Лекция 9.
Сортировка

Постановка задачи сортировки

- Пусть $\mathbf{R_1, R_2, \ldots, R_n}$ – конечное множество записей;
- $\mathbf{K_1, K_2, \ldots, K_n}$ – упорядоченное множество ключей.

Необходимо найти такую перестановку записей $\mathbf{R_{i1}, R_{i2}, \ldots, R_{in}}$, чтобы выполнялось неравенство $\mathbf{K_{i1} \le K_{i2} \le \ldots \le K_{in}}$.

Сортировка называется устойчивой, если записи с равными ключами остаются на месте.

Ключ	Данные
K_1	
K_2	
K_3	
• • •	
K_n	

Классы алгоритмов сортировки

- Вставка. На j-ом этапе j-ый ключ …
- Обмен
- Выбор. На j-ом этапе …
- Распределение
- Слияние

Эффективность алгоритмов сортировки

- Число сравнений ключей
- Число перестановок
- Определяется в трех случаях: лучшем, среднем, худшем

Сортировки вставками

- Простые вставки
- Бинарные вставки
- Метод Шелла

Сортировка простыми вставками

- Пусть $1 < j \le N$
- записи $R_1, \ldots, R_{j-1}$ уже отсортированы
- куда вставить ключ K_j

Простые вставки

K1	K2	K3	K4	K5	
8	7	2	4	6	j=2
7	8	2	4	6	j=3
2	7	8	4	6	j=4
2	4	7	8	6	j=5
2	4	6	7	8	

Алгоритм сортировки простыми вставками

1. Выполнить шаги от 2 до 5 при j = 2, 3, ..., N и после этого завершить алгоритм.
2. Установить: $i = j - 1$, $K = K_j$, $R = R_j$.
3. Если $K > K_i$, то перейти на шаг 5. (Нашли искомое место для записи R.)
4. $R_{i+1} = R_i$, $i = i - 1$. Если $i > 0$, то перейти на 3. (Если $i = 0$, то K—наименьший из ключей, и запись R должна занять первую позицию.)
5. Установить $R_{i+1} = R$.

Простые вставки

```
int temp, j;

for (int i=1; i<N; i++){
   j=i;
   while(a[j]<a[j-1] && j!=0) {
            temp=a[j];
            a[j]=a[j-1];
            a[j-1]=temp;
            j--;    }
}
```

Анализ сложности

$$T_{ср} \approx 1 + 2 + \cdots + j/2 + \ldots + (N-1)/2 \approx N^2/4$$

Необходимо сделать $O(N^2)$ сравнений и $O(N^2)$ перестановок.

Метод Шелла

Сортировка с убывающим шагом

- **h** - шаг сортировки
- последовательность шагов **h**:   **8, 4, 2, 1**

Пусть даны ключи $\mathbf{K_1, K_2, \ldots , K_8}$ и задана последовательность шагов **h**=**{ 4, 2, 1}.**

1. Сортируем ключи, отстоящие друг от друга на четыре позиции ($\mathbf{K_1}$,$\mathbf{K_5}$) ($\mathbf{K_2}$,$\mathbf{K_6}$) ($\mathbf{K_3}$,$\mathbf{K_7}$) ($\mathbf{K_4}$,$\mathbf{K_8}$) методом простых вставок;

2. Сортируем ключи, отстоящие друг от друга на 2 позиции ($\mathbf{K_1}$,$\mathbf{K_3}$,$\mathbf{K_5}$,$\mathbf{K_7}$) ($\mathbf{K_2}$,$\mathbf{K_4}$,$\mathbf{K_6}$,$\mathbf{K_8}$) методом простых вставок;

3. Сортируем весь массив методом простых вставок.

Метод Шелла

$T_{cp} \approx 15 * N^{1.25}$

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	
3	8	5	12	1	10	4	16	13	6	7	2	9	17	14	11	h = 8
3	6	5	2	1	10	4	11	13	8	7	12	9	17	14	16	h = 4
1	6	4	2	3	8	5	11	9	10	7	12	13	17	14	16	h = 2
1	2	3	6	4	8	5	10	7	11	9	12	13	16	14	17	h = 1

```
int temp, j;
int h[]={8,4,2,1};
for(int t=0; t<4; t++){                //перебираем h
    for(int k=0; k<h[t]; k++) {   //создаем h последовательностей
        for (int i=k+h[t]; i<N; i=i+h[t]){  //сортируем с шагом h
            j=i;
            while(a[j]<a[j-h[t]] && j!=k) {
                temp = a[j];
                a[j] = a[j-h[t]];
                a[j-h[t]] = temp;
                j = j-h[t];
} } } }
```

