Lecture 26: Sorting

CS 1110
Introduction to Computing Using Python

[5/3/18]

Plan of Attack

- Insertion Sort
- Partition
- Quick Sort

Insertion Sort

**Insertion Sort: Moving into Position**

```python
def push_down(b, k):
    while k > 0:
        if b[k-1] > b[k]:
            swap(b, k-1, k)
        k = k - 1
    while k < n:
        push_down(b, k)
        k = k + 1
```

**Algorithm Complexity**

```python
def push_down(b, k):
    while k > 0:
        if b[k-1] > b[k]:
            swap(b, k-1, k)
        k = k - 1
    while k < n:
        push_down(b, k)
        k = k + 1
```

Iterating through a sequence of length \( n \) requires \( n \) operations: `push_down` called \( n \) times

**Q: Algorithm Complexity**

Approximately how many operations does this take?

- A: ~ 1 operation
- B: ~ \( n \) operations
- C: ~ \( n^2 \) operations
- D: ~ \( n^3 \) operations
- E: I don't know

Is this how you want to sort 500 exams?
Partition Algorithm

• Given a list segment \( b[h..k] \) with some pivot value \( x \) in \( b[h] \):

\[
\begin{array}{c}
\text{pre: } b[h..k] \\
\end{array}
\]

• Swap elements of \( b[h..k] \) and store in \( i \) to satisfy postcondition:

\[
\begin{array}{c}
\text{post: } b[h..k] \\
\end{array}
\]

Example:

\[
\begin{array}{c}
\text{change: } b = [1, 2, 3, 4, 5, 6, 7, 8, 9] \\
\text{into: } b = [1, 2, 1, 3, 5, 4, 6, 3, 8] \\
\end{array}
\]

Partition: What’s the Invariant?

• Given a list segment \( b[h..k] \) with some pivot value \( x \) in \( b[h] \):

\[
\begin{array}{c}
\text{pre: } b[h..k] \\
\end{array}
\]

• Swap elements of \( b[h..k] \) and store in \( i \) to satisfy postcondition:

\[
\begin{array}{c}
\text{post: } b[h..k] \\
\end{array}
\]

Sorting with Partitions

• Idea: Pick a pivot element \( x \)

\[
\begin{array}{c}
\text{Partition sequence into } \leq x \text{ and } \geq x \\
\end{array}
\]

QuickSort

```python
def quick_sort(b, h, k):
    """Sort the array fragment b[h..k]"""
    if k<=h:
        return
    i = partition(b, h, k)
    # INV: b[h..i-1] \leq b[i] \leq b[i+1..k] post: b[h..i-1] and b[i+1..k]
    quick_sort(b, h, i-1)
    quick_sort(b, i+1, k)
```

"""Sorted!"""