CS2040 – Data Structures and Algorithms

Final Lecture – Revision

rogerz@comp.nus.edu.sg

NUS National University of Singapore
School of Computing

Consultation Slots

- Enzio Kam
  - Thursday, 21 November 2-4 pm
  - Room: TBD
- Roger Zimmermann
  - Friday, 22 November 2-4 pm
  - Room: TBD

  - Please be courteous to your Lab TAs. They are all undergrad students and have exams too.

Assessments: Overview

Activities	Weightages
Tutorial + attendance/participation	3%
Lab attendance	2%
In-lab Assignments	15% (1.5%/problem)
Take-home Assignments	12% (1.5%/problem)
Midterm (open book+calculator)	20%
2 Online Quizzes (open book+calculator)	8% (4% each)
Final Exam (open book+calculator)	40%

– Open Book = allowed to bring in lectures notes, tutorials, quizzes, reference books or any piece of paper you want **<u>but no internet</u>** and **<u>no offline (i.e., local) LLMs</u>!**

Final Exam

Final Exam Scopes

- Entire semester with emphasis on the second half (10% vs 90%)
  - Lecture notes
  - Tutorials and labs

Final Exam Paper Format

- Exam Date: **Wednesday, 27 Nov 2024**
- Duration: **2 hours (5pm to 7pm)**
- Arrive: at **4:45pm**
- Format: similar to Midterm; Examplify on your laptop
- Venue: MPSH1-A and MPSH1-B
  - If you are not present at the exam venue(s) you will get 0 for the exam.

- **Open Book** examination (hard copy and PDFs)
- No offline, local LLM!

Final Exam …

- Please bring a laptop that is fully charged and can last, say, 2.5 hours on Examplify.
- A few charging seats are available.

Review & Data Structures with Multiple Organizations

Week 13 Mix and Match

Basic Data Structures

- Arrays
- Linked Lists
- Map ADT (Hash Table)
- Stacks and Queues
- Trees

▶ We can combine them to implement different data structures for different applications.

Analysis of Array Impl^n of List ADT

- Time complexity of the different list operations
  - Retrieval: *getItemAtIndex(int i), getFirst(), getLast()*
    - O(1) – indexing into an array is constant time due to random access memory of the computer
  - Insertion: *addItemAtIndex(int i, int item), addFront(), addBack()*
    - Best case = O(1) – if adding at the back and no need to enlarge array
    - Worst case = O(n) – if adding to the front due to shifting all item to the right or adding to the back but need to enlarge the array so have to perform copying of n items to new array
    - Amortized analysis (adding at the back) = ? (find out during lecture)
    - Average case = O(n) – on average need to shift ½(n) items to the right
  - Deletion: *removeItemAtIndex(int i), removeFront(), removeBack()*
    - Best case = O(1) – if removing from the back
    - Worst case = O(n) – if removing from the front due to shifting all items to the left
    - Average case = O(n) – on average need to shift ½(n) items to the left

Analysis of Linked List Impl^{n} of List ADT

- Time complexity of the different list operations
  - Retrieval: *getItemAtIndex(int i), getFirst(), getLast()*
    - Best case = O(1) – accessing the first node, return the head
    - Worst case = O(n) – accessing the last node, since you need to move all the way to the back from the head (n moves)
    - Average case = O(n) – need to move about half way through the list to access any node on average so ½(n) iterations of the for loop
  - Insertion: *addItemAtIndex(int i, int item), addFront(), addBack()*
    - Best case = O(1) – if adding at the front (don't have to worry about enlarging the list unlike array)
    - Worst case = O(n) – if adding to the back due to having to move all the way to the back from the head (n moves)
    - Average case = O(n) – on average need to make ½(n) moves

Analysis of Linked List Impl^n of List ADT

- Deletion: *removeItemAtIndex(int i), removeFront(), removeBack()*
  - Best case = O(1) – if removing from the front
  - Worst case = O(n) – if removing from the back, again due to moving all the way to the back from the head
  - Average case = O(n) – on average need to make ½(n) moves

- What about the Space Complexity?
  - We use as much space as there are nodes in the list so exactly O(n) (plus some constant overhead to store each node which requires more space then a simple integer that is stored in an array)

Map ADT Operations

	Sorted List (Array impl. by sorting key)	**Balanced BST**	**HashTable**
Insert	O(*n*)	O(log *n*)	O(1) avg
Delete	O(*n*)	O(log *n*)	O(1) avg
Find	O(log *n*)	O(log *n*)	O(1) avg

Note: Balanced Binary Search Tree (bBST) will be covered in later lectures.

- Hence, hash table supports the Map ADT in constant time on average for the above operations. It has many applications.

