Lecture 18 - March 20

Merge Sort, Quick Sort, BST

MergeSort: Recurrence Relation
QuickSort: Ideas, Java, RT

Announcements/Reminders

- **ProgTest2** info & example questions released
- **Assignment 3** (on linked **Trees**) solution released
- **WrittenTest** and **ProgTest1** results & feedback released
- **Makeup Lecture** (on **Queues**) posted
- Lecture notes template, Office Hours, TA Contact

Merge Sort: Running Time

```
public List<Integer> sort(List<Integer> list) {
  List<Integer> sortedList;
  if(list.size() == 0) { sortedList = new ArrayList<>(); }
  else if(list.size() == 1) {
    sortedList = new ArrayList<>();
    sortedList.add(list.get(0));
  }
  else {
    int middle = list.size() / 2;
    List<Integer> left = list.subList(0, middle);
    List<Integer> right = list.subList(middle, list.size());
    List<Integer> sortedLeft = sort(left);
    List<Integer> sortedRight = sort(right);
    sortedList = merge(sortedLeft, sortedRight);
  }
  return sortedList;
}
```

$T(0) = 1$

$T(1) = 1$

$O(1)$

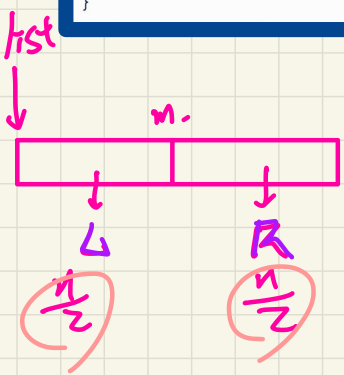
assume: $T(\frac{n}{2})$

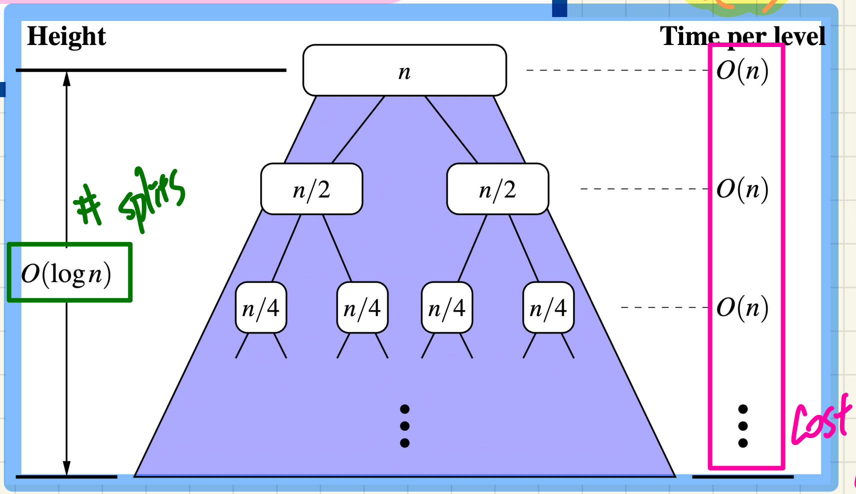
assume: $T(\frac{n}{2})$

$O(n)$

$1 + n + 2 \cdot T(\frac{n}{2})$

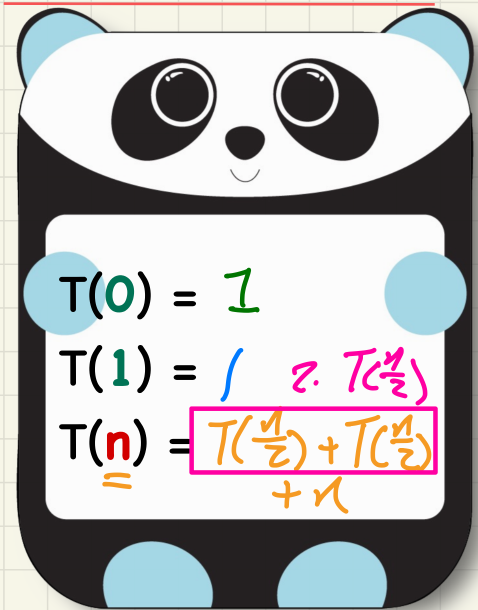
* RT: size(sortedL) + size(sortedR) $= O(n)$





