Программирование на C++

Лекция 8

ПМИ 2 курс

Демяненко Я.М.

ЮФУ 2026

Сортировки массивов

Основные требования к методам сортировок — *экономия памяти* и *быстродействие*.

Первое требование может быть выполнено, если переупорядочивание будет выполняться без создания вспомогательного массива.
Быстродействие зависит от количества выполняемых операций — сравнений и перестановок.

С этой точки зрения выделяют три **простых** метода сортировки, требующих порядка $\mathbf{n^2}$ операций, и **улучшенные** методы, порядок операций которых стремится к $\mathbf{n \cdot \log_2 n}$.

Три простые метода сортировки

- Выбором
- Обменом (методом «пузырька»)
- Включением

Сортировка выбором

Сортировка выбором основана на идее многократного выбора максимального (минимального) элемента и перемещения его в начало (конец) массива.

```
void selectionSort (double * a, int n) {
  int min ;
  for ( int i =0; i < n-1; ++i) {
    min = i ;
    for (int j  = i+1; j < n; ++j)
      if (a[ j ] < a[min]) min = j;
    swap (a[ i ], a[ min ]);
  }
}
```

6 3 8 **2** 4 5

2 3 8 6 4 5

2 **3** 8 6 4 5

2 3 8 6 4 5

2 3 8 6 **4** 5

2 3 4 6 8 5

Сортировка обменом (методом «пузырька»)

Сортировка обменом (методом «пузырька») основана на систематическом обмене значениями пар стоящих рядом элементов, для которых нарушено условие упорядоченности. Процесс обмена продолжается до тех пор, пока таких пар не останется.

Сортировка обменом (методом «пузырька»)

```
void bubbleSort (double * a, int n) {
  bool f = true;
  int i = n-1;
  int j;

  while (f) {
    f = false;
    for (j  = 0; j < i; ++j)
      if (a[ j ] > a[j+1]) {
        swap (a[ j ], a[j+1]);
        f = true;
      }
      i = j;
    }
}
```

6 3 8 2 4 5 3 6 2 4 5 **8**

3 6 8 2 4 5 3 2 6 4 5 **8**

3 6 8 2 4 5 3 2 4 6 5 **8**

3 6 2 8 4 5 3 2 4 5 **6 8**

3 6 2 4 8 5 2 3 4 5 **6 8**

3 6 2 4 5 **8** 2 3 4 **5 6 8**

Сортировка включением

Сортировка включением основана на включении одного элемента в упорядоченный набор с сохранением упорядоченности. Известно, что массив из одного элемента всегда отсортирован.

```
void insertionSort (double * a, int
n) {
  double x; int j;
  for (int i = 1; i <n; ++i) {
    x = a[i];
    j = i-1;
    while (j > =0 && a[ j ] > x) {
      a[j+1] = a[j];
      j --;
    }
    a[j+1] = x;
  }
}
```

3 6 2 4 5 8

3 6 2 4 5 8

2 3 6 4 5 8

2 3 4 6 5 8

2 3 4 5 6 8

Реализация insertionSort для случая параметров, задающих сортируемый сегмент массива посредством указателей

