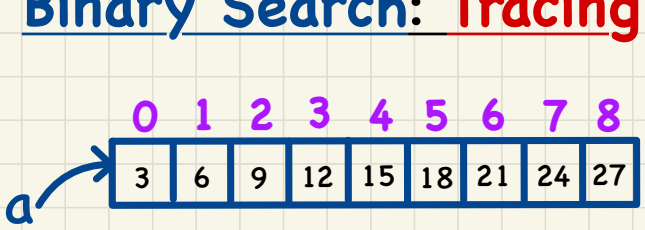
Binary Search: Tracing

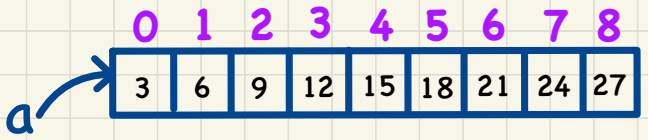


search(a,18)

binarySearchH(a,0,8,18)

binarySearchH(a,5,8,18)

binarySearchH(a,5,5,18)



search(a,7)

binarySearchH(a,0,8,7)

binarySearchH(a,0,3,7)

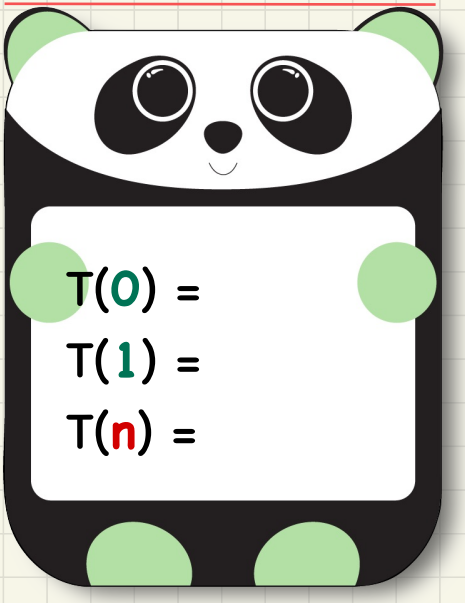
binarySearchH(a,2,3,7)

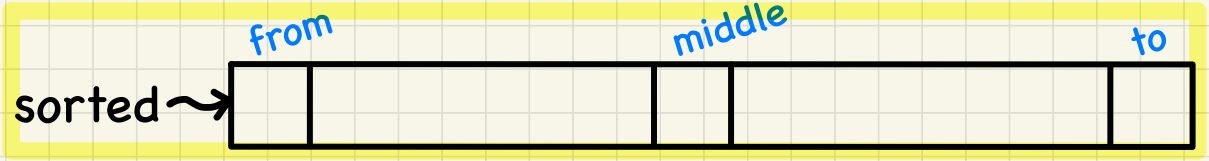
binarySearchH(a,2,1,7)

Binary Search: Running Time

```
boolean binarySearch(int[] sorted, int key) {
  return binarySearchH(sorted, 0, sorted.length - 1, key);
}
boolean binarySearchH(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 binarySearchH(sorted, from, middle - 1, key);
    }
    else if (key > middleValue) {
      return binarySearchH(sorted, middle + 1, to, key);
    }
    else  { return true; }
  }
}
```

Running Time as a Recurrence Relation





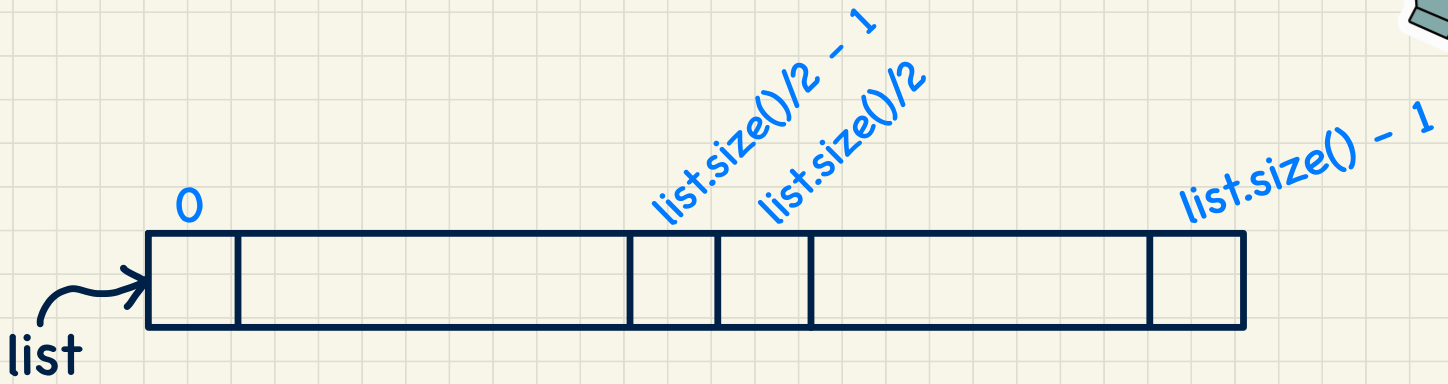
Running Time: Unfolding Recurrence Relation

$T(0) = 1$

$T(1) = 1$

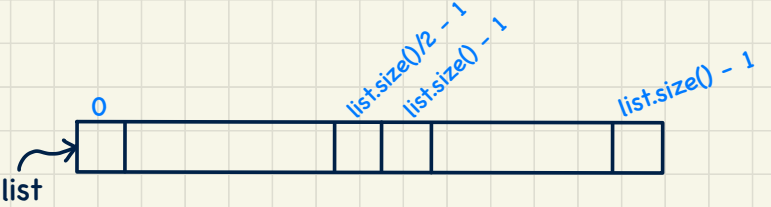
$T(n) = T(n/2) + 1$

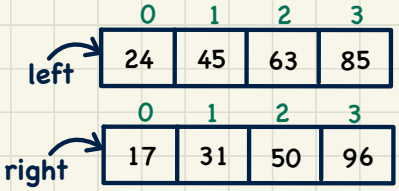
Merge Sort: Ideas

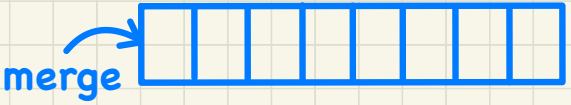


Merge Sort in Java

```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;
}
```








```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;
}
```

Merge Sort: Tracing

→ split
→ merge