Изменение размера массива

Стек: изменение размера массива

- Проблема. От клиента требуется указывать размер стека
- Как увеличивать и уменьшать размер массива?

- Первый подход
  - push(): увеличивать размер массива s[] на 1
  - pop(): уменьшать размер массива s[] на 1

- Стоимость
  - Требуется копировать все элементы в новый массив
  - Стоимость вставки первых N элементов 1 + 2 + 3 + ... + N

Стек: изменение размера массива

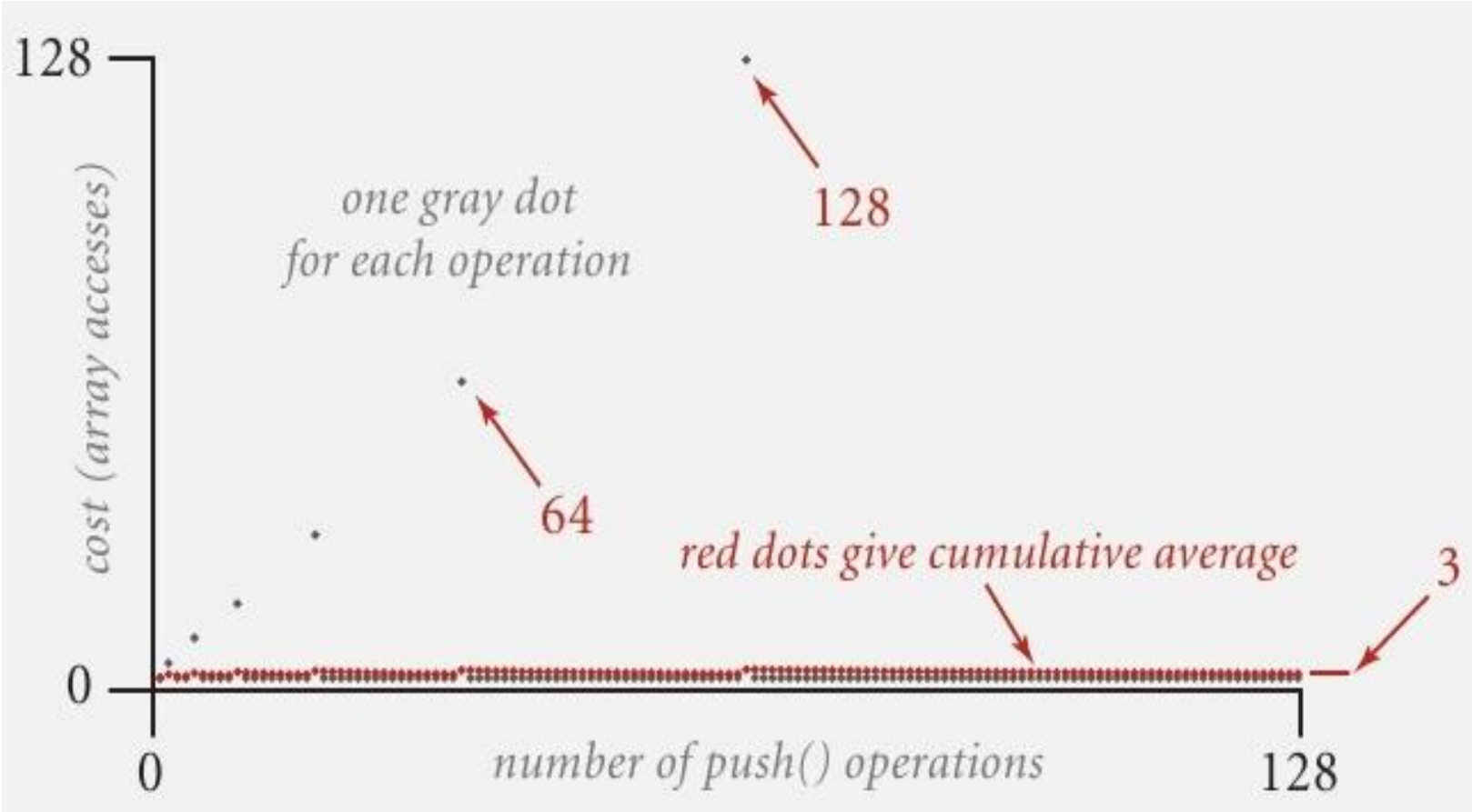
- Если массив полон, то создать новый массив в два раза больше и копировать элементы

```
public ResizingArrayStackOfStrings()
{  s = new String[1];  }

public void push(String item)
{
   if (N == s.length) resize(2 * s.length);
   s[N++] = item;
}

private void resize(int capacity)
{
   String[] copy = new String[capacity];
   for (int i = 0; i < N; i++)
      copy[i] = s[i];
   s = copy;
}
```

- Стоимость. Сложность вставки первых N элементов пропорциональна N

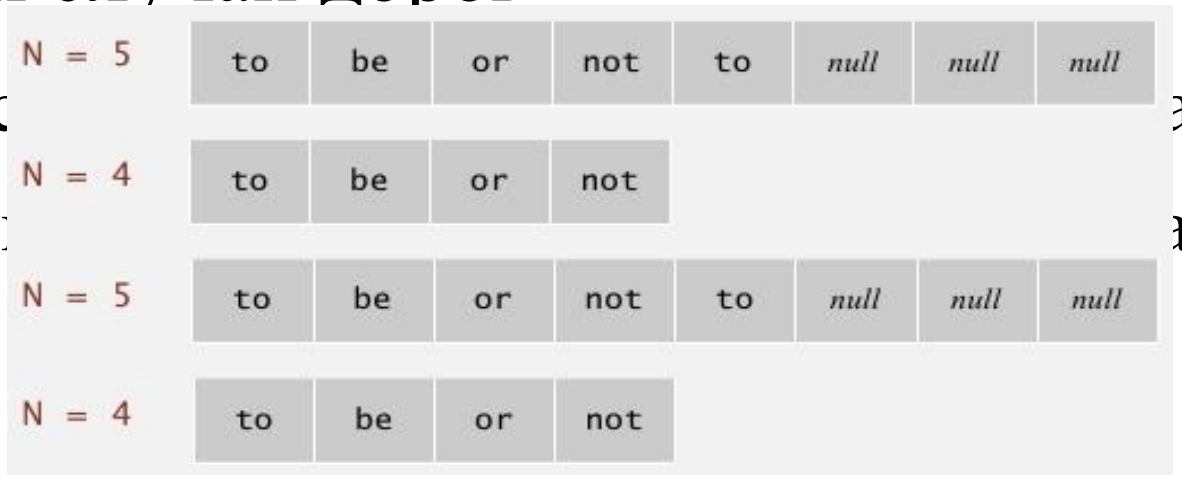
Стек: амортизированная стоимость добавления в стек

- Стоимость добавления первых N элементов: N + (2 + 4 + 8 + … + N) ~ 3N



Стек: изменение размера массива

- Как изменять размер массива?
- Первый подход
  - push(): увеличивать размер массива s[] в два раза, когда массив полон
  - pop(): уменьшать размер массива s[] в два раза, когда массив на половину пуст
- Худший случай дорог
  - Предс ассив полон
  - Слож альна N



Стек: изменение размера массива

- Эффективный подход
  - push(): увеличивать размер массива s[] в два раза, когда массив полон
  - pop(): уменьшать размер массива s[] в два раза, когда массив заполнен на четверть

```
public String pop()
{
   String item = s[--N];
   s[N] = null;
   if (N > 0 && N == s.length/4) resize(s.length/2);
   return item;
}
```

