Algorithm Complexity (Q)

```
def swap(b, h, k):
    ⋮
def push_down(b, k):
    while k > 0 and b[k-1] > b[k]:
        swap(b, k-1, k)
        k= k-1

def insertion_sort(b):
    for i in range(1,len(b)):
        push_down(b, i)
```

Count (approximately) the number of comparisons needed to sort a list of length n

A. ~ 1 comparison
B. ~ n comparisons
C. ~ n^2 comparisons
D. ~ n^3 comparisons
E. I don't know

Algorithm Complexity (A)

- Count the number of comparisons needed
- In the worst case, need i comparisons to push down an element in a sorted segment with i elements.
- For a list of length n
  - 1st push down: 1 comparison
  - 2nd push down: 2 comparisons (worst case)

    ⋮
  - $1+2+\ldots+(n-1) = n*(n-1)/2$, say, $\mathbf{n^2}$ for big n
- For fun, check out this visualization: https://www.youtube.com/watch?v=xxcpvCGrCBc

Complexity of algorithms discussed so far

- Linear search: on the order of n
- Binary search: on the order of $\log_2 n$
  - Binary search is faster but requires sorted data
- Insertion sort: on the order of n^2
- Next, let's look at merge sort