Cost of merge at each level

Running Time as a Recurrence Relation



Running Time: Unfolding Recurrence Relation

$T(0) = 1$

$T(1) = 1$

$T(n) = 2 \cdot T(n/2) + n$

Warm-up: $T\left(\frac{n}{2}\right) = 2 \cdot T\left(\frac{n/2}{2}\right) + \frac{n}{2}$ $\quad n/4$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$$

$$= 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n \qquad \left[4 \cdot T\left(\frac{n}{4}\right) + 2n\right]$$

$$= 2 \cdot \left(2 \cdot \left(2 \cdot T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + \frac{n}{2}\right) + n \qquad \left[8 \cdot T\left(\frac{n}{8}\right) + 3n\right]$$

$2^1, \; 2^2, \; 2^3$

$\vdots$

$$1 = \frac{n}{n} = \frac{n}{2^?} = \frac{n}{2^{\log n}}$$

$$2^? \cdot 1 \; \boxed{T(1)} + ? \cdot n = \underbrace{2^{\log n}}_{n} \cdot 1 + \log n \cdot n = O(n \cdot \log n)$$

$n + n \cdot \log n$

Quick Sort: Ideas

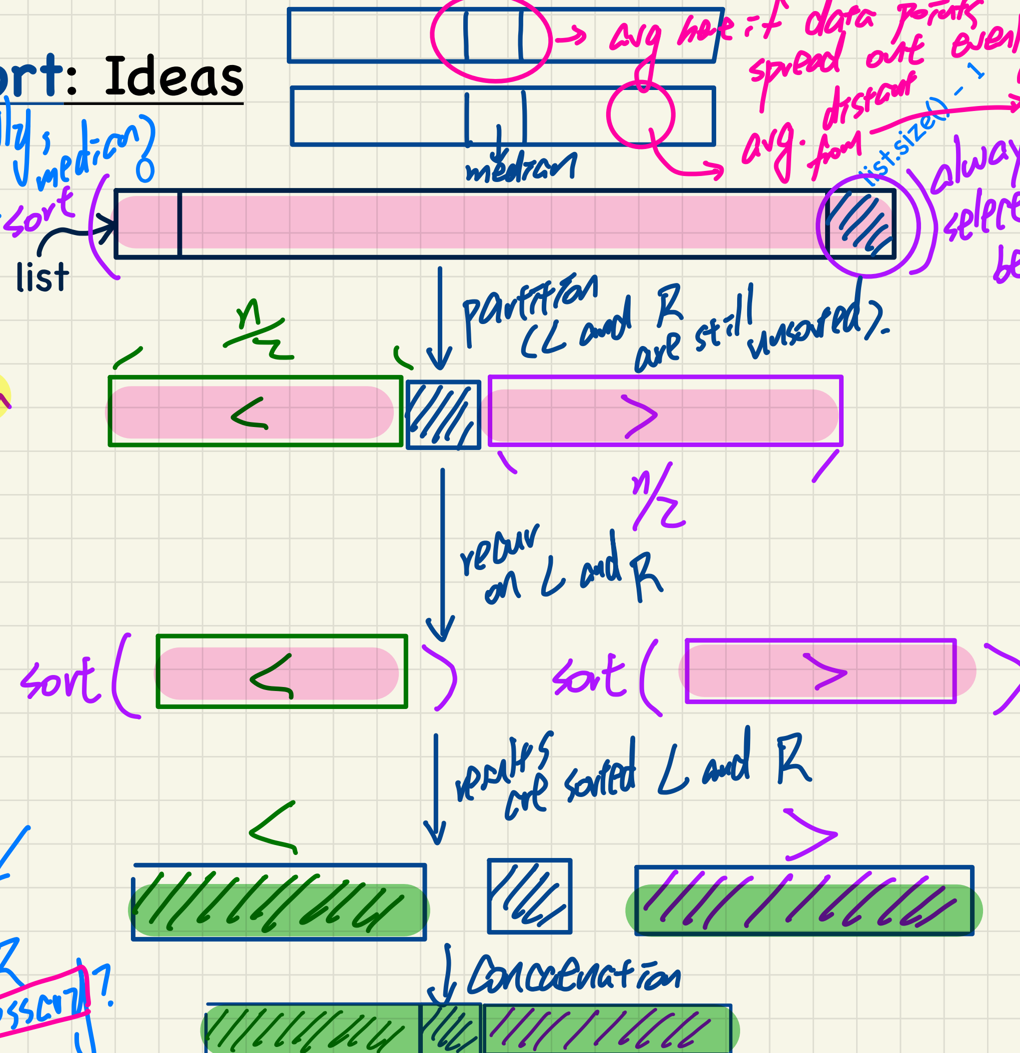
- pivot selection (ideally, the median)
- partition
- concatenation

