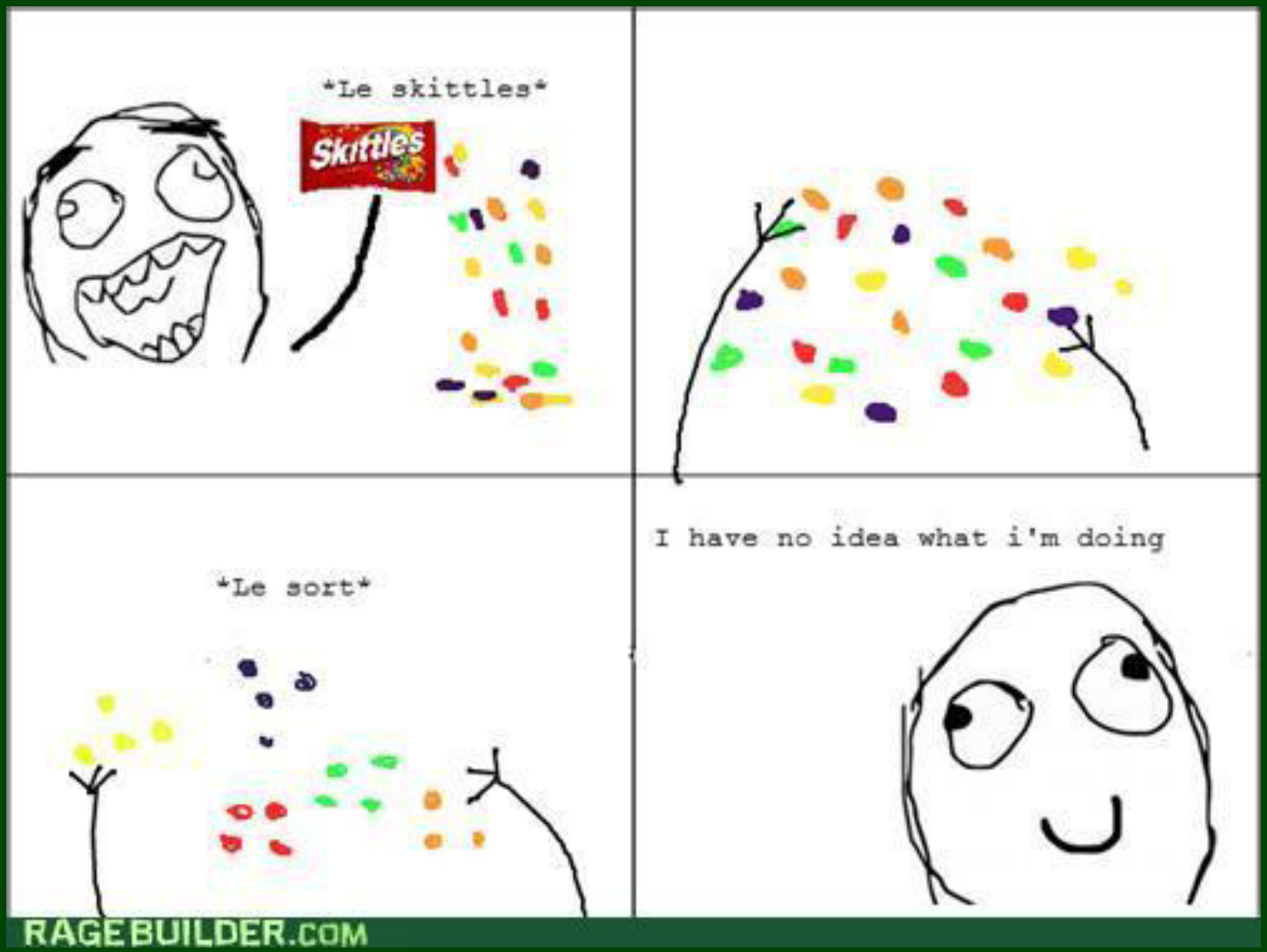
CSE 143

Computer Programming II

Searching and Sorting



Outline

Definition (Formal Language)

A **Formal Language** is a set of words or symbols.

For example:

{1, 2, 3, 4, 5} is a language, and **{hello, goodbye}** is a language.

Definition (Grammar)

A **Grammar** is a set of rules that **generates** a particular language.

Grammars are used to:

- **generate** strings, and to
- **check** if strings are in the language

Definition (Backus-Naur Form (BNF))

BNF is a syntax for describing language grammars in terms of transformation rules, of the form:

$$\langle symbol \rangle ::= \langle expression \rangle \mid \langle expression \rangle \mid ... \mid \langle expression \rangle$$

BNF is made up of two types of symbols:

- **Terminals**: Literals (symbols that are interpreted literally)
- **Non-terminals**: A symbol describing how to generate other symbols based on the rules of the grammar

Example Grammar

$\langle object \rangle$:= $\langle article \rangle$ $\langle thing \rangle$

$\langle article \rangle$:= The | A | That | This

$\langle thing \rangle$:= ball | index card | word | balloon

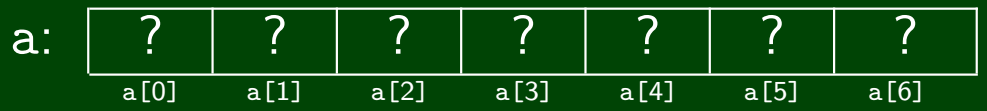
To generate `<object>`s from this grammar, we do the following steps:

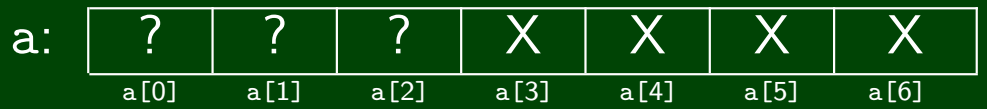
1. Start at `<object>` and look at what to transform to:
   `<article> <thing>`
2. For each non-terminal, look at its rule and choose an option.

