# Оценка сложности

- Одну и ту же задачу можно решить разными способами
- Эквивалентность?
- Сложность
  - затрачиваемое время – временная сложность
  - необходимая память – ёмкостная сложность
  - в худшем случае
  - в среднем
- Учёт самых «дорогих» операций
- Необходим анализ алгоритмов

# Вычисление полинома

```
float poly(float coef[], int n,float x)
{
   float sum = 0f;
   for (int i=0; i<=n; i++)
      sum += coef[i] * power(i.x);
   return sum;
}
float power(int n, float x)
{
  return n==0 ? 1 : x*power(n-1,x);
}
void main()
{
  float binom[] = {1,2,1};
  printf("%d", poly(binom,2,10.0));
}
```

- Одно умножение на каждой из n+1 итерации цикл **for**
- Глубина рекурсии power равна n
  - i умножений на каждой итерации цикла **for**
  - i фреймов для хранения локальных объектов

# Пример оптимизации

```
float poly(float coef[], int n, float x)
{
   float sum = 0f;
   for (int i=0; i<=n; i++)
      sum += coef[i] * power(i,x);
   return sum;
}
float power(int n, float x)
{
  float y = 1;
  while (n) n&1 ? (y*=x,--n) : (x*=x,n/=2);
  return y;
}
void main()
{
  float binom[] = {1,2,1};
  printf("%d", poly(binom,2,10.0));
}
```

вместо

$$x^n = \begin{cases} 1, & n = 0 \\ x * x^{n-1}, & иначе \end{cases}$$

использовать

$$x^n = \begin{cases} 1, & n = 0 \\ x * x^{n-1}, & n - нечётно \\ (x^{n/2})^2, & n - чётно \end{cases}$$

# Пример оптимизации

```
float poly(float coef[], int n, float x)
{
   float sum = 0f;
   for (int i=0; i<=n; i++)
      sum += coef[i] * power(i,x);
   return sum;
}
float power(int n, float x)
{
  float y = 1;
  while (n) n&1 ? (y*=x,--n) : (x*=x,n/=2);
  return y;
}
void main()
{
  float binom[] = {1,2,1};
  printf("%d", poly(binom,2,10.0));
}
```

- Одно умножение на каждой из n+1 итерации цикл **for**
- Максимальное количество итераций цикла **while** равно 2*log(n)
  - 4 * log(i) операций умножения на каждой итерации **for**
- Память - константа

# Пример пессимизации

```
float poly(float coef[], int n, float x)
{
   float sum = 0f;
   int i;
   for (i=0; i<=n; i++)
      sum += coef[i] * exp(log(x)*i);
   return sum;
}
void main()
{
  float binom[] = {1,2,1};
  printf("%d", poly(binom,2,10));
}
```

$$x^n = e^{\log(x)*n}$$

# Пример пессимизации

```
float poly(float coef[], int n, float x)
{
   float sum = 0f;
   int i;
   for (i=0; i<=n; i++)
      sum += coef[i] * exp(log(x)*i);
   return sum;
}
void main()
{
  float binom[] = {1,2,1};
  printf("%d", poly(binom,2,10));
}
```

- Два умножения на каждой итерации **for**
- Неизвестное количество в exp и log
- Память – константа (скорее всего)

# Пример оптимизации

```
float poly(float coef[], int n, float x)
{
   float sum = 0f;
   float power;
   int i;

   for (i=0,power=1f; i<=n; power*=x,i++)
      sum += coef[i] * power;
   return sum;
}
void main()
{
  float binom[] = {1,2,1};
  printf("%d", poly(binom,2,10));
}
```

- На каждой итерации значение *power* увеличивается в *x* раз

# Пример оптимизации

```
float poly(float coef[], int n, float x)
{
   float sum = 0f;
   float power;
   int i;

   for (i=0,power=1f; i<=n; power*=x,i++)
      sum += coef[i] * power;
   return sum;
}
void main()
{
  float binom[] = {1,2,1};
  printf("%d", poly(binom,2,10));
}
```

- Два умножения на каждой итерации **for**
- Память – константа

# Пример оптимизации

```
float poly(float coef[], int n, float x)
{
   float sum = coef[n];
   for (i=n; i>=1; i--)
      sum = sum * x + coef[i];
   return sum;
}
void main()
{
  float binom[] = {1,2,1};
  printf("%d", poly(binom,2,10.0));
}
```

Схема Горнера:

$$\ldots((a_n * 10 + a_{n-1}) * 10 + a_{n-2}) * 10 + \ldots a_0$$

# Сравнение реализаций

| | Время (количество умножений) | Память |
|---|---|---|
| Рекурсивная power | $\sum_{i=0}^{n}(1+i)=\frac{n(n+1)}{2}$ | $O(n)$ |
| Итеративная power | $\sum_{i=0}^{n}(1+2\log_2 i)\le(n+1)*(1+2\log_2 n)$ | константа |
| exp(log(x)*i) | $\sum_{i=0}^{n}(2+T_{\log}(i)+T_{\exp}(x*\log(i)))$ | $\max_i(\max(S_{\exp}(x*\log(i)),S_{\log}(i))$ |
| Накопление power | $\sum_{i=0}^{n}2=2n+2$ | константа |
| схема Горнера | $\sum_{i=1}^{n}1=n$ | константа |

# Коварство O

- Функция $g(n)$ *имеет порядок* $O(f(n))$, если существуют $C_1$, $C_2$ такие, что
  $$C_1 f(n) <= g(n) <= C_2 f(n)$$
  почти для всех n
- Сортировка
  - «пузырёк» - $O(n^2)$
  - слиянием – $O(n \log(n))$

  Кто быстрее?
- Что такое асимптотическое поведение при $n<=2^{32}$ ?

# Мал оператор, да сложен!

Пример:

| | Увеличение целого | Добавление символа к строке |
|---|---|---|
| Pascal | S := S + 123 | S := S + “A”;<br>Inc(S[0]); S[ord[S[0]] = “A”; |
| Visual Basic | S = S + 123 | S = S + “A” |
| C | S += 123; | S = realloc(S,<br>strlen(S) + strlen(“A”) + 1);<br>strcpy(S,”A”); |