Heapsort

Phase 1: Transform arbitrary array to heap

Apply max-heapify() to subtrees rooted at 88 and 18

Apply max-heapify() to subtrees rooted at 37 and 54

Apply max-heapify() to subtree rooted at 21

We now have a heap
Heapsort
Phase 2: Transform heap to sorted array
Five more exchanges, followed by calls to `fixHeap()`, complete the transformation from a heap to a sorted array.
# The Heapsort Algorithm

## max-heapify

```java
void max-heapify(T[] A, Integer n, Integer i) {
    p = i;
    while (2p ≤ n) {
            m = 2p + 1;
        } else {
            m = 2p;
        }
        if (A[p] < A[m]) {
            swap(A[p], A[m]);
            p = m;
        } else {
            return;
        }
    }
}
```

**Initially:** A is an array of size at least \( n \), and \( 1 ≤ i ≤ n \). The max-heap property holds everywhere in the subtree of \( A[1..n] \) rooted at \( A[i] \), except possibly at \( A[i] \) itself.

**Upon return:** The subtree of \( A[1..n] \) rooted at \( A[i] \) is a max-heap. The rest of \( A \) is unchanged.

**Comparisons:** at most \( 2h \), where \( h \) is the height of the subtree.

This height is:
- 1 if \( \lfloor n/2 \rfloor + 1 ≤ i ≤ \lfloor n/2 \rfloor \),
- 2 if \( \lfloor n/2 \rfloor + 1 ≤ i ≤ \lfloor n/2 \rfloor \), etc.

## build-max-heap

```java
void build-max-heap(T[] A) {
    n = A.length;
    for (i = \( \lfloor n/2 \rfloor \), \( \lfloor n/2 \rfloor - 1 \), ..., 1) {
        max-heapify(A, n, i);
    }
}
```

**Initially:** A is an arbitrary array.

**Upon return:** A is a max-heap.

**Note:** Pass \( i \) through the loop makes the subtree of \( A \) rooted at \( A[i] \) into a max-heap.

**Comparisons:** at most \( 2n \).

## sort-max-heap

```java
void sort-max-heap(T[] A) {
    n = A.length;
    for (i = n, n-1, ..., 2) {
        swap(A[1], A[i]);
        max-heapify(A, i-1, 1);
    }
}
```

**Initially:** A is a max-heap.

**Upon return:** A is a sorted array.

**Note:** Pass \( i \) through the loop moves the \( i \)th smallest element to position \( i \), and then rebuilds \( A[1..i-1] \) into a max-heap.

**Comparisons:** at most \( 2n \log(n) \).

## heapsort

```java
void heapsort(T[] A) {
    build-max-heap(A);
    sort-max-heap(A);
}
```

**Initially:** A is an arbitrary array.

**Upon return:** A is sorted.

**Comparisons:** at most \( 2n \log(n) \) + \( O(n) \).