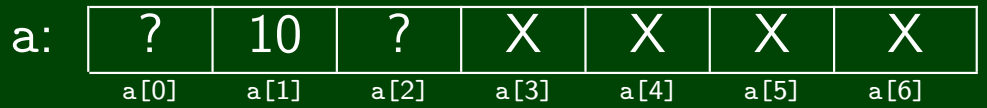
Some `<object>`s in this grammar:

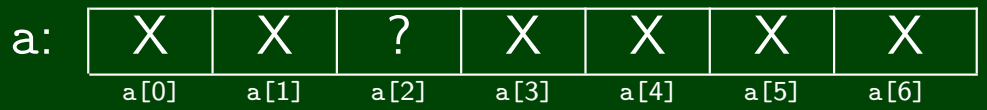
- The ball
- That index card
- The balloon

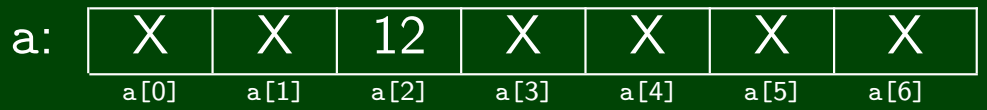
Search for 24 in a











So, 24 is not in a!

Observation

Each time we check an element in the array, Binary Search rules out **half of the remaining possibilities**. If the array is of length n, we can do this $\log_2(n)$ times before getting to one element. So, Binary Search is $\mathcal{O}(\log(n))$.

Using Binary Search in Java

- `Arrays.binarySearch(int[] a, int k);`
- `Collections.binarySearch(int[] a, int k);`

```
 1  private static boolean binarySearch(List<Integer> list, int value,
 2                                      int lo, int hi) {
 3     /* Handle the case where the list is empty */
 4     if (lo == hi) {
 5        return false;
 6     }
 7
 8     /* The base case is when there is only one element left to check */
 9     if (lo == hi - 1) {
10        return list.get(lo) == value;
11     }
12
13     /* Otherwise, figure out of the answer is on the left
14      * or the right, and recurse */
15     int mid = (lo + hi)/2;
16     if (value < list.get(mid)) {
17        /* Since our value is smaller, get rid of the right side
18         * of the array (including mid) */
19        return binarySearch(list, value, lo, mid);
20     }
21     else {
22        /* Since our value is bigger or equal, get rid of everything
23         * smaller than mid */
24        return binarySearch(list, value, mid, hi);
25     }
```

Merging Two Sorted Lists

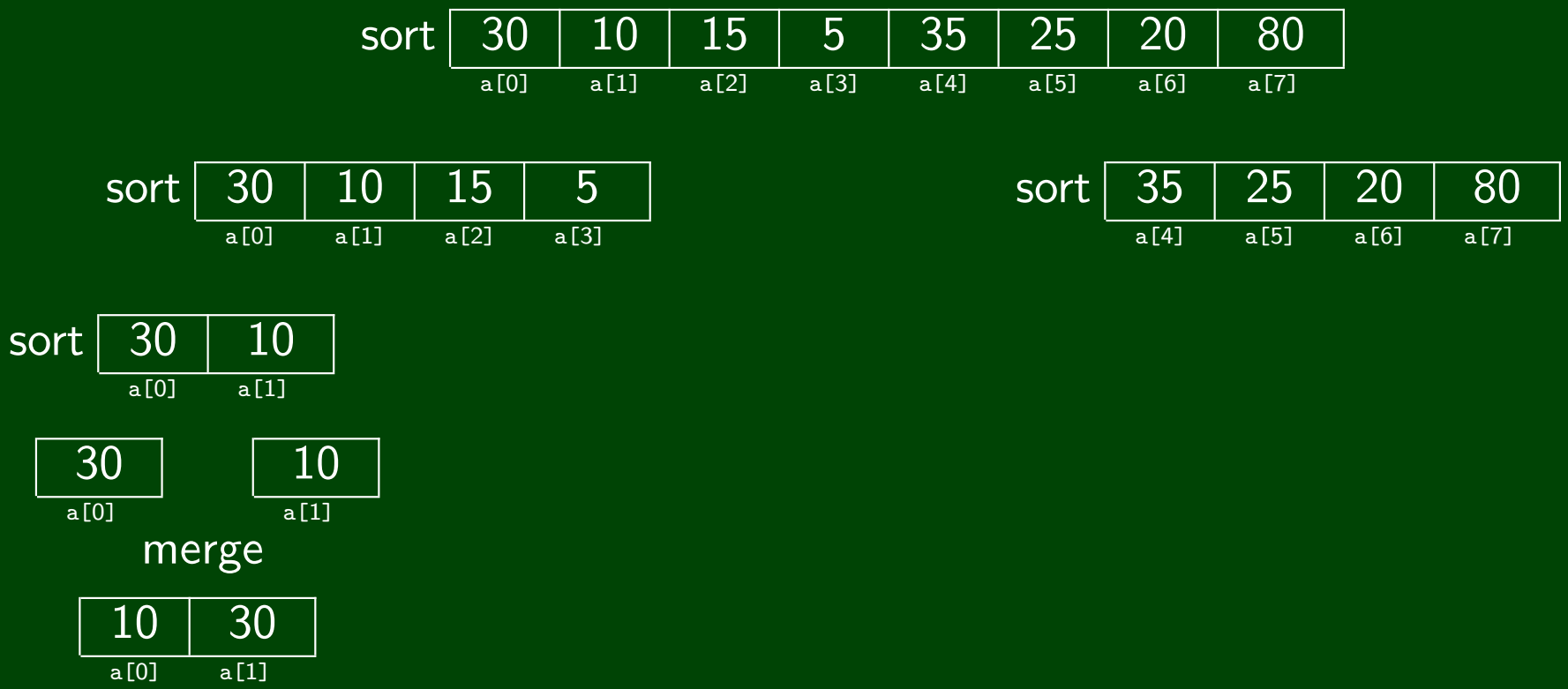
Idea

Keep a finger on the last element we've handled from each list. Figure out which of the elements we're pointing to is smaller. Put the smaller one in the result and move that finger right. Keep on going until both fingers are past the last elements of their lists.

