Linked Lists (DLL reverse function):

```
void reverse(dnode*& head) {
      if(!(head) || !(head -> link())) {
            return;
      }
      dnode* current = head;
      dnode* oldPrevious;
      while(current) {
            oldPrevious = current -> prev();
            current -> set_prev(current -> link());
            current -> set_link(oldPrevious);

            if(current -> prev()) {
                  current = current -> prev();
            }
            else {
                  head = current;
                  return;
            }
      }
}
```

Template Class Item:

```
template <class Item> //add this line before every function that uses it!
void list_head_insert(node<Item>*& head_ptr, const Item& entry);
```

Queues (reverse function for first k elements of queue):
Functions: q.enqueue(), q.dequeue(), q.front(), q.size()

```
void reverse(int k, Queue &q) {
      stack<int> s;

      for(int i = 0; i < k; i++) {
            s.push(q.front());
            q.dequeue();
      }
      for(int i = 0; i < k; i++) {
            q.enqueue(s.top());
            s.pop();
      }
      for(int i = k; i < q.size(); i++) {
            q.enqueue(q.front());
            q.dequeue();
      }
}
```

Runtime Order:
log log n ≺ {log n}≺ √n≺ {n, 2^log n, n+√n}≺ n log n≺ n^1.5≺ n^ log n≺ 2^n

Stacks (Infix to Postfix):
Functions: s.push(), s.top(), s.pop()

```
char expression[100];
cout << "Enter your fully parenthesized infix expression: ";
cin.getline(expression, sizeof(expression));

stack<char> s;
int digits = 0;
double numToAdd = 0;
int i = 0;

while(expression[i]) {
      if(isdigit(expression[i])) {
            digits++;
      }
      else if(digits > 0) {
            numToAdd = 0;
            for(int j = 1; j <= digits; j++) {
                  numToAdd += (expression[i-j] - '0') * pow(10, j-1);
            }
            cout << numToAdd << " ";
            digits = 0;
      }
      if(isalpha(expression[i])) {
            cout << expression[i] << " ";
      }
      else if(expression[i] == '+' || expression[i] == '-' || expression[i]
      == '*' || expression[i] == '/') {
            s.push(expression[i]);
      }
      else if(expression[i] == ')') {
            cout << s.top() << " ";
            s.pop();
      }
      i++;
}
cout << endl;
return 0;
```

Recursion (Towers of Hanoi):

```
int moveDisk(int numDisks, int src, int dest, int temp, int recursion_level) {
  int countMoves = 0;
  if (numDisks == 1) {
    return 1;
  } else {
    countMoves = moveDisk(numDisks - 1, src, temp, dest, recursion_level + 1);
    countMoves++;
    countMoves = countMoves + moveDisk(numDisks - 1, temp, dest, src,
recursion_level + 1);
    return countMoves;
  }
}

numMoves = moveDisk(numDisks, 1, 3, 2, 0);
```