Обменные сортировки

- Метод пузырька
- Шейкер-сортировка
- Сортировка расческой
- Быстрая сортировка Хоара

Метод пузырька

обмен

2	**5**	**7**	**1**	**3**	**8**	**4**	**6**	*1-ый проход*
2	**5**	**1**	**3**	**7**	**4**	**6 |**	**8**	*2-ой проход*
2	**1**	**3**	**5**	**4**	**6 |**	**7**	**8**	*3-ий проход*
1	**2**	**3**	**4**	**5 |**	**6**	**7**	**8**	*4-ый проход*
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	

Метод пузырька

1. Установить BOUND = N. (BOUND—индекс самого верхнего элемента, о котором еще не известно, занял ли он уже свою окончательную позицию);
2. t = 0. Выполнить шаг 3 при j = 1, 2, ..., BOUND − 1.
3. Если $K_j > K_{j+1}$, то поменять местами $R_j \leftrightarrow R_{j+1}$ и установить t = j.
4. Если t = 0, то КОНЕЦ , иначе установить BOUND = t и перейти к шагу 2.

$T_{ср} \approx ½(N^2 - N*\ln N)$ - число сравнений

$T_{об} \approx N^2/4$ - число обменов

Шейкер-сортировка

Шейкер-сортировка является усовершенствованным методом пузырьковой сортировки. Массив просматривается поочередно справа налево и слева направо. Просмотр массива осуществляется до тех пор, пока все элементы не встанут в порядке возрастания (убывания). Количество просмотров элементов массива определяется моментом упорядочивания его элементов.

Лучший случай
 — отсортированный
массив $O(n)$,
худший —
отсортированный
в обратном порядке $O(n^2)$.

```
void Sheiker(int a[], int N){
   int temp flag = 1,  left = 0,  right = N – 1 ;
while ((left < right) && flag > 0)  {  // пока левая граница не сомкнётся с правой
     flag = 0;
     for (int i = left; i < right; i++)  {            //двигаемся слева направо
        if (a[i] > a[i+1])   {      // если следующий элемент меньше текущего,  меняем
              temp = a[i];     a[i] = a[i+1];    a[i+1] = t;
              flag = 1;      }                  }           // перемещения  были
     right--;                          // сдвигаем правую границу на предыдущий элемент
     for (int i = right; i > left; i--)   {            //двигаемся справа налево
        if (a[i-1] > a[i]) {        // если предыдущий элемент больше текущего,  меняем
           temp = a[i];     a[i] = a[i-1];     a[i-1] = t;
           flag = 1;        }                 }                // перемещения были
    left++;                              // сдвигаем левую границу на следующий элемент
    if(flag == 0) break;                              //если перемещений больше нет
}  }
```

Сортировка расческой

Почему пузырек медленно всплывает?

Потому что за проход он перемещается на 1 позицию. А почему он перемещается только на 1 позицию? Потому, что сравниваются и переставляются соседние элементы. Можно сравнивать не соседние элементы, а находящиеся на некотором расстоянии (постепенно уменьшая расстояние с каждым проходом).

Расчёска лучше пузырьковой сортировки, потому что в ней намного меньше операций перестановки. Именно перестановка занимает основное время процессора. В самом худшем случае алгоритм сортировки расчёской будет работать так же, как и пузырьковая, а в среднем — алгоритм работает быстрее пузырьковой.

Оптимальное значение фактора уменьшения равно $1/(1-e^{-\varphi}) \approx 1.247$, где *e* – основание натурального логарифма, а φ= 1,618 – золотое сечение.

$T_{ср}$ = O(N lnN), в худшем – O(N^2).

```
void Comb( int  a[], int N ) {
          int  step = N;
          int  flag=1, temp;
   while ( step > 1 ||  flag==1 ) {
          step  = step / 1.25;
          flag = 0;
    for(int  i=0; i < N - step;  i++) {
          int j = i + step;
         if (a[i] > a[j]) {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                    flag=1; }
 }  }  }
```

Быстрая сортировка Хоара

$T_{ср} \approx N*\log_2 N$ - число сравнений

Ключи $\mathbf{K_1, K_2, \dots, K_n}$, i=1, j=n.

1. Выберем опорным некоторый элемент массива.

Например, $\mathbf{K_1}$.

1. Сравниваем $\mathbf{K_1}$ и $\mathbf{K_j}$. Пока $\mathbf{K_1} < \mathbf{K_j}$ **j=j-1** и повторяем сравнение.

