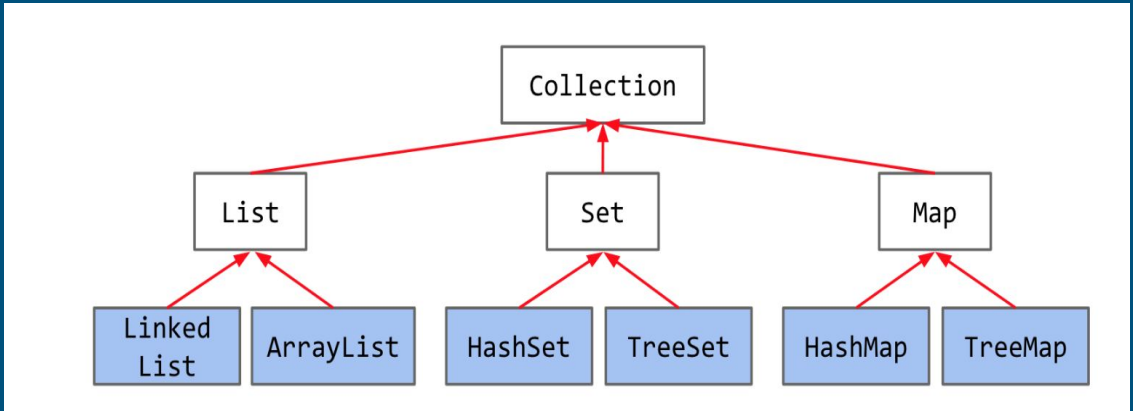
ADTs and and Sorting

CS61B Sp19 Discussion 12

ADT's

- An Abstract Data Type (ADT) is defined only by its operations, not by its implementation.
- For example in proj1a, we developed an ArrayDeque and a LinkedListDeque that had the same methods, but how those methods were written was very different. In this case, we say that ArrayDeque and LinkedListDeque are *implementations* of the Deque ADT.
- From this description, we see that ADT's and interfaces are somewhat related.



List

- An ordered collection or sequence
- add(element); // adds element to the end of the list
- add(index, element); // adds element at the given index
- get(index); // returns element at the given index
- size(); // the number of elements in the list

Set

- A (usually unordered) collection of **unique** elements
  - Calling add(“hi”) twice doesn’t change anything about the set
- add(element); // adds element to the collection
- contains(object); // checks if set contains object
- size(); // number of elements in the set
- remove(object); // removes specified object from set

Map

- A collection of key-value mappings
  - The keys of a map are unique
  - Analogous to a dictionary in python
- put(key, value); // adds key-value pair to the map
- get(key); // returns value for the corresponding key
- containsKey(key); // checks if map contains the specified key
- keySet(); // returns set of all keys in map

Stacks and Queues and Deques

- Linear collections with rules about item addition/removal
- Stacks are Last In First Out (LIFO)
- Queues are First In First Out (FIFO)
- Deques combine both

Priority Queue

- Like a queue but each element has a priority associated with it that determines the ordering of removal
- add(e); // adds element e to the priority queue
- peek(); // looks at the highest priority element, but does not remove it from the PQ
- poll(); // pops the highest priority element from the PQ

Solving problems with ADTs

- Don’t worry about implementation, assume the ADTs correctly implemented their given set of methods
- Writing pseudocode might help you
  - It also might not, depends
- Try out a few of the ADTs and compare their usefulness in the situation
- Think about correctness first, and then efficiency

(a) Given a news article, find the frequency of each word used in the article.

Use a map. When you encounter a word for the first time, put the key into the map with a value of 1. For every subsequent time you encounter a word, get the value, and put the key back into the map with its value you just got, plus 1.

(b) Given an unsorted array of integers, return the array sorted from least to greatest.

Use a priority queue. For each integer in the unsorted array, enqueue the integer with a priority equal to its value. Calling dequeue will return the largest integer; therefore, we can insert these values from index length-1 to 0 into our array to sort from least to greatest.

(c) Implement the forward and back buttons for a web browser

Use two stacks, one for each button. Each time you visit a new web page, add the previous page to the back button's stack. When you click the back button, add the current page to the forward button stack, and pop a page from the back button stack. When you click the forward button, add the current page to the back button stack, and pop a page from the forward button stack. Finally, when you visit a new page, clear the forward button stack.

