CSE 12 – Basic Data Structures and Object-Oriented Design
Lecture 13

Greg Miranda, Spring 2021

This lecture is being recorded

Announcements

- Quiz 13 due Friday @ 12pm
- PA4 due tonight @ 11:59pm
- Survey 5 due Friday @ 11:59pm
- PA5 released tomorrow (closed)

Topics

- Partition/Sort
- Questions on Lecture 13?

Quicksort: Another magical (recursive) algorithm

https://www.youtube.com/watch?v=ywWBy6J5gz8

14	4	9	12	15	8	19	2

Select a **pivot** element:

14	4	9	**12**	15	8	19	2

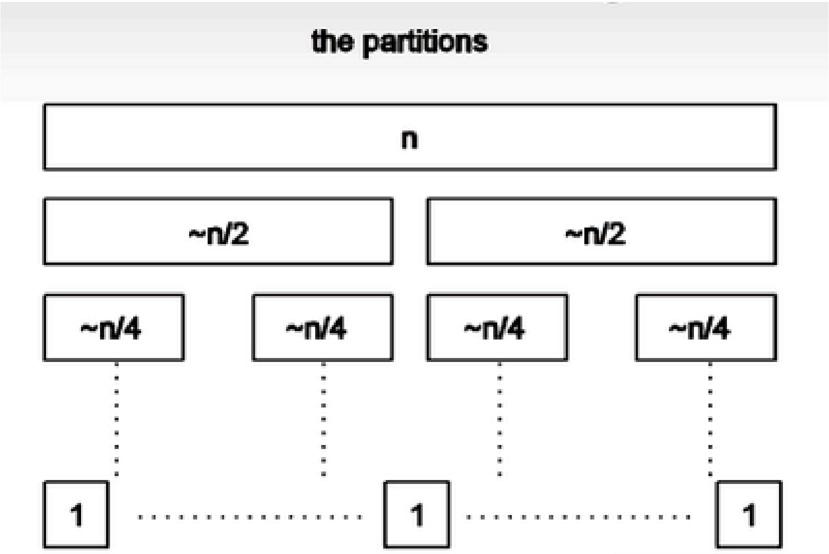
“Partition” the elements in the array (smaller or equal to pivot, larger or equal to pivot)

2	4	9	8	15	**12**	19	14

Magically sort the smaller elements and the larger elements (Quicksort)

2	4	8	9	12	15	19	21

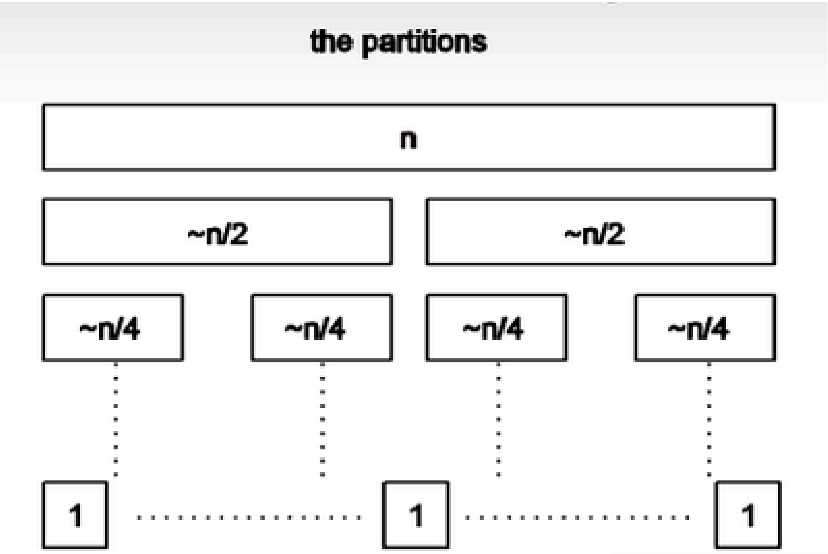
Quick Sort: Using a “good” pivot



How many levels will there be if you choose a pivot that divides the list in half?