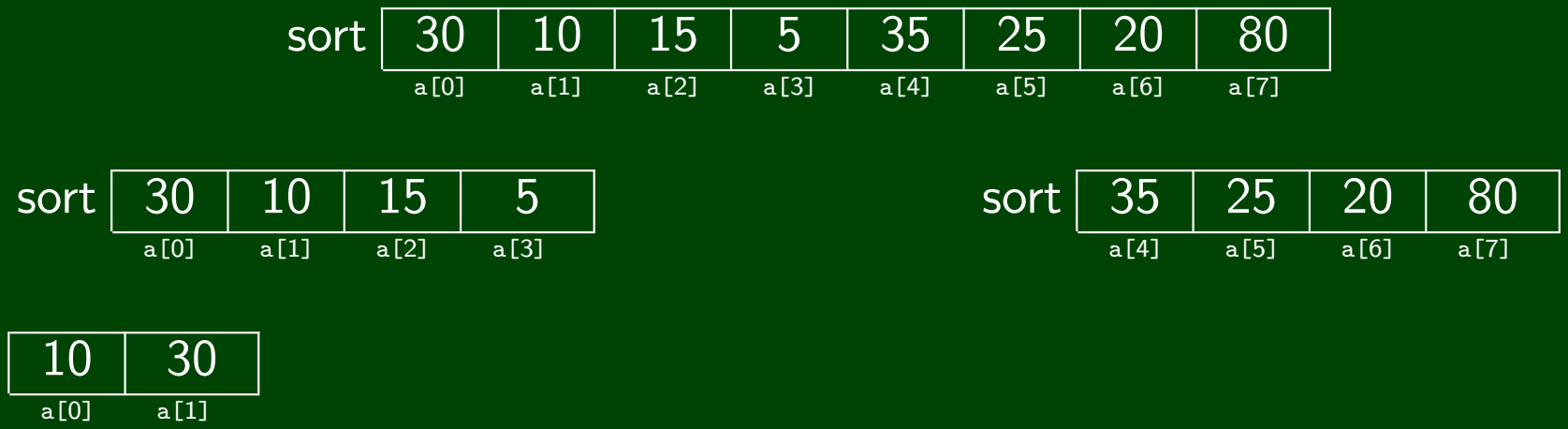
Code

```
1  public static List<Integer> merge(List<Integer> list1, List<Integer> list2) {
2     List<Integer> result = new ArrayList<Integer>(list1.size() + list2.size());
3     int i1 = 0;
4     int i2 = 0;
5     while (i1 + i2 < list1.size() + list2.size()) {
6        Integer a = null, b = null;
7        if (i1 < list1.size()) { a = list1.get(i1); }
8        if (i2 < list2.size()) { b = list2.get(i2); }
9        if (a != null && (b == null || a < b)) {
10          result.add(a);
11          i1++;
12       }
13       else {
14          result.add(b);
15          i2++;
16       }
17    }
18    return result;
19 }
```