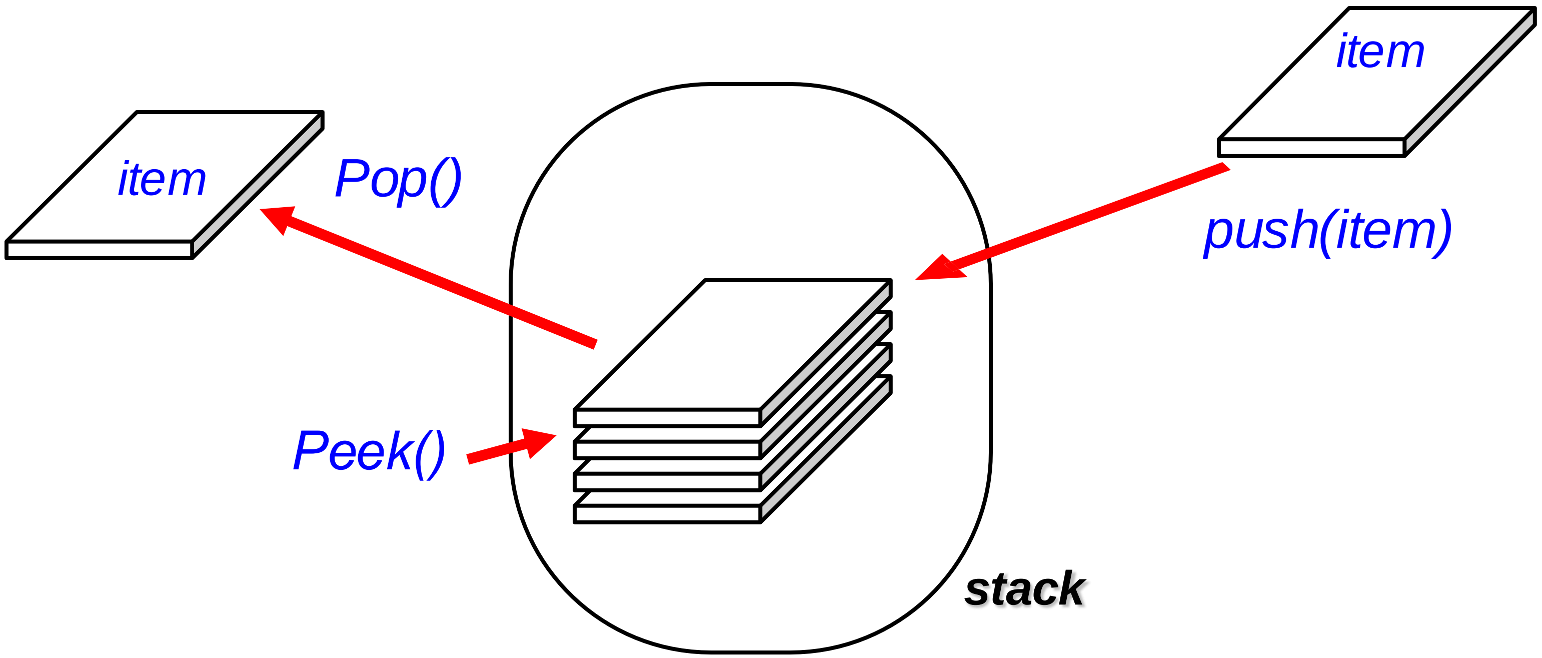
Map ADT – HashTable Summary

- How to hash? Criteria for good hash functions?
- How to resolve collision?

  Collision resolution techniques:
  - separate chaining
  - linear probing
  - quadratic probing
  - double hashing
- Problem on deletions
- Primary clustering and secondary clustering.

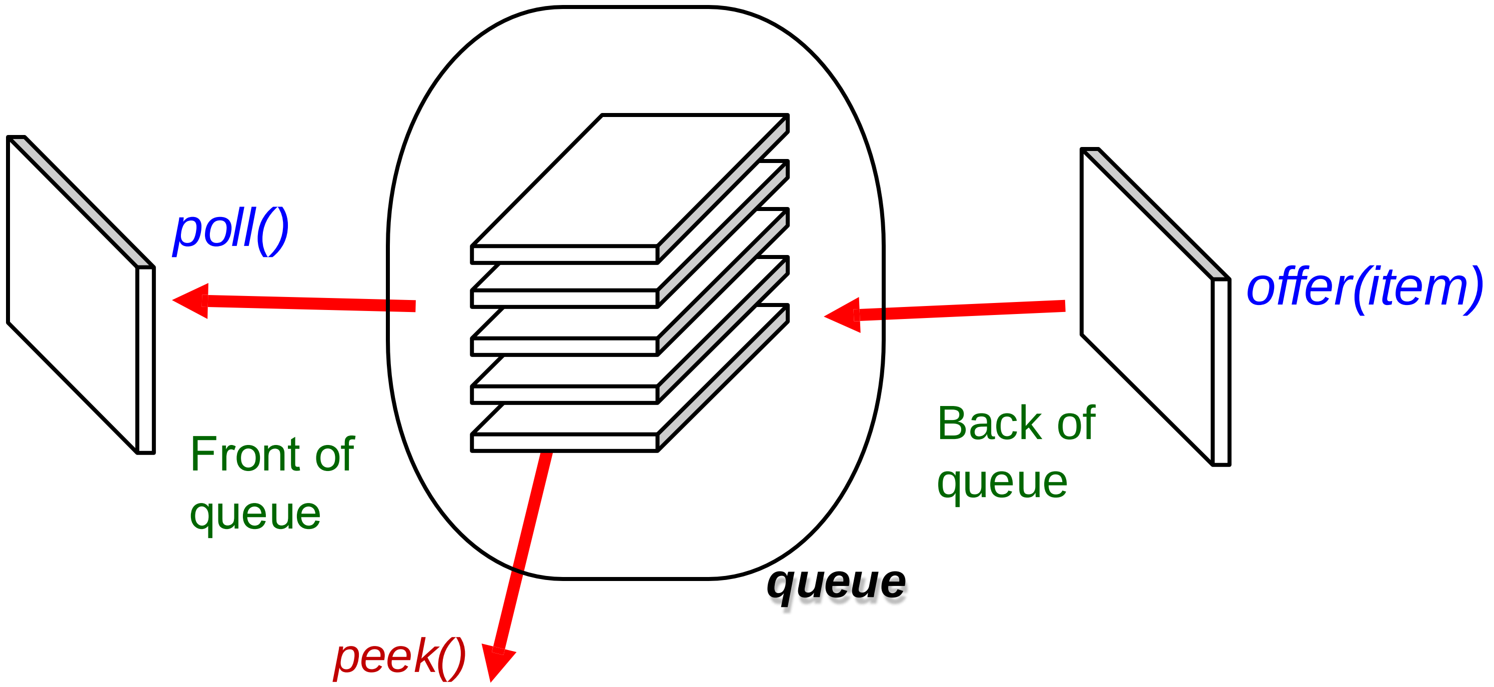
Stack ADT: Operations

- A Stack is a collection of data that is accessed in a last-in-first-out (LIFO) manner
- Major operations: "push", "pop", and "peek".



Queue ADT: Operations

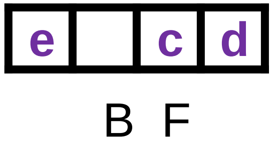
- A Queue is a collection of data that is accessed in a first-in-first-out (FIFO) manner
- Major operations: "poll" (or "dequeue"), "offer" (or "enqueue"), and "peek".



Stacks & Queues Summary

- We learn to create our own data structures from array and linked list
  - LIFO vs FIFO – a simple difference that leads to very different applications
  - Drawings can often help in understanding the different cases for operations on the Stack and Queue
- Stacks and Queues can be implemented with either arrays or linked lists.
- Queues can be circular.

Need to distinguish full from empty.