- Массив заполнен от 25% до 100%

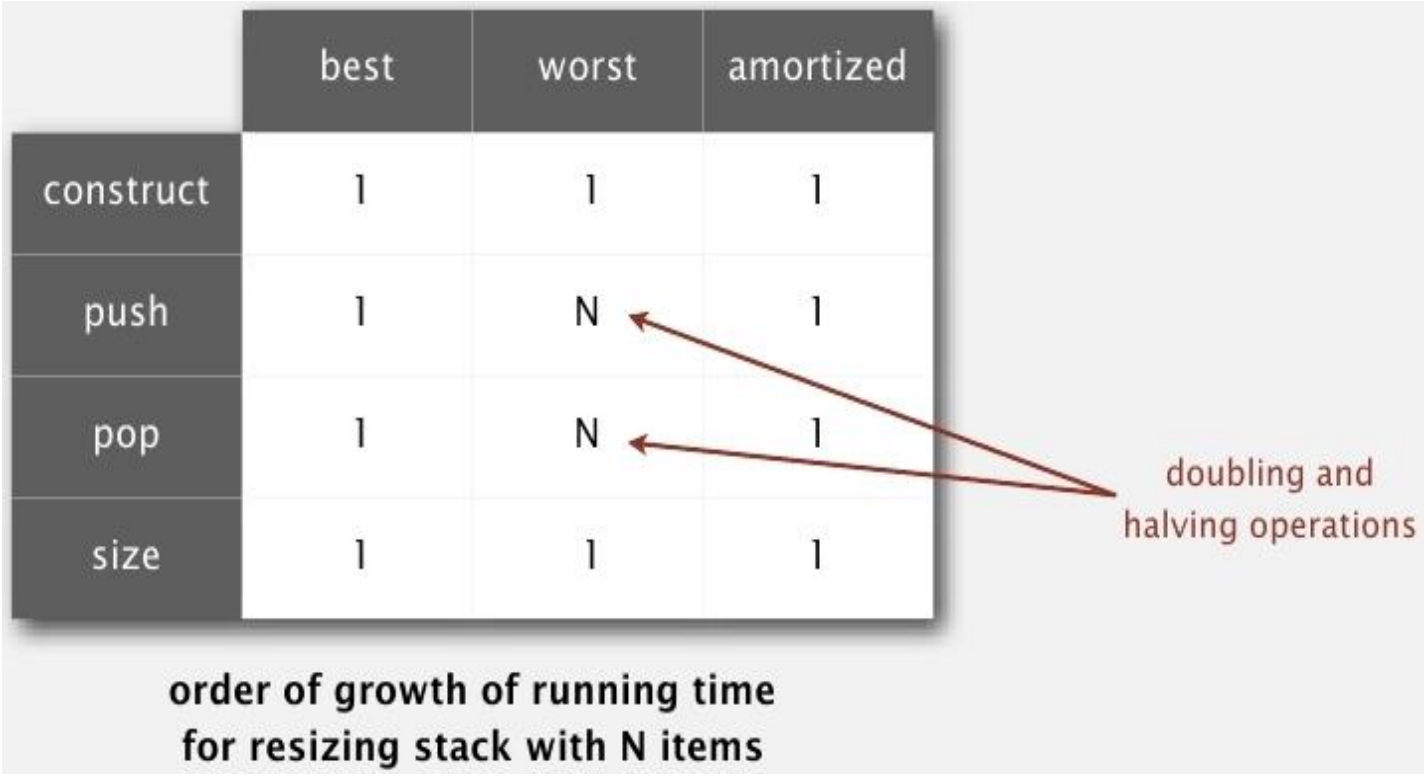
Стек: изменение размера массива

push()	pop()	N	a.length	a[] 0	1	2	3	4	5	6	7
		0	1	*null*							
to		1	1	to							
be		2	2	to	be						
or		3	4	to	be	or	*null*				
not		4	4	to	be	or	not				
to		5	8	to	be	or	not	to	*null*	*null*	*null*
-	to	4	8	to	be	or	not	*null*	*null*	*null*	*null*
be		5	8	to	be	or	not	be	*null*	*null*	*null*
-	be	4	8	to	be	or	not	*null*	*null*	*null*	*null*
-	not	3	8	to	be	or	*null*	*null*	*null*	*null*	*null*
that		4	8	to	be	or	that	*null*	*null*	*null*	*null*
-	that	3	8	to	be	or	*null*	*null*	*null*	*null*	*null*
-	or	2	4	to	be	*null*	*null*				
-	be	1	2	to	*null*						
is		2	2	to	is						

Trace of array resizing during a sequence of push() and pop() operations

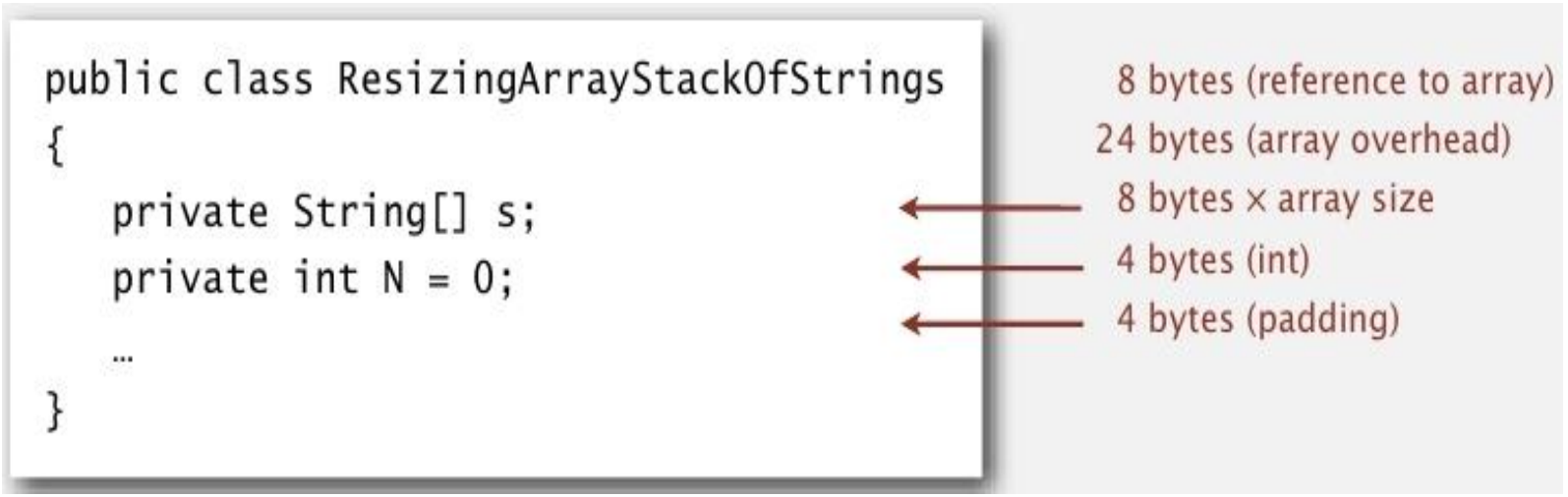
Стек: амортизированный анализ

- Предположение. Начиная с пустого стека, последовательность из M push/pop операций занимает время пропорциональное M



Стек: использование памяти

- Предположение. Используется от ~ 8N до ~ 32N байт для стека из N элементов
  - ~ 8N когда стек полон
  - ~ 32N когда стек заполнен на четверть



- Учитывается память, занимаемая самим стеком, но не данными

Очередь с приоритетом
(Priority Queue)

