

Worst-Case Running Time of Heapsort $(n = \text{length}(A))$

- Build - Max-Heap $\leftarrow O(n)$
- for $i \leftarrow \text{length}(A)$ down to 2 $\leftarrow n-1$ iterations
 - do exchange $A[i] \leftrightarrow A[1]$ each takes $O(\lg \text{heapsize})$
 - $\text{heapsize} \leftarrow \text{heapsize} - 1$
 - Max-Heapify $(A, 1)$

$$\text{heapsize} \leq n \rightarrow \lg \text{heapsize} \leq \lg n$$

$$\sum_{\text{heapsize}=2}^n \lg \text{heapsize} \leq \sum_{\text{heapsize}=2}^n \lg n = (n-1) \lg n = O(n \lg n)$$

No improvement will be made by computing $\sum \lg \text{heapsize}$ instead of $\sum \lg n$ $\left. \begin{array}{l} \text{except a} \\ \text{constant} \\ \text{factor.} \end{array} \right\}$

Thm: Sorting an array of n elements takes $\Omega(n \log n)$ time, worst case

I.e., there is no sorting algorithm which runs in guaranteed $O(n \log n)$ Time.

Proof: There are $n!$ permutations of n distinct values. A sorting algorithm must be capable of dealing with (fixing) all $n!$ of those permutations. With each comparison, it knows more about which permutation it has been given. With a good question, the space of remaining permutations can be cut in half, roughly. With a bad question, you may have more than half the permutations still left.

The array is sorted when the space of possible permutations has only a single element left.

How many steps does that take?

At best, you cut the space in half each time.

At start, size of space = $n!$

questions $\geq \lg(n!) = \Theta(n \lg n)$ ~~Q~~
(Stirling's Formula)

This is called an "information-theoretic" proof.

~~This proof applies only to algorithms that sort by comparison.~~

Max-Priority-Queues.

We want to maintain a data structure consisting of entries, each of which has some priority.

We should be able to:

- Add some element to the queue, $\text{Max-Heap-Insert } O(\lg n)$
- Ask for ~~the~~^{an} element of highest priority $\text{return } A[1] \quad O(1)$
- Ask for and remove ~~the~~ an element of highest priority
- Increase the priority of some element.

↓
like Max-Heap-Insert ,
except start with the
element concerned.

$O(\lg n)$.

$O(\lg n)$

```
return A[1]
exchange A[1] ↔ A[heapsize]
heapsize -- 1
max-Heapify(A, 1)
```