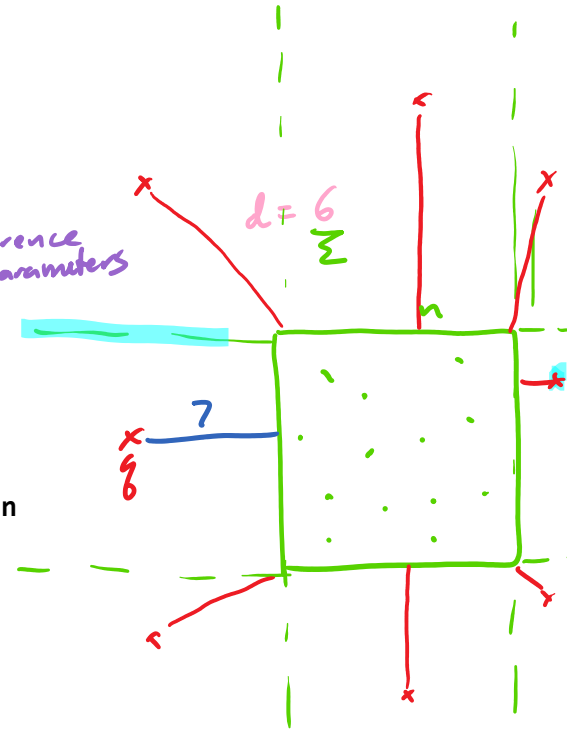


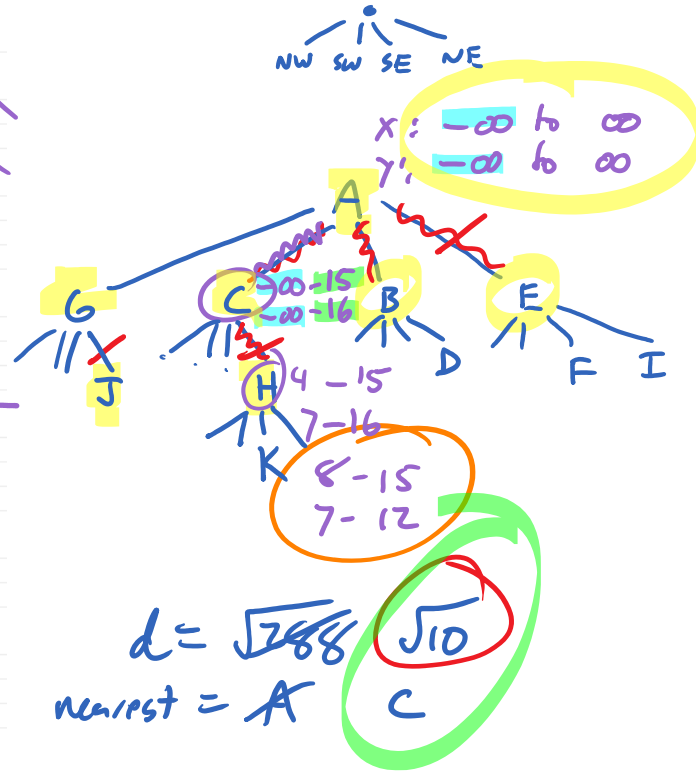
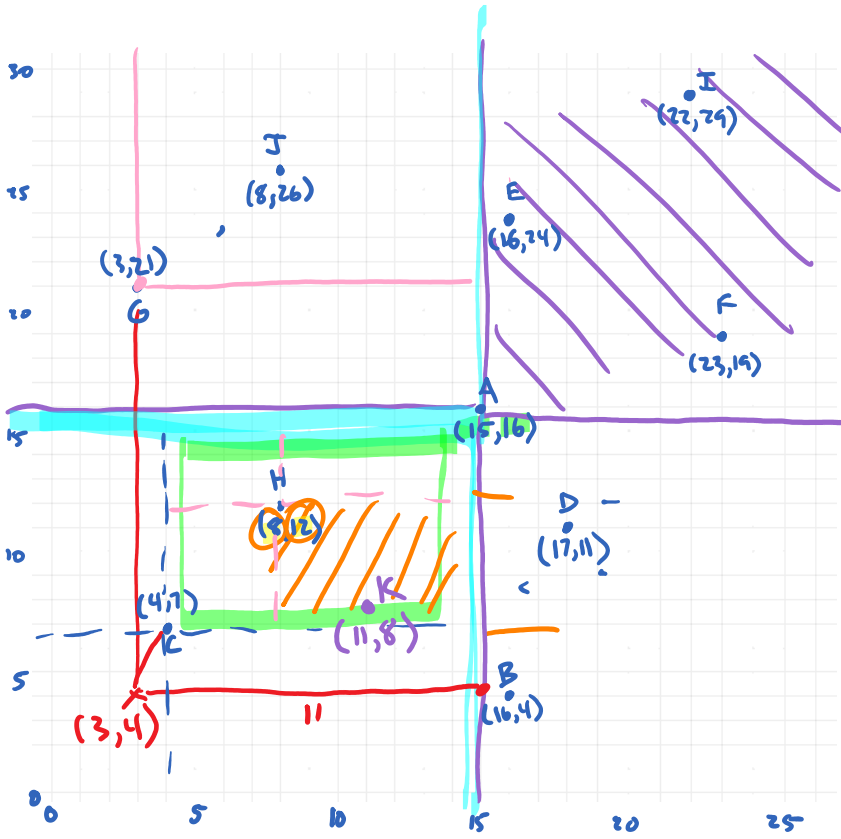
Nearest Neighbor

```
pointset_nearest_neighbor(points, q)
  point2d nearest
  double d = INFINITY
  pointset_nearest_neighbor(points->root, q, nearest, d)
```

