Quick Sort

```
int partition(int[] A, int lower, int upper)
//@requires 0 <= lower && lower < upper && upper <=
\length(A);
//@ensures lower <= \result && \result < upper;
//@ensures gt(A[\result], A, lower, \result);
//@ensures leq(A[\result], A, \result+1, upper);
{
        int pivot_index = lower+(upper-lower)/2;
        int pivot = A[pivot_index];
        swap(A, pivot_index, upper-1);
        int left = lower;
        int right = lower;
        while (right < upper-1)
        //@loop_invariant lower <= left && left <= right && right
< upper;
        //@loop_invariant gt(pivot, A, lower, left);
        //@loop_invariant leq(pivot, A, left, right);
        //@loop_invariant pivot == A[upper-1];
        {
                if (pivot <= A[right])
                        {
                        right++;
                        }
                else
                        {
                        swap(A, left, right);
                        left++; right++;
                        }
        }
        swap(A, left, upper-1);
        return left;
}

void qsort(int[] A, int lower, int upper)
//@requires 0 <= lower && lower <= upper && upper <=
\length(A);
//@ensures is_sorted(A, lower, upper);
{
        if (upper-lower <= 1) return;
                int i = partition(A, lower, upper);
        qsort(A, lower, i);
        qsort(A, i+1, upper);
        return;
}

swap(A, pivot_index, upper-1);
int left = lower;
int right = lower;
while (right < upper-1)
//@loop_invariant lower <= left && left <= right && right < upper;
//@loop_invariant gt(pivot, A, lower, left);
//@loop_invariant leq(pivot, A, left, right);
//@loop_invariant pivot == A[upper-1];
{
        ...
}
```

Binary Search

```
int bsearch(int x, int[] A, int n)
//@requires 0 <= n && n <= \length(A);
//@requires is_sorted(A, n);
/*@ensures (-1 == \result && !is_in(x, A, n))
      || ((0 <= \result && \result < n) && A[\result] == x); @*/
{
 int lo = 0;
 int hi = n;
 while (lo < hi)
  //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
  //@loop_invariant (lo == 0 || A[lo-1] < x);
  //@loop_invariant (hi == n || A[hi] > x);
  {
   int mid = lo + (hi-lo)/2;
   //@assert lo <= mid && mid < hi;
   if (A[mid] == x) return mid;
   else if (A[mid] > x) hi = mid;
   else //@assert A[mid] < x;
        lo = mid+1;
  }
 return -1;
}
```

Linked Lists

```
struct list {
  string data;
  struct list next;
};
typedef struct list* list
```

Safe Add

x + (y - x) / 2

Overflow

assert(maxint – x > y)

Two's Compliment of x

~x + 1

Min/Max

8 bit	-128	127
16 bit	-32768	32767
32 bit	-2147483648	2147483648

Big O

f(n) = O(g(n)) means there are positive constants c and k such that 0 <= f(n) <= cg(n) for all n >= k.

Complexity Proof

Prove that $3n^2 + n \log(n) + 6n + 6$ is $O(n^2)$

$3n^2 + n \log(n) + 6n + 6 <= c\,n^2$ for $c > 0$, $n >= n0$

$3n^2 + n \log(n) + 6n + 6 <= 3n^2 + n^2 + n^2 + n^2$

$= 6n^2$