Full Case: (((B+1) % maxsize) == F)
Empty Case: F == B

Sorting Algorithms

- *Comparison based and Iterative algorithms*

1. Selection Sort
2. Bubble Sort
3. Insertion Sort

- *Comparison based and Recursive algorithms*

4. Merge Sort
5. Quick Sort

- *Non-comparison based*

6. Radix Sort

7. Comparison of Sort Algorithms
   - In-place sort
   - Stable sort
8. Use of Java Sort Methods

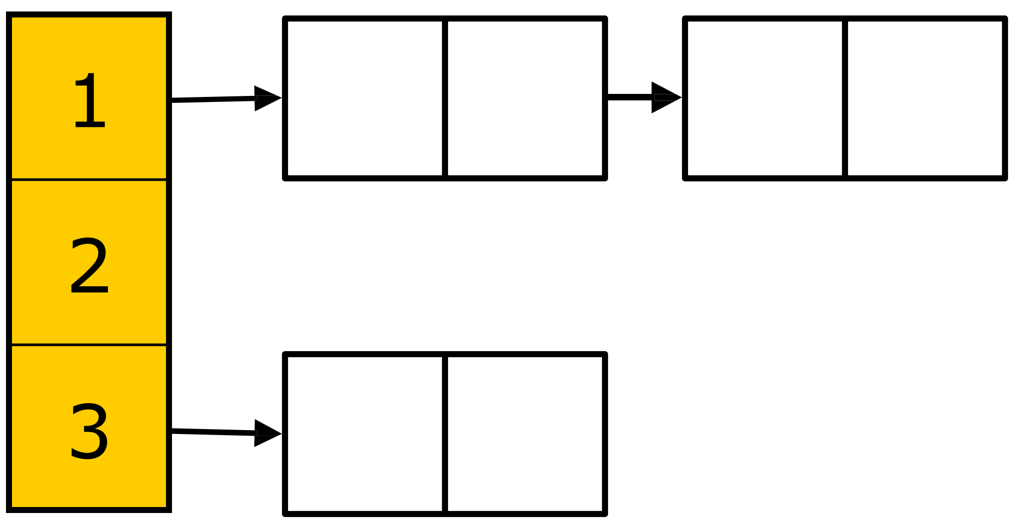
Summary of Sorting Algorithms

	Worst Case	Best Case	In-place?	Stable?
Selection Sort	$O(n^2)$	$O(n^2)$	Yes	No
Insertion Sort	$O(n^2)$	$O(n)$	Yes	Yes
Bubble Sort	$O(n^2)$	$O(n^2)$	Yes	Yes
Bubble Sort 2 (improved with flag)	$O(n^2)$	$O(n)$	Yes	Yes
Merge Sort	$O(n \log n)$	$O(n \log n)$	No	Yes
Radix Sort (non-comparison based)	$O(n)$ (see Notes 1)	$O(n)$	No	Yes
Quick Sort	$O(n^2)$	$O(n \log n)$	Yes	No

Notes: 1. **$O(n)$** for Radix Sort is due to non-comparison based sorting.
2. **$O(n \log n)$** is the <u>best possible</u> for comparison based sorting.

Mix-and-Match

- Array of Linked-Lists
  - E.g.: Adjacency list for representing graph
  - E.g.: Hash table with separate chaining



Problem

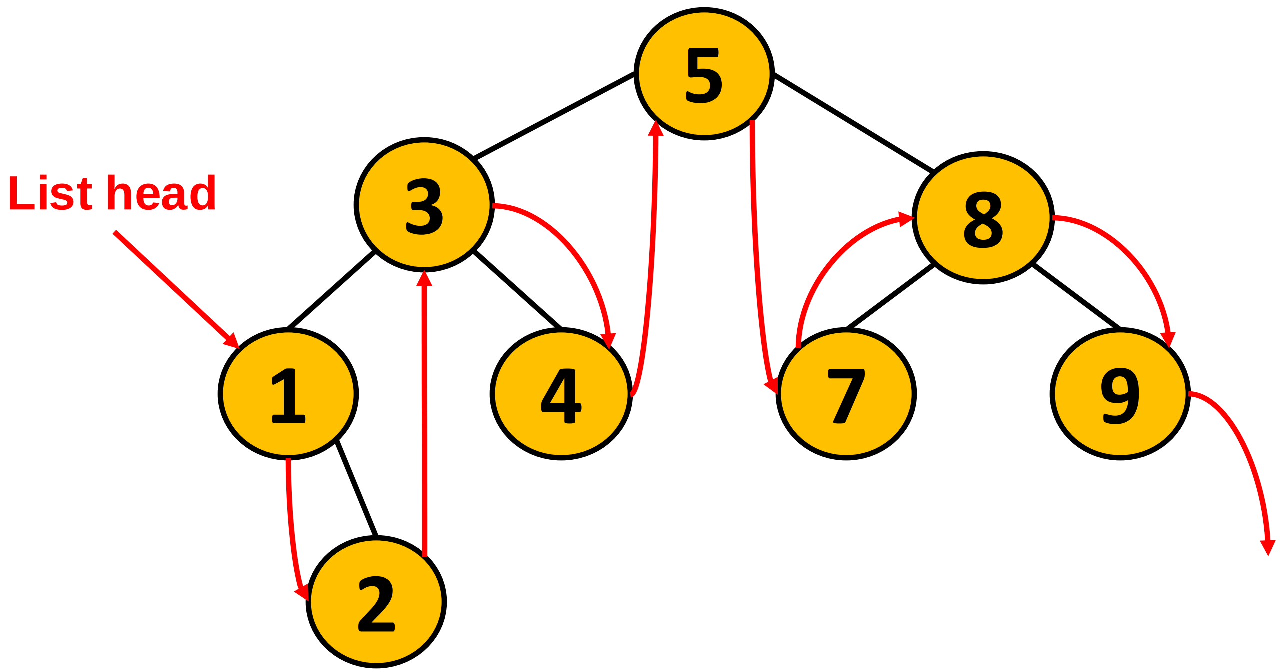
- Searching on an unsorted linked list is always O(n)
- How to improve it to O(1)?

Use hashing.

(i, j) as key and the hash value returned by hash function to be index to a hash table where (i, j) is stored together with the reference to the node in the linked list.

Mix-and-Match 2

- Binary Search Tree + Linked-List
- Can find the successors easily



Q: How to handle updates?

More Examples

- Suppose we need an ADT that support the following operations
  - enqueue(item)
  - dequeue()
  - peek()
  - printInOrder()

Use a Queue

- If we use a queue, we can support the queue operations efficiently O(1).
- But to print the items in order, we need to first sort the items in the queue, which is O(N log N) time.

