CS2040: Lecture 14

Week 13
Mix and Match

Data Structures with Multiple Organizations

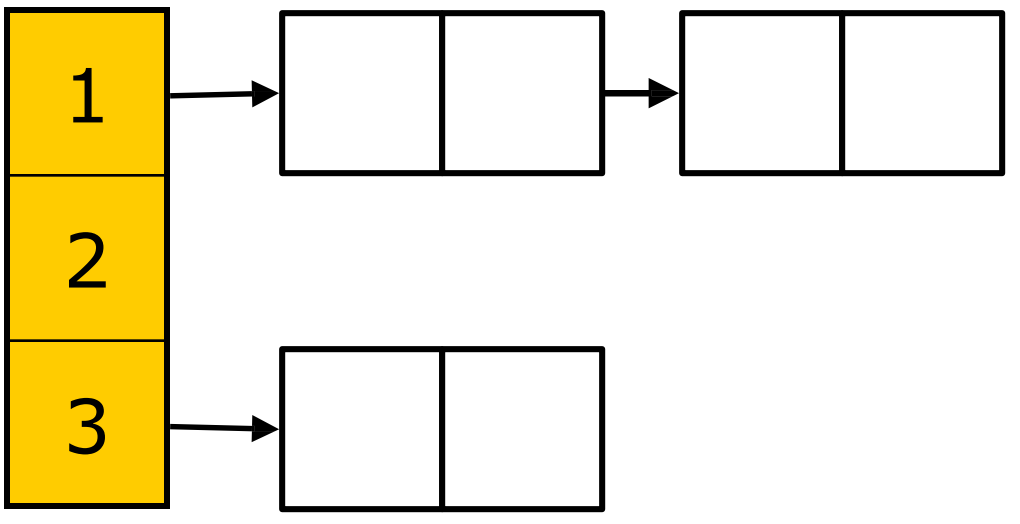
Basic Data Structures

- Arrays
- Linked Lists
- Trees

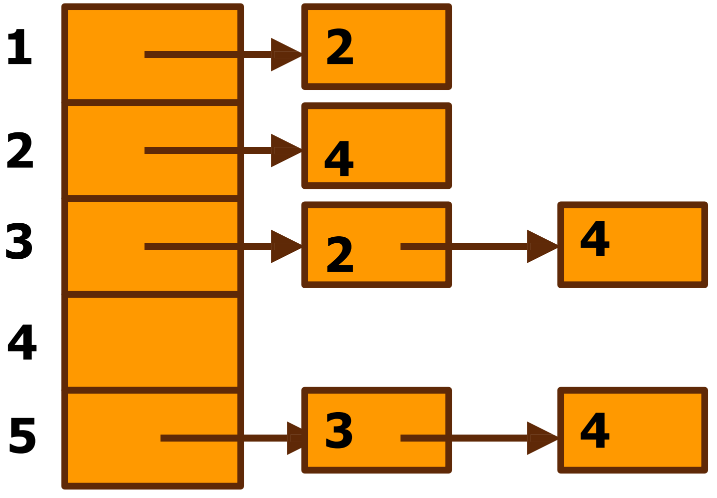
We can combine them to implement different data structures for different applications.

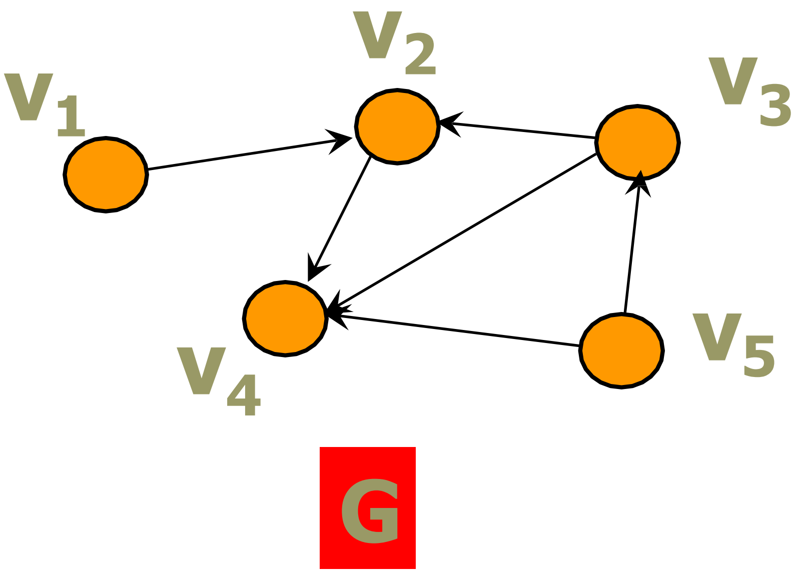
Mix-and-Match

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



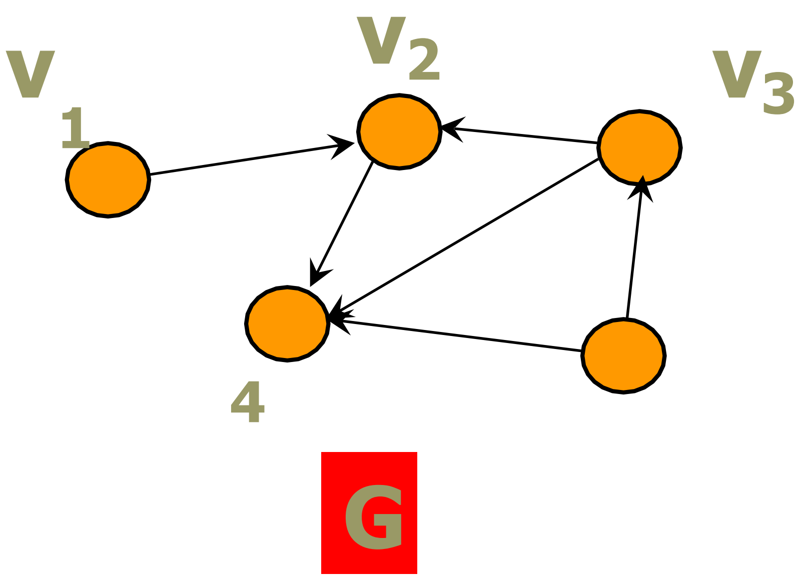
Adjacency list for directed graph





Adjacency matrix for directed graph

Matrix[i][j] = 1 if $(v_i, v_j) \in E$
0 if $(v_i, v_j) \notin E$



		1	2	3	4	5
		v_1	v_2	v_3	v_4	v_5
1	v_1	0	1	0	0	0
2	v_2	0	0	0	1	0
3	v_3	0	1	0	1	0
4	v_4	0	0	0	0	0
5	v_5	0	0	1	1	0

CS2040 2003 (Exam Q)

(16 points) Let n_i be the number of vertices adjacent to a vertex i. Suppose we want to support the following four operations on a directed graph:

- **insert**(i, j), which adds an edge (i, j) into the graph;
- **delete**(i, j), which removes the edge (i, j) from the graph;
- **exists**(i, j), which checks if edge (i, j) exists in the graph; and
- **neighbours**(i), which returns the list of vertices adjacent to i.

Describe a data structure that supports **insert**(i, j), **delete**(i, j) and **exists**(i, j) in O(1) time, and **neighbours**(i) in O(n_i) time. You may use diagrams to illustrate your data structure. You may simply quote data structures taught in this class without going into details.

