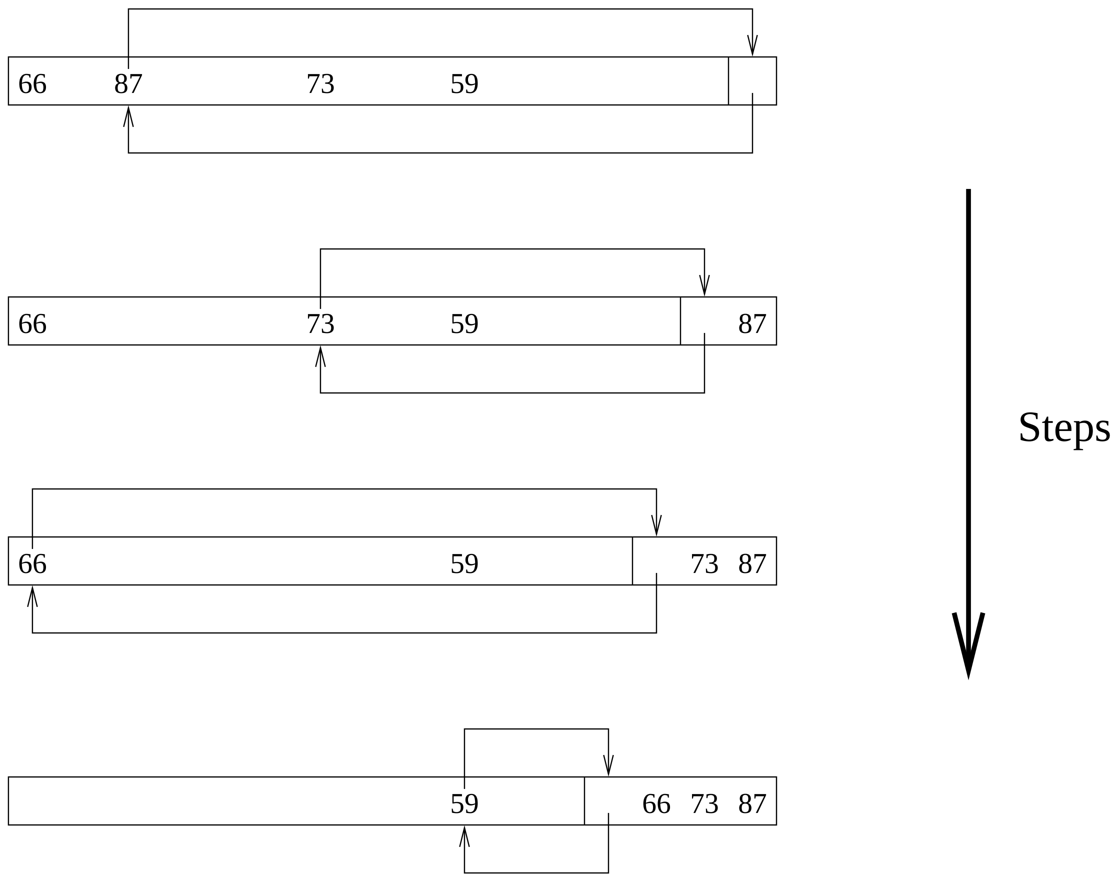
Review: Selection Sort

- A variation of Selection Sort that finds the element with the largest value at each step:

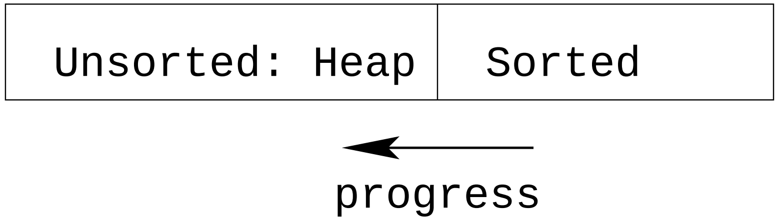


Heap Sort

- Heap Sort, like Selection 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 maximum (minimum) 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.