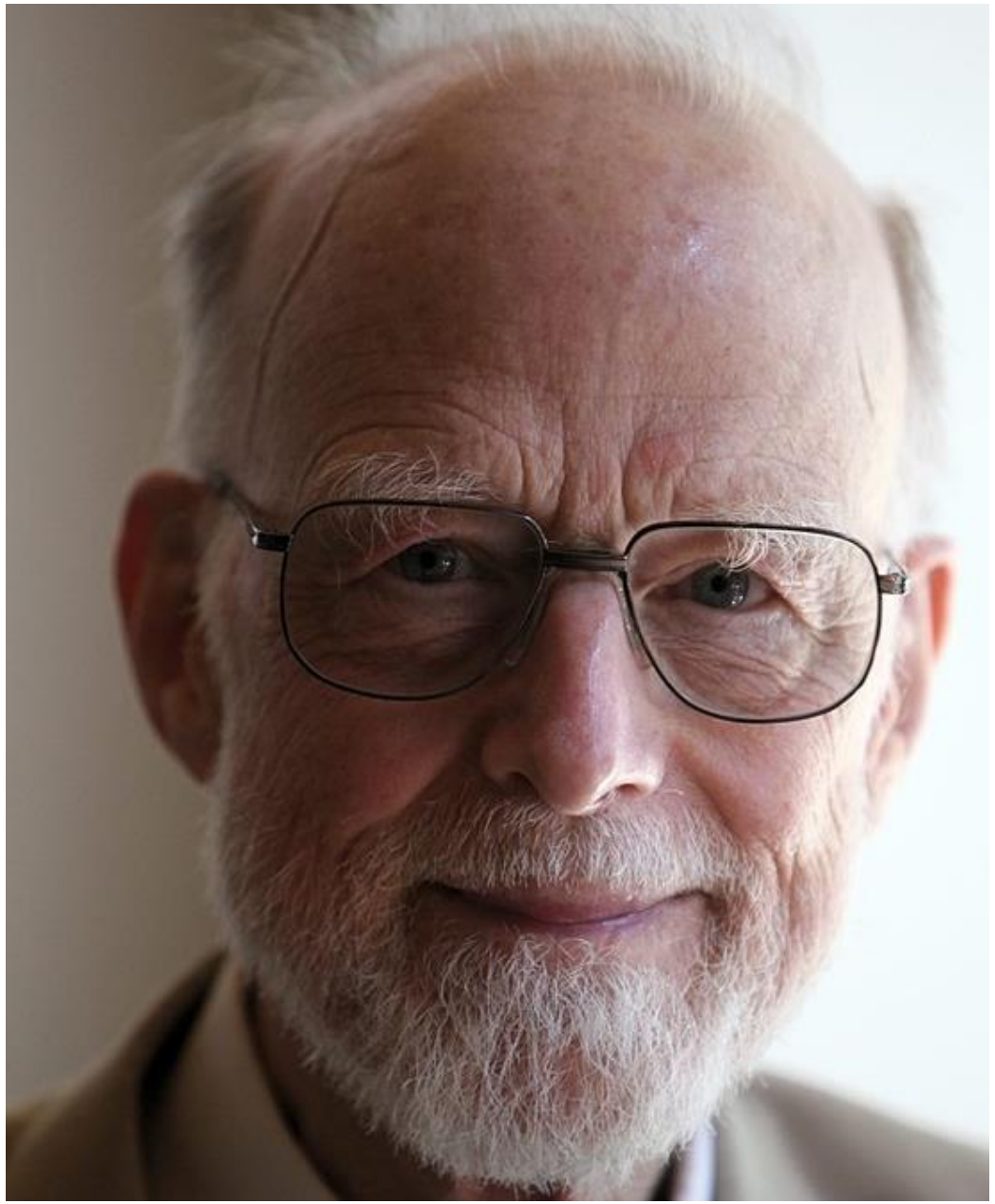
```
void insertionSort (double * a, double * b) {
  double x; double *d;
  for (double *c=a+1; c<b; ++c) {
    x =*c;
    d = c-1;
    while (d >=a && *d > x) {
      *(d+1) = *d;
      d--;
    }
    *(d+1) = x;
  }
}
```

- **Сэр Чарльз Э́нтони Ри́чард Хо́ар** (род. 11 января 1934, Коломбо, Цейлон, Британская империя, ныне Шри-Ланка) — английский учёный, специализирующийся в области информатики и вычислительной техники.

- Наиболее известен как разработчик алгоритма «быстрой сортировки» (1960), на сегодняшний день являющегося наиболее популярным алгоритмом сортировки.

- Образование: Московский государственный университет, Мертон-колледж (Оксфорд), Оксфордский университете

