Lecture 10. Recursion

- **A *recursive function* is a function that calls itself**

```
int sum(int n) {
    if (n == 0)
        return 0;
    else
        return sum(n - 1) + n;
}
```

- **To compute f(n) using recursion**

  compute f(0) **'basis' case**
  compute f(n) using f(k) for k < n **'recursive' cases**

- **Recursion is like *mathematical induction***

  To prove S(*n*) : prove S(0) , then prove S(*n*) assuming S(*k*) for all $k < n$

  $0 + 1 + 2 + 3 + \ldots + n = n(n+1)/2$

  Trivially true for $n = 0$

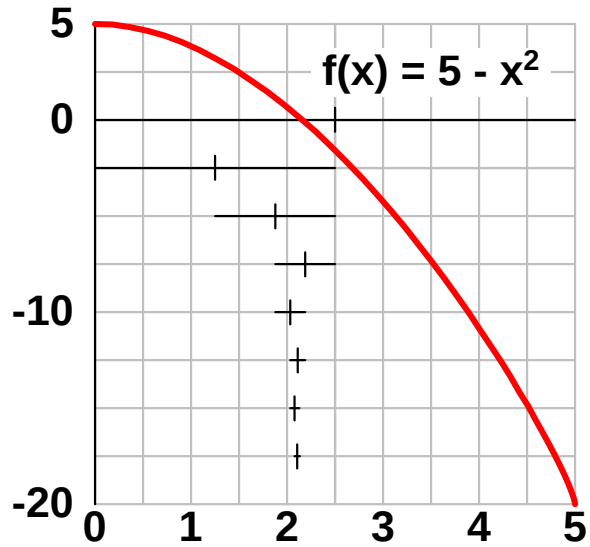
  Assume it is true for $0 + \ldots + n - 1$

  Is it true for $0 + \ldots + (n-1) + n$?

  $0 + 1 + 2 + \ldots + n = \underline{0 + \ldots + (n-1)} + n = \underline{(n-1)(n-1+1)/2} + n = n(n+1)/2$

Divide and Conquer

- **Solve a problem by *dividing it* into smaller ones:**
  To compute $\sqrt{n}$, find x such that $n - x^2 = 0$



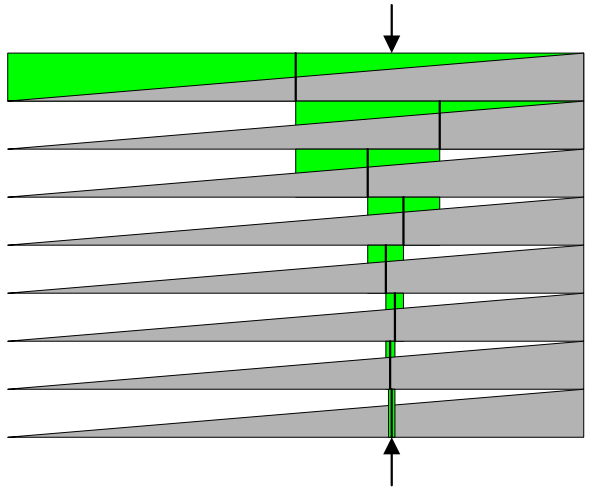

```
float sqroot(float n, float l, float r) {
    float x = (l + r)/2.0;
    if (r - l < 0.000001)
        return x;
    else if (n - x*x < 0.0)
        return sqroot(n, l, x);
    else
        return sqroot(n, x, r);
}

int main(int argc, char *argv[]) {
    int i;
    for (i = 1; i < argc; i++) {
        int n;
        sscanf(argv[i], "%d", &n);
        printf("sqrt(%d) = %f (should be %f)\n", n,
               sqroot(n, 0.0, n), sqrt(n));
    }
    return 0;
}
```

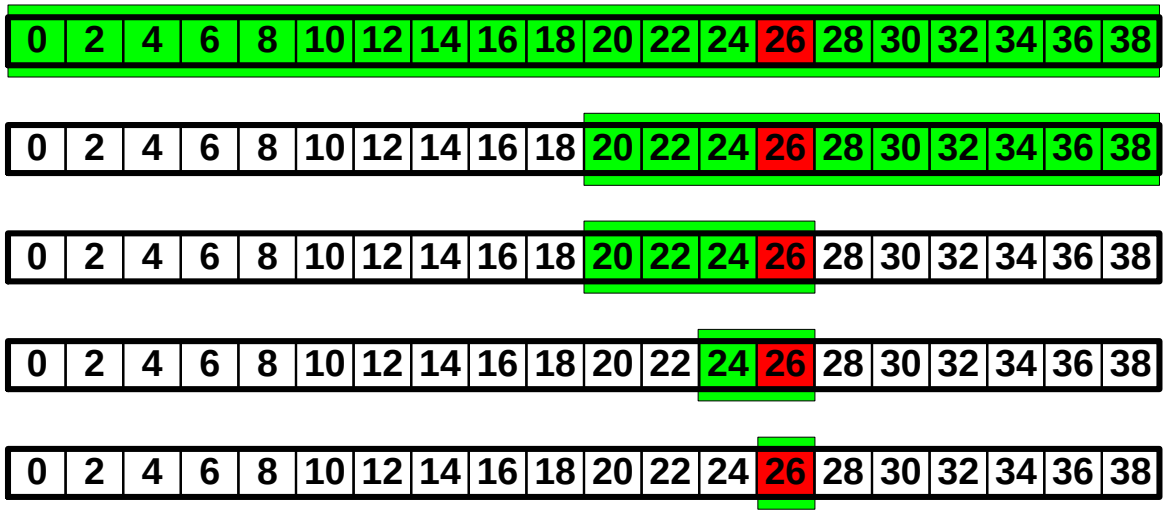
```
% lcc sqroot.c
% a.out 5
sqrt(5) = 2.236068 (should be 2.236068)
```