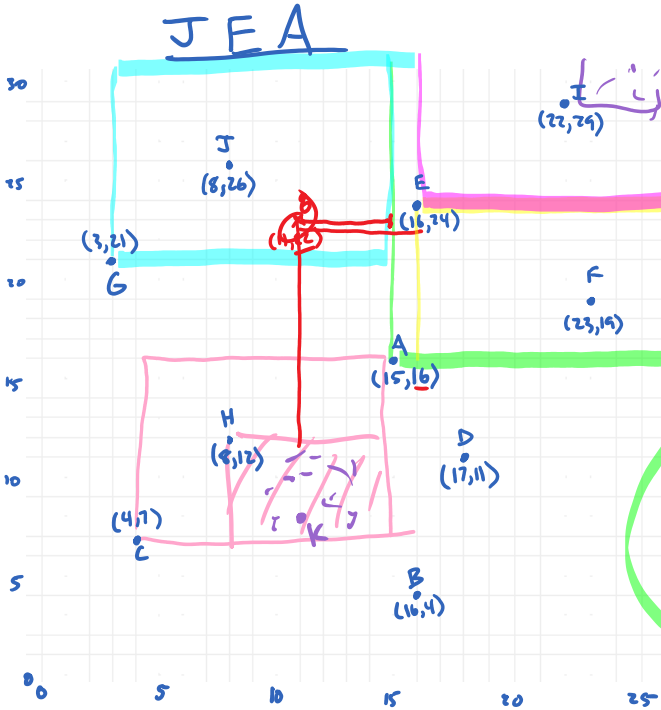
```
pointset_nearest_neighbor_in_subtree(n, q, nearest, d)
  if n == NULL
    return
  if the region for n is further than d from q
    return
  if point in n is closer than nearest to q
    nearest, d = point in n, distance from q to point in n
  pointset_nearest_neighbor(n->nw, q, nearest, d)
  pointset_nearest_neighbor(n->sw, q, nearest, d)
  pointset_nearest_neighbor(n->se, q, nearest, d)
  pointset_nearest_neighbor(n->ne, q, nearest, d)
```