A. 1
B. log(N)
C. N
D. N*log(N)
E. N^2

Quick Sort: Using a “good” pivot



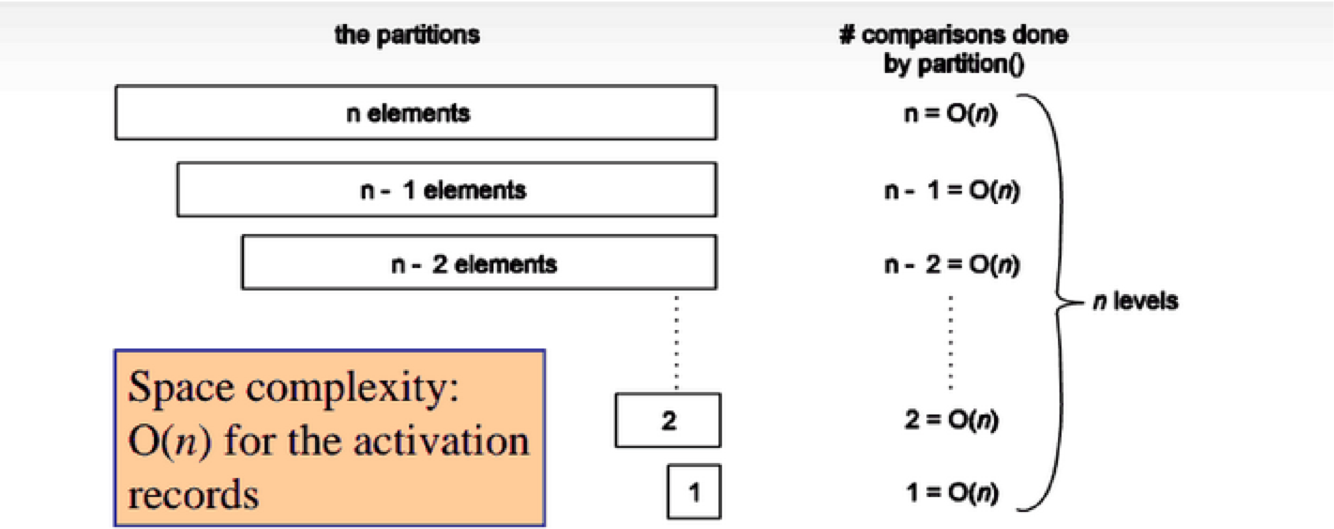
If the time to partition on each level takes N comparisons, how long does Quicksort take with a good partition?

A. O(1)
B. O(log(N))
C. O(N)
D. O(N*log(N))
E. O(N^2)

Which of these choices would be the *worst* choice for the pivot?

A. The minimum element in the list
B. The last element in the list
C. The first element in the list
D. A random element in the list

Quick sort with a bad pivot



Space complexity: O(n) for the activation records

If the pivot always produces one empty partition and one with $n - 1$ elements, there will be n levels, each of which requires O(n) comparisons: O(n^2) time complexity

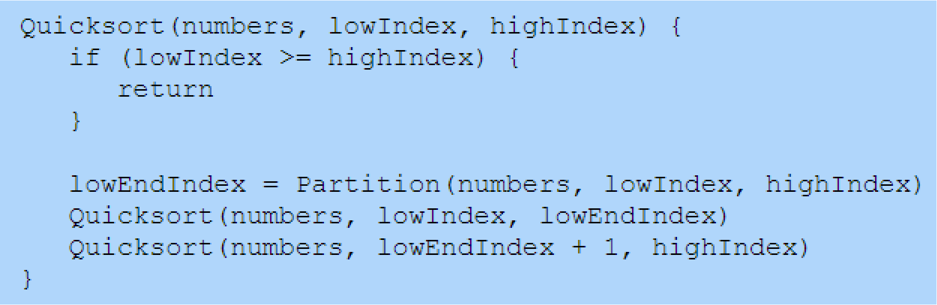
Which of these choices is a better choice for the pivot?

A. The first element in the list
B. A random element in the list
C. They are about the same

Quick sort – Middle Pivot

sort {12, 4, 9, 3, 15, 8, 19, 2}

There are many ways to partition!



```
Quicksort(numbers, lowIndex, highIndex) {
   if (lowIndex >= highIndex) {
      return
   }

   lowEndIndex = Partition(numbers, lowIndex, highIndex)
   Quicksort(numbers, lowIndex, lowEndIndex)
   Quicksort(numbers, lowEndIndex + 1, highIndex)
}
```

Quick sort – Middle Pivot

1. We always pick the middle location as pivot
2. The data we sort is {2, 3, 1, 5, 4, 6, 7}

After the first split, what is the order of elements in the list that was <= pivot?

A. 1 2 3 4

B. 2 3 1 4

C. 4 3 2 1

D. 3 4 1 2

E. None of the above

QuickSort – Draw the picture of sort()

```
public class Sort {
  public static void swap(String[] array, int i1, int i2) {
    String temp = array[i1];
    array[i1] = array[i2];
    array[i2] = temp;
  }
  public static int partition(String[] array, int low, int high) {
    int pivotStartIndex = high - 1;
    String pivot = array[pivotStartIndex];
    int smallerBefore = low, largerAfter = high - 2;

    while (smallerBefore <= largerAfter) {
      if (array[smallerBefore].compareTo(pivot) < 0) {
        smallerBefore += 1;
      }
      else {
        swap(array, smallerBefore, largerAfter);
        largerAfter -= 1;
      }
    }

    swap(array, smallerBefore, pivotStartIndex);
    return smallerBefore;
  }
```

```
public static void qsort(String[] array, int low, int high) {

   if (high - low <= 1) { return; }

   int splitAt = partition(array, low, high);

   qsort(array, low, splitAt);

   qsort(array, splitAt + 1, high);

 }


 public static void sort(String[] array) {

   qsort(array, 0, array.length);

 }
```

```
main() {

  String[] str = {"f", "b", "a", "e", "d", "c" };

  int[] result = Sort.sort(str);

  System.out.println(Arrays.deepToString(result));

}
```