median of medians algorithm → find median in O(n)

avg here if data points spread out evenly

avg. distance from list.size() - 1 → median if data skewed.

always select pivot being the last element.



Is merging sL and sR $\boxed{\text{necessary}}$? no.

Median of medians algo. : O(n)

↳ as opposed to sort and select.

Quick Sort in Java

T(n)

```
public List<Integer> sort(List<Integer> list) {
  List<Integer> sortedList;
  if(list.size() == 0) { sortedList = new ArrayList<>(); }
  else if(list.size() == 1) {
    sortedList = new ArrayList<>(); sortedList.add(list.get(0)); }
  else {
    int pivotIndex = list.size() - 1;
    int pivotValue = list.get(pivotIndex);
    List<Integer> left = allLessThanOrEqualTo (pivotIndex, list);
    List<Integer> right = allLargerThan (pivotIndex, list);
    List<Integer> sortedLeft = sort(left);
    List<Integer> sortedRight = sort(right);
    sortedList = new ArrayList<>();
    sortedList.addAll(sortedLeft);
    sortedList.add(pivotValue);
    sortedList.addAll(sortedRight);
  }
  return sortedList;
}
```

O(1)

→ ideally, this is close to the index of median

O(n)

$T(\frac{n}{2})$

$T(\frac{n}{2})$

→ only if pivot ≈ median

concat. O(1)

```
List<Integer> allLessThanOrEqualTo(int pivotIndex, List<Integer> list)
  List<Integer> sublist = new ArrayList<>();
  int pivotValue = list.get(pivotIndex);
  for(int i = 0; i < list.size(); i ++) {
    int v = list.get(i);
    if(i != pivotIndex && v <= pivotValue) { sublist.add(v); }
  }
  return sublist;
}
List<Integer> allLargerThan(int pivotIndex, List<Integer> list) {
  List<Integer> sublist = new ArrayList<>();
  int pivotValue = list.get(pivotIndex);
  for(int i = 0; i < list.size(); i ++) {
    int v = list.get(i);
    if(i != pivotIndex && v > pivotValue) { sublist.add(v); }
  }
  return sublist;
}
```

