CS 223

Exam #2

Heaps

Code for Percolate Up & Percolate Down in Heaps

Difference Between Min-heap and Max-heap

Event Simulation - Application of a Heap

Anything that was on the quiz

B-Trees

Code for Insertion (not for construction)

K-D Tree

Know insertion/construction algorithm

Range queries

Nearest Neighbor Search

Sorting algorithms

Heap sort

Best	$O(nlog(n))$
Average	$O(nlog(n))$
Worst	$O(nlog(n))$

Merge Sort

Best	$O(nlog(n))$
Average	$O(nlog(n))$
Worst	$O(nlog(n))$

Quick Sort

Best	$O(nlog(n))$
Average	$O(nlog(n))$
Worst	$O(n^2)$

Code:

```
int partition(int* array, int left, int right, int pivotIndex)
    int storeIndex = left;
    pivotValue = array[pivotIndex];
    swap(array[pivotIndex], array[right]);
    // For loop moves everything smaller than pivot (which is
    // in the rightmost position of this sub-array) to the rig
    // once to for loop is finished, the "storeIndex" will be
    // element that is larger than the pivot.
    for (int i = left; i < right-1; ++i){
        if (array[i] <= array[right]) {
            swap(array[i], array[storeIndex]);
            storeIndex++;
        }
    }
    // Now we swap the pivot into the border between smaller a
    swap(array[storeIndex], array[right]);
    // Then we return the "storeIndex" since that's where the
    // is. This allows for further recursive partitioning.
}

void quicksort(int* array, int left, int right){
    if (left < right) {
        int pivotIndex = medianOfThree(array, left, right);
        int newPivotIndex = partition(array, left, right, pivo

        // Recurse sort smaller sub-array
        quicksort(array, left, newPivotIndex-1);

        // Recurse sort larger sub-array
        quicksort(array, newPivotIndex+1, right);
    }
}
```