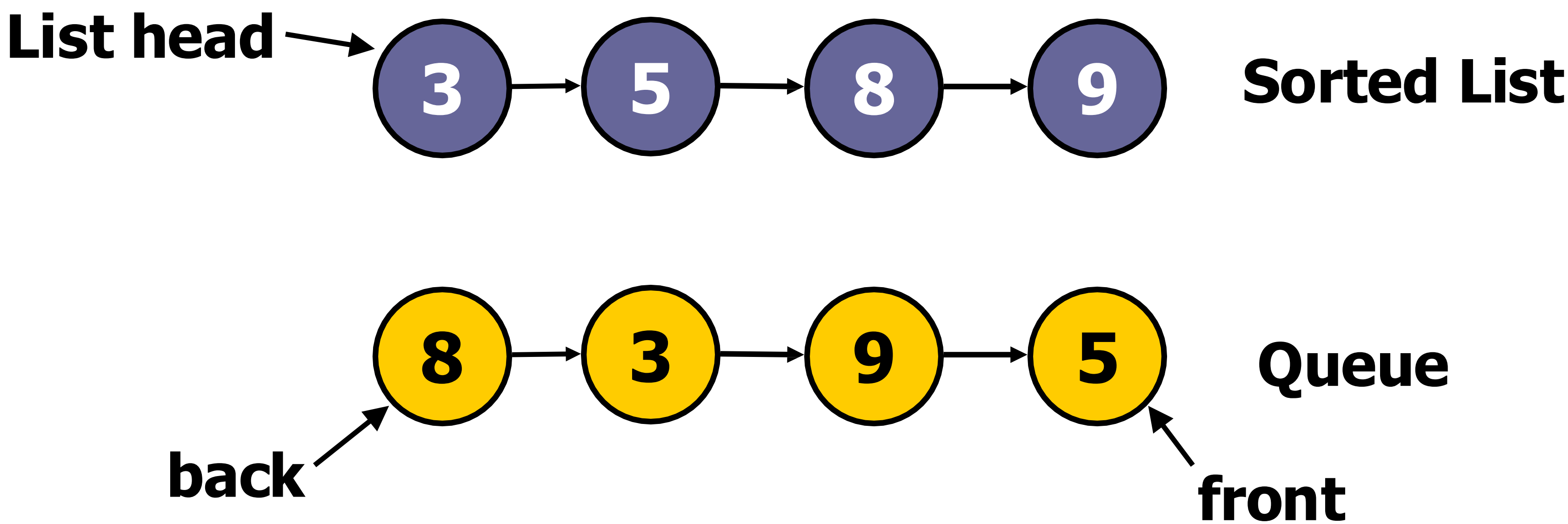
enqueue(item)	O(1)
dequeue()	O(1)
peek()	O(1)
printInOrder()	O(N log N)

Use a Sorted Linked List

- We can reduce printInOrder() to O(N) using a sorted linked list instead.
- But the queue operations are not supported.

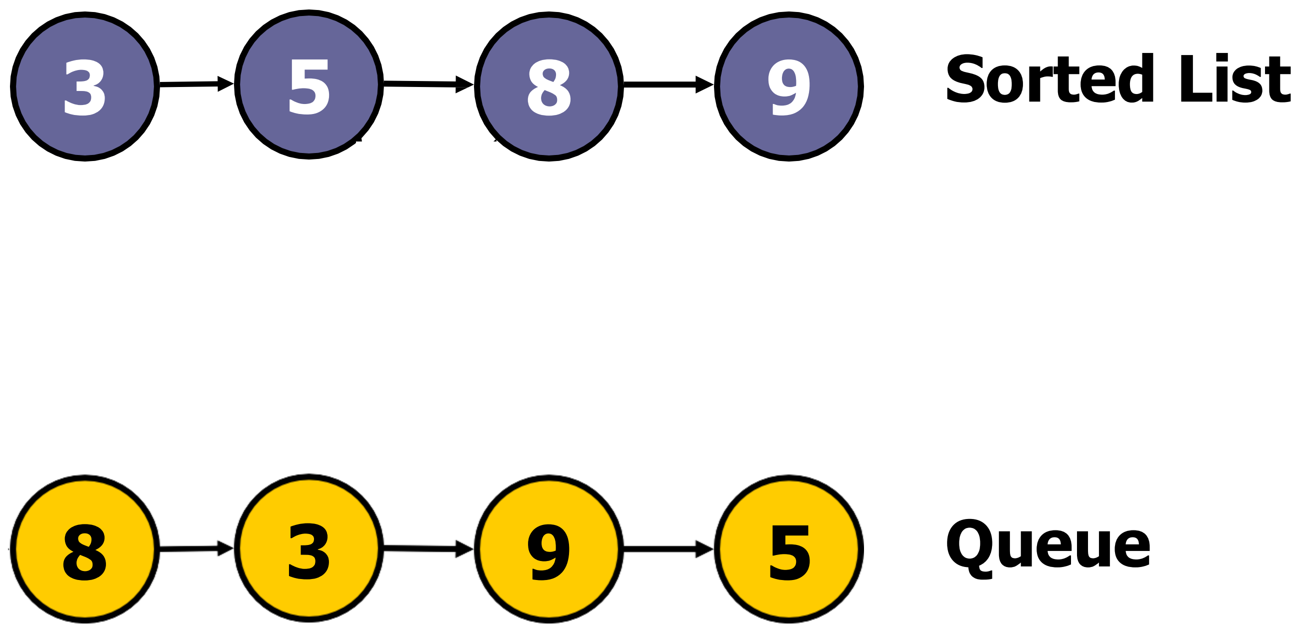
enqueue(item)	?
dequeue()	?
peek()	?
printInOrder()	O(N)

Use both: Queue + Sorted List ?