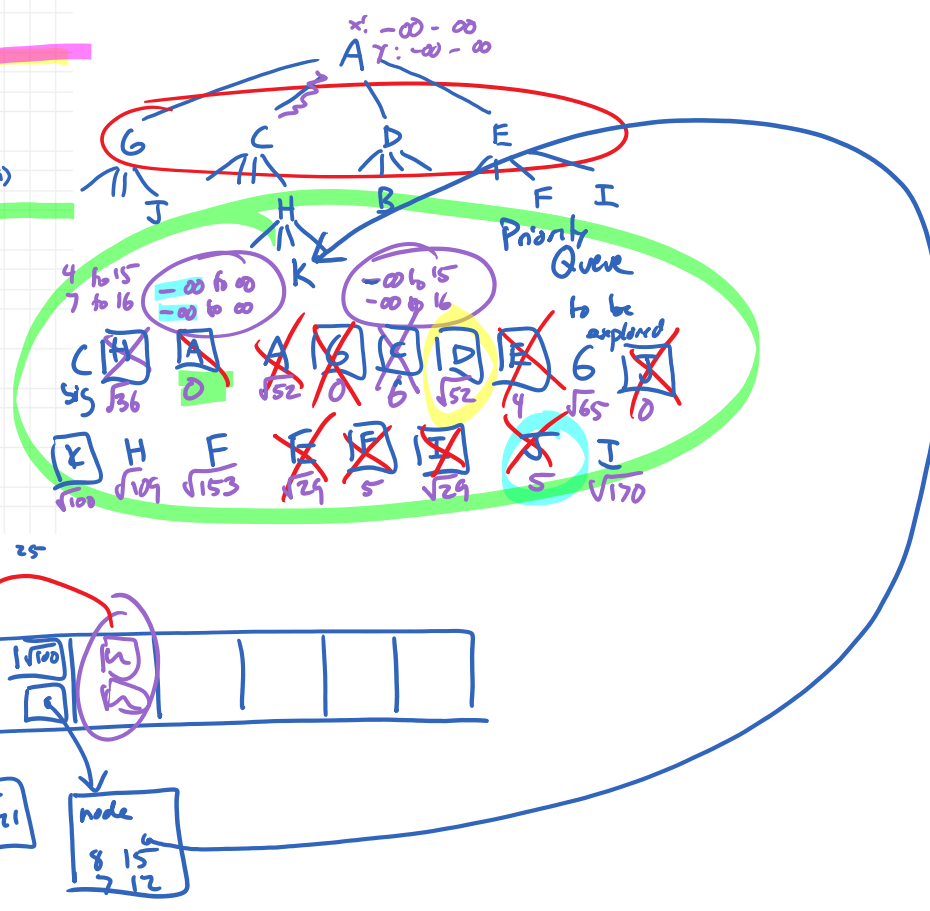
Nearest neighbor



k-nearest



find 3 closest pts to G
 NW SW SE NE



k Nearest Neighbors

```
pointset_k_nearest(points, query, k)
```

```
  q = pqueue_create()
```

```
  point2d nearest[k]
```

```
  found = 0
```

```
  pqueue_enqueue(q, point->root, 0)
```

```
  while (q is not empty and found < k)
```

```
    item = pqueue_dequeue(q)
```

```
    if item is a point
```

```
      add to item to nearest
```

```
      found++
```

```
    else (item must be a node in this case)
```

```
      pqueue_enqueue(q, pt inside node, distance from query to that point)
```

```
      for each non-NULL child of node
```

```
        pqueue_enqueue(q, child, distance from query to region of child)
```

adds an item to priority queue with given priority
unsorted array: amortized $O(1)$
balanced BST (keys: priorities) : worst-case $O(\log n)$

returns item with lowest numeric priority

unsorted array : $\Theta(n)$

balanced BST : worst case $O(\log n)$