Sorting by Divide and Conquer

- Recall the abstract class *ASorter* in the handout.

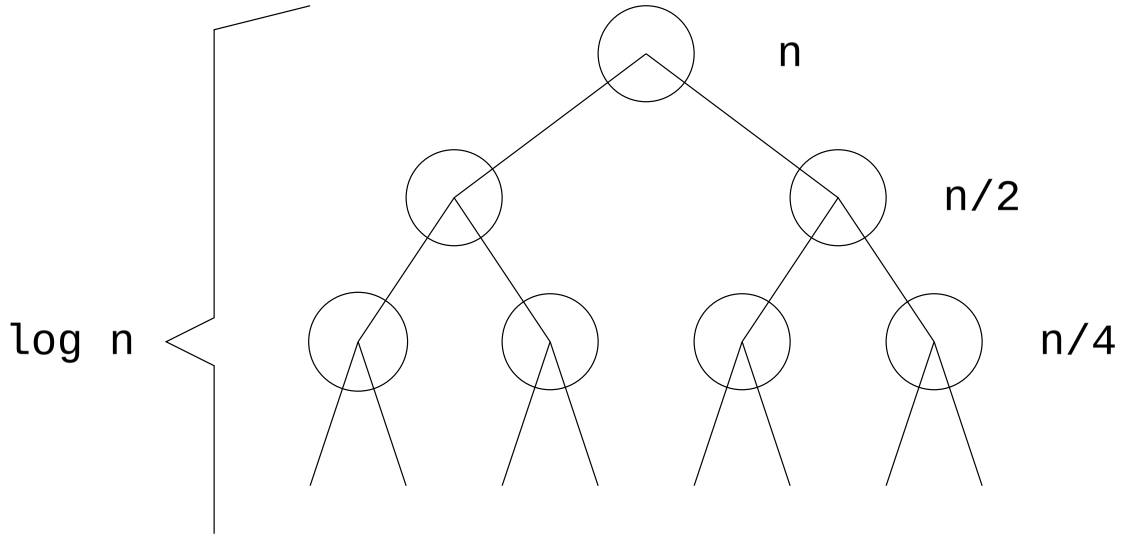
```
public final void sort(int[] A, int lo, int hi)
{
  if (lo < hi) {
    int s = split(A, lo, hi);
    sort(A, lo, s-1);
    sort(A, s, hi);
    join(A, lo, s, hi);
  }
}

public abstract int split(int[] A, int lo, int hi);

public abstract void join(int[] A, int lo, int s, int hi);
```

Merge Sort

- Merge Sort is a *easy-split, hard-join* method.
- Merge Sort takes $O(n\ log\ n)$ steps.
  - Because each `split()` divides the array into two (almost) equal-sized parts, each element is `join()`'ed $log\ n$ times.

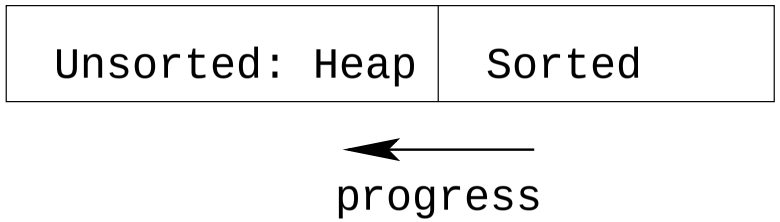


Heap Sort

- Heap Sort is a *hard-split, easy-join* method.
- Think of Heap Sort as an improved (faster) version of Selection Sort.
  - Specifically, `split()`, which finds the minimum (maximum) element in the subarray, is made to run in $O(log\ n)$ steps instead of $O(n)$ steps, where n is the subarray length.
    - Since `split()` is performed n times, where n is the (overall) array length, Heap Sort takes $O(n\ log\ n)$ steps.

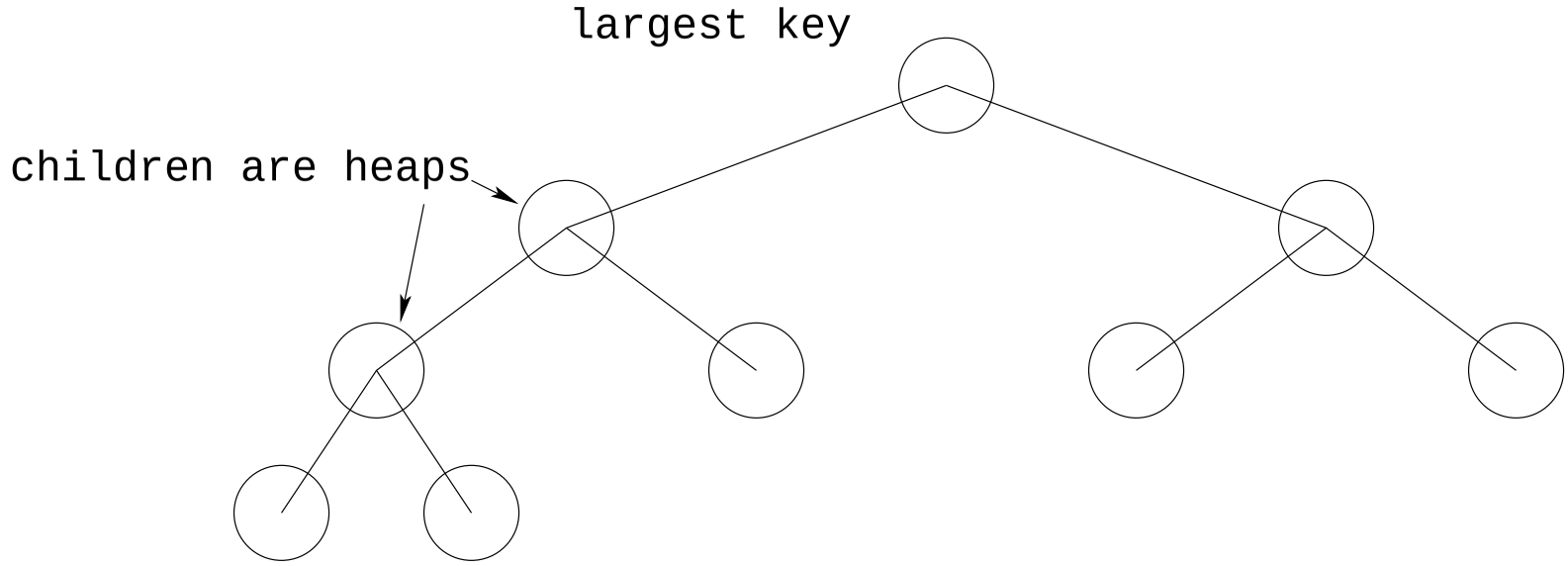
How is `split()` sped up?

- The elements in the unsorted portion of the array are organized into a *heap*.



What is a *Heap*?

- A heap is a binary tree that is almost balanced (we allow a variation of at most 1 in path lengths from the root to the leaves) and that further exhibits the heap property:
  - the root, if non-null, is the largest key in the tree, and its left and right subtrees are themselves heaps.



Implementing a *Heap*?