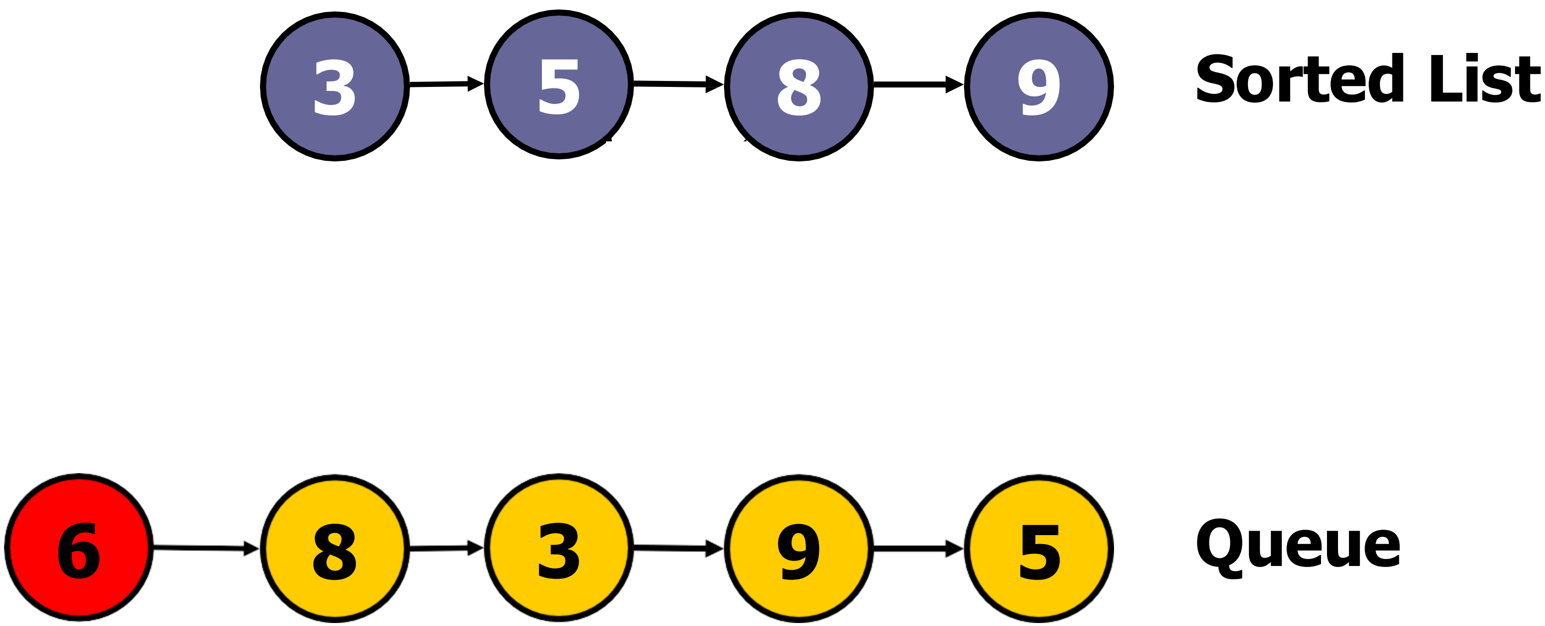


Trivial problem: Need to duplicate the data.

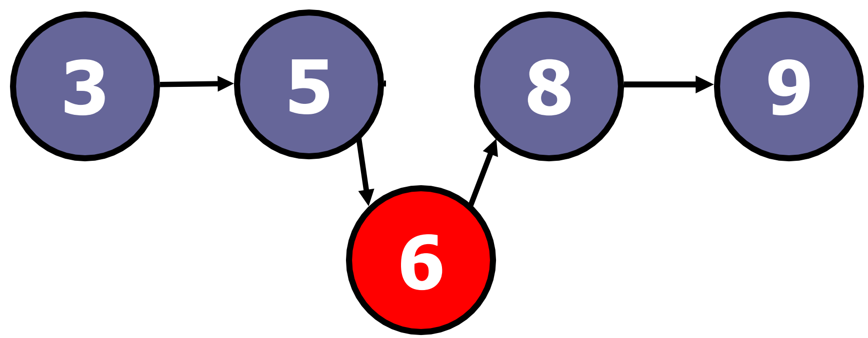
Enqueue(6)



Enqueue(6)



Enqueue(6)



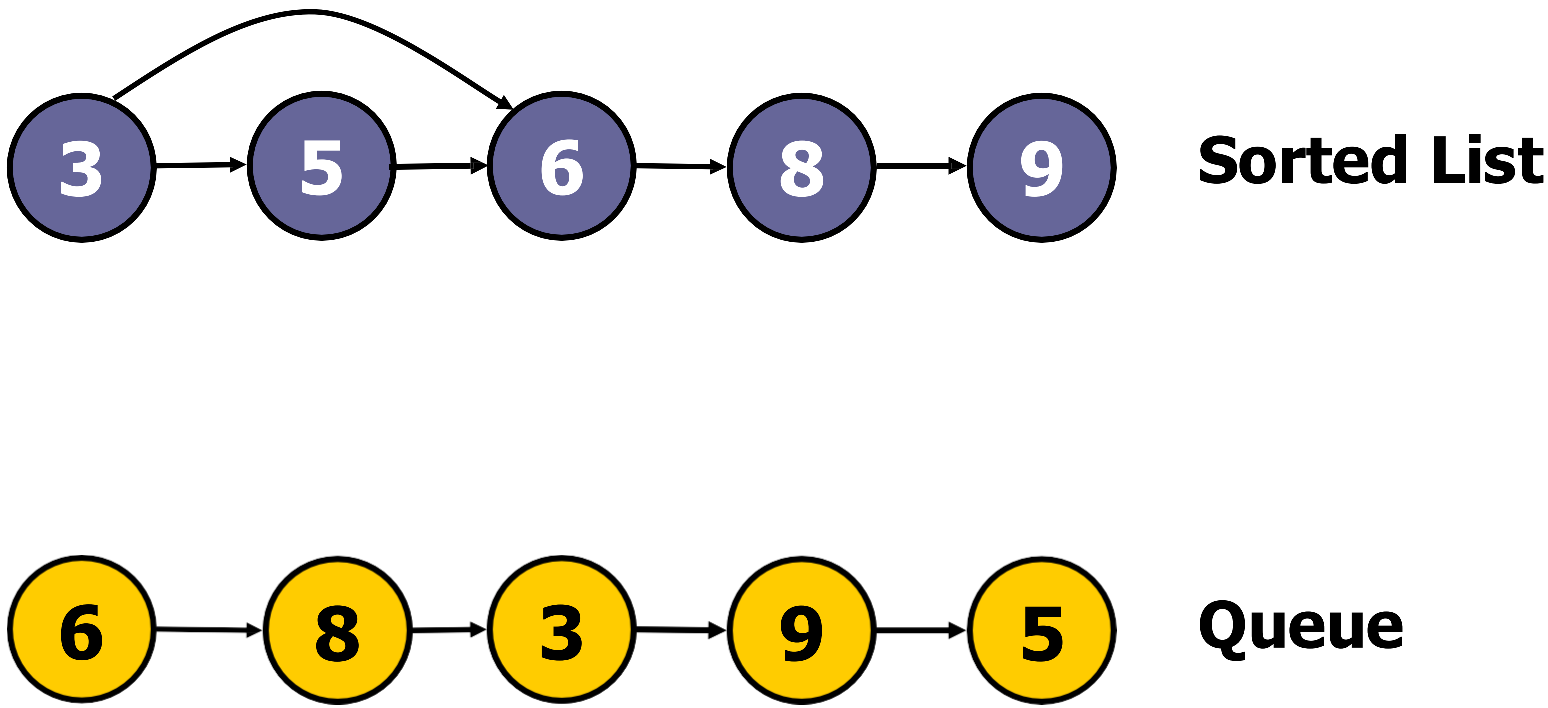
Sorted List

O(N)