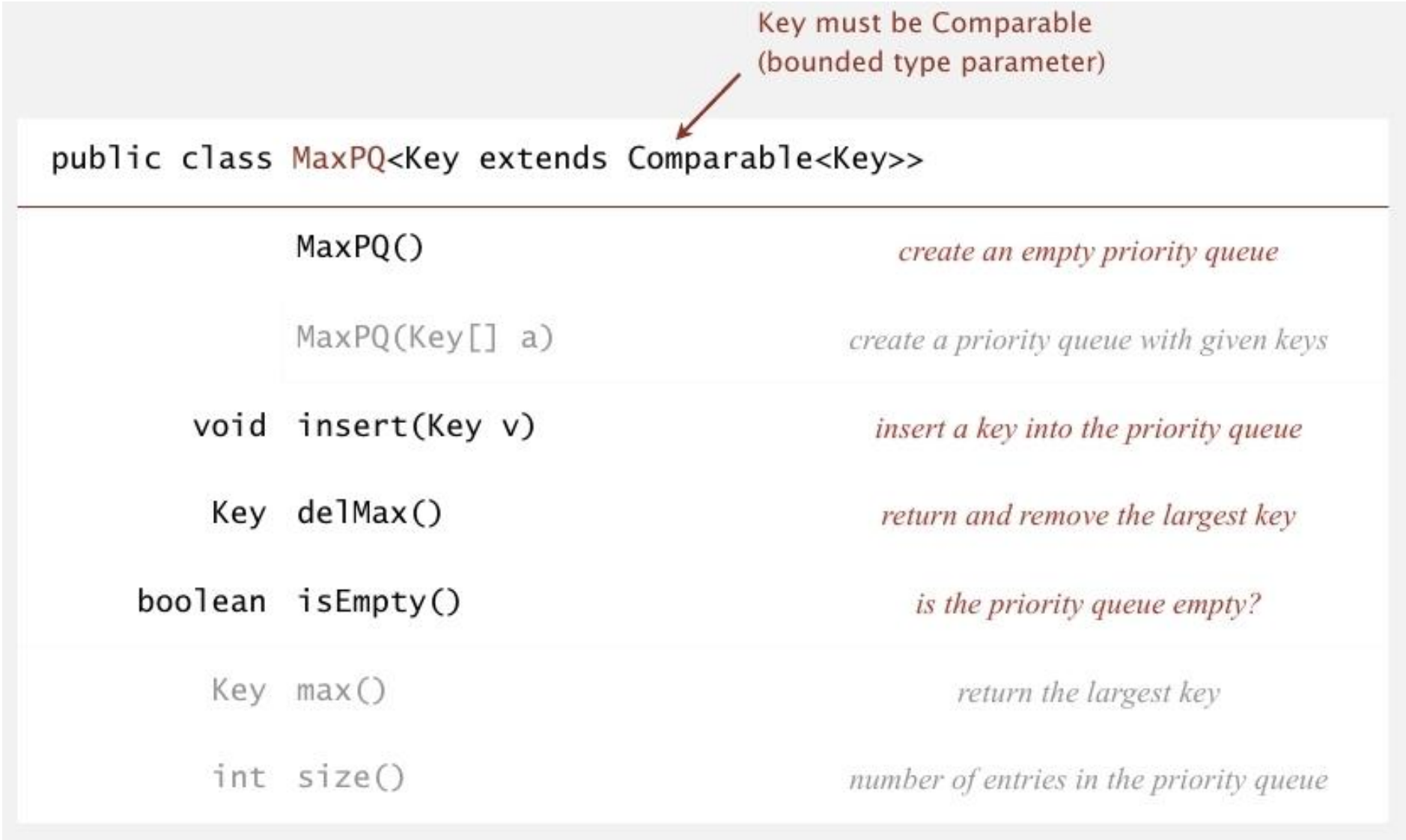
Очередь с приоритетом

- Коллекции. Вставка и удаление элементов. Какой элемент удалять?
- Стек. LIFO
- Очередь. FIFO
- Рандомизирова… …даляется случайный элем…
- Очередь с прио… …тся самый большой (или м… …ент

operation	*argument*	*return value*
insert	P	
insert	Q	
insert	E	
remove max		Q
insert	X	
insert	A	
insert	M	
remove max		X
insert	P	
insert	L	
insert	E	
remove max		P

API очереди с приоритетом

- Требование. Элементы должны быть сравнимы



Использование очереди с приоритетам

- Event-driven simulation. [customers in a line, colliding particles]
- Numerical computation. [reducing roundoff error]
- Data compression. [Huffman codes]
- Graph searching. [Dijkstra's algorithm, Prim's algorithm]
- Number theory. [sum of powers]
- Artificial intelligence. [A* search]
- Statistics. [maintain largest M values in a sequence]
- Operating systems. [load balancing, interrupt handling]
- Discrete optimization. [bin packing, scheduling]
- Spam filtering. [Bayesian spam filter]

Пример очереди с приоритетом

- Задача. Найти наибольшие M элементов в потоке из N элементов
  - Отслеживание транзакций

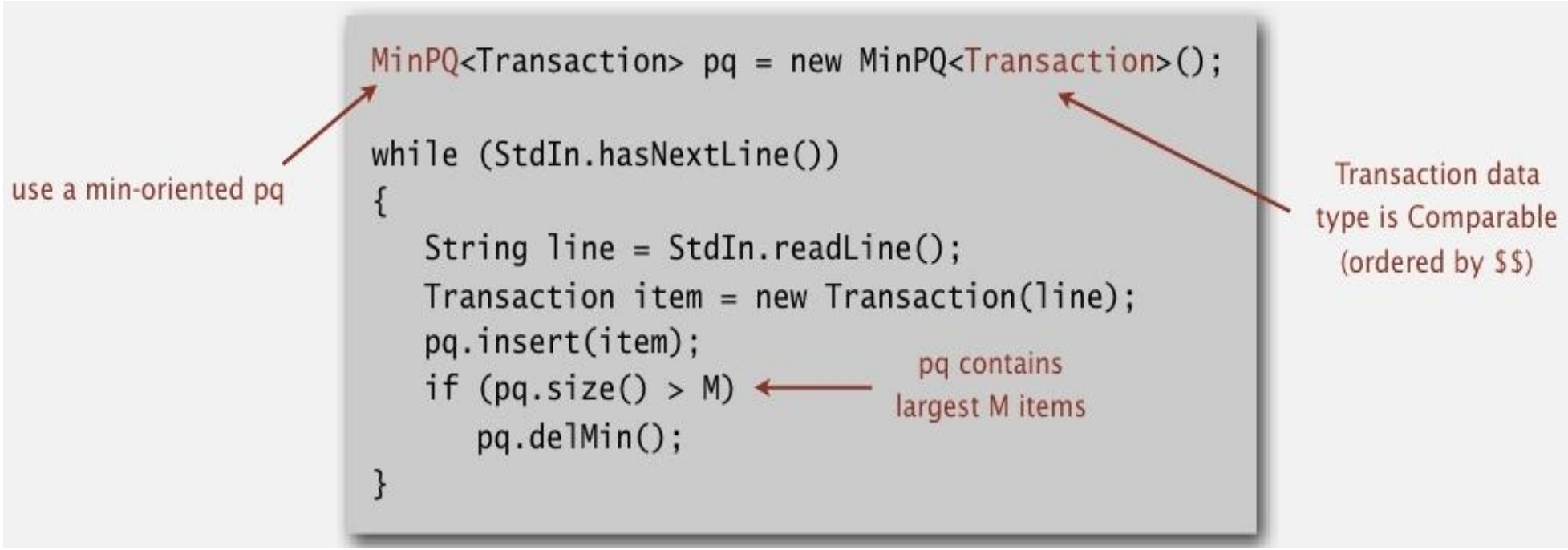
```
% more tinyBatch.txt
Turing       6/17/1990   644.08
vonNeumann   3/26/2002  4121.85
Dijkstra     8/22/2007  2678.40
vonNeumann   1/11/1999  4409.74
Dijkstra    11/18/1995   837.42
Hoare        5/10/1993  3229.27
vonNeumann   2/12/1994  4732.35
Hoare        8/18/1992  4381.21
Turing       1/11/2002    66.10
Thompson     2/27/2000  4747.08
Turing       2/11/1991  2156.86
Hoare        8/12/2003  1025.70
vonNeumann  10/13/1993  2520.97
Dijkstra     9/10/2000   708.95
Turing      10/12/1993  3532.36
Hoare        2/10/2005  4050.20
```

```
% java TopM 5 < tinyBatch.txt
Thompson     2/27/2000  4747.08
vonNeumann   2/12/1994  4732.35
vonNeumann   1/11/1999  4409.74
Hoare        8/18/1992  4381.21
vonNeumann   3/26/2002  4121.85
```

sort key

Пример очереди с приоритетом

- Задача. Найти наибольшие M элементов в потоке из N элементов
  - Отслеживание транзакций
  - Поиск файлов и директорий