Define a Queue using Stacks

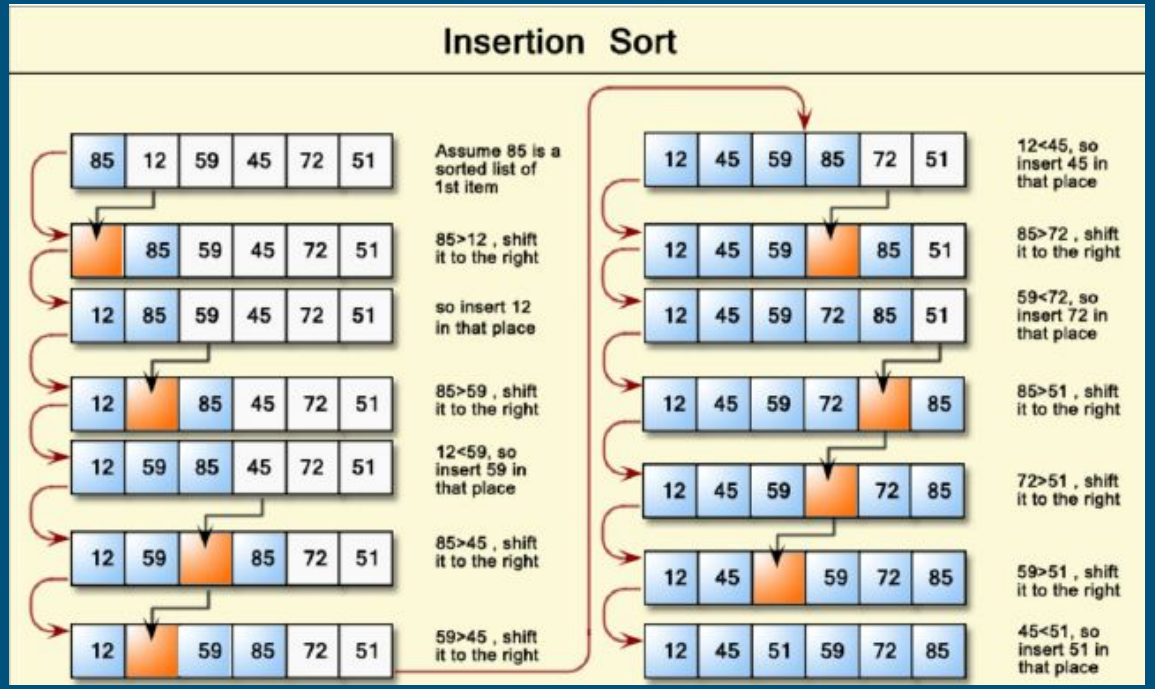
- Brainstorm first before writing any code
- Check your work with a small example

```
public class Queue {
    private Stack stack = new Stack();
    public void push(E element) {
        Stack temp = new Stack();
        while (!stack.isEmpty()) {
            temp.push(stack.poll());
        }
        stack.push(element);
        while (!temp.isEmpty()) {
            stack.push(temp.poll());
        }
    }
    public E poll() {return stack.poll(); }
}
```

```
public class Queue {
    private Stack stack = new Stack();
    public void push(E element) { stack.push(element); }
    public E poll() { return poll(stack.pop()); }
    private E poll(E previous) {
        if (stack.isEmpty()) {
            return previous;
        }
        E current = stack.poll();
        E toReturn = poll(current);
        push(previous);
        return toReturn;
    }
}
```