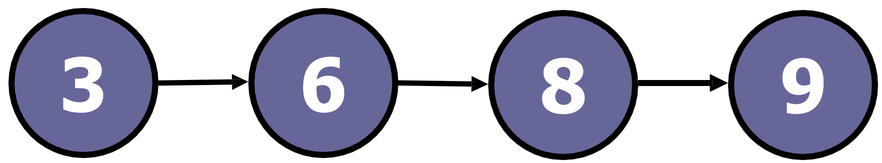
Queue

O(1)

Dequeue()

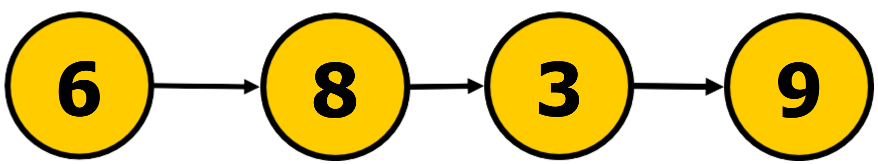


Dequeue()



Sorted List

O(N)



Queue

O(1)

Use Queue + Sorted List

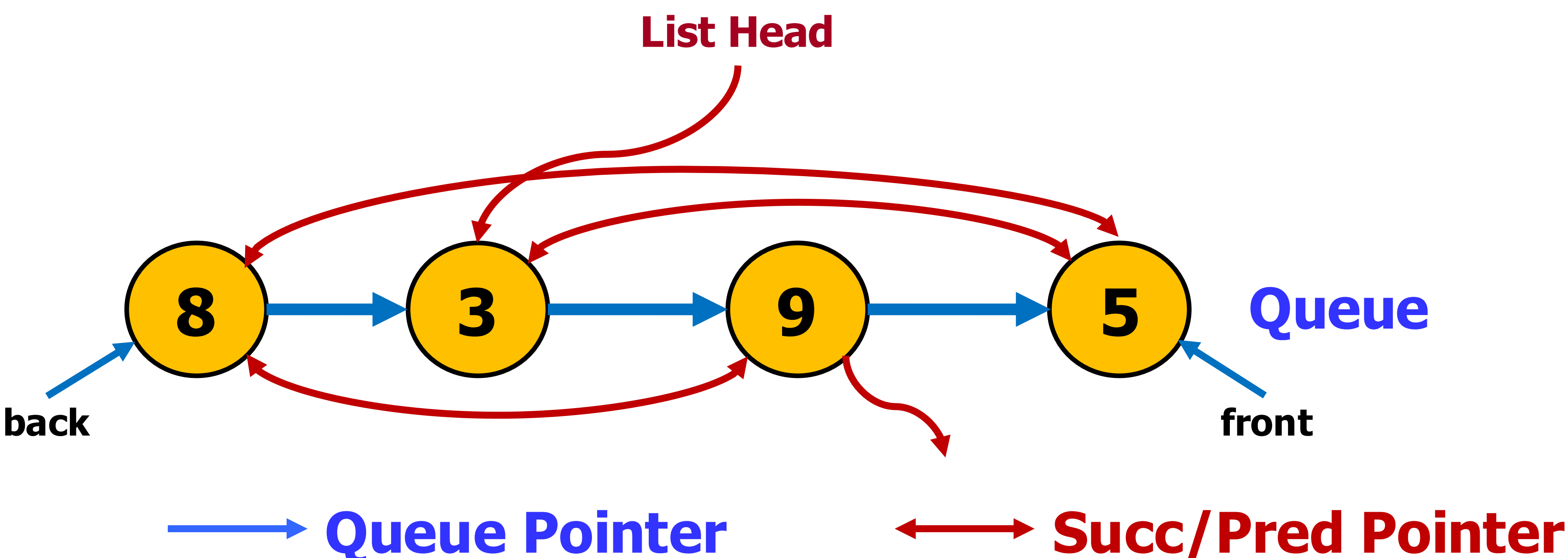
But then enqueue and dequeue take linear time O(N), because we have to look for the position of the item in the linked list to insert/delete. Too slow.

enqueue(item)	O(N)
dequeue()	O(N)
peek()	O(1)
printInOrder()	O(N)

Q: Can we improve them?

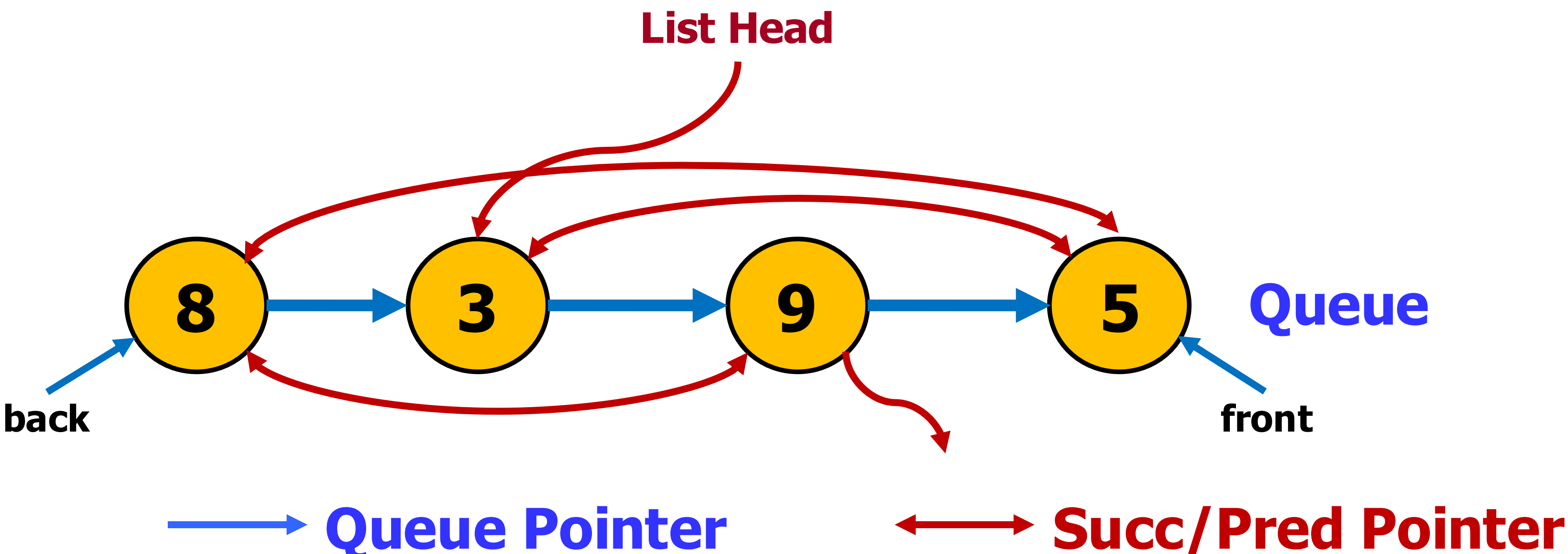
Improvement:
Queue combines with DLinked List