Пример очереди с приоритетом

- Задача. Найти наибольшие M элементов в потоке из N элементов

order of growth of finding the largest M in a stream of N items

implementation	time	space
sort	N log N	N
elementary PQ	M N	M
binary heap	N log M	M
best in theory	N	M

Очередь с приоритетом: неупорядоченная и упорядоченная реализация

operation	*argument*	*return value*	*size*	*contents (unordered)*							*contents (ordered)*						
insert	P		1	P							P						
insert	Q		2	P	Q						P	Q					
insert	E		3	P	Q	E					E	P	Q				
remove max		Q	2	P	E						E	P					
insert	X		3	P	E	X					E	P	X				
insert	A		4	P	E	X	A				A	E	P	X			
insert	M		5	P	E	X	A	M			A	E	M	P	X		
remove max		X	4	P	E	M	A				A	E	M	P			
insert	P		5	P	E	M	A	P			A	E	M	P	P		
insert	L		6	P	E	M	A	P	L		A	E	L	M	P	P	
insert	E		7	P	E	M	A	P	L	E	A	E	E	L	M	P	P
remove max		P	6	E	M	A	P	L	E		A	E	E	L	M	P	

A sequence of operations on a priority queue

Очередь с приоритетом: неупорядоченная реализация

```
public class UnorderedMaxPQ<Key extends Comparable<Key>>
{
   private Key[] pq;    // pq[i] = ith element on pq
   private int N;       // number of elements on pq

   public UnorderedMaxPQ(int capacity)
   {  pq = (Key[]) new Comparable[capacity];  }

   public boolean isEmpty()
   {  return N == 0; }

   public void insert(Key x)
   {  pq[N++] = x;  }

   public Key delMax()
   {
      int max = 0;
      for (int i = 1; i < N; i++)
         if (less(max, i)) max = i;
      exch(max, N-1);
      return pq[--N];
   }
}
```

no generic array creation

less() and exch() similar to sorting methods

null out entry to prevent loitering

Пример очереди с приоритетом

- Задача. Все операции эффективны

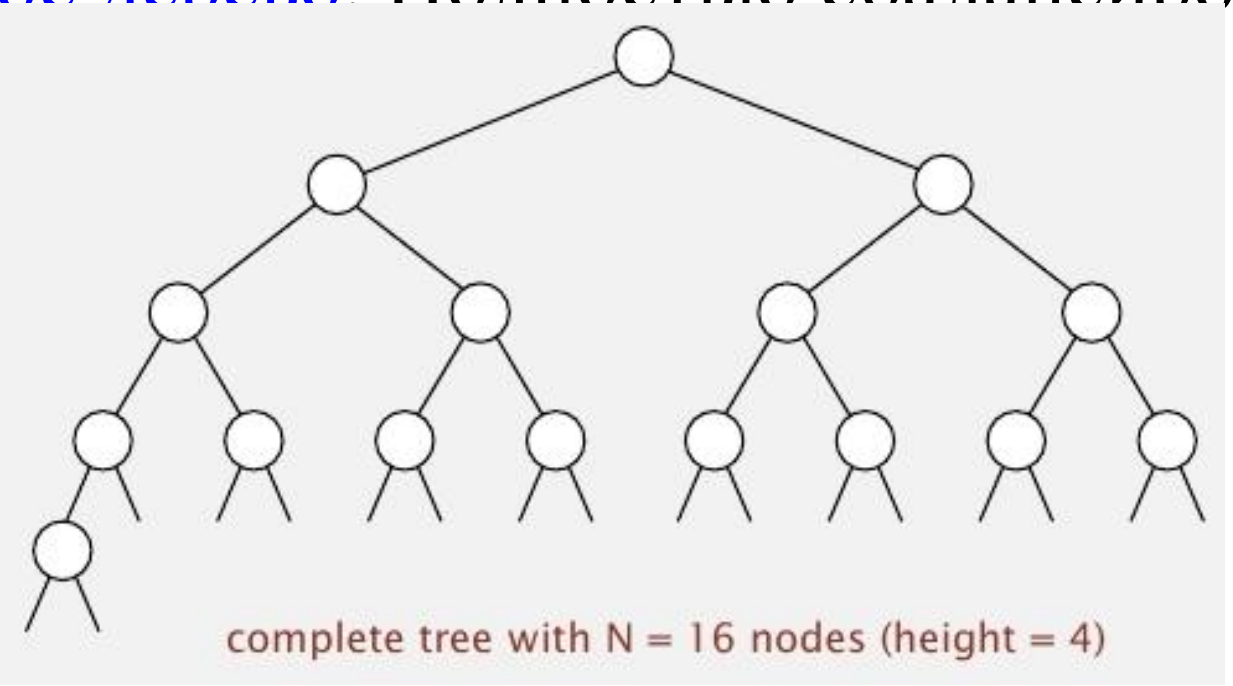
order of growth of running time for priority queue with N items

implementation	insert	del max	max
unordered array	1	N	N
ordered array	N	1	1
goal	log N	log N	log N

Бинарная пирамида
(Binary Heaps)

Полное бинарное дерево

- Бинарное дерево. Пустота или узел с левым и правым бинарным поддеревом
- Полное дерево. Полностью сбалансированное, за исклю