Use adjacency Matrix

Operation	Big-O
Insert (i, j)	
Delete (i, j)	
Exist (i, j)	
Neighbour(i)	

	1	2	3	4
1		T		
2	T		T	
3		T		
4	T			

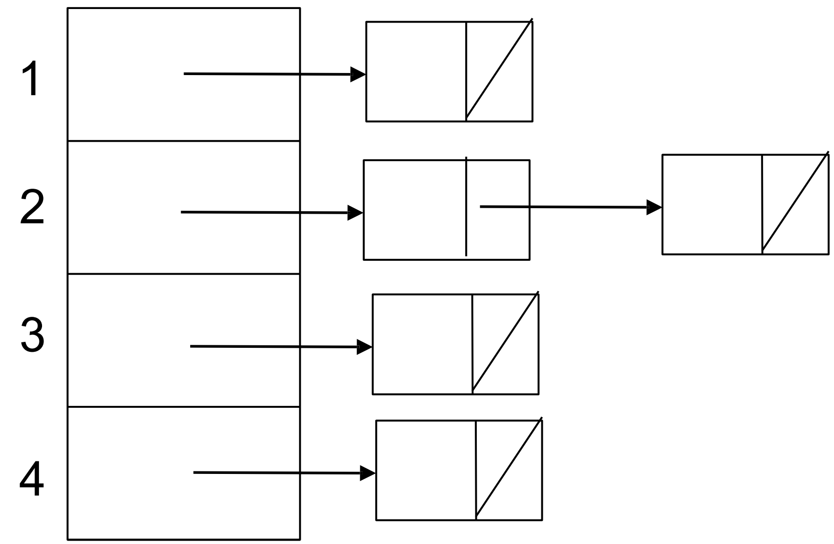
Use adjacency Matrix

Operation	Big-O
Insert (i, j)	**O(1)**
Delete (i, j)	**O(1)**
Exist (i, j)	**O(1)**
Neighbour(i)	**O(n)**

	1	2	3	4
1		T		
2	T		T	
3		T		
4	T			

Use adjacency list

Operation	Big-O
Insert (i, j)	**O(1)**
Delete (i, j)	**O(n)**
Exist (i, j)	**O(n)**
Neighbour(i)	**$O(n_i)$**



Problem

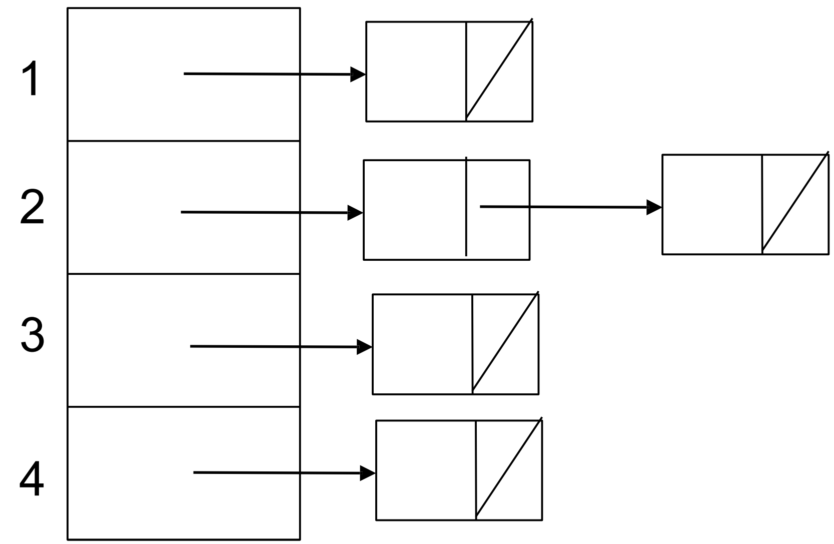
- Searching on an unsorted linked list is always O(n)
- How to improve it to O(1)?

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.

Use adjacency list

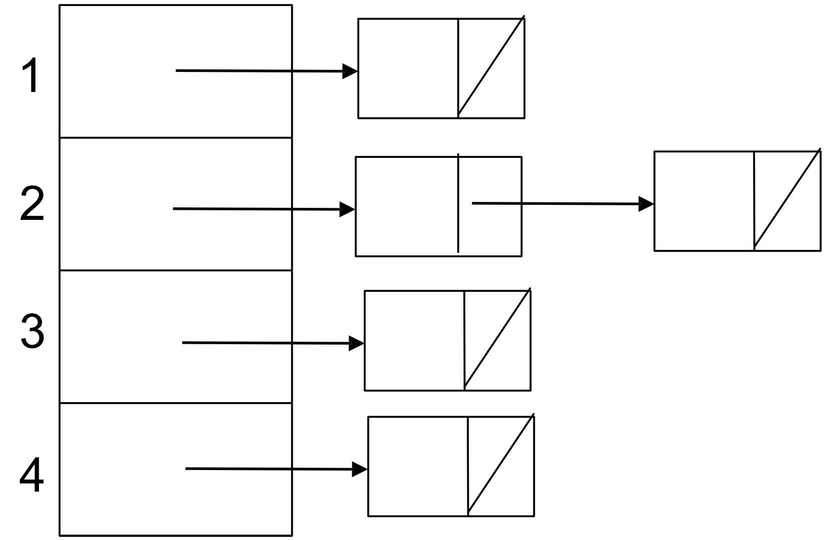
Operation	Big-O
Insert (i, j)	**O(1)**
Delete (i, j)	**O(n)**
Exist (i, j)	**O(1)**
Neighbour(i)	**$O(n_i)$**



Is delete (i, j) O(1)?

Use adjacency list

Operation	Big-O
Insert (i, j)	**O(1)**
Delete (i, j)	**O(n)**
Exist (i, j)	**O(1)**
Neighbour(i)	**$O(n_i)$**



No, hash table will find the node to be deleted, but you need to find the previous node

CS2040 2003

(16 points) Let n_i be the number of vertices adjacent to a vertex i. Suppose we want to support the following four operations on a directed graph:

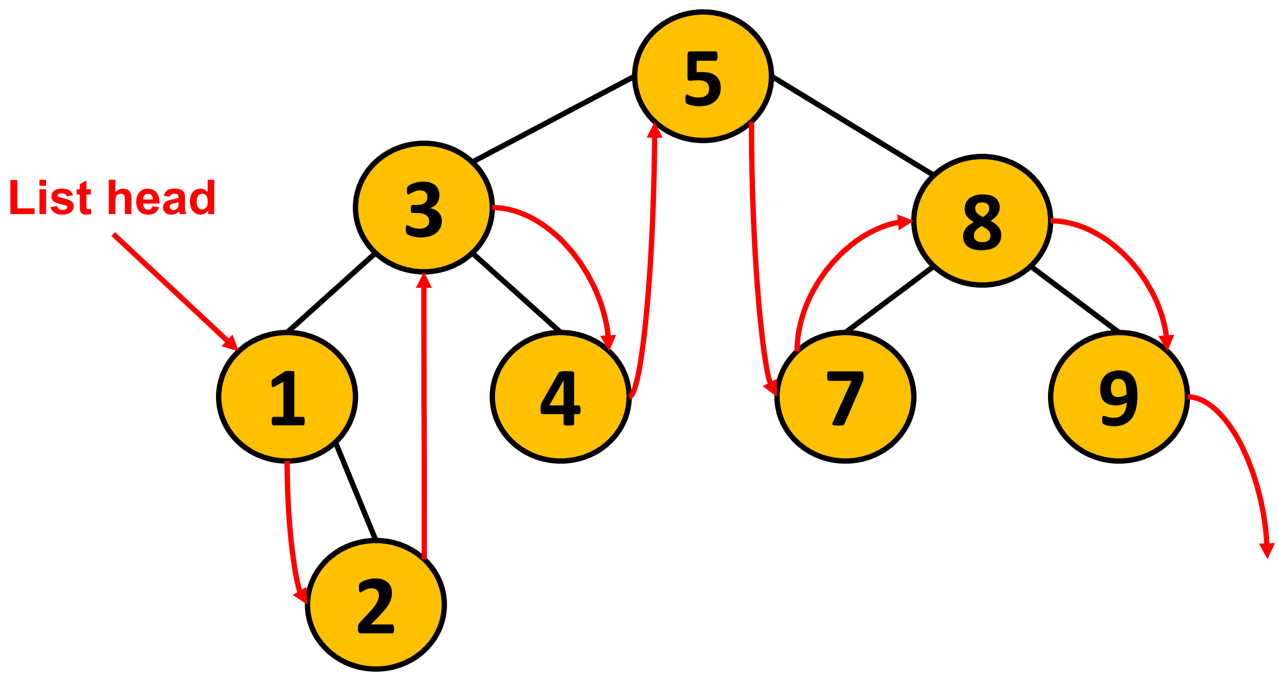
- **insert**(i, j), which adds an edge (i, j) into the graph;
- **delete**(i, j), which removes the edge (i, j) from the graph;
- **exists**(i, j), which checks if edge (i, j) exists in the graph; and
- **neighbours**(i), which returns the list of vertices adjacent to i.