- **Сайт:** cs.ox.ac.uk/people/tony.hoare

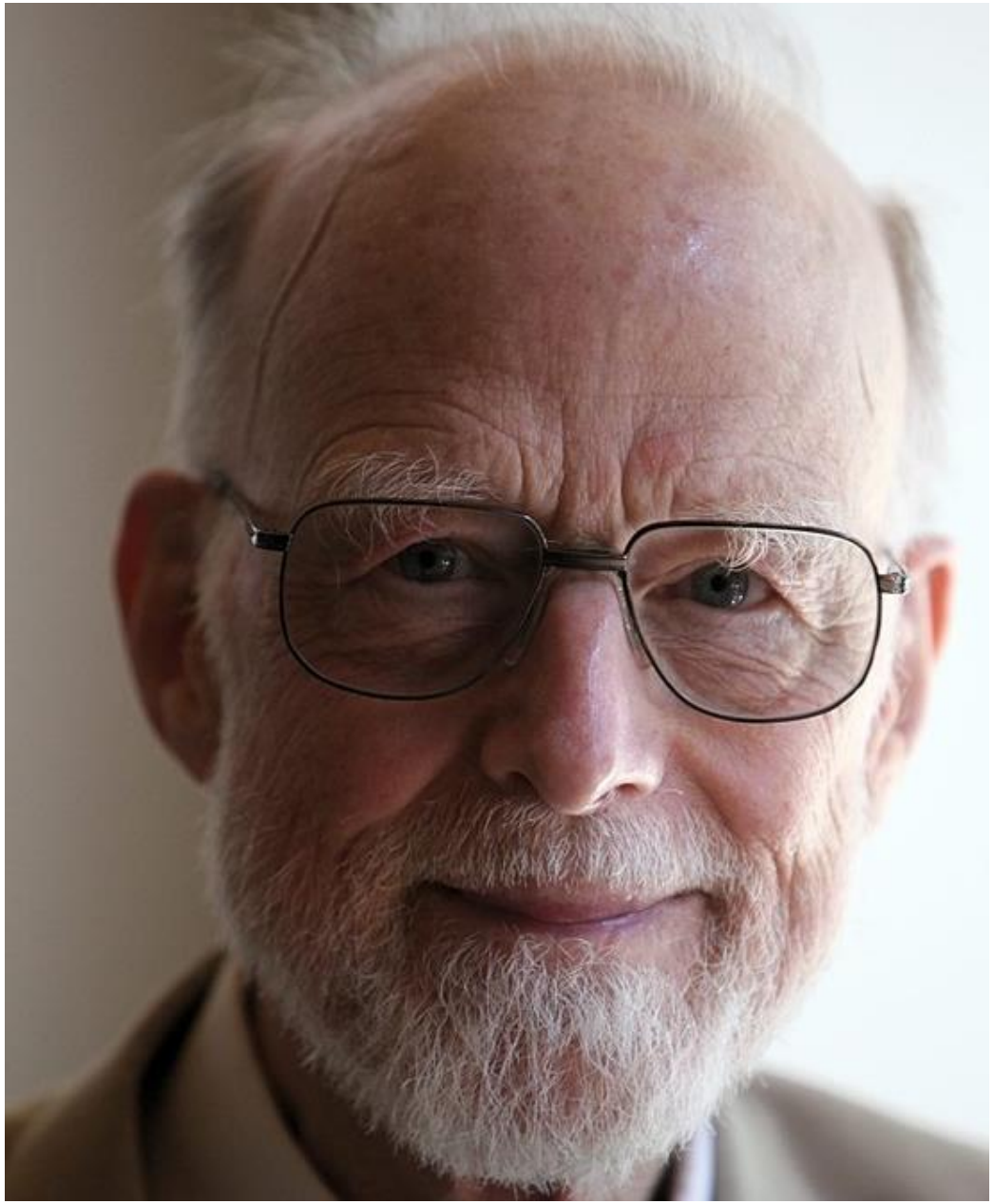


- **Сэр Чарльз Э́нтони Ри́чард Хо́ар**
- Другие известные результаты его работы: язык Z спецификаций и параллельная модель взаимодействия последовательных процессов (CSP, Communicating Sequential Process).

- В числе его заслуг — разработка логики Хоара (англ. Hoare Logic), научной основы для конструирования корректных программ, используемой для определения и разработки языков программирования.

- Хоар создал ряд трудов по созданию спецификаций, проектированию, реализации и сопровождению программ, показывающих важность научных результатов для увеличения производительности компьютеров и повышения надежности программного обеспечения.

- **Сайт:** cs.ox.ac.uk/people/tony.hoare







В конце июня в Санкт-Петербурге состоялась Девятая международная Ершовская конференция PSI'2014 (Перспективы систем информатики)

Сортировка Хоара или быстрая сортировка

Сортировка, носящая имя А. Хоара часто называется быстрой сортировкой.

Быстрая сортировка является **улучшенным методом сортировки обменом**.

Хоаром было отмечено, что для достижения большей эффективности необходимо производить обмены на больших расстояниях.

Алгоритм состоит из двух этапов

- Выполнение перестановки элементов таким образом, **чтобы массив разделился на две части относительно** некоторого элемента, называемого **опорным**.
- Левая часть после перестановки будет содержать элементы меньшие или равные опорному, правая часть — большие или равные опорному.
- Рекурсивное применение этого алгоритма к каждой из частей, пока размер сортируемой части больше единицы.