Binary Search

- **Suppose an array x contains n integers in increasing order; is q in `x[0..n-1]`?**

```
int bsearch(int x[], int lb, int ub, int q) {
    if (lb <= ub) {
        int m = (lb + ub)/2;
        if (x[m] < q)
            return bsearch(x, m + 1, ub, q);
        else if (x[m] > q)
            return bsearch(x, lb, m - 1, q);
        else
            return m;
    } else
        return -1;
}
k = bsearch(x, 0, 20 - 1, 26);
```





Number Conversion

- **Print an integer in base *b* (between 2 and 16)**

```
void convert(int n, int b) {
    if (n/b > 0)
        convert(n/b, b);
    printf("%c", "0123456789ABCDEF"[n%b]);
}
```

Printing 876 in base 5

$175\times5+\underline{1}$
$(35\times5+\underline{0})\times5+1$
$((7\times5+\underline{0})\times5+0)\times5+1$
$(((1\times5+\underline{2})\times5+0)\times5+0)\times5+1$
$((((0\times5+\underline{1})\times5+2)\times5+0)\times5+0)\times5+1$
$1\times5^4+2\times5^3+0\times5^2+0\times5^1+1\times5^0$
12001_5

```
convert(876, 5)
    convert(175, 5)
        convert(35, 5)
            convert(7, 5)
                convert(1, 5)
                    printf("%c", '1')
                printf("%c", '2')
            printf("%c", '0')
        printf("%c", '0')
    printf("%c", '1')
```

In base 16

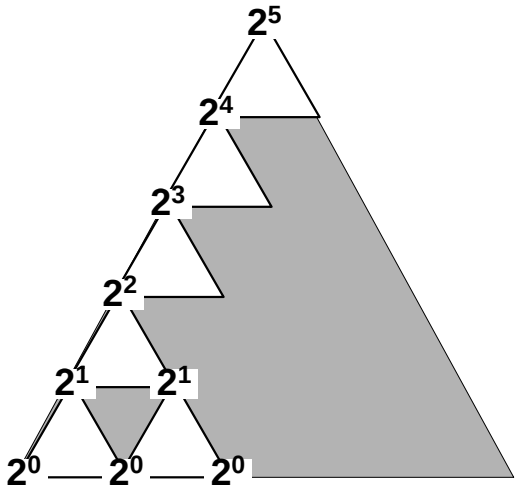
$54\times16+\underline{12}$
$(3\times16+\underline{6})\times16+12$
$((0\times16+\underline{3})\times16+6)\times16+12$
$3\times16^2+6\times16^1+12\times16^0$
$36C_{16}$

Pitfalls

- **Many computations are expressed naturally as recursive functions**
- ***But*, some simple recursive functions consume excessive resources: compute 2^n**

```
int f(int n) {
    if (n == 0)
        return 1;
    else
        return f(n-1) + f(n-1);
}
```

Hard way to compute 2^n because f *recomputes intermediate results*



- **Even 'natural' recursive function may consume excessive resources**

```
int fib(int n) {
    if (n == 0 || n == 1)
        return 1;
    else
        return fib(n-2) + fib(n-1);
}
```

- **Despite pitfalls, thinking and writing recursively yields correct implementations**
- ***Make it right before you make it fast***

Memo Functions

- **Recursive functions can avoid recomputing intermediate results by *saving* them**

```
int fibs[51] = { 1, 1, 0 };

int fib(int n) {
    if (n <= 50 && fibs[n] != 0)
        return fibs[n];
    else {
        int f;
        if (n == 0 || n == 1)
            f = 1;
        else
            f = fib(n-2) + fib(n-1);
        if (n <= 50)
            fibs[n] = f;
        return f;
    }
}
```

```
% lcc fib2.c
% a.out 30
fib(30) = 1346269
30: 1              14: 1597
29: 1              13: 2584
28: 2              12: 4181
27: 3              11: 6765
26: 5              10: 10946
25: 8               9: 17711
24: 13              8: 28657
23: 21              7: 46368
22: 34              6: 75025
21: 55              5: 121393
20: 89              4: 196418
19: 144             3: 317811
18: 233             2: 514229
17: 377             1: 832040
16: 610             0: 514229
15: 987
```

Changing Recursion to Iteration

- **If the *last action* of a function is to call itself — 'tail recursion' — the call can be replaced with assignments and a loop; use labels and gotos, then a loop statement**

```
float sqroot(float n, float l, float r) {
    float x;

    x = (l + r)/2.0;
    if (r - l < 0.000001)
        return x;
    else if (n - x*x < 0.0)
        return sqroot(n, l, x);
    else
        return sqroot(n, x, r);
}
```

```
Loop:   x = (l + r)/2.0;
        if (r - l < 0.000001)
            return x;
        else if (n - x*x < 0.0)
            { r = x; goto Loop; }
        else
            { l = x; goto Loop; }
```

```
float sqroot(float n, float l, float r) {
    float x;

    x = (l + r)/2.0;
    while (r - l > 0.000001) {
        if (n - x*x < 0.0)
            r = x;
        else
            l = x;
        x = (l + r)/2.0;
    }
    return x;
}
```