2. Сравниваем $\mathbf{K_1}$ и $\mathbf{K_i}$. Пока $\mathbf{K_1} > \mathbf{K_i}$ **i=i+1** и повторяем сравнение.

3. Нашли слева $\mathbf{K_i} > \mathbf{K_1} > \mathbf{K_j}$ справа. Меняем местами $\mathbf{K_i}$ и $\mathbf{K_j}$.
i=i+1, j=j-1 и снова выполняем шаги 1 и 2.

4. Если просмотрены все элементы (**i>=j**), то выполняем шаги 1, 2, 3 для левой части (0,j) и для правой части (i,n-1).

```
void qs(int a[], int l, int r){
  int i,j,temp;
  i=l;
  j=r;
  int x=a[i];
   do {
          while(a[i]<x && i<r)  i++;
          while(x<a[j] && j>l)  j--;

          if(i<=j){
               temp=a[i];
               a[i]=a[j];
               a[j]=temp;
               i++;
               j--;
                   }
      } while(i<=j);

    if(l<j) qs(a,l,j);
    if(i<r) qs(a,i,r);
}
```

Быстрая сортировка Хоара

Два указателя: i=1, j=N.

Последовательность ключей: 7 5 10 8 4 9 2 1 12 6

Выбираем первый ключ: 7

7 5 10 8 4 9 2 1 12 6

____

6 5 1 2 4 9 8 10 12 7

4 5 1 2 6 | 9 8 10 12 7

2 1 5 4 6 | 9 8 10 12 7

1 2 5 4 6 | 9 8 10 12 7

1 2 4 5 6 | 9 8 10 12 7

1 2 4 5 6 | 7 8 10 12 9

1 2 4 5 6 | 7 8 10 12 9

1 2 4 5 6 | 7 8 9 12 10

1 2 4 5 6 | 7 8 9 10 12

Сортировка выбором

2 5 7 1 3 8 4 6

1 |5 7 2 3 8 4 6

1 2 |7 5 3 8 4 6

1 2 3 |5 7 8 4 6

• • •

Сортировка выбором

1. Выполнить шаги 2 и 3 при $j = 1, ..., N\text{-}1$.
2. Найти наименьший из ключей
   $K_j, ..., K_{N-1}, K_N$; пусть это будет K_i.
3. Поменять местами записи $R_i \leftrightarrow R_j$.

$T_{ср} \approx ½*N^2$ - число сравнений

Сортировка слиянием

Сортировки слиянием работают по такому принципу:

- ищутся (или формируются) упорядоченные подмассивы.
- упорядоченные подмассивы соединяются в общий упорядоченный подмассив.

Алгоритм был предложен Джоном фон Нейманом.

D1: $i = 1;\ j = 1;\ k = 1;$

D2: если $x_i \le y_j$, то перейти к шагу D5;

D3: $z_k = y_j$; $k = k+1$; $j = j+1$; если $j \le n$, то перейти к шагу D2;

D4: $z_k \ldots z_{m+n} = x_i \ldots x_m$; СТОП;

D5: $z_k = x_i$; $k = k+1$; $i = i+1$; если $i \le m$, то перейти к шагу D2;

D6: $z_k \ldots z_{m+n} = y_j \ldots y_n$.