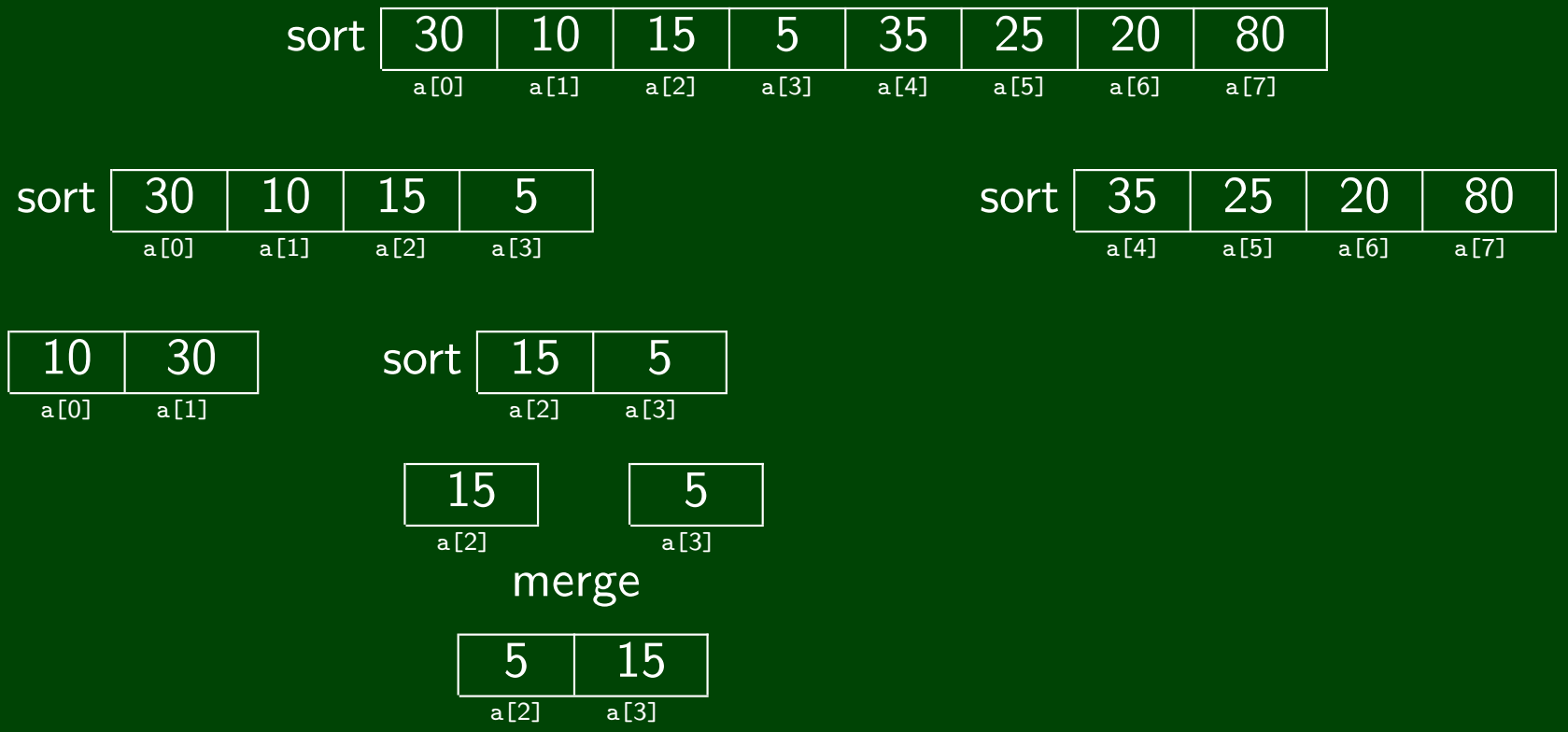
Merge Sort By Picture



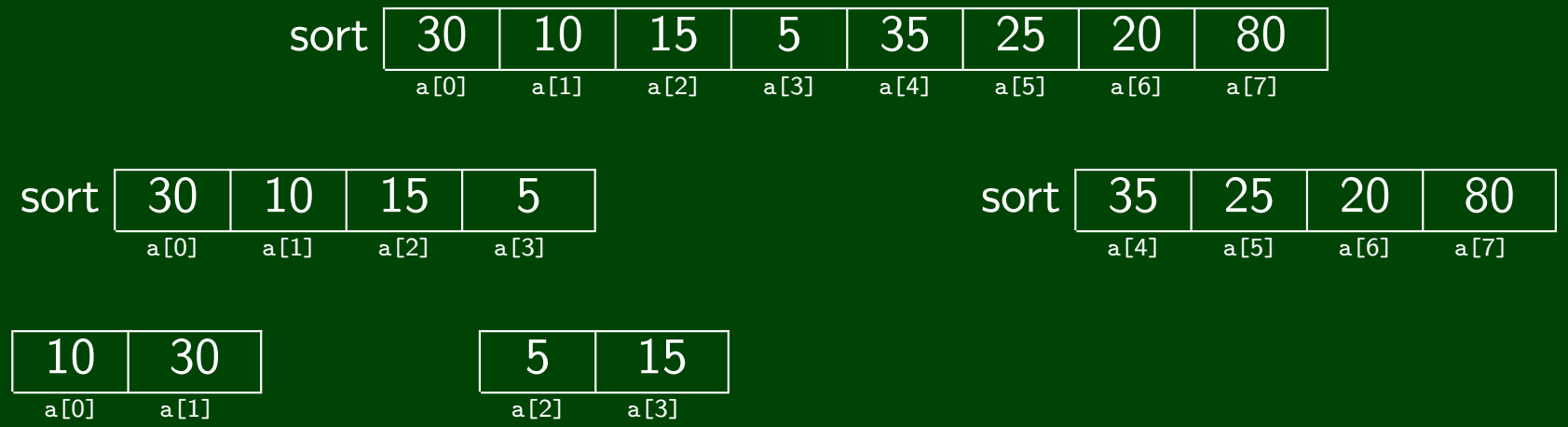
Merge Sort By Picture



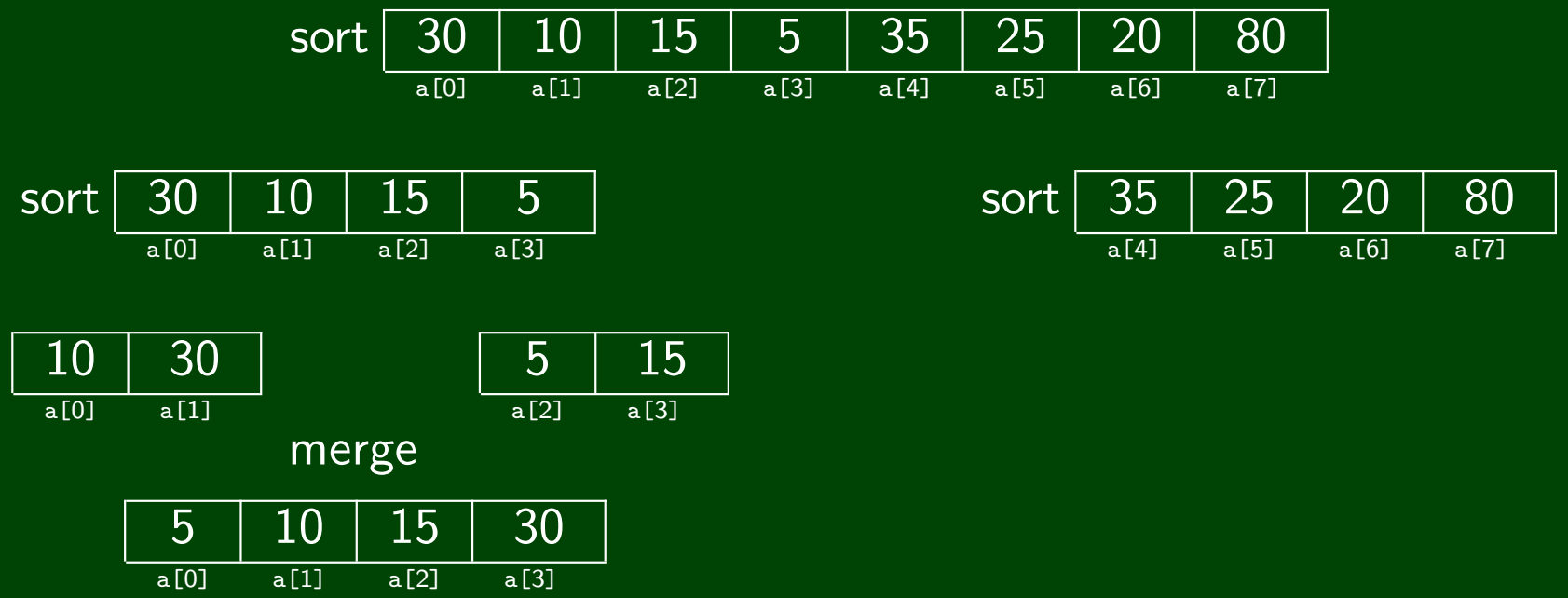
Merge Sort By Picture



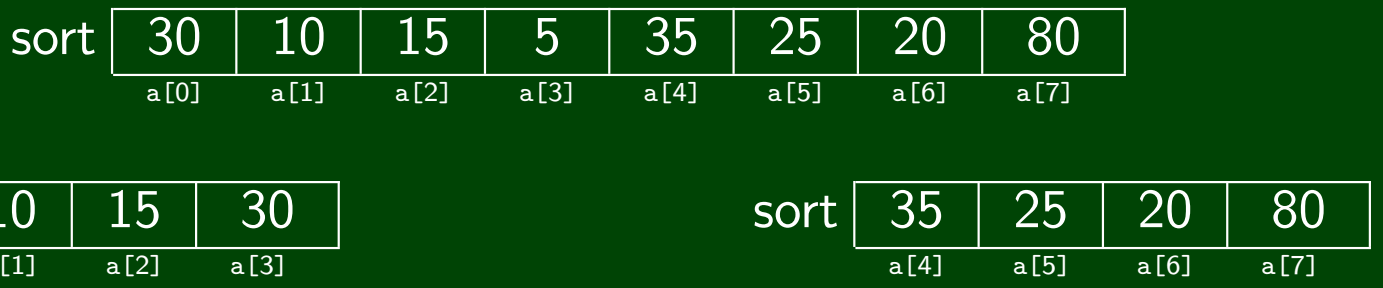
Merge Sort By Picture



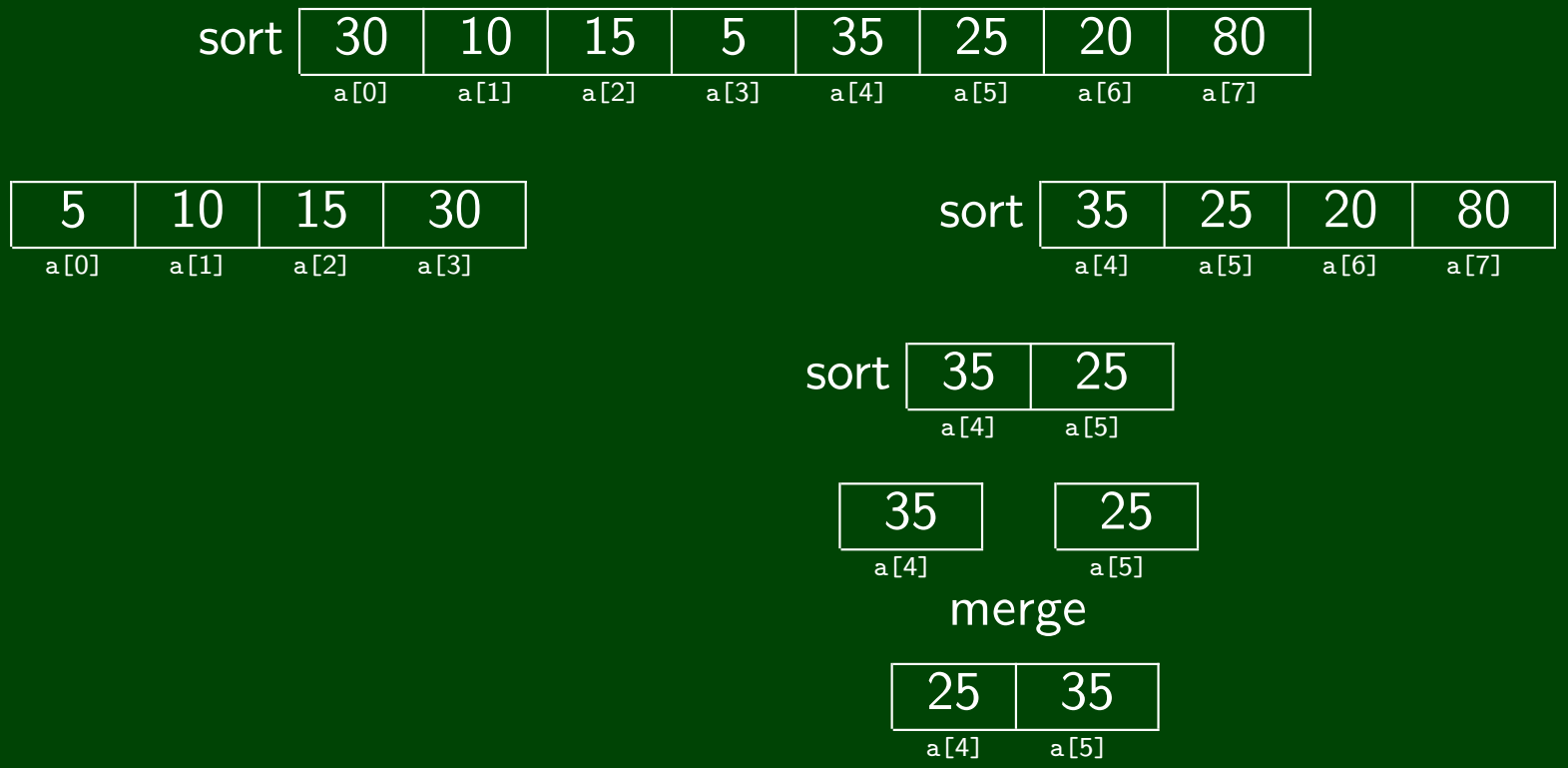