Describe a data structure that supports **insert**(i, j), **delete**(i, j) and **exists**(i, j) in O(1) time, and **neighbours**(i) in O(n_i) time. You may use diagrams to illustrate your data structure. You may simply quote data structures taught in this class without going into details.

Build an adjacency list of the graph, where the lists are doubly linked. Build a hash table with (i, j) as key, and a reference to the node representing (i, j) in the adjacency list as value.

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.

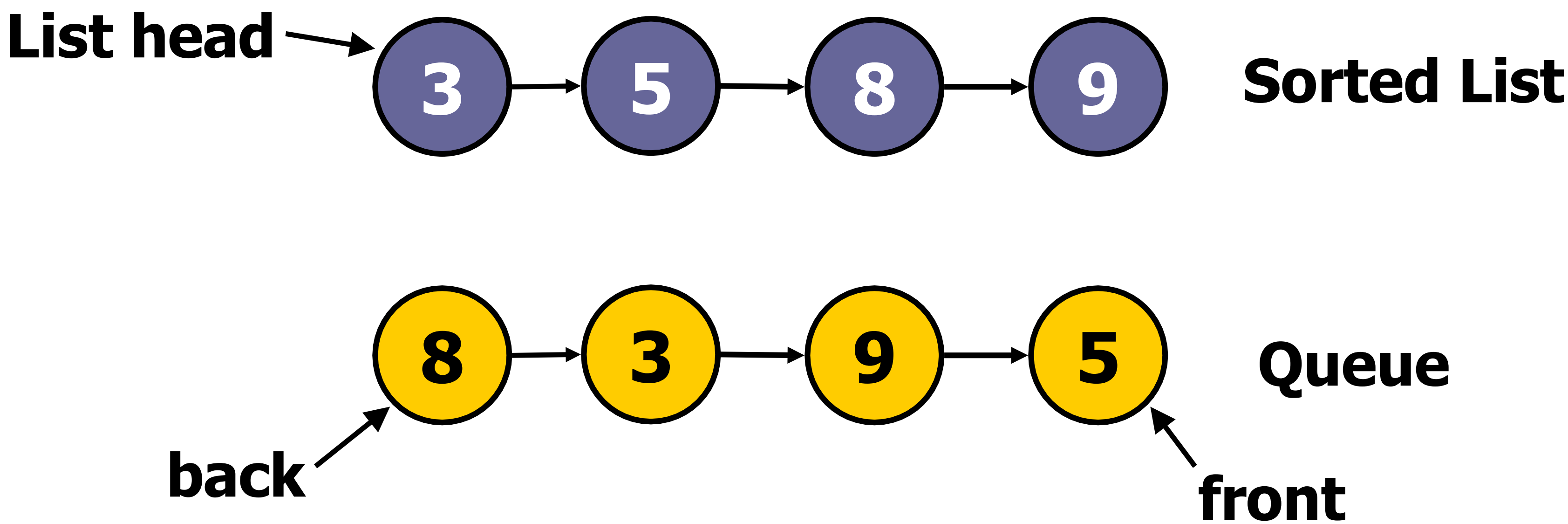
enqueue(item)	O(1)
dequeue()	O(1)
peek()	O(1)
printInOrder()	O(N log N)

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.

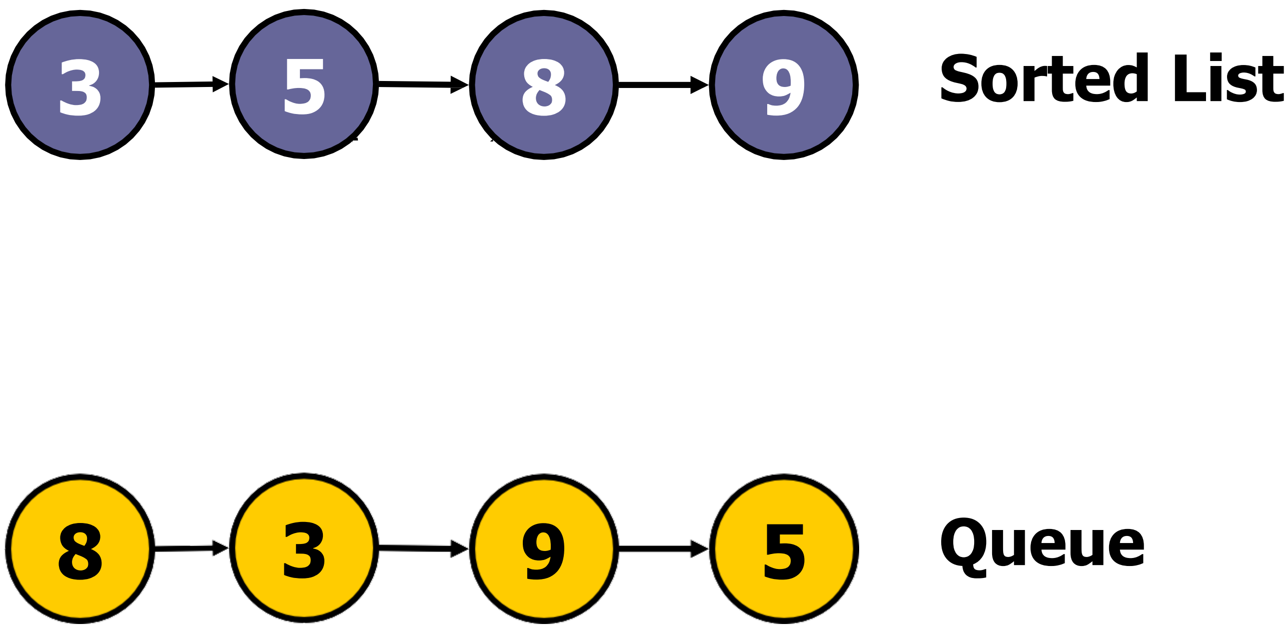
enqueue(item)	?
dequeue()	?
peek()	?
printInOrder()	O(N)

Use both:
Queue + Sorted List ?