list

85	24	63	45	17	31	96	50

pivot

<50

left

24	45	17	31

>50

right

85	63	96

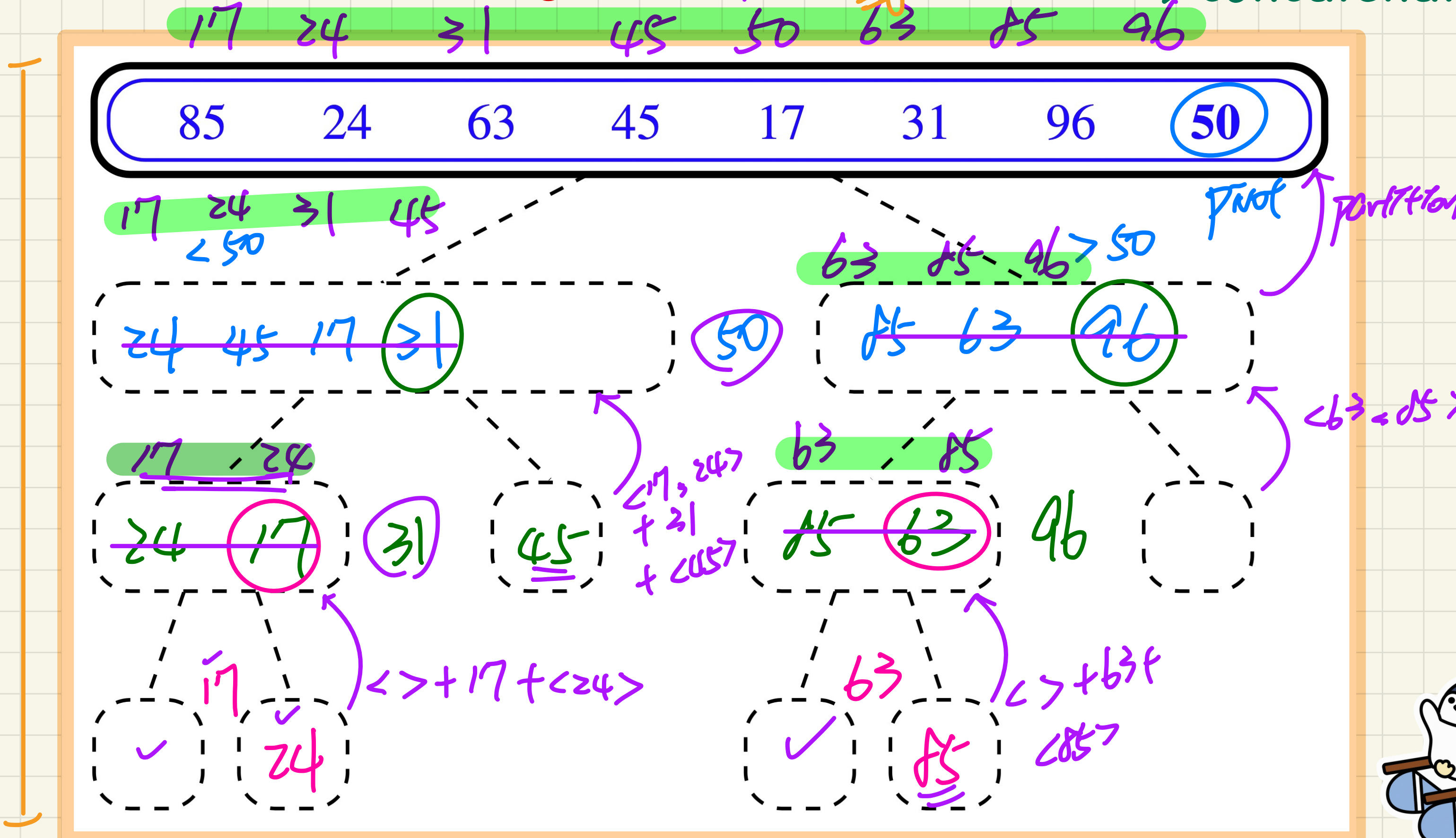
not sorted yet

Quick Sort: Tracing

best case for # partitions: log n

→ split

→ concatenate



Quick Sort: Worst-Case Running Time

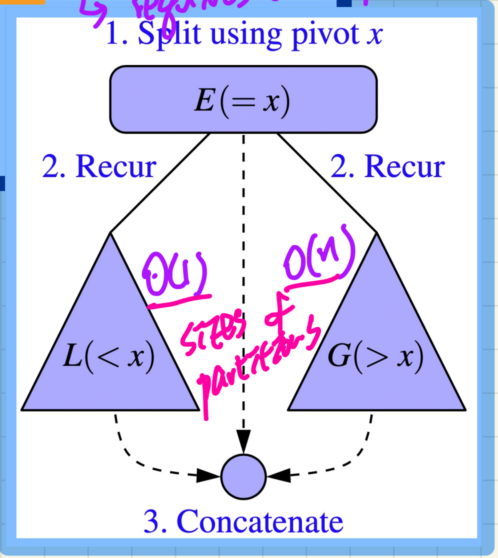
```java
public List<Integer> sort(List<Integer> list) {
  List<Integer> sortedList;
  if(list.size() == 0) { sortedList = new ArrayList<>(); }
  else if(list.size() == 1) {
    sortedList = new ArrayList<>(); sortedList.add(list.get(0)); }
  else {
    int pivotIndex = list.size() - 1;
    int pivotValue = list.get(pivotIndex);
    List<Integer> left = allLessThanOrEqualTo(pivotIndex, list);
    List<Integer> right = allLargerThan(pivotIndex, list);
    List<Integer> sortedLeft = sort(left);
    List<Integer> sortedRight = sort(right);
    sortedList = new ArrayList<>();
    sortedList.addAll(sortedLeft);
    sortedList.add(pivotValue);
    sortedList.addAll(sortedRight);
  }
  return sortedList;
}
```