Merge Sort By Picture



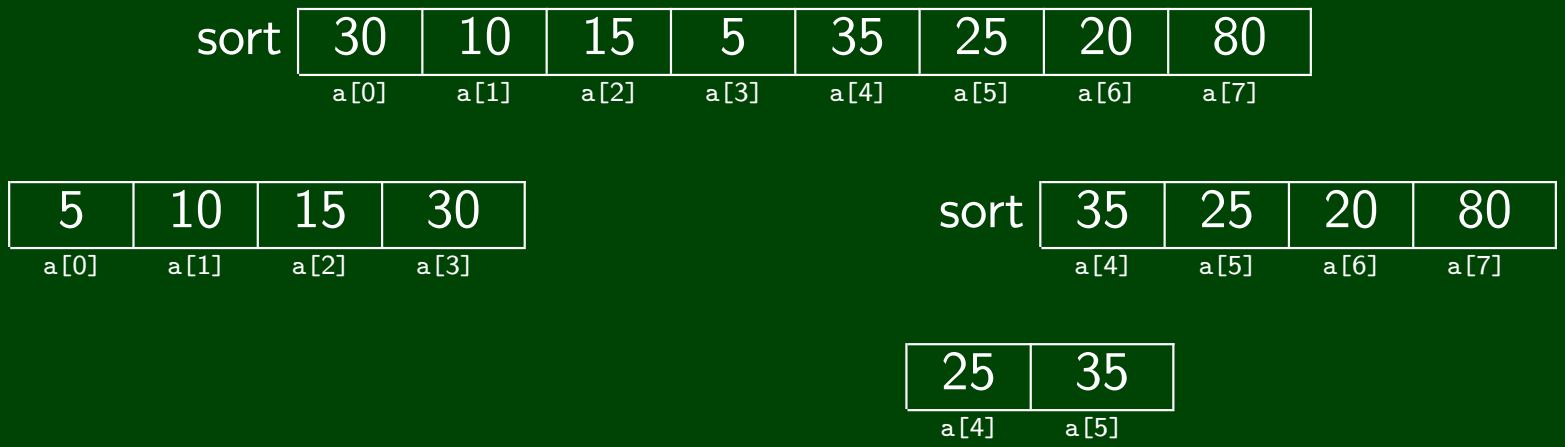
Merge Sort By Picture



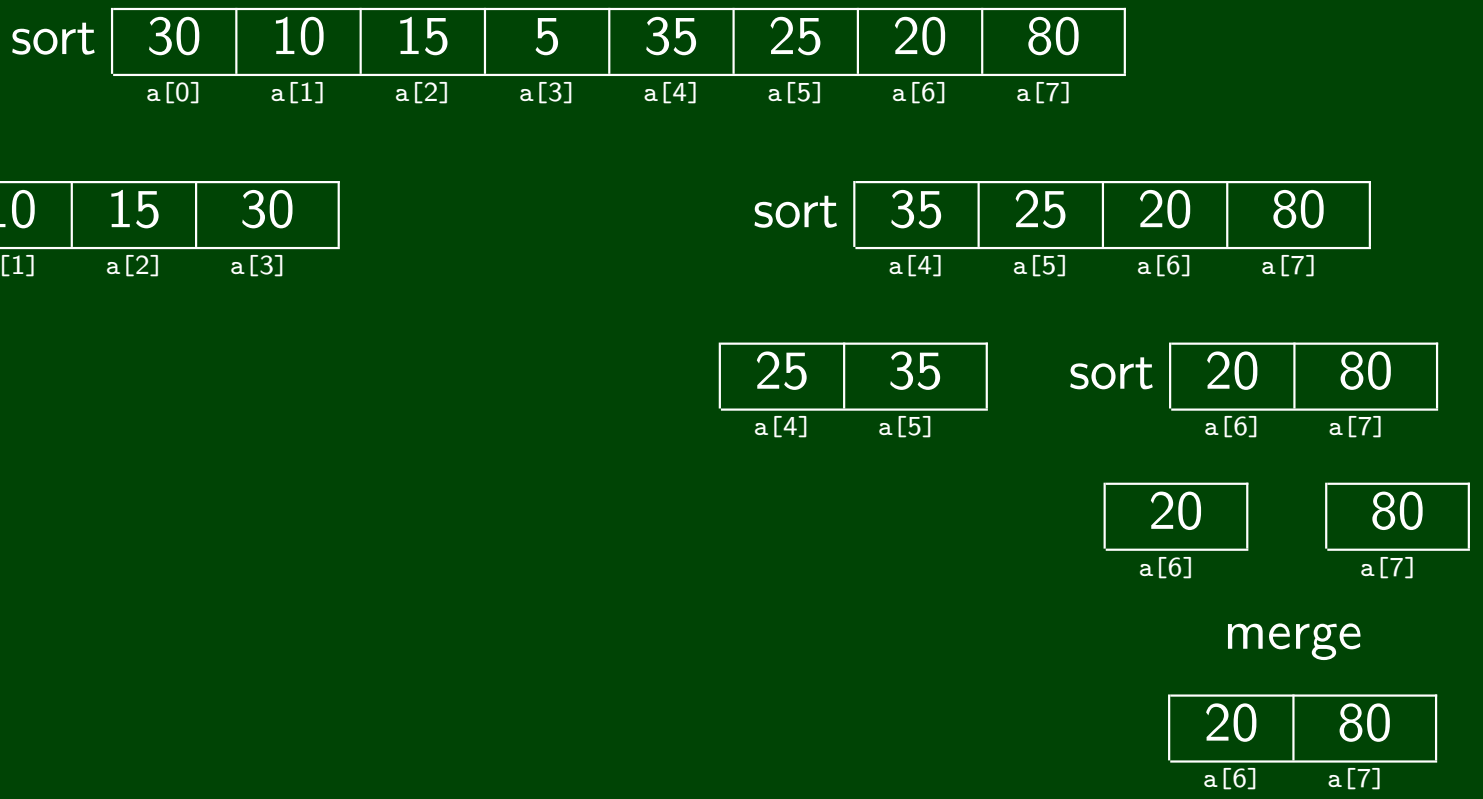
Merge Sort By Picture



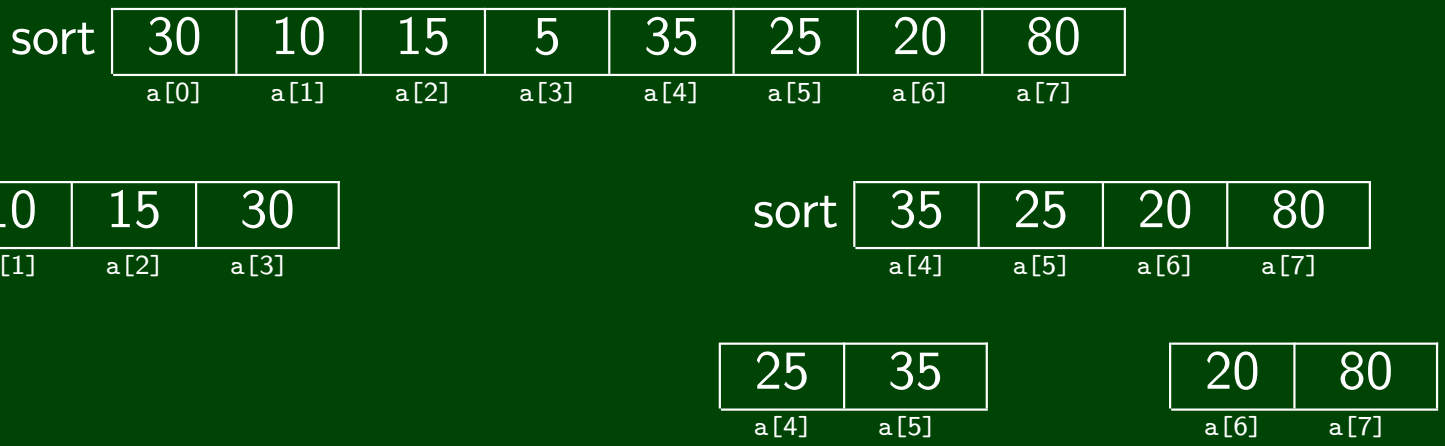
Merge Sort By Picture



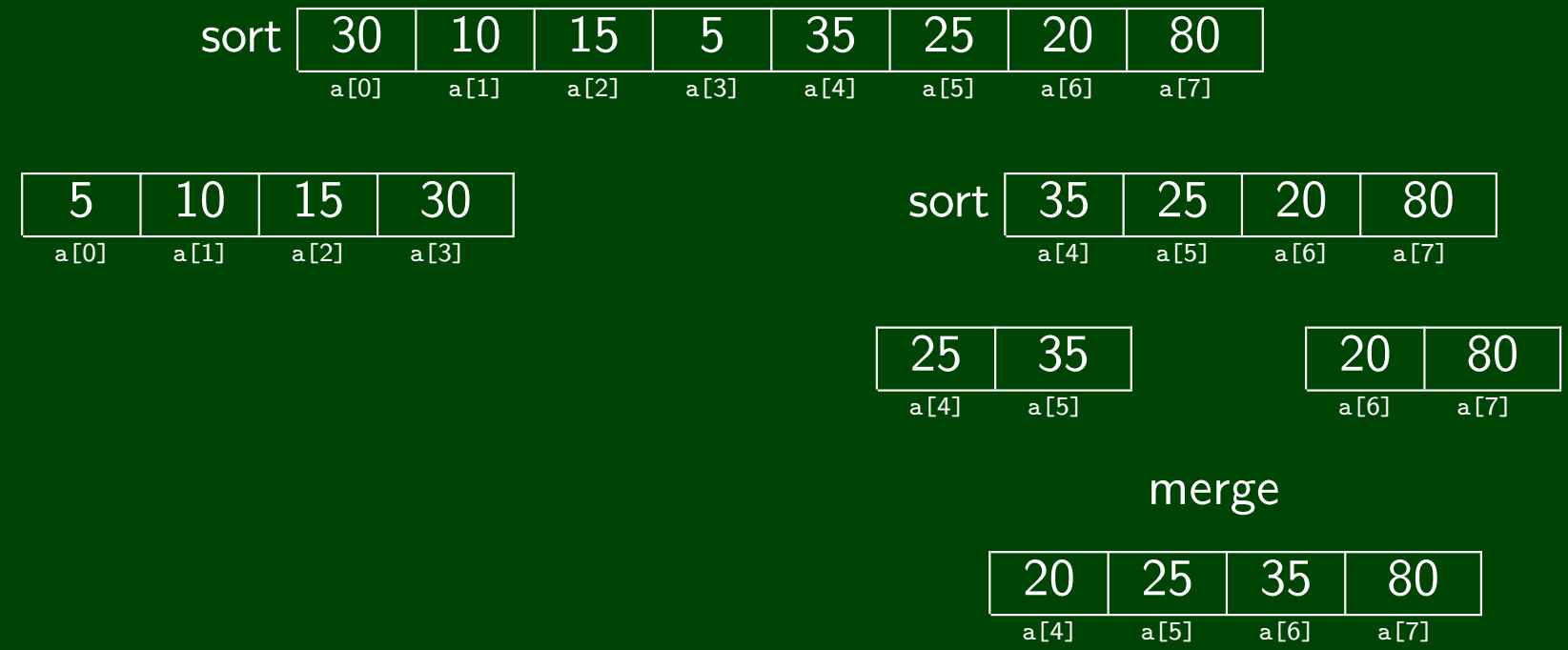
Merge Sort By Picture