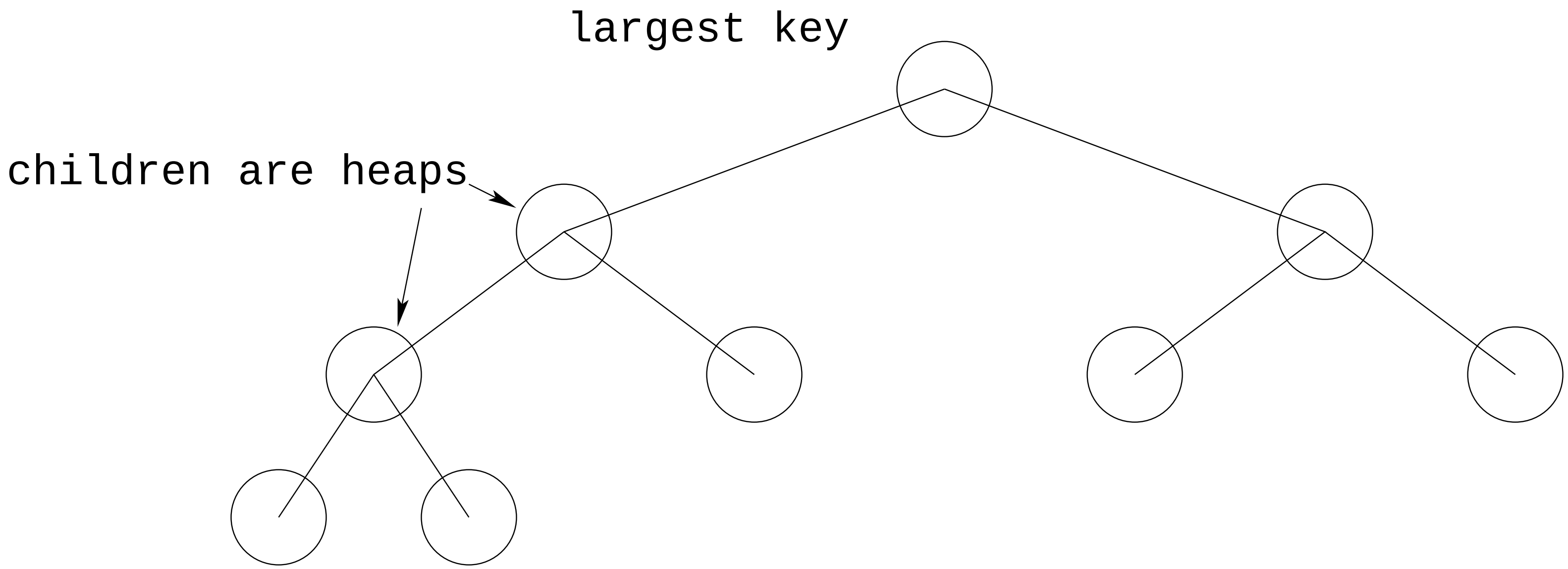
How is `split()` sped up?

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

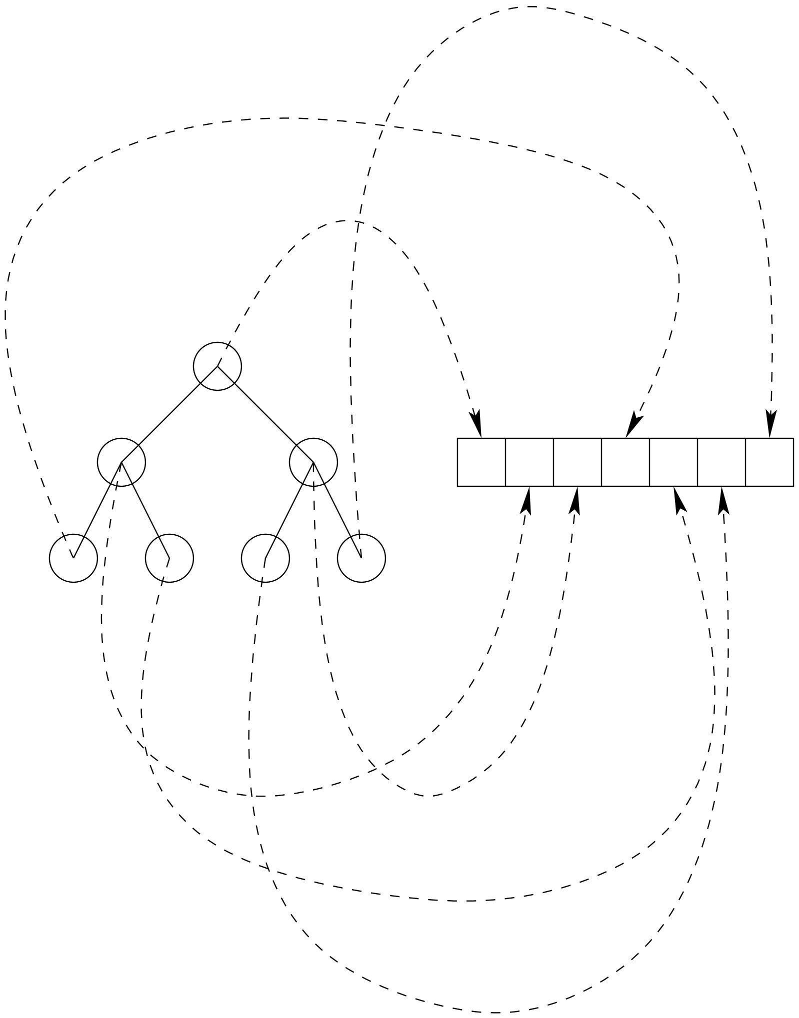


What Is A *Heap*?

