Monday Dec. 3

Lecture 24

Review Sessions for Exam

1pm ~ 3pm LAS C

Monday (Dec. 10)

Wednesday (Dec. 12)

Confirm your attendance on Moodle!

Tower of Hanoi: Strategy

45 10

Consider 3 disks A < B < C

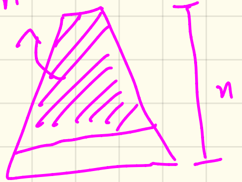
move A B C from p1 to p3

n-1 disks

ori

d

des

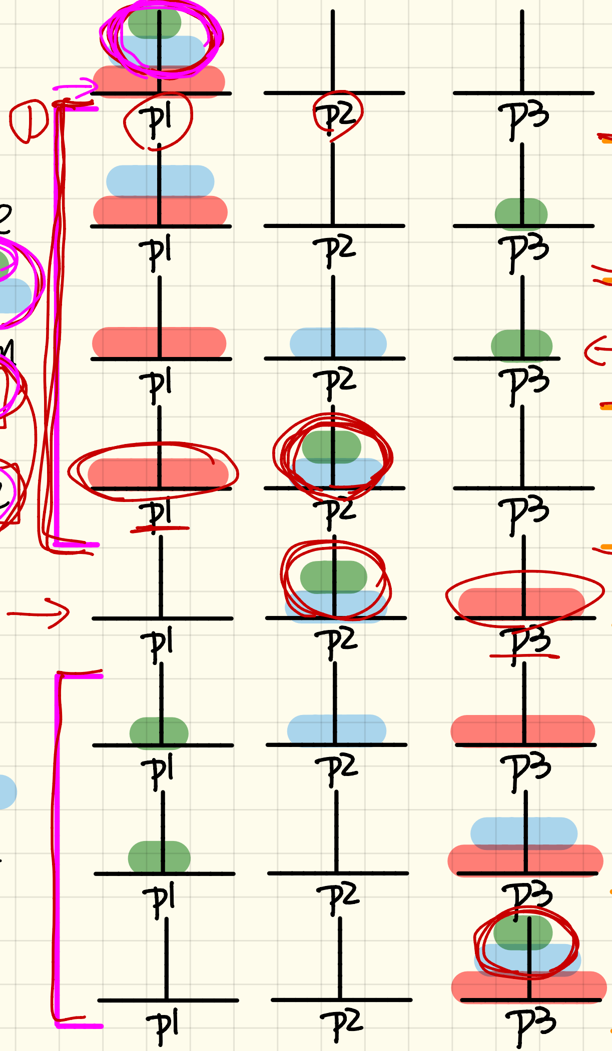


n disks

p2

move from p1 to p2

move from p2 to p3



move from p1 to p3

move from p3 to p2

move from p2 to p1

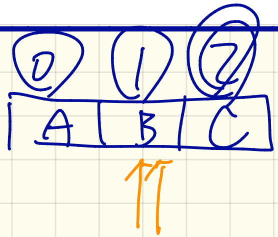
move from p1 to p3

Tower of Hanoi : Java

1 2 3
1 3 2

```
void towerOfHanoi(String[] disks) {
  tohHelper (disks, 0, disks.length - 1, 1, 3);
}
void tohHelper(String[] disks, int from, int to, int ori, int des){
  if(from > to) { }
  else if(from == to) {
    print("move " + disks[to] + " from " + ori + " to " + des);
  }
  else {
    int intermediate = 6 - ori - des;
    tohHelper (disks, from, to - 1, ori, intermediate);
    print("move " + disks[to] + " from " + ori + " to " + des);
    tohHelper (disks, from, to - 1, intermediate, des);
  }
}
```

Say disks = {A,B,C}.
Consider towerOfHoni(disks) which calls:
tohHelper(disks, 0, disks.length - 1, 1, 3)



Tower of Hanoi: Tracing

Say *ds* (disks) is $\{A, B, C\}$, where $A < B < C$.