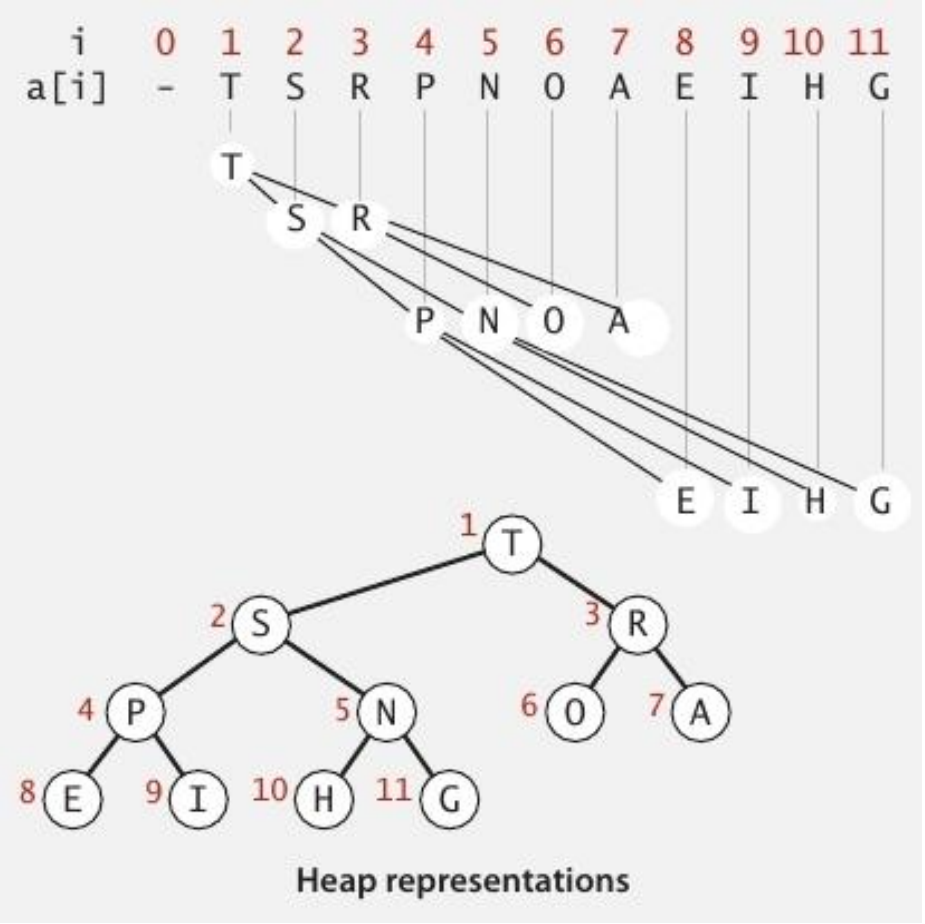


- Свойство. Высота полного дерева из N-1 узлов равна

Полное бинарное дерево

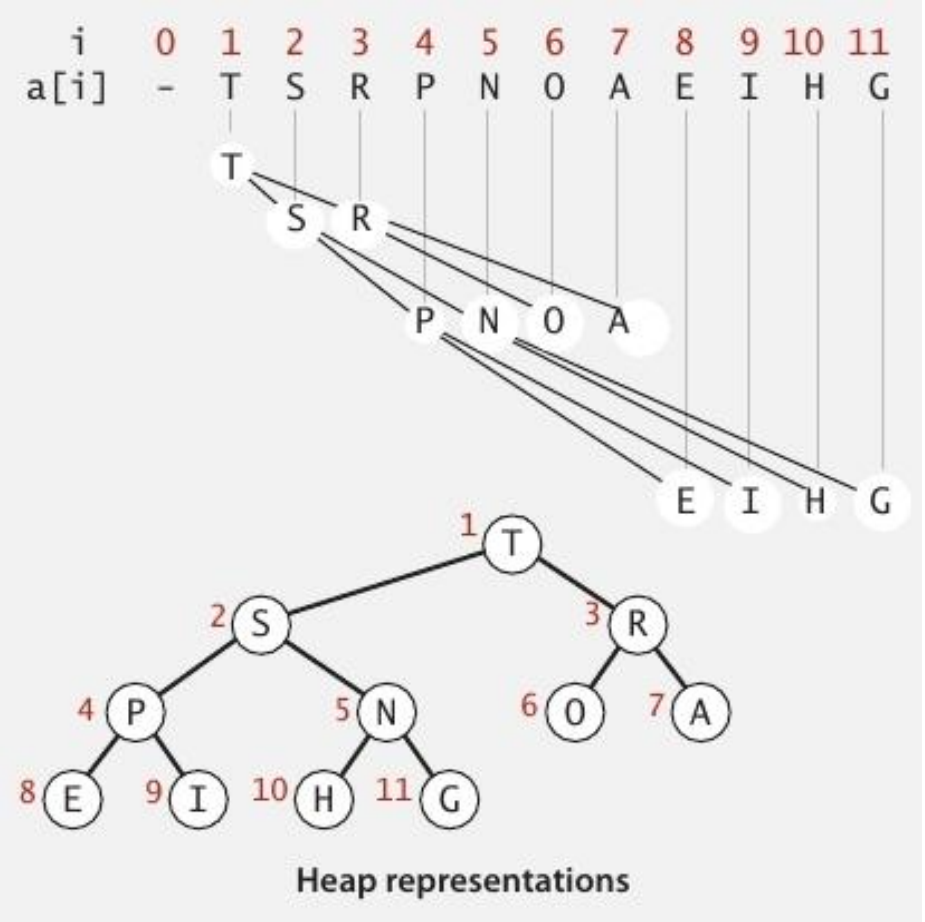
Бинарная пирамида

- Бинарная пирамида. Пирамидально упорядоченное полное бинарное дерево можно представить в виде массива
- Пирамидально упорядоченное бинарное дерево
  - Ключи в узлах
  - Ключ родительского узла не меньше чем дочернего
- Представление в массиве
  - Индексы начинаются с 1
  - Узлы упорядочены по уровням



Бинарная пирамида

- Самый большой ключ находится в корне по адресу a [1]
- Пользуйтесь индексами для перемещения по массиву
  - Родитель узла k находится в k/2
  - Потомки узла k находятся в 2k и 2k+1



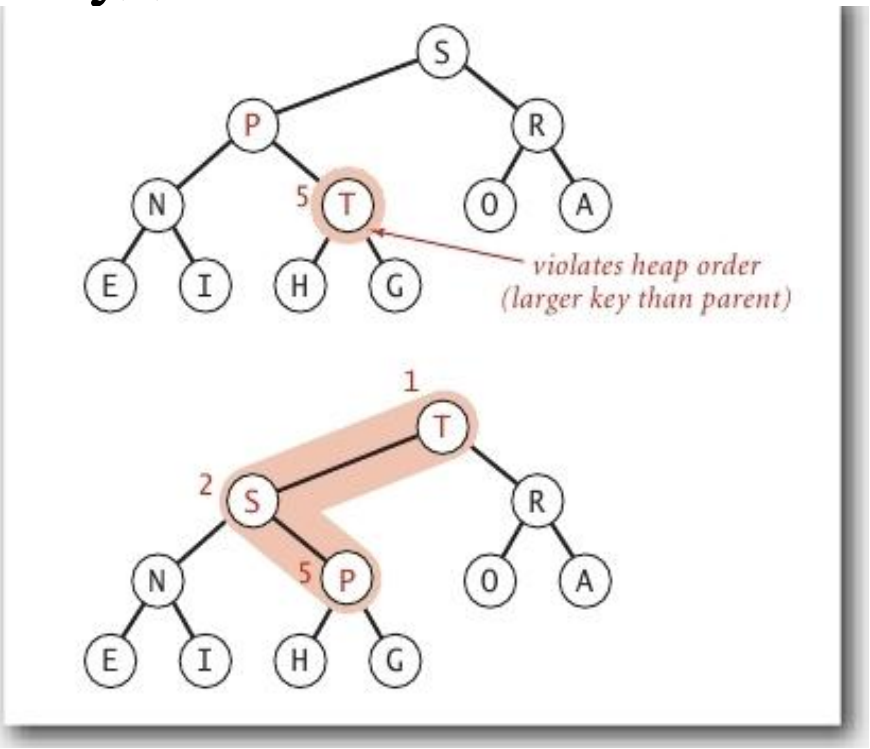
Heap representations

Продвижение в пирамиде

- Если дочерний узел больше родительского
  - Поменять местами дочерний и родительский узел
  - Повторять до тех пор пока не будет восстановлена

```
private void swim(int k)
{
   while (k > 1 && less(k/2, k))
   {
      exch(k, k/2);
      k = k/2;
   }
}
```

