Prolegomena unit outline:

- Algorithms and correctness (last week Wednesday and Friday)
- Algorithms and efficiency (Wednesday and **today**)
- Abstract data types (next week Monday)
- Data Structures (next week Wednesday and Friday)

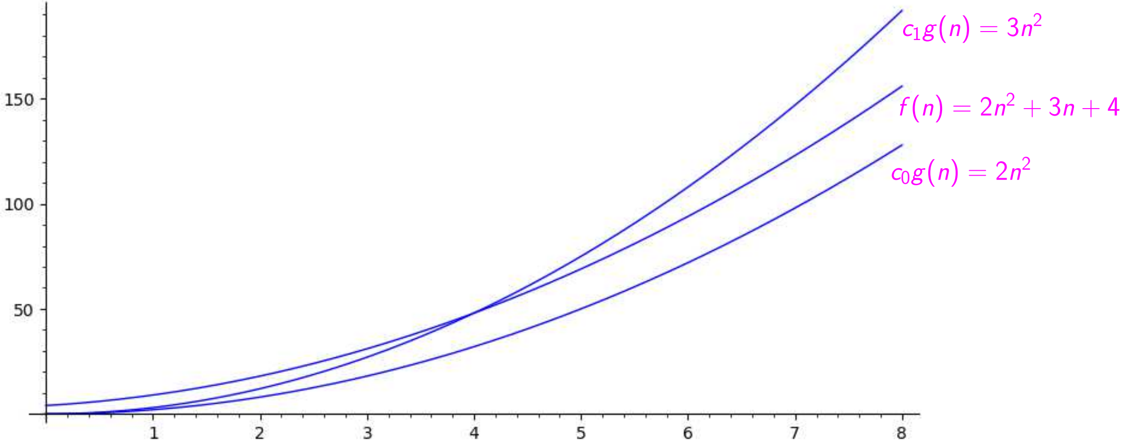
Today and Friday:

- Go over Ex 1.(6 & 7)
- The general meaning of efficiency
- The analyses of bounded linear search, binary search, and selection sort
- The precise meaning of big-oh, big-theta, and big-omega
- The costs of elemental algorithms
- The analysis of quick sort

Objections to and misconceptions of big-oh notation take forms such as

- Big-oh notation specifies only an upper bound of running time, which might be widely imprecise.
- Big-oh notation measures only the worst case, when the best case or the typical case might be much better.
- Big-oh ignores constants, which can greatly affect running time in practice.
- Algorithms that have the same big-oh category can have widely different running times in practice.
- Big-oh considers only the *size* of the input, when in fact other attributes of the input can greatly affect running time.

$$\Theta(g) = \{f : \mathbb{N} \to \mathbb{N} \mid \exists\ c_0, c_1, n_0 \in \mathbb{N} \text{ such that } \forall\ n \in [n_0, \infty), c_0 g(n) \leq f(n) \leq cg(n)\}$$



$$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{d-1} x^{d-1} + c_d x^d$$

$$\begin{aligned}
p(x) &= c_0 + c_1 x + c_2 x^2 + \cdots + c_{d-1} x^{d-1} + c_n x^d \\
&= c_0 + x(c_1 + c_2 x + \cdots + c_{n-1} x^{d-2} + c_d x^{d-1}) \\
&= c_0 + x(c_1 + x(c_2 + \cdots + c_{n-1} x^{d-3} + c_d x^{d-2})) \\
&= c_0 + x(c_1 + x(c_2 + \cdots + x(c_{d-1} + c_d x) \cdots))
\end{aligned}$$

```python
def eval_poly(coefficients, x):
    x_pow = 1.0
    result = 0.0
    for c in coefficients:
        result += c * x_pow
        x_pow *= x
    return result
```

```python
def eval_poly_horner(coefficients, x):
    result = 0.0
    for c in reversed(coefficients) :
        result *= x
        result += c
    return result
```

$g(n) \sim f(n)$ means $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 1$.

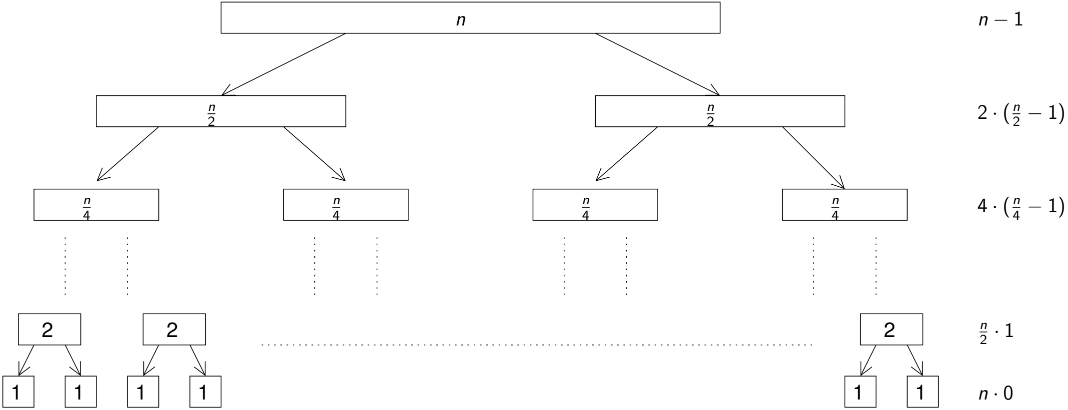
`eval_poly` is $\sim 3n$, `eval_poly_horner` is $\sim 2n$

$f \sim g$ means the functions are asymptotically *equal*, that is, that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$. For example, $\frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3} \sim \frac{n^3}{6}$.