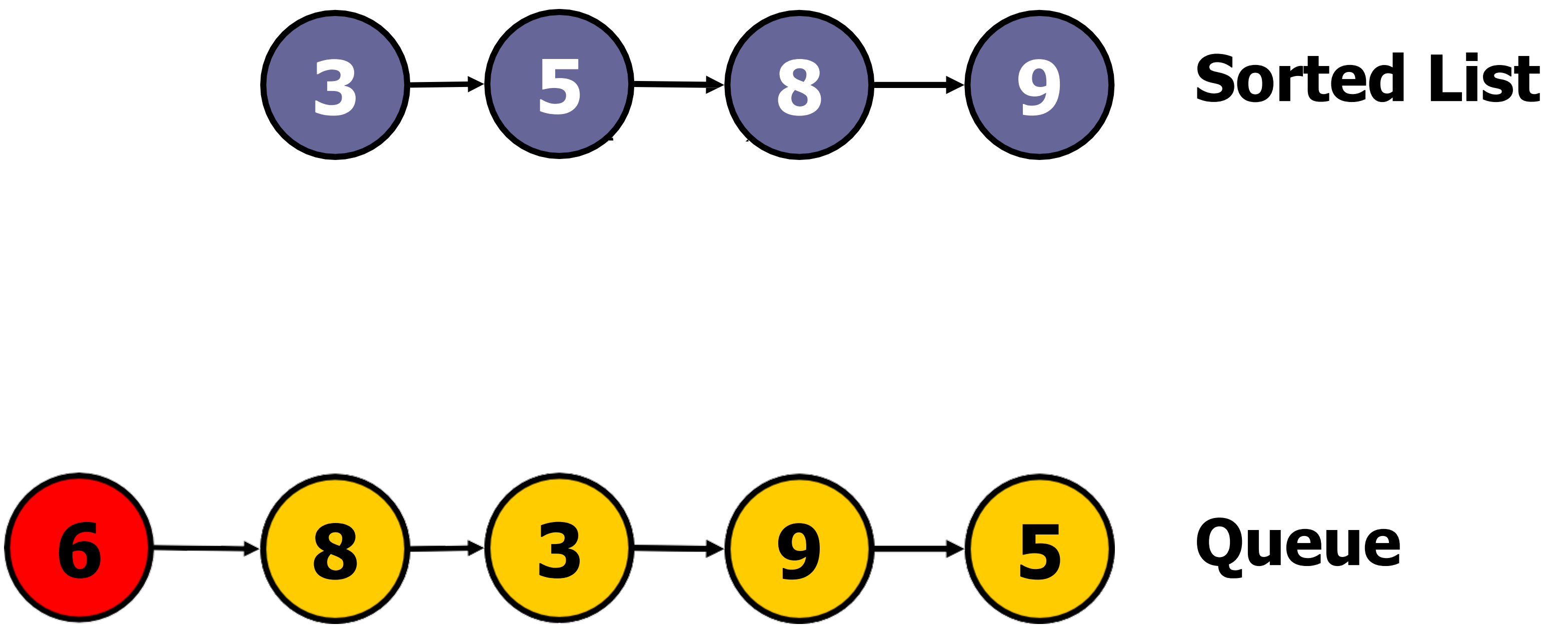


Trivial problem: Need to duplicate the data.

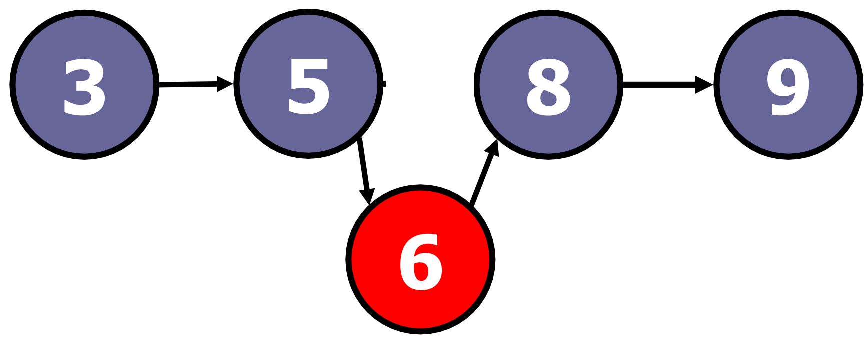
Enqueue(6)



Enqueue(6)

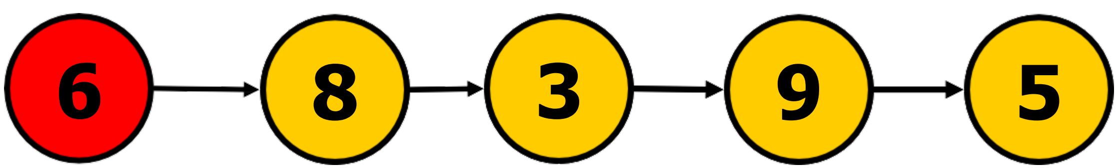


Enqueue(6)



Sorted List

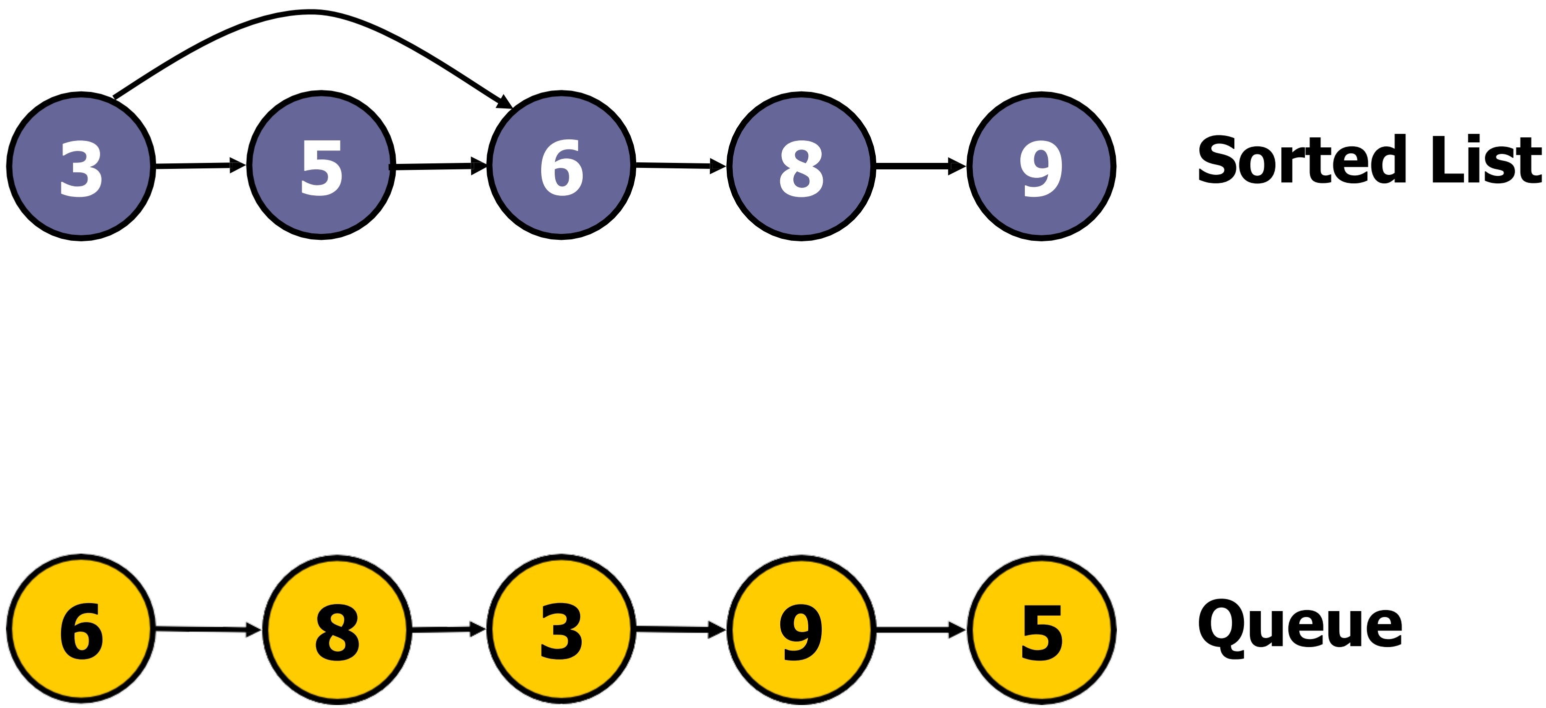
O(N)



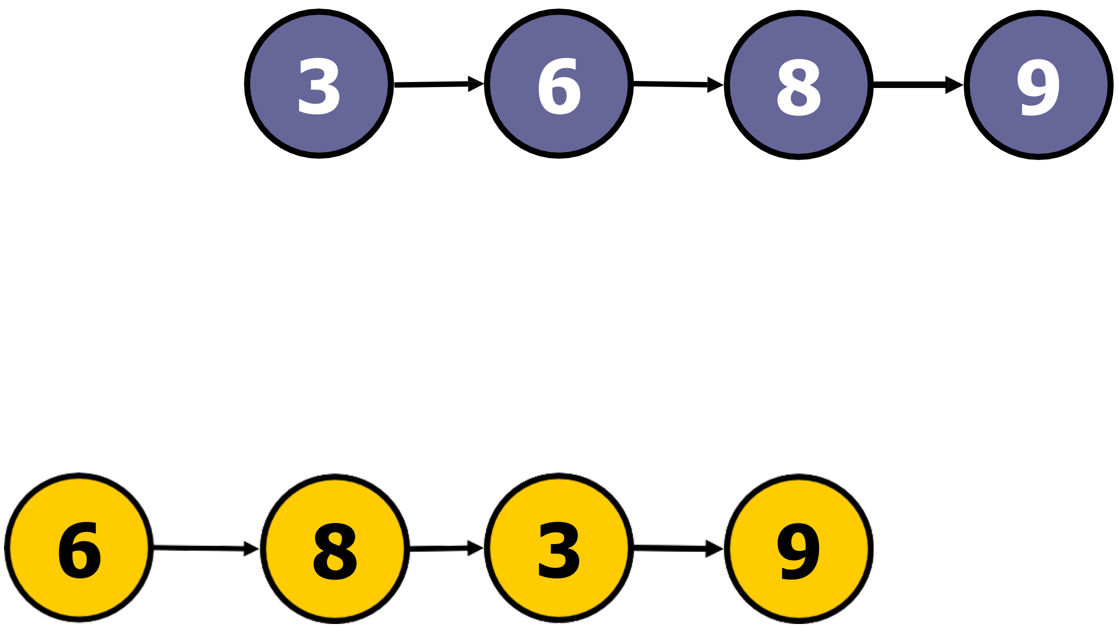
Queue

O(1)