parent of node at k is at k/2

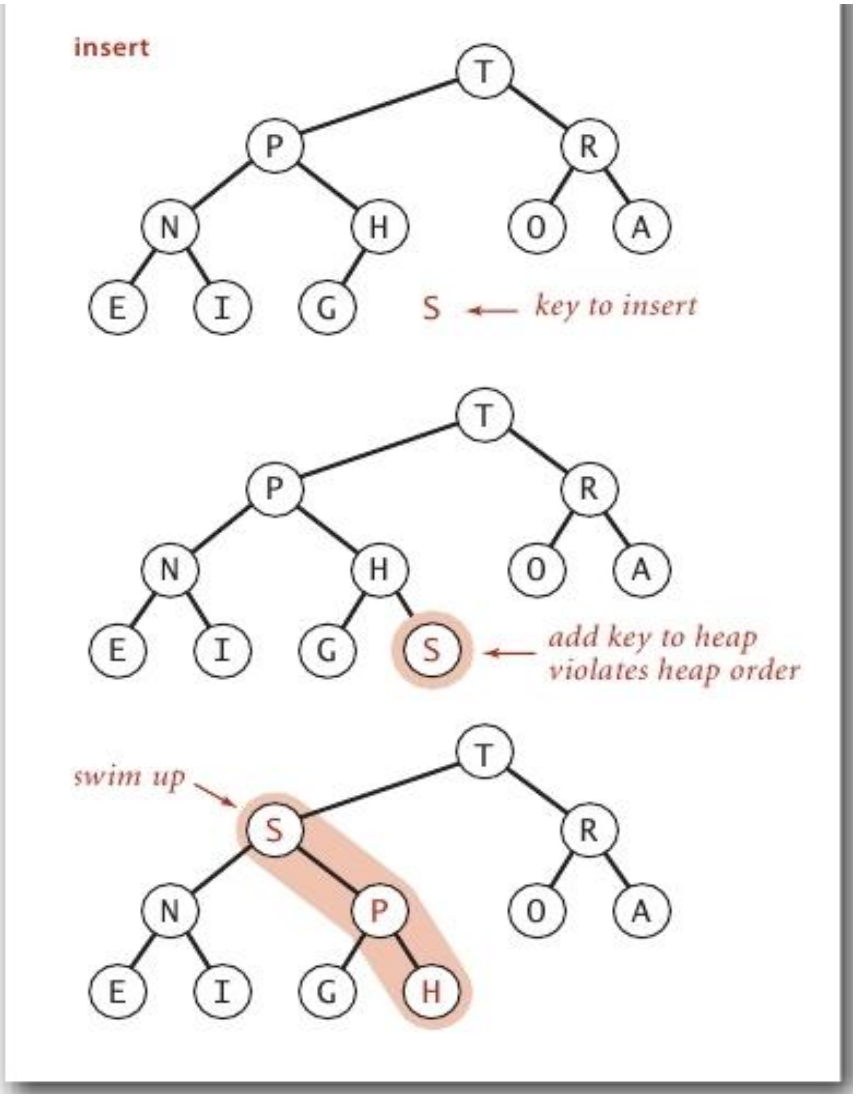


- Принцип Питера. Узел продвигается до уровня своей некомпетентности

Вставка в пирамиде

- Вставка. Добавить узел в конец и поднимать его выше
-

```
public void insert(Key x)
{
   pq[++N] = x;
   swim(N);
}
```

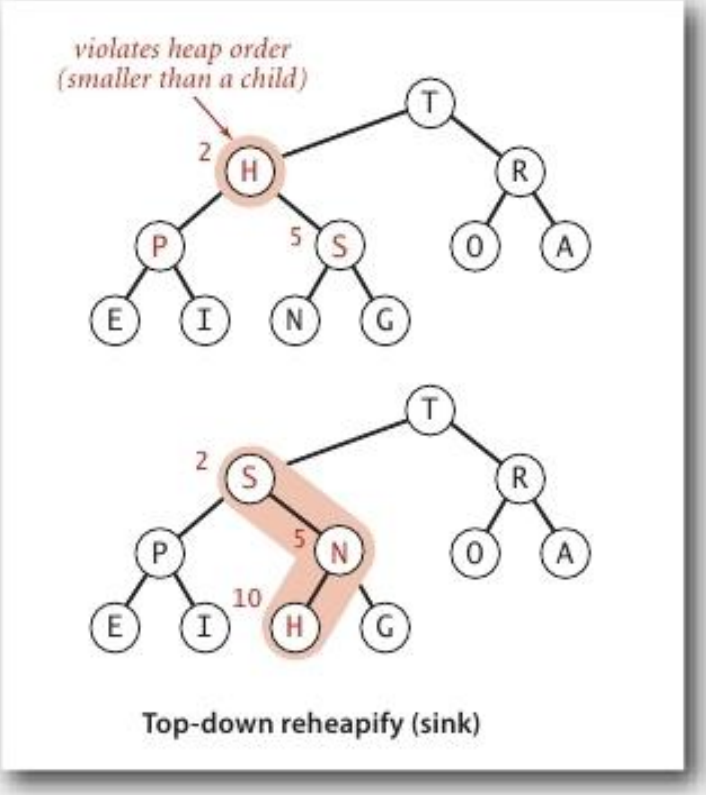


Спуск в пирамиде

- Если родительский узел меньше одного (или двух) дочерних
  - Поменять местами родительский и больший дочерний

```
private void sink(int k)
{
   while (2*k <= N)
   {
      int j = 2*k;
      if (j < N && less(j, j+1)) j++;
      if (!less(k, j)) break;
      exch(k, j);
      k = j;
   }
}
```

children of node at k are 2k and 2k+1



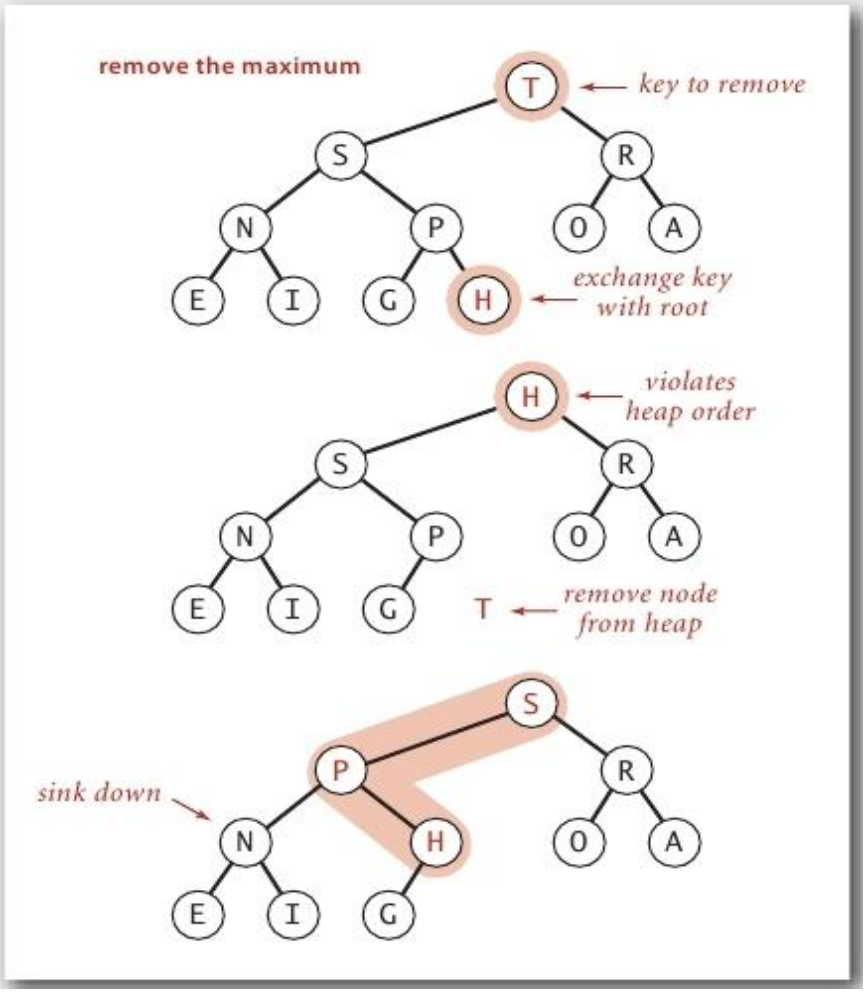
Top-down reheapify (sink)