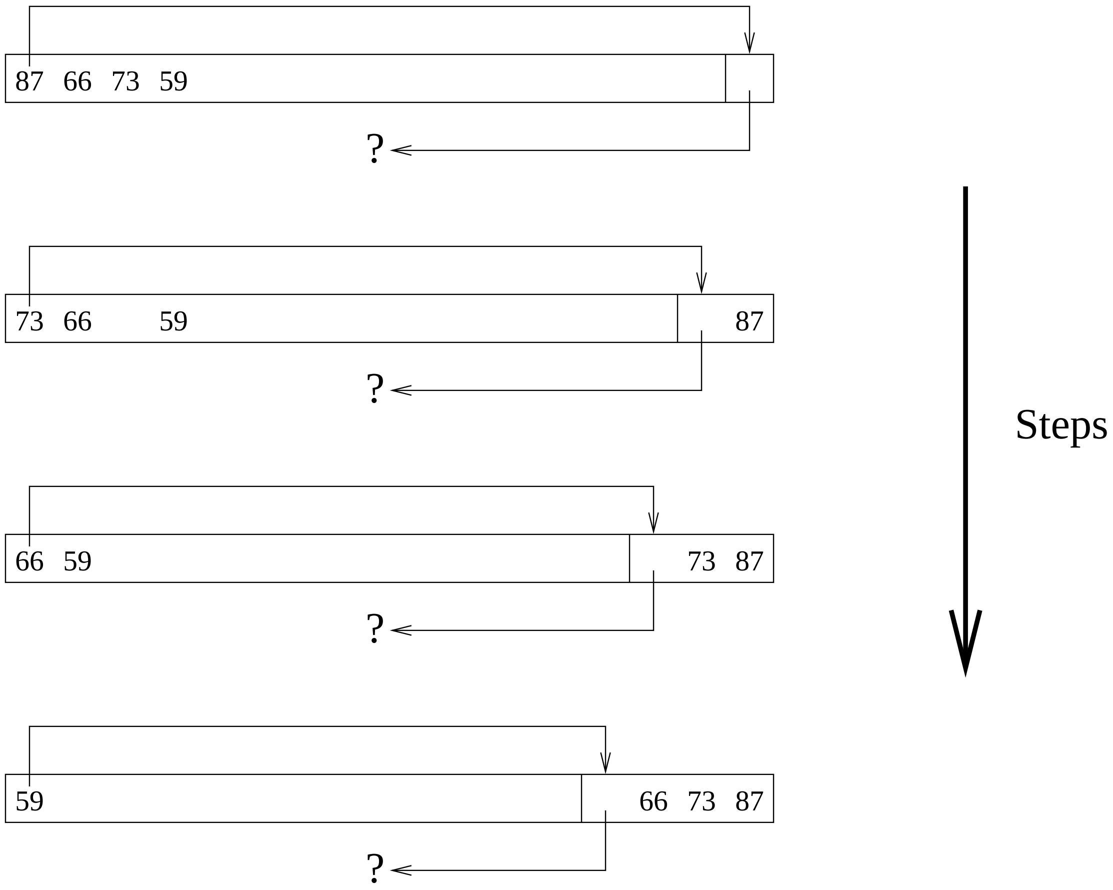
- A heap is a binary tree that (1) is almost balanced (we allow a variation of at most one in path lengths from the root to the leaves) and (2) exhibits the heap property:
  - the root, if non-null, is the largest key in the tree, and its left and right subtrees are themselves heaps.



Implementing A *Heap* Within An Array?