Dequeue()



Dequeue()



Sorted List

O(N)

Queue

O(1)

Use Queue + Sorted List

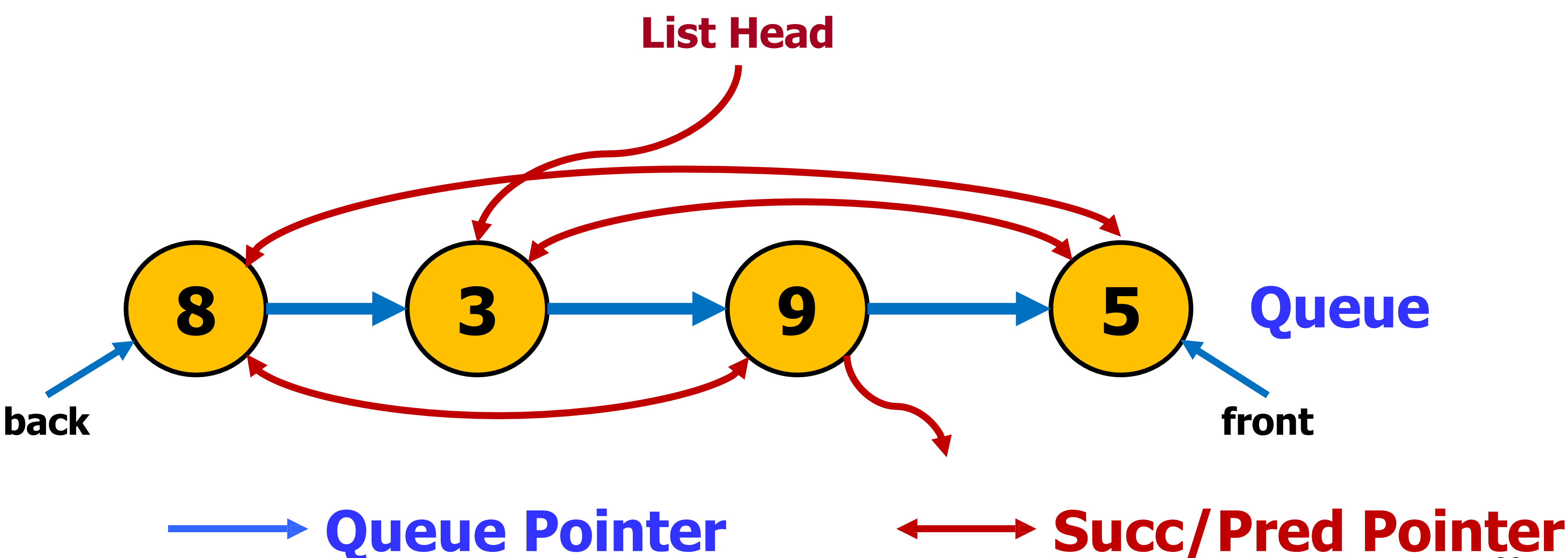
But then enqueue and dequeue take linear time O(N), because we have to look for the position of the item in the linked list to insert/delete. Too slow.

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

Q: Can we improve them?

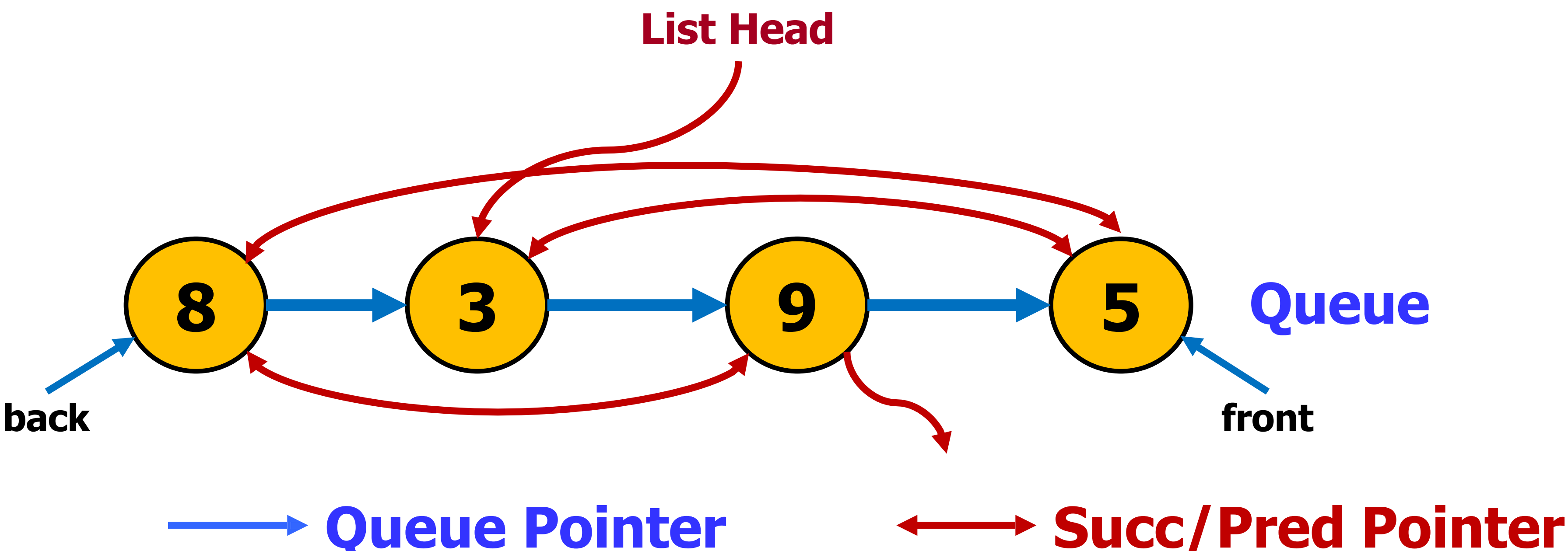
Improvement: Queue combines with DLinked List

- 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?