Удалить максимальный узел в пирамиде

- Удаление максимального узла. Поменять корень с последним узлом и спустить его ниже

```
public Key delMax()
{
   Key max = pq[1];
   exch(1, N--);
   sink(1);
   pq[N+1] = null;
   return max;
}
```

prevent loitering



Бинарная пирамида

- Вставка. Добавить узел в конец и поднимать его выше
- Удаление максимального узла. Поменять корень с последним узлом и спустить его ниже
- Видео 1

Бинарная пирамида: реализация на Java

```
public class MaxPQ<Key extends Comparable<Key>>
{
   private Key[] pq;
   private int N;

   public MaxPQ(int capacity)
   {  pq = (Key[]) new Comparable[capacity+1];  }

   public boolean isEmpty()
   {   return N == 0;   }
   public void insert(Key key)
   public Key delMax()
   {   /* see previous code */  }

   private void swim(int k)
   private void sink(int k)
   {   /* see previous code */  }

   private boolean less(int i, int j)
   {   return pq[i].compareTo(pq[j]) < 0;  }
   private void exch(int i, int j)
   {   Key t = pq[i]; pq[i] = pq[j]; pq[j] = t;  }
}
```

fixed capacity (for simplicity)

PQ ops

heap helper functions

array helper functions

Реализация очереди с приоритетом

order-of-growth of running time for priority queue with N items

implementation	insert	del max	max
unordered array	1	N	N
ordered array	N	1	1
binary heap	log N	log N	1
d-ary heap	$\log_d N$	$d \log_d N$	1
Fibonacci	1	log N †	1
impossible	1	1	1

← why impossible?

† amortized

Binary heap considerations

Immutability of keys.

- Assumption: client does not change keys while they're on the PQ.
- Best practice: use immutable keys.

Underflow and overflow.

- Underflow: throw exception if deleting from empty PQ.
- Overflow: add no-arg constructor and use resizing array.

leads to log N amortized time per op (how to make worst case?)

Minimum-oriented priority queue.

- Replace `less()` with `greater()`.
- Implement `greater()`.

Other operations.

- Remove an arbitrary item.
- Change the priority of an item.

can implement with `sink()` and `swim()` [stay tuned]

Immutability: properties

Data type. Set of values and operations on those values.

Immutable data type. Can't change the data type value once created.

Advantages.

- Simplifies debugging.
- Safer in presence of hostile code.
- Simplifies concurrent programming.
- Safe to use as key in priority queue or symbol table.

Disadvantage. Must create new object for each data type value.

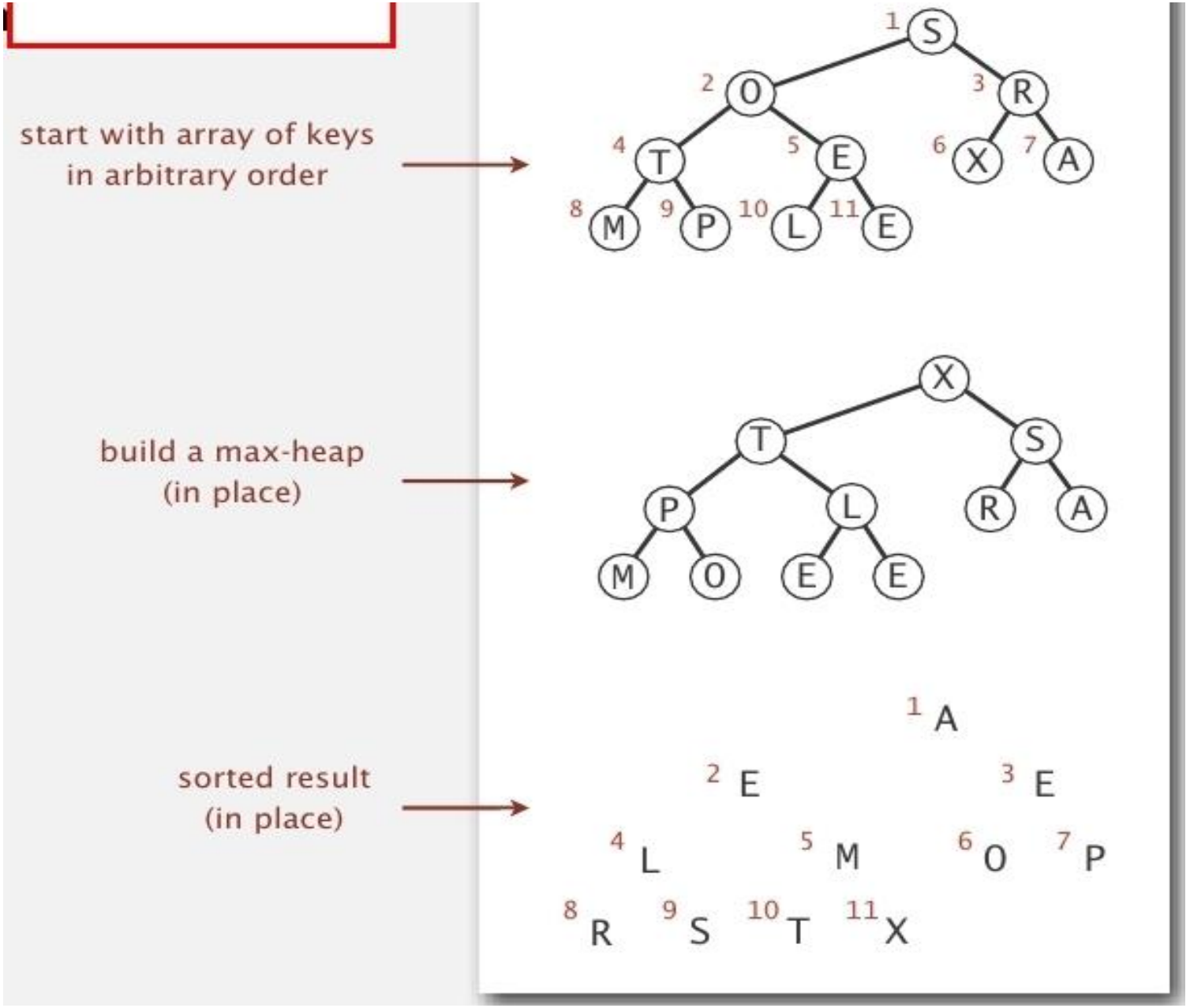
> *" Classes should be immutable unless there's a very good reason to make them mutable.... If a class cannot be made immutable, you should still limit its mutability as much as possible. "*
>
> *— Joshua Bloch (Java architect)*

Пирамидальная сортировка (Heapsort)