Heap Sort Basics



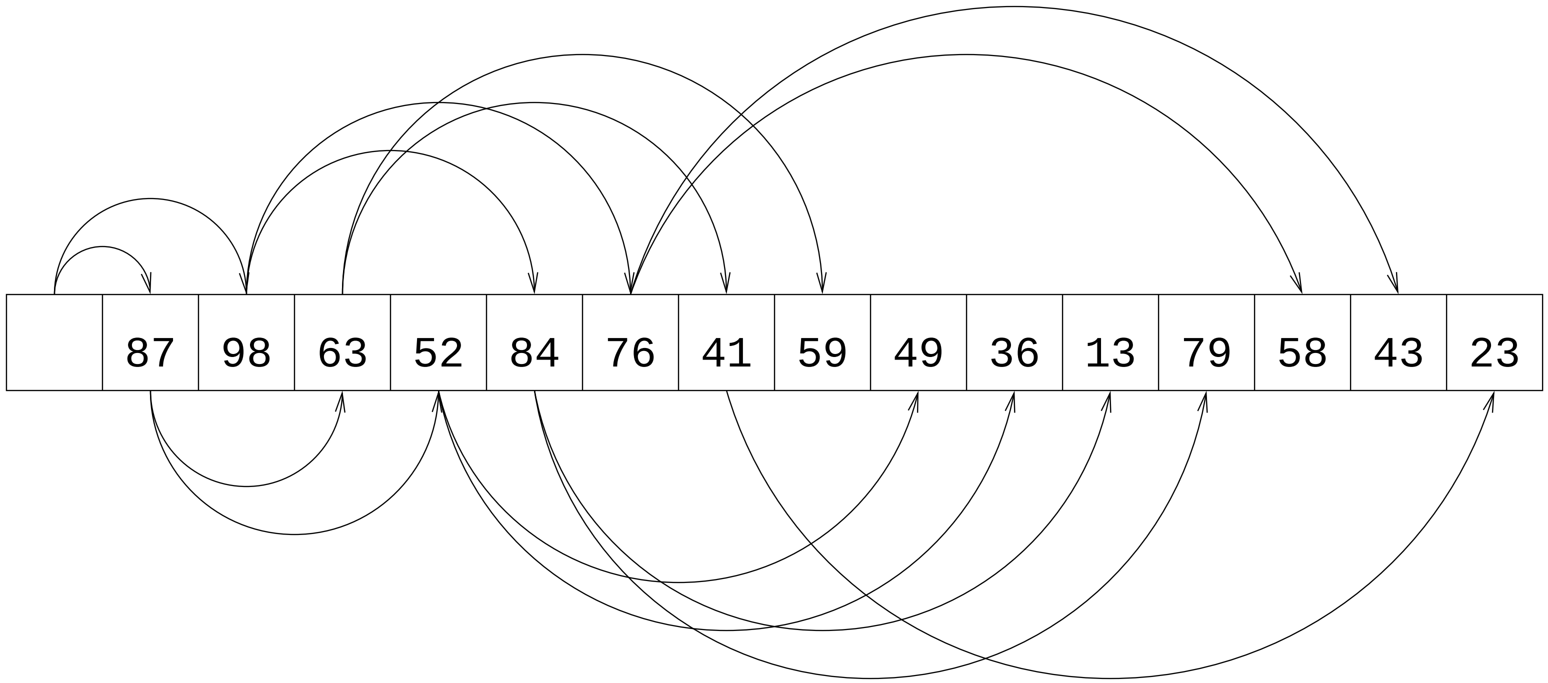
Heap Sort: `split()`

```
public int split(int[] A, int lo, int hi)
{
  // Swap A[hi] and A[lo].
  int temp = A[hi];
  A[hi] = A[lo];
  A[lo] = temp;

  // Restore the heap property by ``sifting down''
  //  the element at A[lo].
  Heapifier.Singleton.siftDown(A, lo, lo, hi - 1);

  return hi;
}
```

Example of `siftDown()`



siftDown(): The Implementation

```
public void siftDown(int[] A, int lo, int cur, int hi)
{
  int dat = A[cur];               // hold on to data.
  int child = 2 * cur + 1 - lo; // index of left child of A[cur].
  boolean done = hi < child;

  while (!done) {

    if (child < hi && A[child + 1] < A[child]) {
      child++;
    } // child is the index of the smaller of the two children.

    if (A[child] < dat) {
      A[cur] = A[child];
      cur = child;
      child = 2 * cur + 1 - lo;
      done = hi < child;
```

```
    }                            // A[cur] is less than its children.
    else {                       // A[cur] <= A[child].
      done = true;               // heap condition is satisfied.
    }                            // A[cur] is less than its children.
  }                              // location found for temp.
  A[cur] = dat;
}
```

Initializing the Heap: `HeapSorter()`

```
public class HeapSorter extends ASorter
{
  public HeapSorter(int[] A, int lo, int hi)
  {
    for (int cur = (hi - lo + 1) / 2; cur >= lo; cur--) {
      Heapifier.Singleton.siftDown(A, lo, cur, hi);
    }
  }
  . . .
```