inter: p3

inter: p2

$$tohH(ds, \underbrace{0, 2}_{\{A,B,C\}}, p1, p3) = \begin{cases} tohH(ds, \underbrace{0, 1}_{\{A,B\}}, p1, p2) = \begin{cases} tohH(ds, \underbrace{0, 0}_{\{A\}}, p1, p3) = \left\{ \texttt{Move } \mathbf{A}\texttt{: p1 to p3} \right. \\ \texttt{Move } \mathbf{B}\texttt{: p1 to p2} \\ tohH(ds, \underbrace{0, 0}_{\{A\}}, p3, p2) = \left\{ \texttt{Move } \mathbf{A}\texttt{: p3 to p2} \right. \end{cases} \\ \texttt{Move } \mathbf{C}\texttt{: p1 to p3} \\ tohH(ds, \underbrace{0, 1}_{\{A,B\}}, p2, p3) = \begin{cases} tohH(ds, \underbrace{0, 0}_{\{A\}}, p2, p1) = \left\{ \texttt{Move } \mathbf{A}\texttt{: p2 to p1} \right. \\ \texttt{Move } \mathbf{B}\texttt{: p2 to p3} \\ tohH(ds, \underbrace{0, 0}_{\{A\}}, p1, p3) = \left\{ \texttt{Move } \mathbf{A}\texttt{: p1 to p3} \right. \end{cases} \end{cases}$$

$\{A, B, C\}$: 0 … 2

dfs

Tower of Hanoi: Running Time

$T(1) \to n-(n-1)$

```
void towerOfHanoi(String[] disks) {
  tohHelper(disks, 0, disks.length - 1, 1, 3);
}
void tohHelper(String[] disks, int from, int to, int ori, int des){
  if(from > to) { }
  else if(from == to) {
    print("move " + disks[to] + " from " + ori + " to " + des);
  }
  else {
    int intermediate = 6 - ori - des;
    tohHelper(disks, from, to - 1, ori, intermediate);
    print("move " + disks[to] + " from " + ori + " to " + des);
    tohHelper(disks, from, to - 1, intermediate, des);
  }
}
```

1 disk

base case

n disks

n-1 disks

recursive

n-1 disks

formulate

$T(1) = 1$

$T(n) = 2 * T(n - 1) + 1$

$T(n) = 2 * T(n-1) + 1$

$= 2 * (2 * T(n-2) + 1) + 1$

$= 2 * (2 * (2 * T(n-3) + 1) + 1) + 1$

$= \ldots$ $n-1$ $n-(n-1)$ $n-1$

$= 2 * (2 * (\ldots T(1)) + 1) + 1 \ldots) + 1$

$O(2^n) \leftarrow 2^{n-1} + (n-1)$

Binary Search: Running Time

1024 8 4

assume $n = 2^i$

$\log 1024$

$1024 = 2^{10}$ $\quad n = 2^{\log n}$

```
boolean binarySearch(int[] sorted, int key) {
  return binarySearchHelper (sorted, 0, sorted.length - 1, key);
}
boolean binarySearchHelper (int[] sorted, int from, int to, int key
  if (from > to) { /* base case 1: empty range */
    return false; }
  else if(from == to) { /* base case 2: range of one element */
    return sorted[from] == key; }
  else {
    int middle = (from + to) / 2;
    int middleValue = sorted[middle];
    if(key < middleValue) {
      return binarySearchHelper (sorted, from, middle - 1, key);
    }
    else if (key > middleValue) {
      return binarySearchHelper (sorted, middle + 1, to, key);
    }
    else { return true; }
  }
}
```

calc. mid. pos $O(1)$

formulate

$$T(0) = 1$$
$$T(1) = 1$$
$$T(n) = T(n/2) + 1$$

n/4

mid. pos.