- Only store one copy of each item
- Each node have 2 sets of pointers:
  - One for queue and one for a doubly linked list



Combine Queue and DLinked List

- Dequeue of a doubly linked list can be done in O(1) time.
  Q: How?
- However, enqueue is still O(N). Why? E.g., enqueue 4?
  A: Need to find the insertion point in the DLinked List



Combine Queue and DLinked List

- Dequeue of a doubly linked list can be done in O(1) time.
  Q: How?
- However, enqueue is still O(N). Why? E.g. enqueue 4?

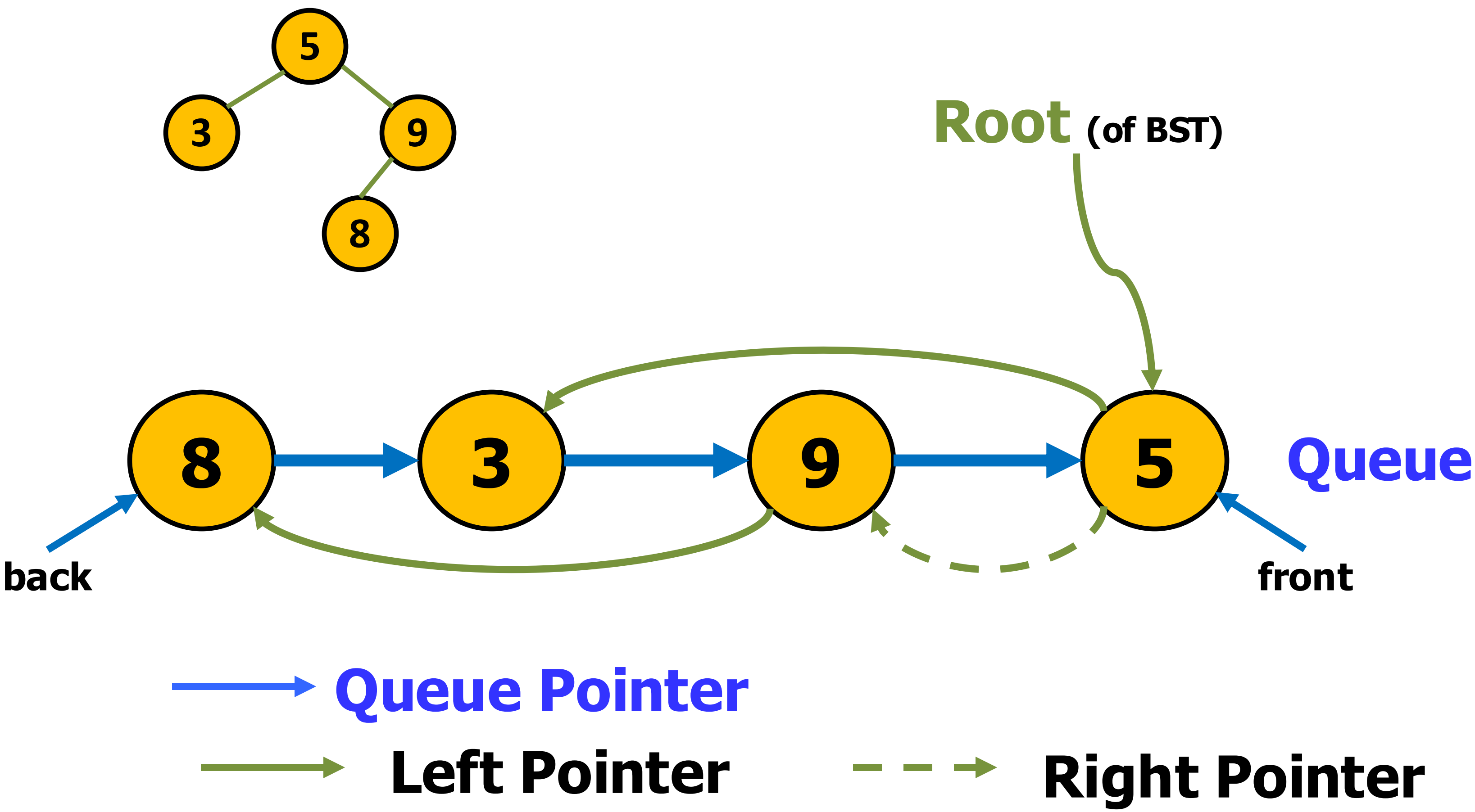
enqueue(item)	O(N)
dequeue()	O(1)
peek()	O(1)
printInOrder()	O(N)

Q: Can we improve it?

Combine Queue and BST

- We can improve enqueue to O(log N) by combing a queue with a BST instead of a linked list.

More improvement: Queue combines with BST



Combine Queue and BST

- But now dequeue also takes O(log N).

enqueue(item)	O(log N)
dequeue()	O(log N)
peek()	O(1)
printInOrder()	O(N)

Q: Is there a way to make dequeue O(1)?

