

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

```
void recSearch(kd_node*& node, kd_point*& sPoint, kd_point*& nn, int k) {
    k = (k+1) % _DIMS_;
    if (node){
        if (oCompCheck[k](sPoint, node->getVal())){ // sPoint is smaller
            if (node->getLeft()){
                recSearch(node->getLeft(), sPoint, nn, k);
            } else {
                if (oCalcDistance(sPoint, node->getVal()) < oCalcDistance(sPoint, nn)){
                    nn = node->getVal();
                }
                return;
            }
        } else { // sPoint larger, go right.
            if (node->getRight()){
                recSearch(node->getRight(), sPoint, nn, k);
            } else {
                if (oCalcDistance(sPoint, node->getVal()) < oCalcDistance(sPoint, nn)){
                    nn = node->getVal();
                }
            }
        }
    }
}
```

```

        return;
    }
}
} else {
    return;
}

// If the Cartesian distance is less, then current node th
if (oCalcDistance(sPoint, node->getVal()) < oCalcDistance(
    nn = node->getVal();
}
// If there's any possibility that there could be nodes in
// fork of the subtree, then we explore that as well.
if (sphereCheck[k](sPoint, node->getVal()) <= oCalcDistance
// if (isInsideHyperSphere(oCalcDistance(sPoint, nn), nn,
    if ( oCompCheck[k](sPoint, node->getVal()) ){// spoint
                                                // we che
                                                // direct
        recSearch(node->getRight(), sPoint, nn, k);
    } else {
        recSearch(node->getLeft(), sPoint, nn, k);
    }
}
return;
}

```

Sorting algorithms

Heap sort

Best	$O(n\log(n))$
Average	$O(n\log(n))$
Worst	$O(n\log(n))$

Merge Sort

Best	$O(n\log(n))$
Average	$O(n\log(n))$
Worst	$O(n\log(n))$

Quick Sort

Best	$O(n\log(n))$
Average	$O(n\log(n))$
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 right
    // once the 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 and larger
    swap(array[storeIndex], array[right]);
    // Then we return the "storeIndex" since that's where the pivot
    // 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, pivotIndex);

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

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