L or R

$$T(n) = T\left(\frac{n}{2}\right) + 1 \qquad \frac{n}{2^1}$$
$$= \left(T\left(\frac{n}{4}\right) + 1\right) + 1 \qquad \frac{n}{2^2}$$
$$= \left(\left(T\left(\frac{n}{8}\right) + 1\right) + 1\right) + 1$$
$$= \ldots$$
$$= T(1) + 1 + \ldots + 1$$

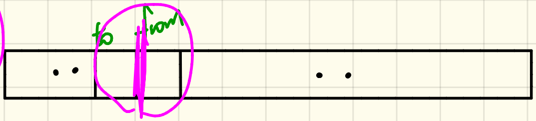
$\frac{n}{2^?}$ $\quad 1$ $\quad \log n$

$O(\log n) \leftarrow 1 + \log n$

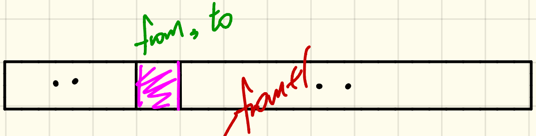
Correctness Proofs: Ideas

```
boolean allPositive(int[] a) { return allPosH(a, 0, a.length - 1); }
boolean allPosH (int[] a, int from, int to) {
  if (from > to) { return true; }
  else if (from == to) { return a[from] > 0; }
  else { return a[from] > 0 && allPosH(a, from + 1, to); } }
```

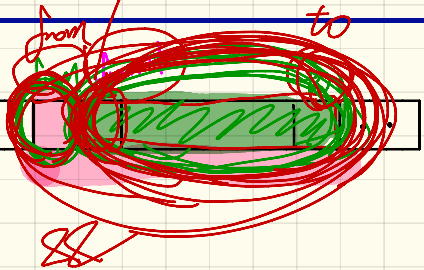
Base Case:
Empty Array



Base Case:
Array of size 1



Recursive Case:



Correctness Proofs

```
1 boolean allPositive(int[] a) { return allPosH (a, 0, a.length - 1); }
2 boolean allPosH (int[] a, int from, int to) {
3   if (from > to) { return true; }
4   else if(from == to) { return a[from] > 0; }
5   else { return a[from] > 0 && allPosH (a, from + 1, to); } }
```

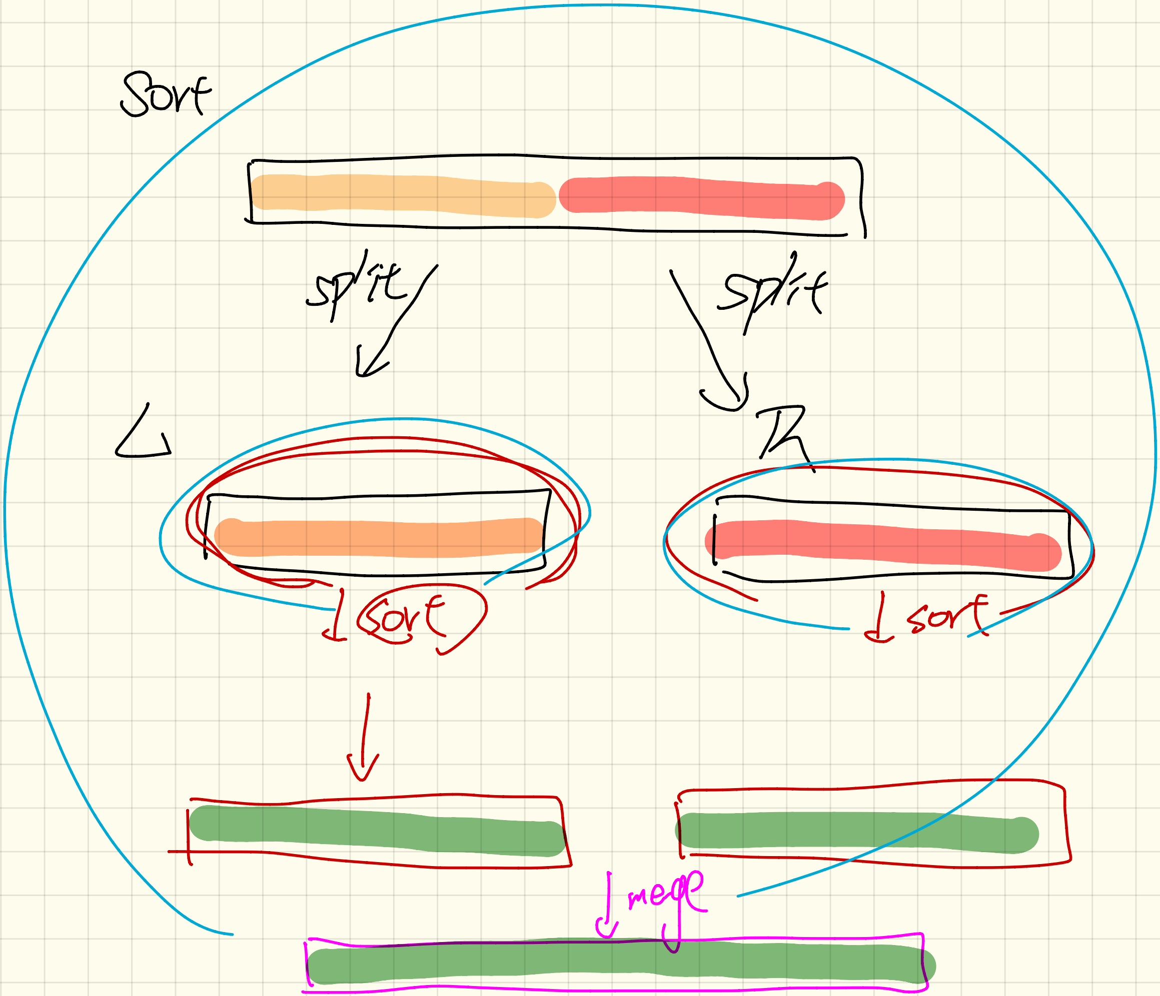
- Via mathematical induction, prove that `allPosH` is correct:

  Base Cases

  - In an empty array, there is no non-positive number ∴ result is ***true***. **[L3]**
  - In an array of size 1, the only one elements determines the result. **[L4]**

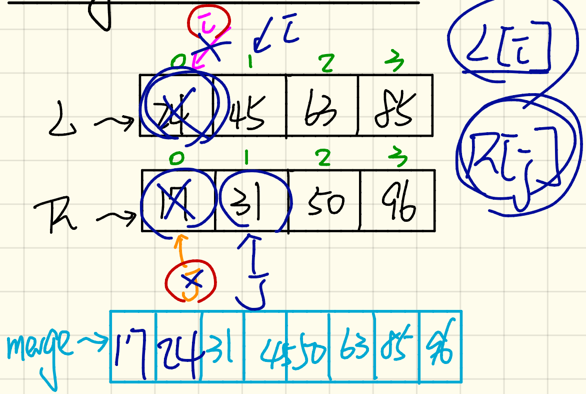
  Inductive Cases

  - **Inductive Hypothesis**: `allPosH(a, from + 1, to)` returns ***true*** if a[from + 1], a[from + 2], . . . , a[to] are all positive; ***false*** otherwise.
  - `allPosH(a, from, to)` should return ***true*** if: **1)** a[from] is positive; and **2)** a[from + 1], a[from + 2], . . . , a[to] are all positive.
  - By *I.H.*, result is $a[from] > 0 \wedge$ `allPosH(a, from + 1, to)`. **[L5]**
- `allPositive(a)` is correct by invoking `allPosH(a, 0, a.length - 1)`, examining the entire array. **[L1]**



Sort
split
split
L
R
sort
sort
merge

Merge Sort : Java




```java
/* Assumption:  L and R are both already sorted.  */
private List<Integer> merge(List<Integer> L, List<Integer> R) {
 List<Integer> merge = new ArrayList<>();
 if(L.isEmpty()||R.isEmpty()) { merge.addAll(L); merge.addAll(R); }
 else {
  int i = 0;
  int j = 0;
  while(i < L.size() && j < R.size()) {
   if(L.get(i) <= R.get(j)) { merge.add(L.get(i)); i ++; }
   else { merge.add(R.get(j)); j ++; }
  }
  /* If i >= L.size(), then this for loop is skipped. */
  for(int k = i; k < L.size(); k ++) { merge.add(L.get(k)); }
  /* If j >= R.size(), then this for loop is skipped. */
  for(int k = j; k < R.size(); k ++) { merge.add(R.get(k)); }
 }
 return merge;
}
```

```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 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;
}
```