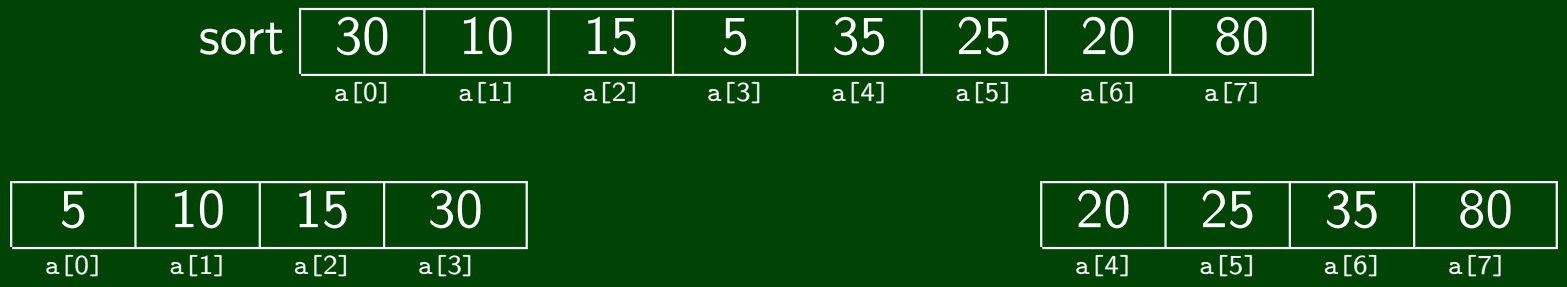
Merge Sort By Picture



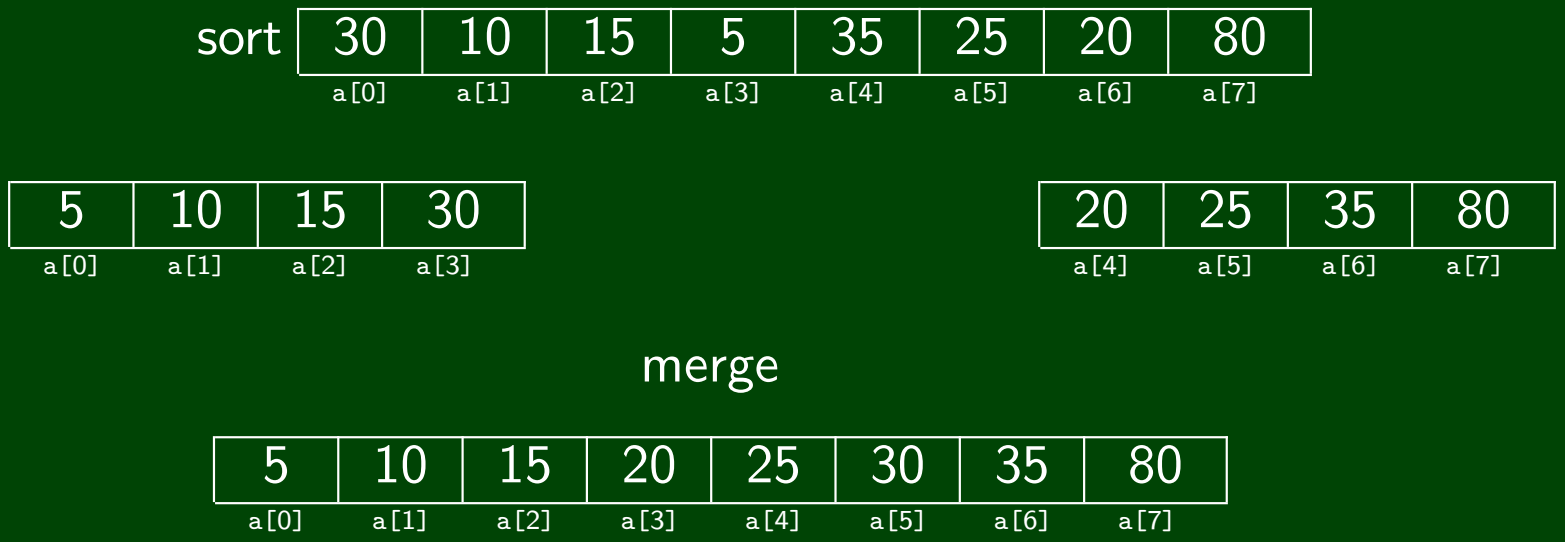
Merge Sort By Picture

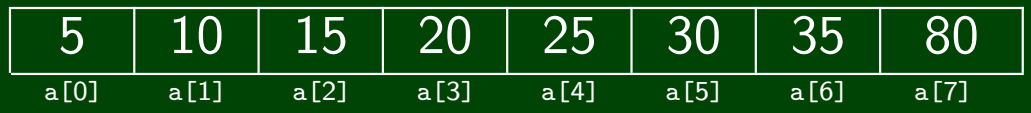


Merge Sort By Picture



Merge Sort By Picture




5
10
15
20
25
30
35
80
a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]

Idea

Break the list into two equal pieces. Sort the left piece; sort the right piece. Use our previous algorithm to merge the two sorted pieces together.

Code

```
1  public static void mergesort(List<Integer> list) {
2     if (list.size() > 1) {
3        int mid = list.size() / 2;
4        List<Integer> left = list.subList(0, mid);
5        List<Integer> right = list.subList(mid, list.size());
6
7        mergesort(left);
8        mergesort(right);
9
10       List<Integer> merged = merge(left, right);
11       for (int i = 0; i < merged.size(); i++) {
12          list.set(i, merged.get(i));
13       }
14    }
15 }
```