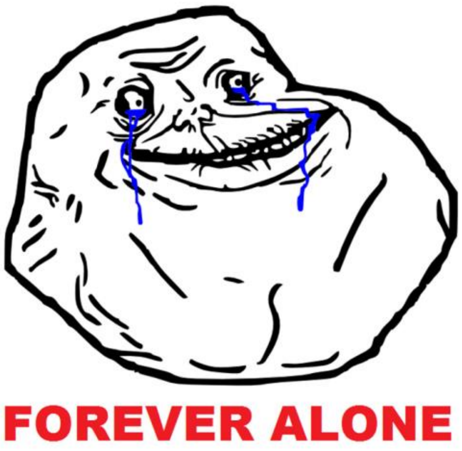
Complexity

Minimum

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

- 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

- 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

```
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)

```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")
```

- 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")
```

- 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
  - 1^{st} iteration = N/2
  - 2^{nd} 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
  - 1^{st} iteration = N/2
  - 2^{nd} 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")
```

- 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

```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")
```

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

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

- 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….

- 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…

- 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.

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.

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

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

- These are all comparison based
- $\Omega(N \log N)$

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

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

Questions?

Complexity

Time – Output Sensitive

- What is it?

- 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?

- 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

- 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

- 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

- 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?

- 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

- 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!

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

- 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!

Questions?

- What is it?

Space

- How much space is used

Complexity

Space

- How much space is used

- Consider our functions earlier…

- How much space is used

- Consider our functions earlier…

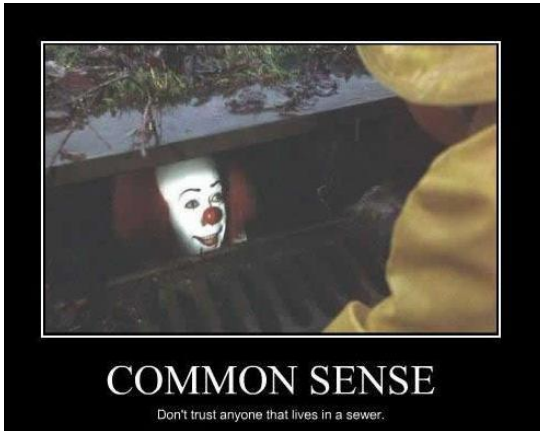
```
def find_minimum(my_list):
    minimum = None
    for i in range(0, len(my_list)):
        if minimum is None:
            minimum = my_list[i]
        else:
            if minimum > my_list[i]:
                minimum = my_list[i]
    return minimum
```

```
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")
```

- How much space is used

- Consider our functions earlier…

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




```
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")
```

Questions?

- 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?

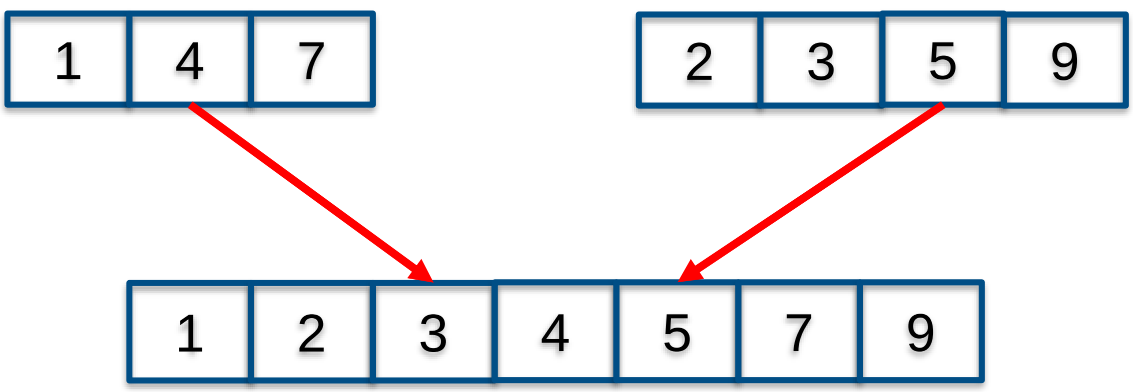
4	7	1

3	2	9	5

Complexity
Auxiliary Space

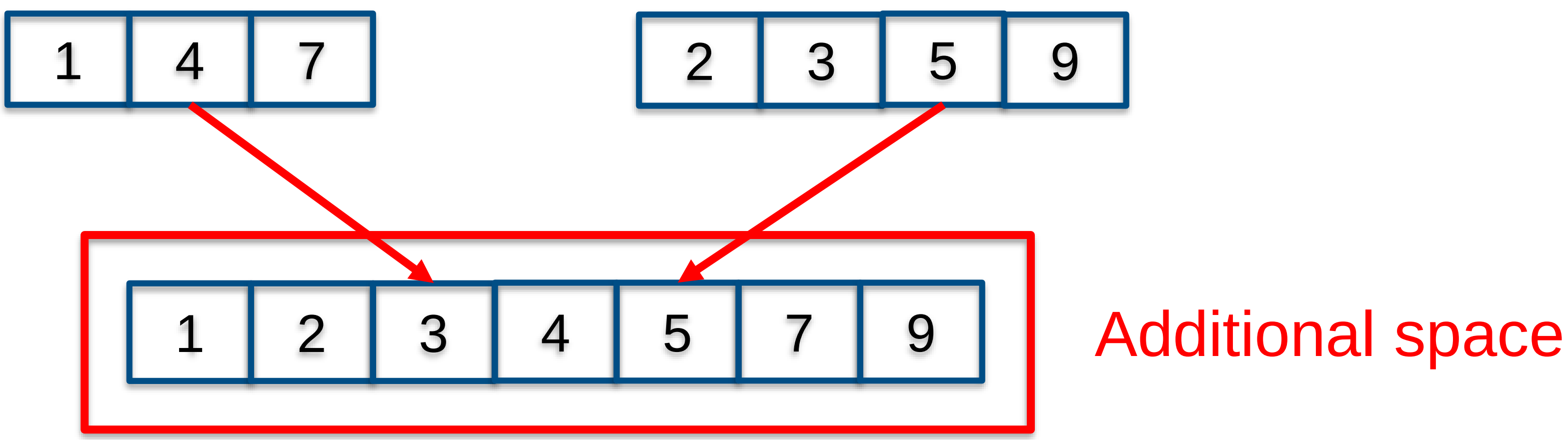
- What is this now then?
- Additional space required in addition to the input

- Remember the merge sort's merge operation?



MONASH University

- What is this now then?
- Additional space required in addition to the input
- Remember the merge sort's merge operation?



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

- 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")
```

- 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")
```

Questions?

MONASH University

Thank You