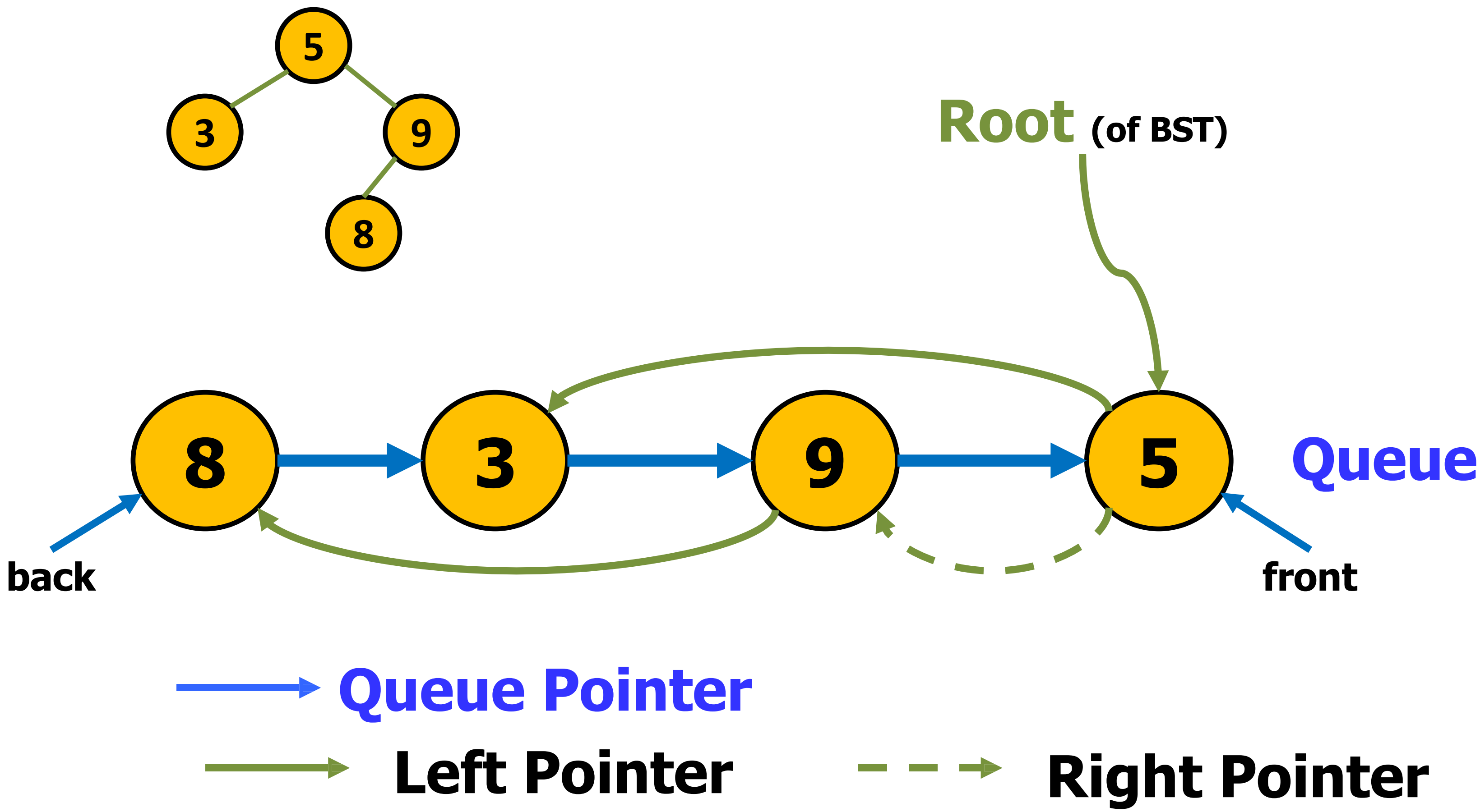
enqueue(item)	O(N)
dequeue()	O(1)
peek()	O(1)
printInOrder()	O(N)

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

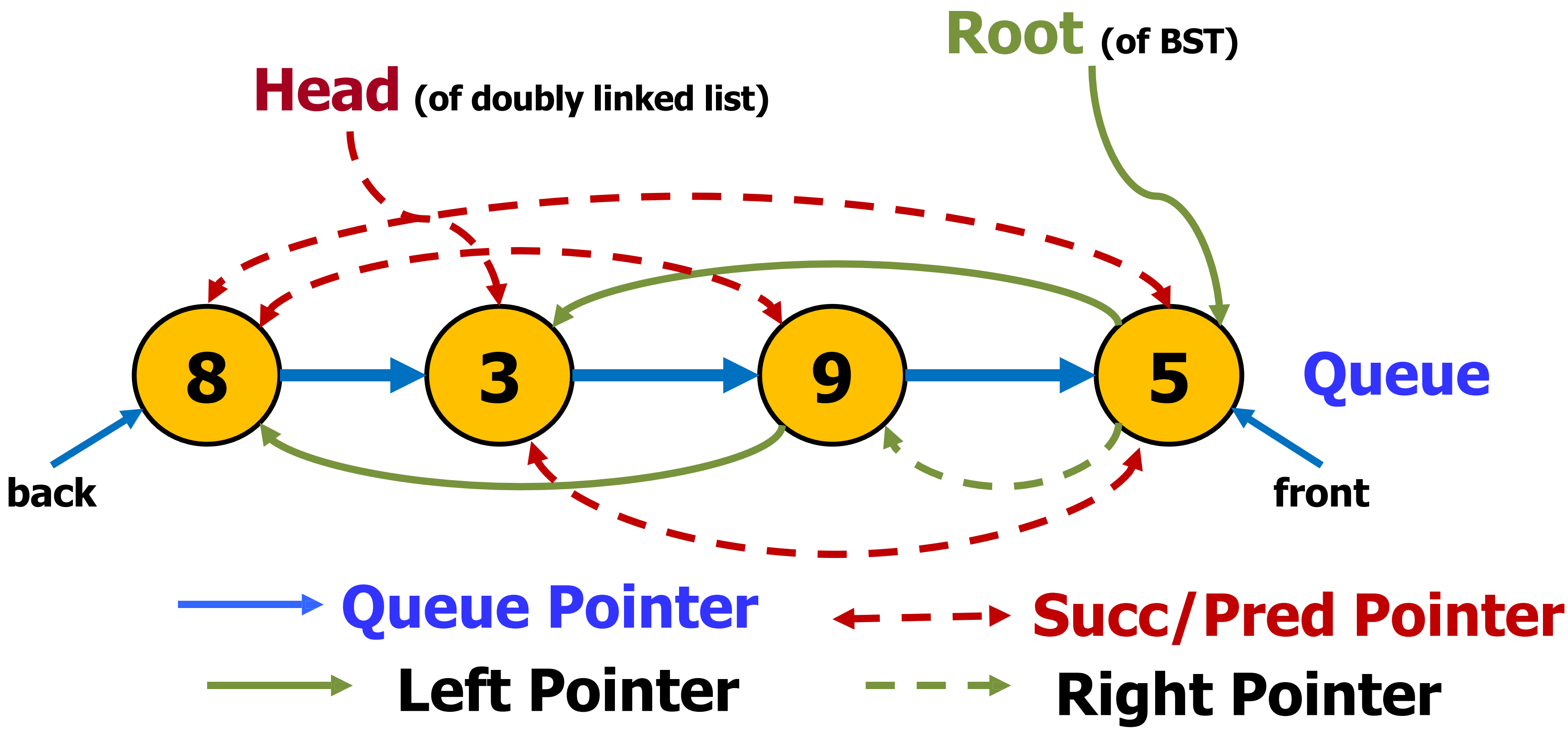
enqueue(item)	O(log N)
dequeue()	O(1) **?**
peek()	O(1)
printInOrder()	O(N)

