selection sort:
4 3 1 5 2 -> 1 | 3 4 5 2 -> 1 2 | 4 5 3 -> 1 2 3 | 5 4 -> 1 2 3 4 | 5
o(n^2)

insertion sort:
4 3 1 5 2 -> 3 4 | 1 5 2 -> 1 3 4 | 5 2 -> 1 3 4 5 2 -> 1 2 3 4 5
best: n; average: n^2 worst: n^2

bubble sort:
4 3 1 5 2 -> 3 4 1 5 2 -> 3 1 4 5 2 -> 3 1 4 2 5 | -> 1 3 4 2 5 -> 1 3 2 4 5 -> | 1 2 3 4 5
best:n; average: n^2, worst n^2

```
int kthSmallest(int arr[], int l, int r, int k)
{
     int pos = partition(arr, l, r);
     if (pos-l == k-1)
          return arr[pos];
      if (pos-l > k-1)  // If position is more, recur for left subarray
          return kthSmallest(arr, l, pos-1, k);
      return kthSmallest(arr, pos+1, r, k-pos+l-1);
   }
}
```


```
Preorder: 5 3 0 2 4 7 6 8 10
Inorder: 0 2 3 4 5 6 7 8 10
Postorder: 2 0 4 3 6 10 8 7 5
Levelorder: 5 3 7 0 4 6 8 2 10 BFS
Someorder: Same as preorder: DFS

int BinaryTree::numOfNodes(Node *node)
{
    if(node == nullptr)
        return 0;
    return 1 + numOfNodes(node->left) + numOfNodes(node -> right);
}

int BinaryTree::numOfLeafNodes(Node *node)
{
    if(node == nullptr)
        return 0;
    if(node->left == nullptr && node->right == nullptr)
```

```
        return 1;
    return numOfLeafNodes(node->left) + numOfLeafNodes(node->right);
}

int BinaryTree::numOfNonLeafNodes(Node *node)
{
    if(node == nullptr || (node->left == nullptr && node -> right ==
nullptr))
        return 0;
    return 1+numOfNonLeafNodes(node->left) + numOfNonLeafNodes(node-
>right);
}

int BinaryTree::height(Node *node)
{
    if (node == nullptr) return 0;
    int left = height(node->left);
    int right = height(node->right);
    return 1+ (left > right ? left : right);
}

void BinarySearchTree::FreeTree(Node *cur)
{
     if (cur == nullptr ) return;
     FreeTree(cur->left);
     FreeTree(cur-> right);
     delete cur;
}
```