$f = O(g)$, which really should be written $f(n) \in O(g(n))$, means that a scaled version of g asymptotically *bounds* f above. It means there exists a c such that when n is large enough, $f(n) \leq cg(n)$. For example, $\frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3} = O(\frac{n^3}{6})$ but also $\frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3} = O(n^3)$ and $\frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3} = O(n^4)$.

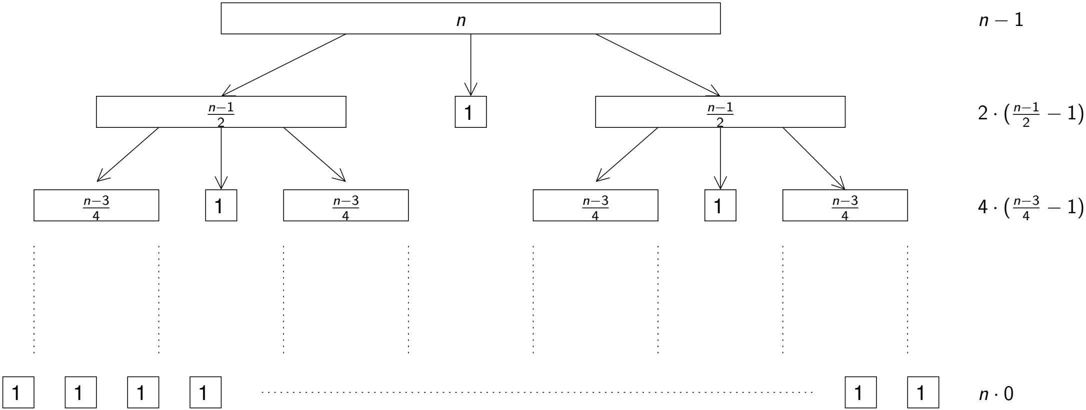
```c
int merge_sort_r(int sequence[], int aux[], int low, int high)
{
  if (low + 1 >= high)
    return 0;
  else {
    int compars = 0;  // the number of comparisons
    int midpoint = (low + high) / 2; // index to the middle of the range
    int k, n;
    n = high - low;
    compars += merge_sort_r(sequence, aux, low, midpoint);
    compars += merge_sort_r(sequence, aux, midpoint, high);
    compars = merge(sequence, aux, low, high);
    return compars;
  }
}
```

$$C_{ms}(n) = \begin{cases} 0 & \text{if } n \le 1 \\ n - 1 + 2C_{ms}(\frac{n}{2}) & \text{otherwise} \end{cases}$$

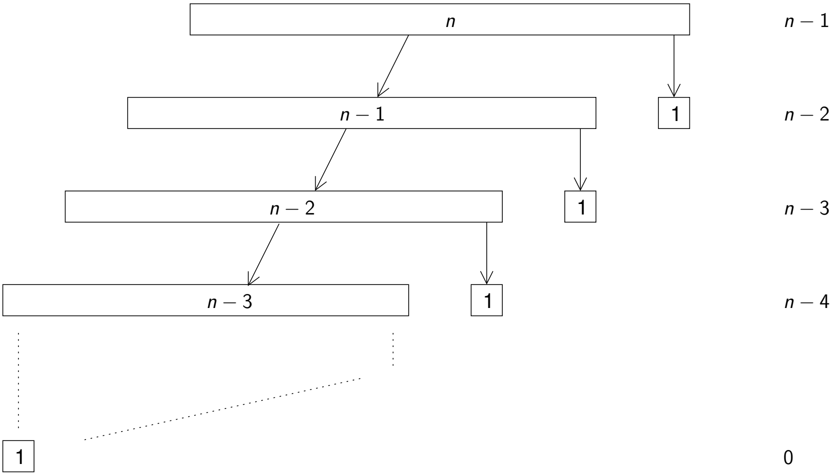


$$\begin{aligned} \sum_{i=0}^{\lg n - 1} 2^i \cdot (\tfrac{n}{2^i} - 1) &= \sum_{i=0}^{\lg n - 1} n \quad - \quad \sum_{i=0}^{\lg n - 1} 2^i \\ &= n \lg n \quad - \quad n + 1 \end{aligned}$$

```c
int quick_sort_r(int sequence[], int low, int high)
{
  if (low + 1 >= high) return 0;
  int i, j, temp;
  int compars = 0;
  for (i = j = low; j < high-1; j++) {
    compars++;
    if (sequence[j] < sequence[high-1])
      {
        temp = sequence[j];
        sequence[j] = sequence[i];
        sequence[i] = temp;
        i++;
      }
  }
  temp = sequence[i];
  sequence[i] = sequence[j];
  sequence[j] = temp;
  return compars + quick_sort_r(sequence, low, i)
    + quick_sort_r(sequence, i+1, high);
}
```



n
n−1
n−1/2
1
2 · (n−1/2 − 1)
n−3/4
4 · (n−3/4 − 1)
1
n · 0



$$(n-1)+(n-2)+(n-3)+\cdots+1+0=\sum_{i=1}^{n-1} i=\frac{n\cdot(n-1)}{2}=\frac{n^2-n}{2}$$

Coming up:

Due Today:
Read Sections 1.(3 & 4)
Do Exercises 1.(27, 28, 42, 43)
Take quiz

Due Tues, Jan 25:
Read Section 2.1
Do Exercise 1.11
Take quiz