$x_i=(7,10,12)$ $y_j=(3,8,11)$

$z_k=(3, 7, 8, 10, 11, 12)$

$$\begin{cases} 7 \quad 10 \quad 12 \\ 3 \quad 8 \quad 11 \end{cases}$$

$$3 \begin{cases} 7 \quad 10 \quad 12 \\ 8 \quad 11 \end{cases}$$

$$3 \quad 7 \begin{cases} 10 \quad 12 \\ 8 \quad 11 \end{cases}$$

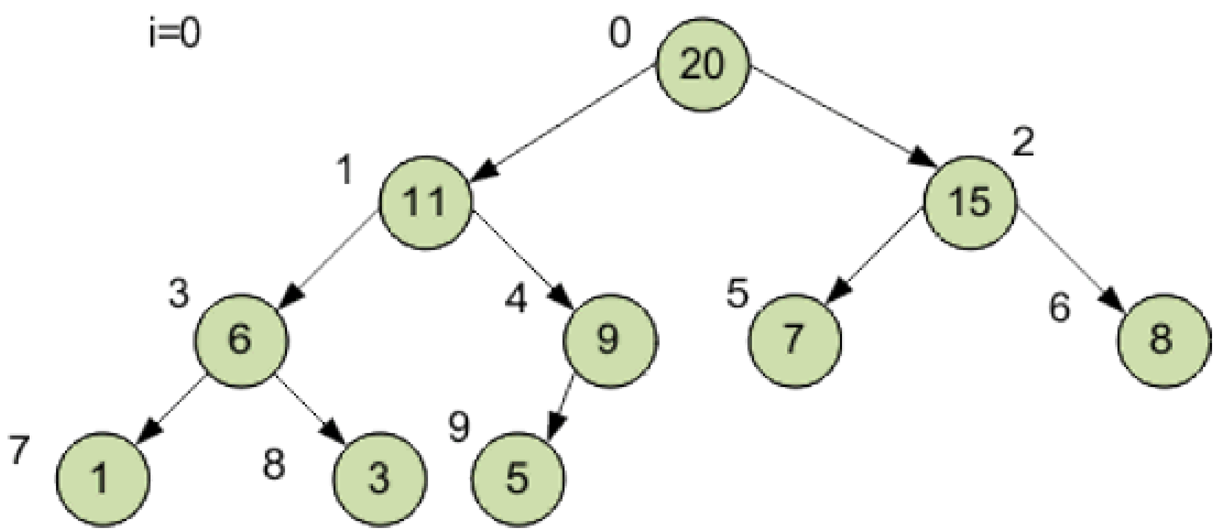
$$3 \quad 7 \quad 8 \begin{cases} 10 \quad 12 \\ 11 \end{cases}$$

Пирамидальная сортировка
(сортировка с помощью кучи)

int a[] = {20, 11, 15, 6 , 9 , 7 , 8 , 1 , 3 , 5}

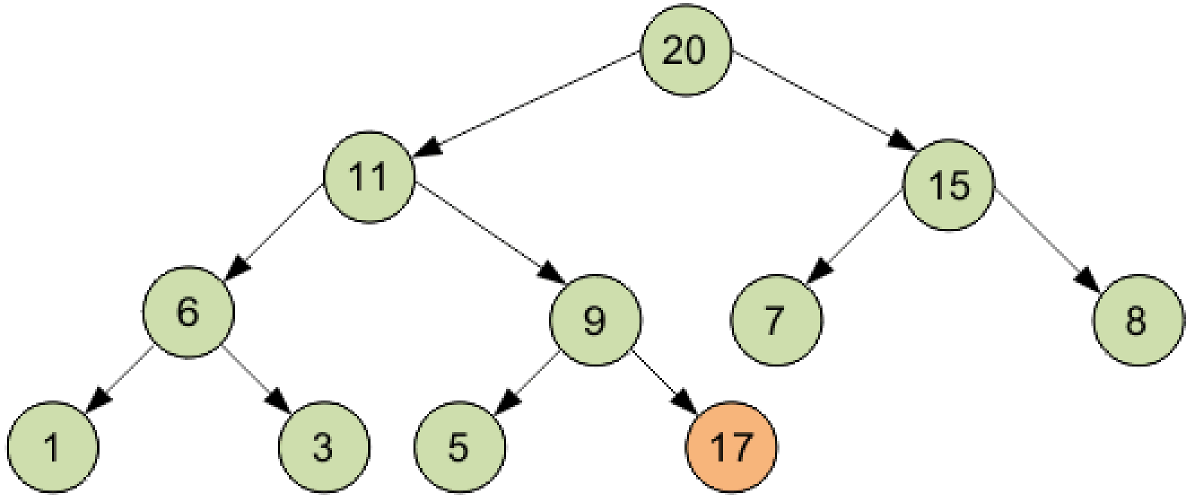
0 1 2 3 4 5 6 7 8 9

Двоичную кучу удобно хранить в виде одномерного массива. Левый потомок вершины с индексом i имеет индекс 2*i+1, правый потомок вершины с индексом i имеет индекс 2*i+2.

