CSE 12 – Basic Data Structures and Object-Oriented Design
Lecture 14

Greg Miranda, Spring 2021

This lecture is being recorded

Announcements

- Quiz 14 due Monday @ 12pm
- PA5 due tonight @ 11:59pm (closed)
- Survey 5 due tonight @ 11:59pm

Topics

- Sorting Wrapup
- Questions on Lecture 14?

```
import java.util.Arrays;
public class Sort {
static void selectionSort(int[] arr) {
  for(int i = 0; i < arr.length; i += 1) {
    int minIndex = i;
    for(int j = i; j < arr.length; j += 1) {
      if(arr[minIndex] > arr[j]) { minIndex = j; }
    }
    int temp = arr[i];
    arr[i] = arr[minIndex];
    arr[minIndex] = temp;
  }
}
```

```
static void insertionSort(int[] arr) {
  for(int i = 0; i < arr.length; i += 1) {
    for(int j = i; j > 0; j -= 1) {
      if(arr[j] < arr[j-1]) {
        int temp = arr[j-1];
        arr[j-1] = arr[j];
        arr[j] = temp;
      }
      else { break; } // new! exit inner loop early
    }
  }
}
}
```

```
import java.util.Arrays;
public class SortFaster {

  static int[] combine(int[] p1, int[] p2) {...}

  static int[] mergeSort(int[] arr) {
    int len = arr.length
    if(len <= 1) {return arr; }
    else {
      int[] p1 = Arrays.copyOfRange(arr, 0, len / 2);
      int[] p2= Arrays.copyOfRange(arr, len / 2, len);
      int[] sortedPart1 = mergeSort(p1);
      int[] sortedPart2 = mergeSort(p2);
      int[] sorted = combine(sortedPart1, sortedPart2);
      return sorted;
    }
  }
```

```
  static int partition(String[] array, int l, int h) {...}

  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);
  }

}
```

	Insertion	**Selection**	**Merge**	**Quick**
Best case time				
Worst case time				
Key operations	swap(a, j, j-1) (until in the right place)	swap(a, i, indexOfMin) (after finding minimum value)	l = copy(a, 0, len/2) r = copy(a, len/2, len) ls = sort(l) rs = sort(r) merge(ls, rs)	p = partition(a, l, h) sort(a, l, p) sort(a, p + 1, h)

Last note about sorting

- Not only do we care about runtime, we also care about
  - Space: do we need extra storage?
  - Stable: if we have duplicates, do we maintain the same ordering?

Algorithm	Space	Stable
Bubble sort	O(1)	Yes
Selection sort	O(1)	No
Insertion sort	O(1)	Yes
Heap sort	O(1)	No
Merge sort	O(n)	Yes
Quick sort	O(logn)	No