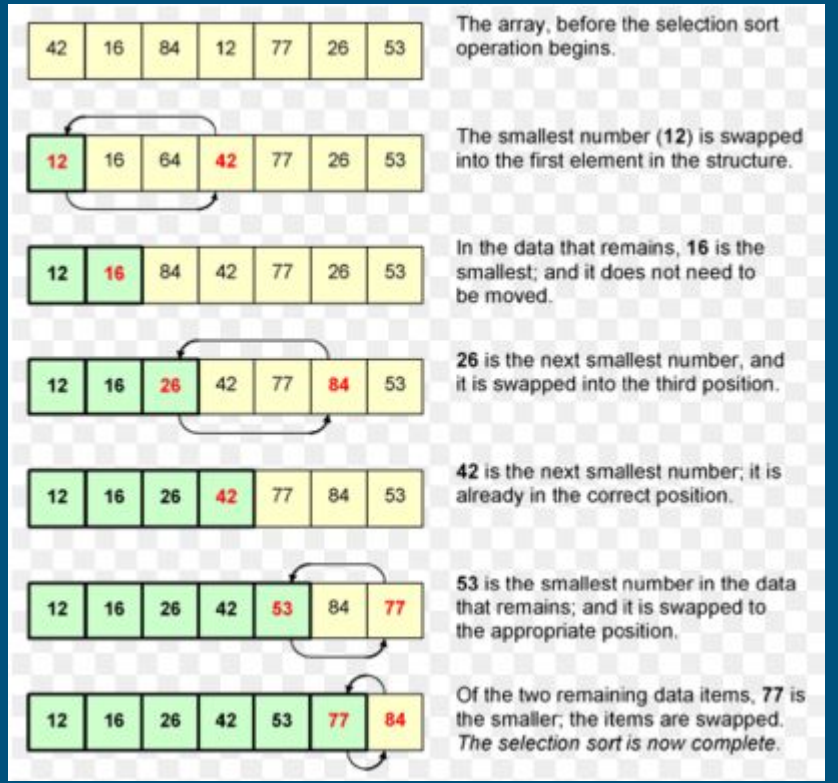
Slow sorts

- Insertion sort: For every item, swap until it is larger than the element to its left (or is first).
- Runtime: O(N^2), Omega(N)



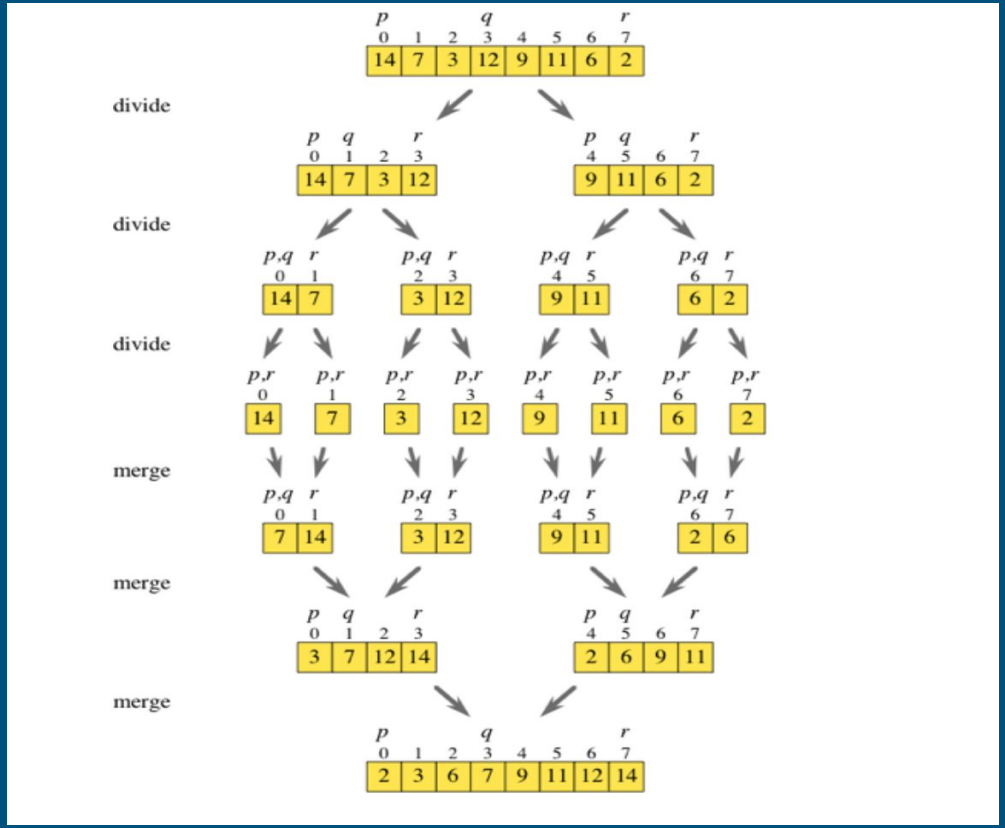
Slow sorts

- Selection sort:
  Repeatedly find the smallest element and put it in the front
- Runtime: Theta(N^2)



Fast sorts

- Merge sort: Recursively sorts halves of the array
- Procedure:
  - If base case (single element):
    - Return single element
  - Cut the items in half
  - Recursively sort both halves
  - Merge the two halves
  - Return
- Runtime: Theta(NlogN)



Fast sorts

- Heap Sort: Puts everything into a max heap and then repeatedly pops off the max element
- Runtime: Theta(NlogN)

2.1 a-c

Please sort this list with insertion, selection, merge sort

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

2.1 d

Use heap sort to sort the following array

0, 6, 2, 7, 4

https://www.cs.usfca.edu/~galles/visualization/Heap.html