$T(n-1)$

$O(n)$

$T(1)$

not balanced ↳ requires $O(n)$ partitions



9 7 5 3 1

9 7 5 3

9 7 5

9 7

9

4 partitions

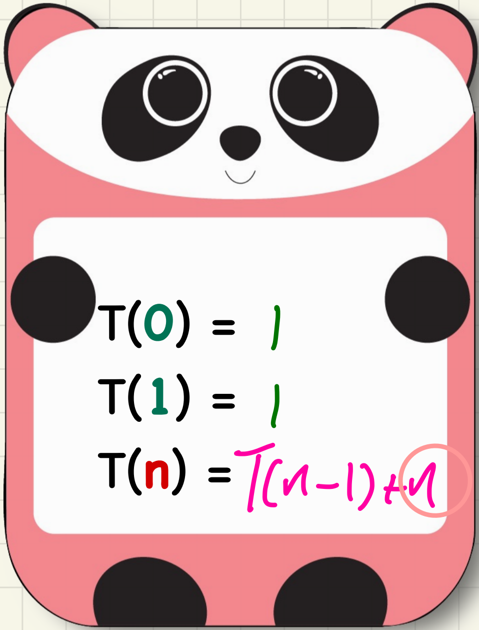
n-1

$O(n)$

≈ selection insertion sort

overall RT is $O(n^2)$

Running Time as a Recurrence Relation



$T(0) = 1$

$T(1) = 1$

$T(n) = T(n-1) + n$

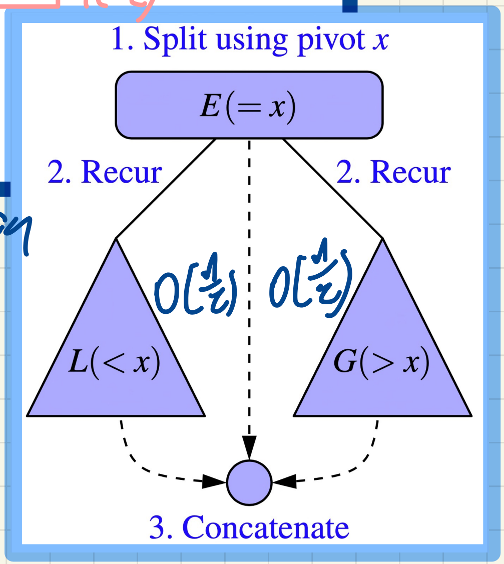
Quick Sort: Best-Case Running Time

```
public List<Integer> sort(List<Integer> list) {
  List<Integer> sortedList;
  if(list.size() == 0) { sortedList = new ArrayList<>(); }
  else if(list.size() == 1) {
    sortedList = new ArrayList<>(); sortedList.add(list.get(0)); }
  else {
    int pivotIndex = list.size() - 1;
    int pivotValue = list.get(pivotIndex);
    List<Integer> left = allLessThanOrEqualTo(pivotIndex, list);
    List<Integer> right = allLargerThan(pivotIndex, list);
    List<Integer> sortedLeft = sort(left);
    List<Integer> sortedRight = sort(right);
    sortedList = new ArrayList<>();
    sortedList.addAll(sortedLeft);
    sortedList.add(pivotValue);
    sortedList.addAll(sortedRight);
  }
  return sortedList;
}
```

$O(n)$

$T(\frac{n}{2})$

$T(\frac{n}{2})$



Running Time as a Recurrence Relation

$T(0) = 1$

$T(1) = 1$

$T(n) = 2 \cdot T(\frac{n}{2}) + n$

$O(n \cdot \log n)$

$\approx$ merge sort