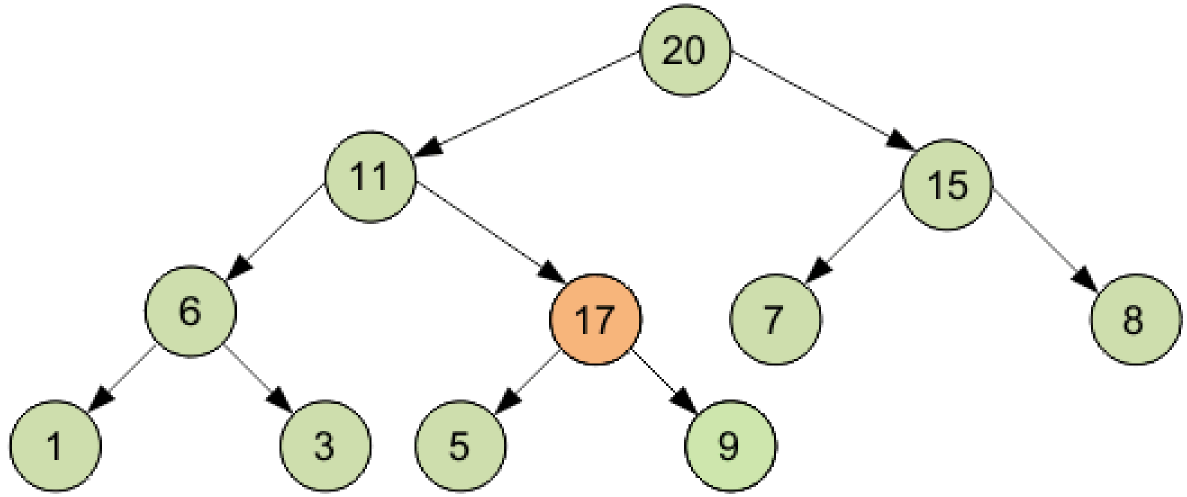


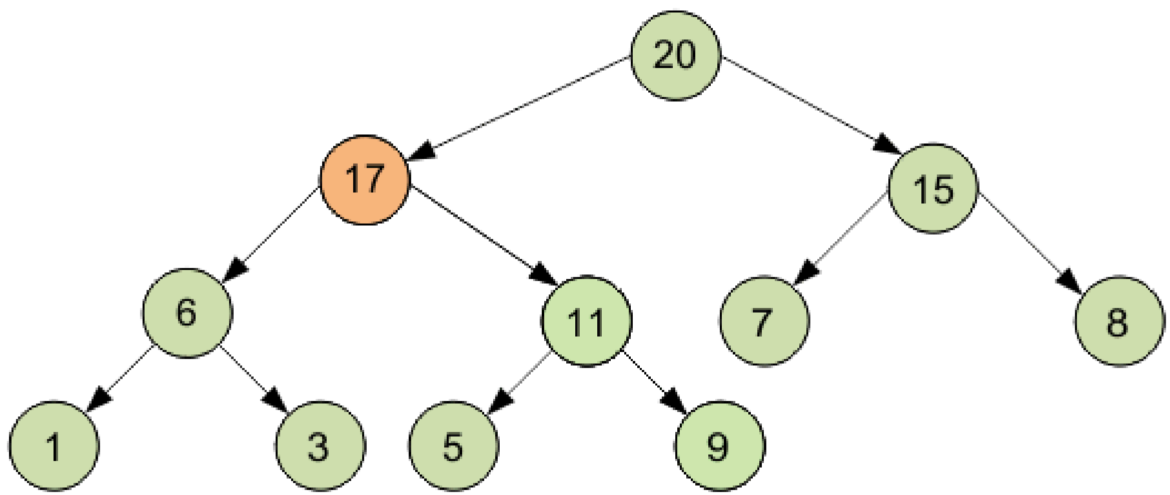
Добавление элемента кучи

Новый элемент добавляется на последнее место в массиве, то есть позицию с максимальным индексом.



Возможно, что при этом будет нарушено основное свойство кучи, так как новый элемент может быть больше родителя. В таком случае новый элемент «поднимается» на один уровень (менять с вершиной-родителем) до тех пор, пока не будет соблюдено основное свойство кучи.





Количество «подъемов» не больше высоты дерева, то есть равна $\log_2 N$.

Построение кучи в массиве

```
void heap(int a[], int n, int i)   {

    int  largest = i;      //  предположим, наибольший элемент - корень
    int  l = 2*i + 1;      // левый = 2*i + 1
    int  r = 2*i + 2;     // правый = 2*i + 2

   if (l < n && a[l] > a[largest])   // Если левый дочерний элемент больше корня
            largest = l;

   if (r < n && a[r] > a[largest])   //Если правый дочерний элемент больше
            largest = r;

    if (largest != i)    {    // Если самый большой элемент не корень
          int   temp=a[i];
          a[i]=a[largest];
          a[largest]=temp;
          heap(a, n, largest);  // рекурсивно преобразуем в двоичную кучу
                        }
 }
```

```
void   heapSort(int a[], int n){
   int temp;
  for (int i = n / 2 - 1; i >= 0; i--)   // Построение кучи (перегруппируем массив)
        heap(a, n, i);                      // максимальный в корень a[0]

// извлекаем элементы из кучи

 for (int i=n-1; i>=0; i--)   {        // Перемещаем текущий корень в конец
     temp=a[0];
     a[0]=a[i];
     a[i]=temp;
     heap(a, i, 0);    }      // вызываем  heap на уменьшенной куче
 }
```