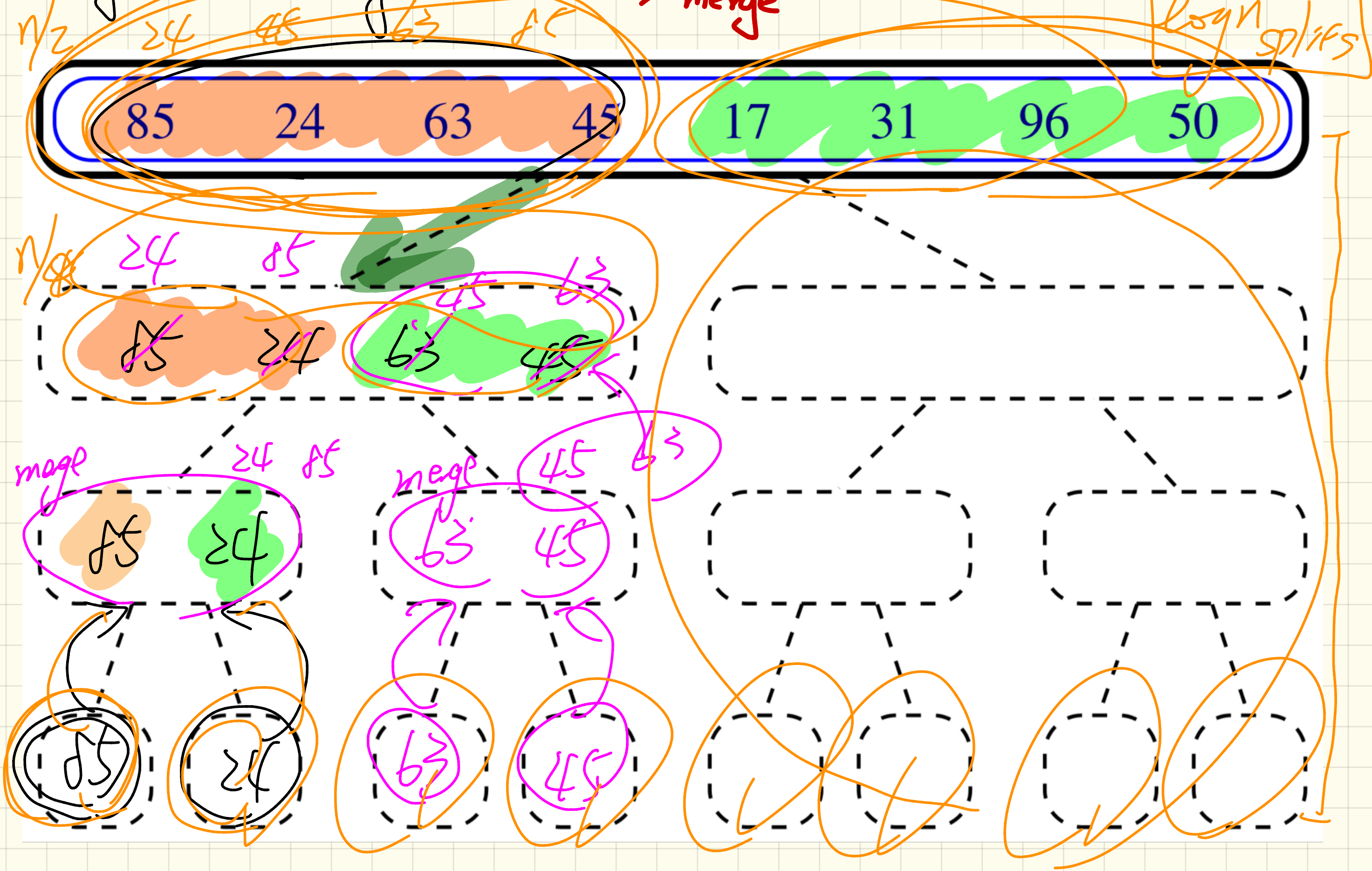
Merge Sort : Tracing

→ split

→ merge

n → n/2 → n/4 → .. 1

log n splits



Merge Sort : Running Time

$n \cdot \log n$