Sorting

4/22/26

Be sure to email me code from Lab 4

A common problem: Sorting

- Have collection of objects (numbers, strings, dates, ...) and want to put them into a defined order

- How do we specify the order?

- How do we do this efficiently?
  - assume objects are in an array
  - assume you can compare any pair of objects

Generalizing comparisons

- Comparable<T> interface

  int compareTo(T other)

  negative: this < other

  positive: this > other

  zero: this == other

- Comparator<T> interface

  int compare(T o1, T o2)

  as above, with o1 taking role of “this”

Selection sort

(code adapted from Wikipedia)

- Grow sorted part of array by finding smallest value in the remaining

```
for(int i=0; i < A.length-1; i++) {
    int minIndex = i;                          //index of min
    for(int j=i+1; j < A.length; j++) {    //find the min
        if(A[j] < A[minIndex])
            minIndex = j;
    }
    swap A[i] and A[minIndex];
}
```

Selection sort

(code adapted from Wikipedia)

- Grow sorted part of array by finding smallest value in the remaining

```
for(int i=0; i < A.length-1; i++) {
    int minIndex = i;                          //index of min
    for(int j=i+1; j < A.length; j++) {        //find the min
        if(A[j] < A[minIndex])
            minIndex = j;
    }
    swap A[i] and A[minIndex];
}
```

What most accurately characterizes the running time of selection sort?

A. O(1)
B. O(n)
C. $O(n^2)$
D. $O(n^3)$
E. None of the above

Selection sort

(code adapted from Wikipedia)

- Grow sorted part of array by finding smallest value in the remaining

```
for(int i=0; i < A.length-1; i++) {
    int minIndex = i;                          //index of min
    for(int j=i+1; j < A.length; j++) {        //find the min
        if(A[j] < A[minIndex])
            minIndex = j;
    }
    swap A[i] and A[minIndex];
}
```

What most accurately characterizes the running time of selection sort?

A. O(1)
B. O(n)
C. $O(n^2)$
D. $O(n^3)$
E. None of the above