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

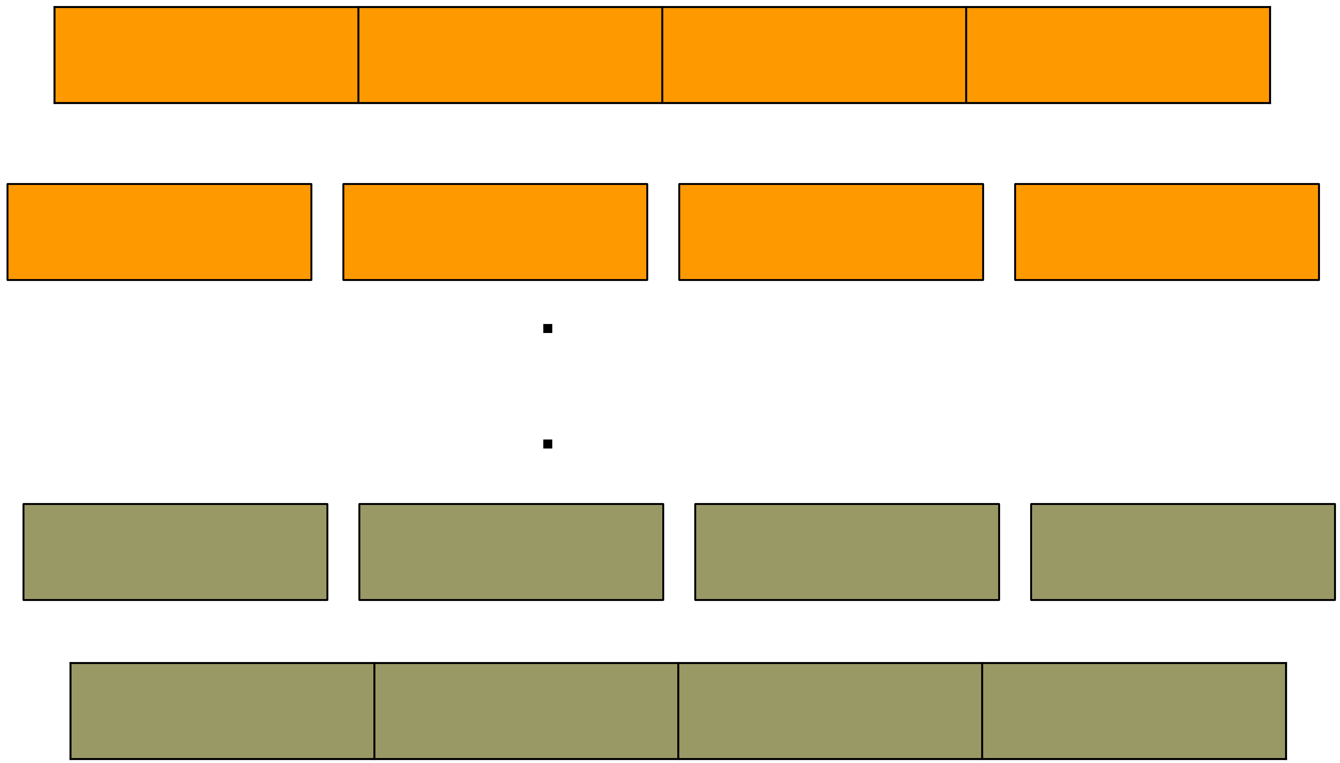
- use a queue and a linked list
- combine queue with doubly linked list
- combine queue and BST
- combine queue, BST, and doubly linked list

Q: Which improvement should be used?

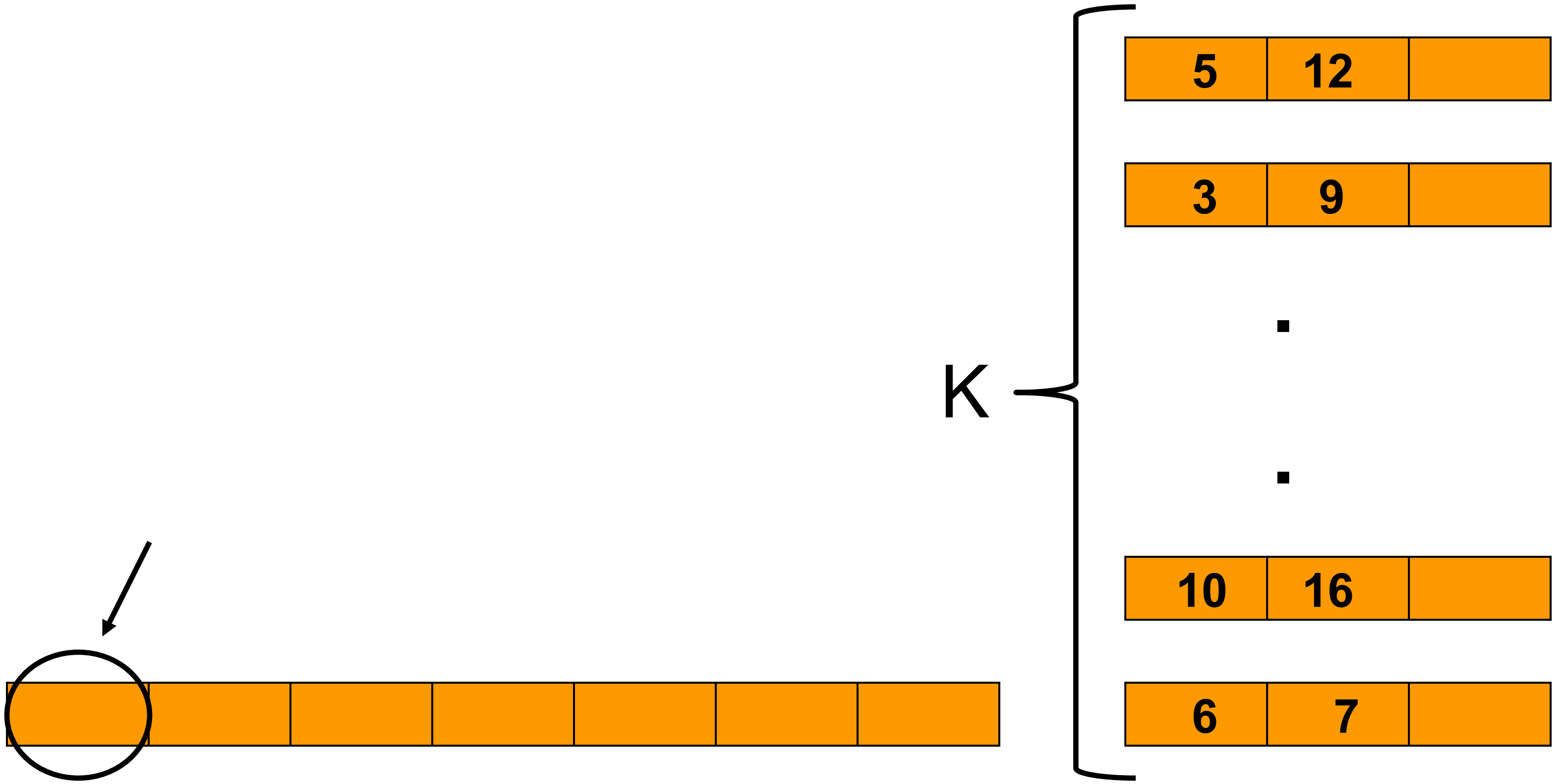
Depends on the application.
E.g., it depends how often certain operations are executed.

K-way Merge Sort

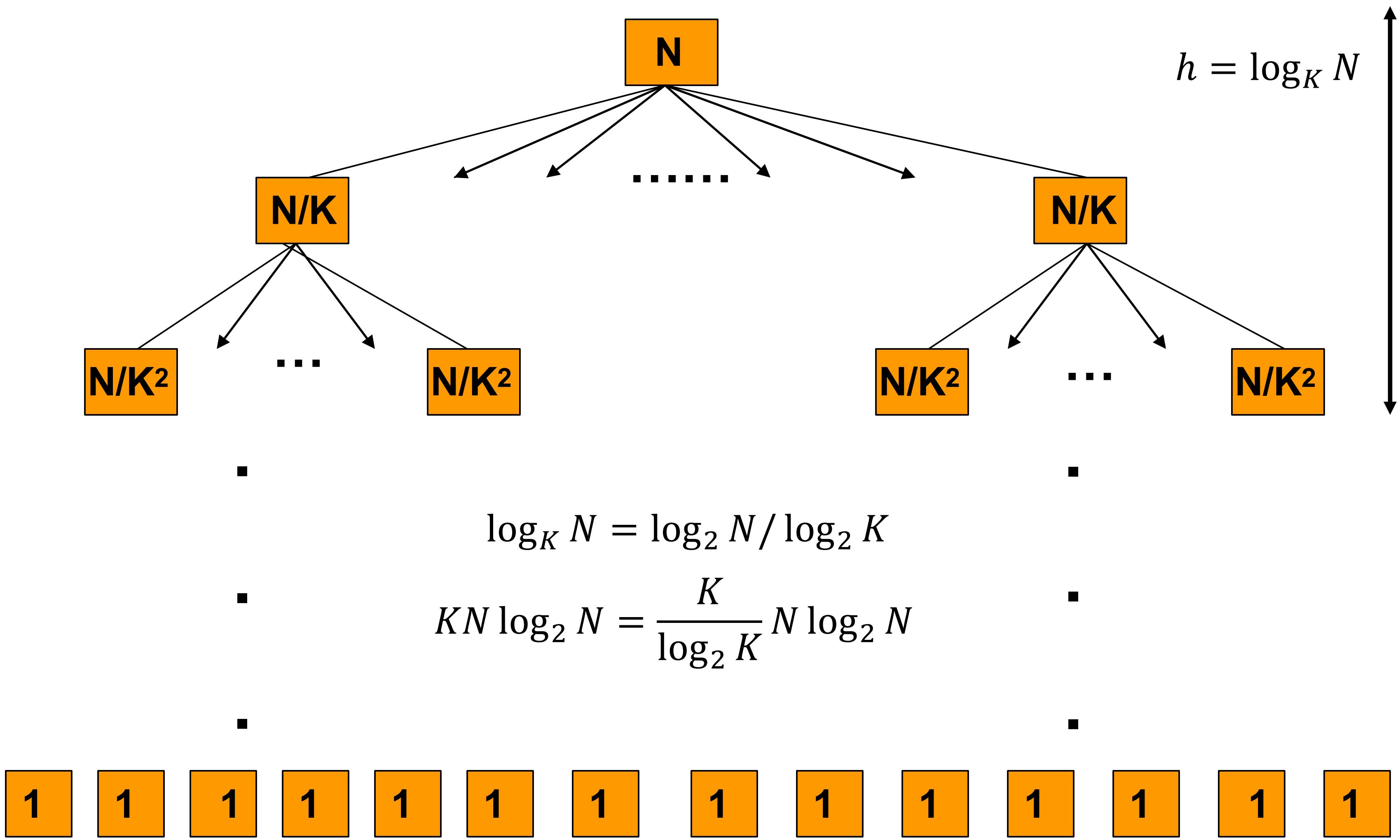
Can we make merge sort more efficient by dividing by k instead of 2?