Combine Queue and BST

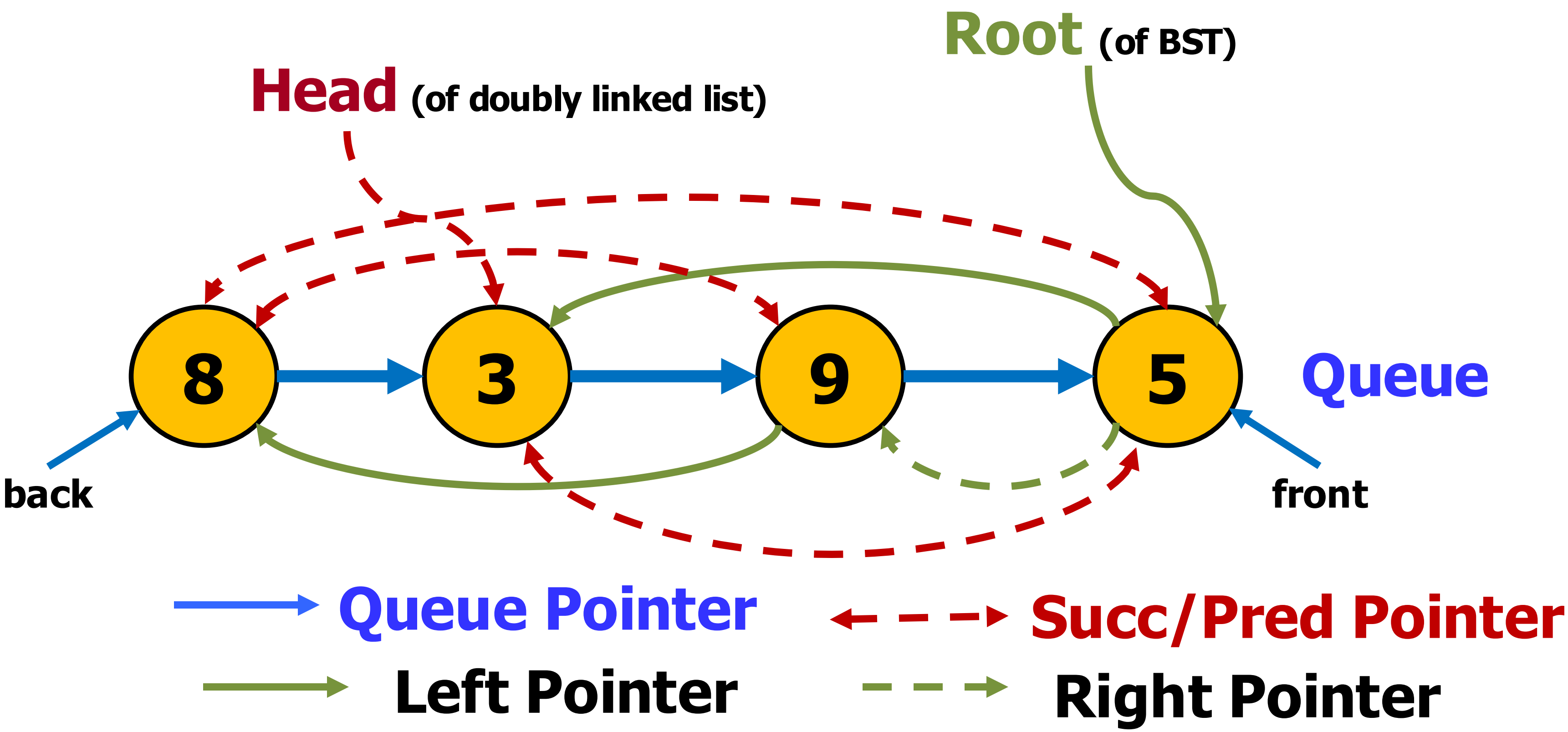
enqueue(item)	O(log N)
dequeue()	O(1) **?**
peek()	O(1)
printInOrder()	O(N)

Q: Is there a way to make dequeue O(1)?

Yes, use another doubly linked list, so that finding the replacement for BST deletion can be done in O(1) instead of O(log N).

More improvement: combine Queue + BST + DList

- Use another doubly linked list.



Combine queue + BST + DList

enqueue(item)	O(log N)
dequeue()	O(1)
peek()	O(1)
printInOrder()	O(N)

Recall: use another doubly linked list, so that finding the replacement for BST deletions can be done in O(1) instead of O(log N). Why?

Improvement summary

- use a queue and a linked list
- combine queue with doubly linked list
- combine queue and BST
- combine queue, BST, and doubly linked list

Q: Which improvement should be used?

Depends on the application.

E.g., it depends how often certain operations are executed.

End of Mix and Match

CS2040 Objectives

- Give an introduction to data structures and algorithms for constructing **efficient** computer programs.
- Emphasize on **data abstraction** issues (through ADTs) in the code development.
- Emphasize on **efficient implementations** of chosen data structures and algorithms.

CS2040 Objectives

- Include arrays, lists, stacks, queues, hash tables, and BST/AVL trees, heaps, graphs together with their algorithms (insert, delete, find, etc.).
- Simple algorithmic paradigms, such as **sorting** and **search** algorithms and **greedy** algorithms were introduced.
- Elementary **analysis of algorithmic complexities** were taught.

What is Next?

- Continue to program ☺!
- For non-CS, take more CS modules
  - CS2103 – Software Engineering
  - CS3230 – Design and Analysis of Algorithms
  - CS3217/CS3216 – Software development on Modern Platforms
  - CS3233 – Competitive Programming
  - CS3247/CS4213 – Game Development
  - Others: AI, Cybersecurity, Blockchain, etc.

What is Next?

- Do a minor in CS
- Do a major in CS
- Transfer to CS
- Create the next software for the whole world/universe ☺.
- Be a TA

Student Feedback Exercise

- Give your honest feedback to all in the teaching team.
- Not a chance to get back at them.
- Help them to help yourself.
- Help them to help your juniors.


شكراً جزيلاً
ngiyabonga
рахмат
danke
謝謝
teşekkür ederim
Баярлалаа
спасибо
faafetai lava
mersi
kia ora
barka
welalin
tack
spas
dank je
misaotra
matondo
paldies
grazzi
mahalo
tapadh leat
vinaka
спасибі
blagodaram
thank you
gracias
хвала
asante
manana
obrigada
tenki
murakoze
kiitos
dankie
nanni
nandri
dhanyavad
mauruuru
köszönöm
hvala
gràçie
bayarlalaa
bedankt
enkosi
akun
dankon
aciū
djiere dieuf
tau
mochchakkeram
дякую
mamnun
chokrane
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpáy
go raibh maith agat
sobodi
dėkuji
obrigado
sagolun
sukriya
kop khun krap
taiku
arigatô
takk
dakujem
trugarez
mèsi
didi madloba
najis tuke
ありがとう
tanemirt
grazie
rahmet
terima kasih
kam sah hamnida
rahmat
shukriya
mercê
мерси
diolch
dhanyavadagalu
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ
merci