Реализация сортировки Хоара

```
void qSort(double*A, int low, int high) {
  int i = low, j = high;
  double x = A[(low+high)/2];  // x - опорный
  do {
    while(A[i] < x) ++i;  // поиск элемента для переноса
    while(A[j] > x) --j;  // поиск элемента для переноса
    if (i <= j){          // обмен элементов местами:
      swap(A[i],A[j]);
      ++i; --j;           // переход к следующим элементам:
    }
  } while(i < j);
  if (low < j) qSort(A, low, j);
  if (i < high) qSort(A, i, high);
}
```

3 6 **2** 4 5 8

2 6 **3** 4 5 8

2 6 3 4 5 8

Реализация qSort для случая параметров, задающих сортируемый сегмент массива посредством указателей.

```
void qSort(double*a, double *b) {
  double x = *(a+(b-a)/2);  // x - опорный
  double *p=a;
  double *q=b-1;
  do {
    while(*p < x) ++p;  // поиск элемента для переноса
    while(*q > x) --q;  // поиск элемента для переноса
    if (p < q){       // обмен элементов местами:
      swap(*p,*q);
    ++p; --q;         // переход к следующим элементам:
    }
  } while(p < q);
  if (a < q)  qSort(a, q+1);
  if (p < b-1) qSort(p, b);
}
```

Проблема и решение

Из **рекурсивной** природы быстрой сортировки следует **большое количество вызовов**, поэтому для **небольших** массивов она будет давать **худшие результаты** по сравнению с простыми методами.

На практике рекомендуется учитывать размер сортируемого сегмента, и **при уменьшении** его **ниже некоторой границы заменять** вызов рекурсивной сортировки **на один из простых методов** сортировки.

```
void qSortWithCut(double*a, double *b) {
  if (b-a<CUTOFF) {
    insertionSort(a,b);
    return;
  }
  double x = *(a+(b-a)/2);  // x - опорный
  double *p=a;
  double *q=b-1;
  do {
    while(*p < x) ++p;  // поиск элемента для переноса
    while(*q > x) --q;  // поиск элемента для переноса
    if (p <= q){       // обмен элементов местами:
      swap(*p,*q);
      ++p; --q;        // переход к следующим элементам:
    }
  } while(p < q);
  if (a < q)  qSort(a, q+1);
  if (p < b-1) qSort(p, b);
}
```

Задача

Упорядочить строки матрицы по возрастанию сумм их элементов

Индексная сортировка

Для индексной сортировки помимо сортируемого массива характерно использование **массива ключей** и **массива индексов**.

Массив ключей содержит значения, по которым производится сортировка. Он может отсутствовать, если ключи хранятся в самом сортируемом массиве.

Массив индексов на каждом шаге отражает размещение элементов в сортируемом массиве.

Процесс сортировки заключается в сравнении ключей (доступ к ним осуществляется через индекс) и перестановке индексов key[ind[i]]

Индексная сортировка

key 6 3 4 1 0 9

ind 0 1 2 3 4 5

1 2 3 4 0 **5**

1 0 2 3 4 5

1 **2 0** 3 4 5

1 2 **3 0** 4 5

1 2 3 **4 0** 5

4 3 1 2 0 5

```
void ind_sort(int a[][10],int n, int m, int ind[]) {
  int sum[10];
  for (int i=0; i<n; ++i) {
    ind[i]=i;  sum[i]=0;
    for(int j=0;j<m; ++j)
       sum[i]+=a[i][j];
  }
 bool flag=true;   int j=n-1;
 while (flag) {
   flag=false;
   for (int i=0; i<j; ++i) {
      if (sum[ind[i]]>sum[ind[i+1]]) {
         int k=ind[i];   ind[i]=ind[i+1];  ind[i+1]=k;
         flag=true;
       }
    }
    j--;
 }
}
```

Пример. Упорядочить строки матрицы по возрастанию их первых элементов

```
void sort(int **a, int n, int m) {
  bool flag=true;
  int j=n-1;
  while (flag) {
    flag=false;
    for (int i=0; i<j; ++i) {
      if (a[i][0]>a[i+1][0]) {
        int * k=a[i];
        a[i]=a[i+1];
        a[i+1]=k;
        flag=true;
      }
    }
    j--;
  }
}
```