K-way merge sort



Running time: $O(K\ N\log_k N)$

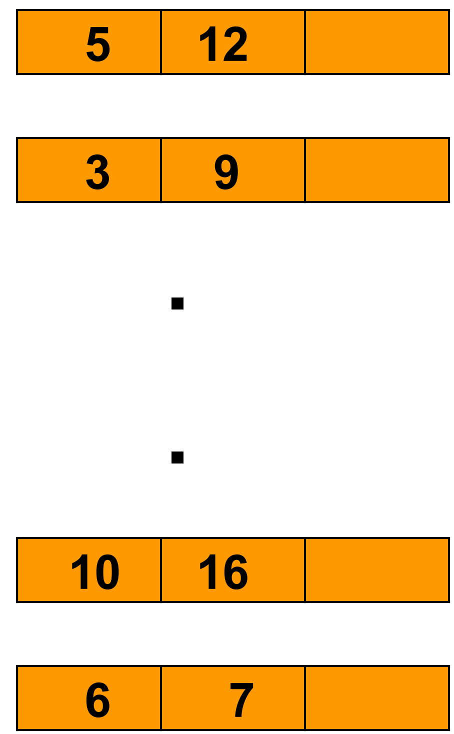


$$\log_K N = \log_2 N / \log_2 K$$

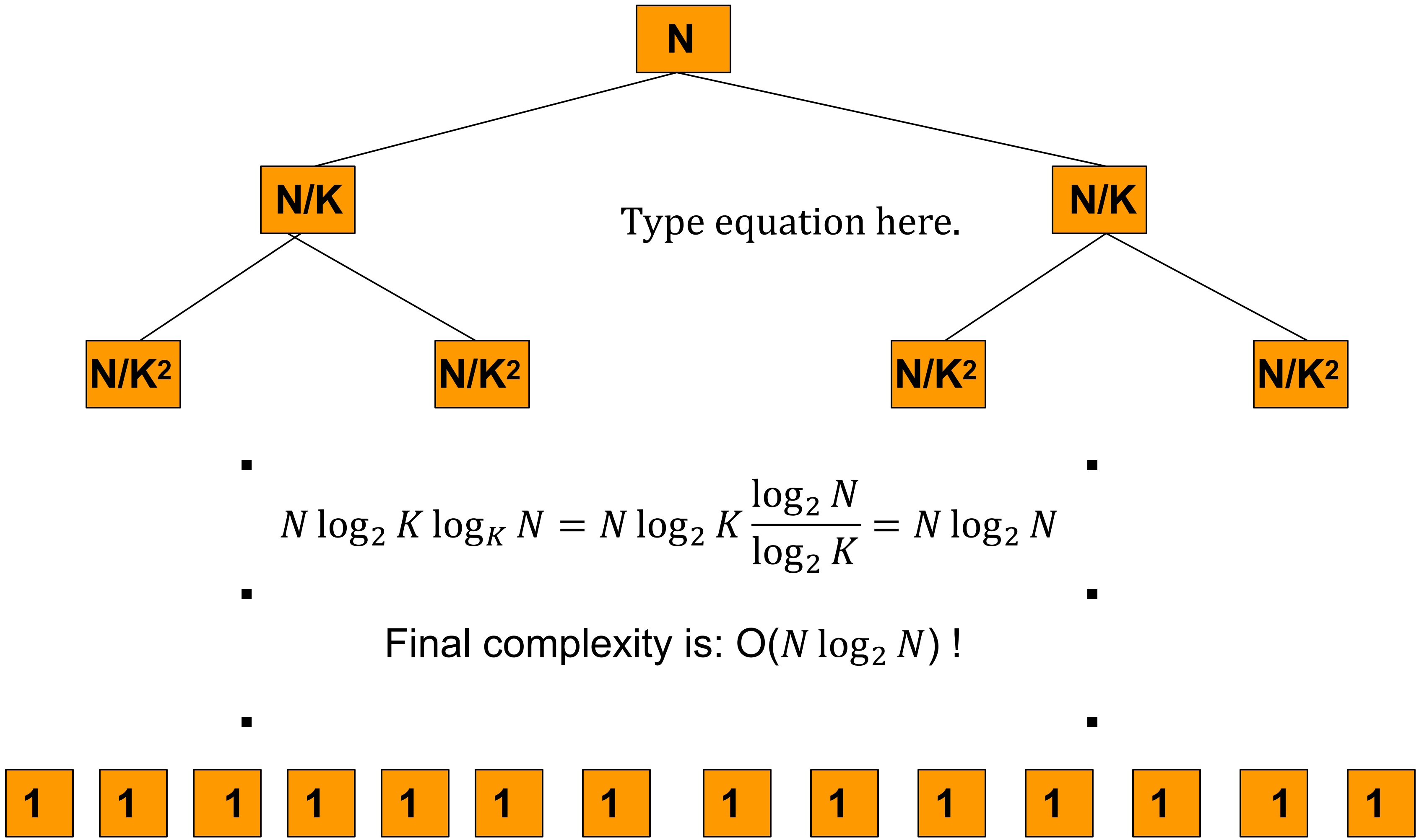
$$KN \log_2 N = \frac{K}{\log_2 K} N \log_2 N$$

Improved K-way merge sort

- K-way mergesort is more expensive than 2-way mergesort
- Can we improve it further?
- Improve selection of smallest among K elements => use heap!
- Instead of factor NK we now have $N \log_2 K$.



Running time: $O(N \log_2 K \log_k N)$



Type equation here.

$$N \log_2 K \log_K N = N \log_2 K \frac{\log_2 N}{\log_2 K} = N \log_2 N$$

Final complexity is: $O(N \log_2 N)$!

Running time: $O(N \log_2 K \log_k N)$

- By changing the base, we get

  $O(N \log_2 N)$

- It is not really an improvement over 2-way merge sort.
- But it has real applications.

End of Mix and Match


thank you
شكراً جزيلاً
danke
謝謝
рахмат
ngiyabonga
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
kiitos
dankie
nanni
nandri
dhanyavad
maururu
gracias
хвала
asante
manana
obrigada
tenki
murakoze
bedankt
bayarlalaa
gràcie
hvala
köszönöm
dankon
aciu
akun
enkosi
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpáy
djiere dieuf
tau
mochchakkeram
дякую
mamnun
chokrane
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
mercè
мерси
diolch
dhanyavadagalu
xièxie
ευχαριστώ
감사합니다
merci