Пирамидальная сортировка

- Создать пирамиду из всех N ключей
- Повторять удаление максимального ключа

Пирамидальная сортировка



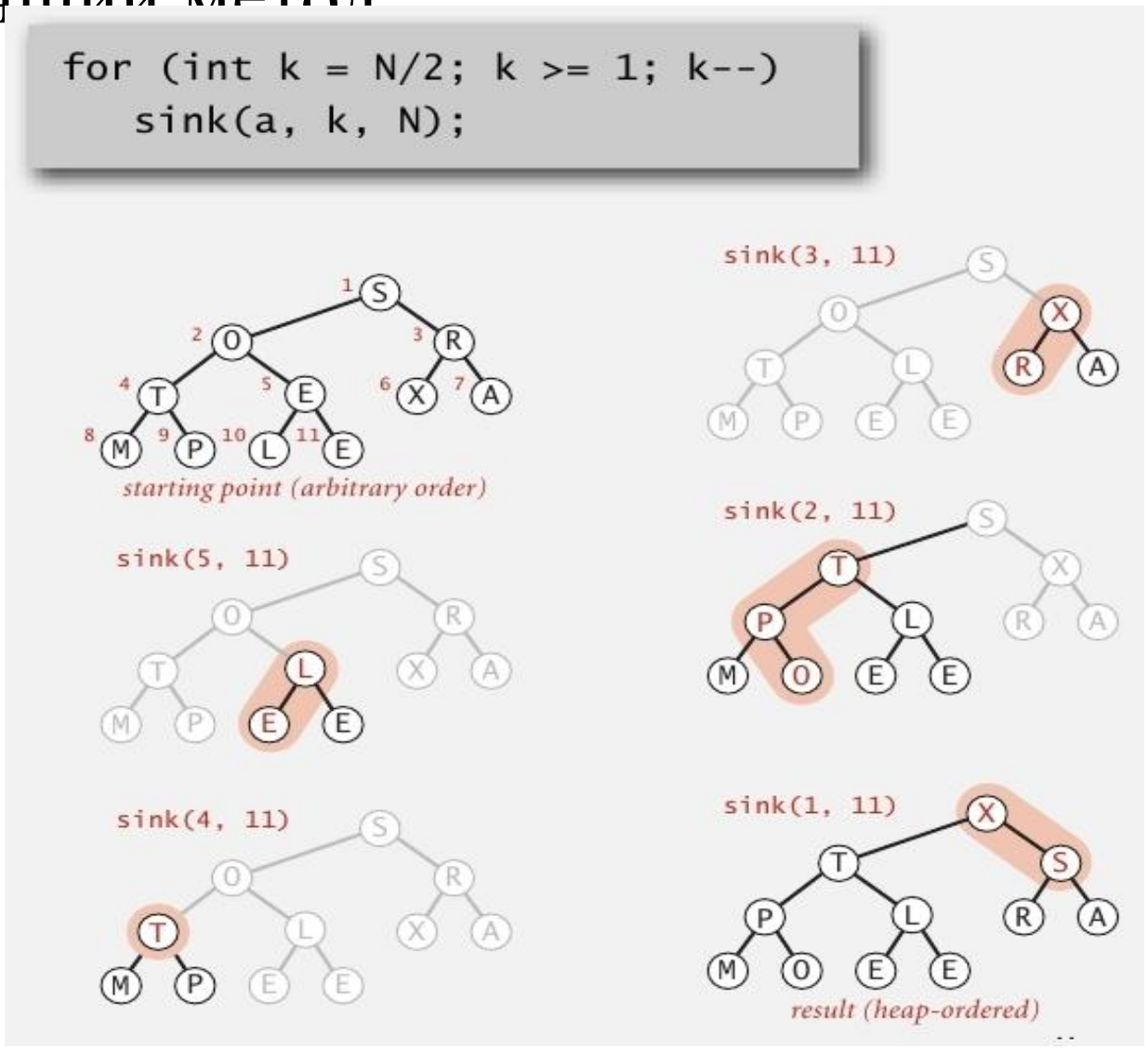
Пирамидальная сортировка

- Конструктор пирамиды. Создать max пирамиду восходящим методом
- Видео 2
- Видео 3

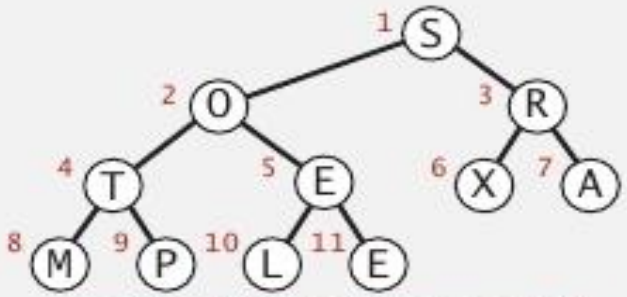
Пирамидальная сортировка: конструктор

- Первый проход. Создать пирамиду используя восходящий метод

```
for (int k = N/2; k >= 1; k--)
   sink(a, k, N);
```

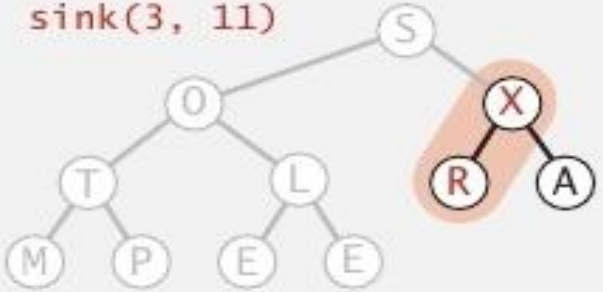


```
for (int k = N/2; k >= 1; k--)
   sink(a, k, N);
```

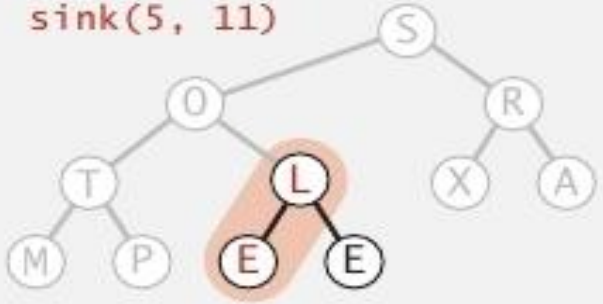


starting point (arbitrary order)

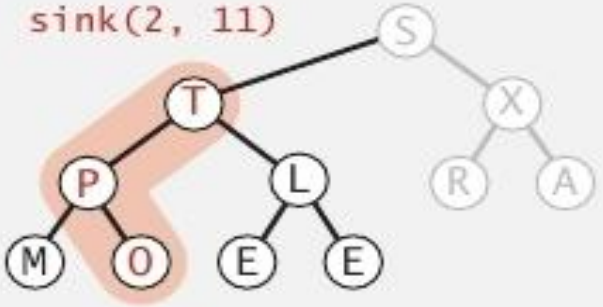
sink(3, 11)



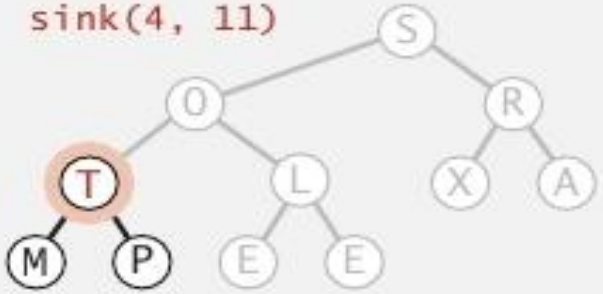
sink(5, 11)



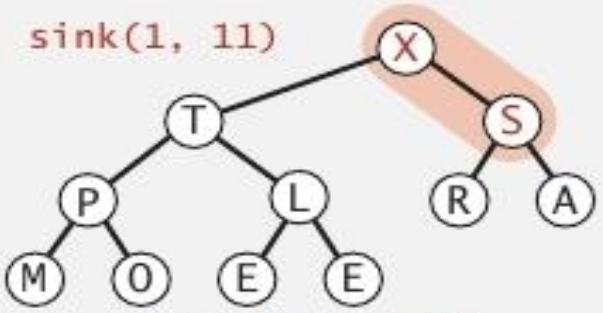
sink(2, 11)



sink(4, 11)



sink(1, 11)

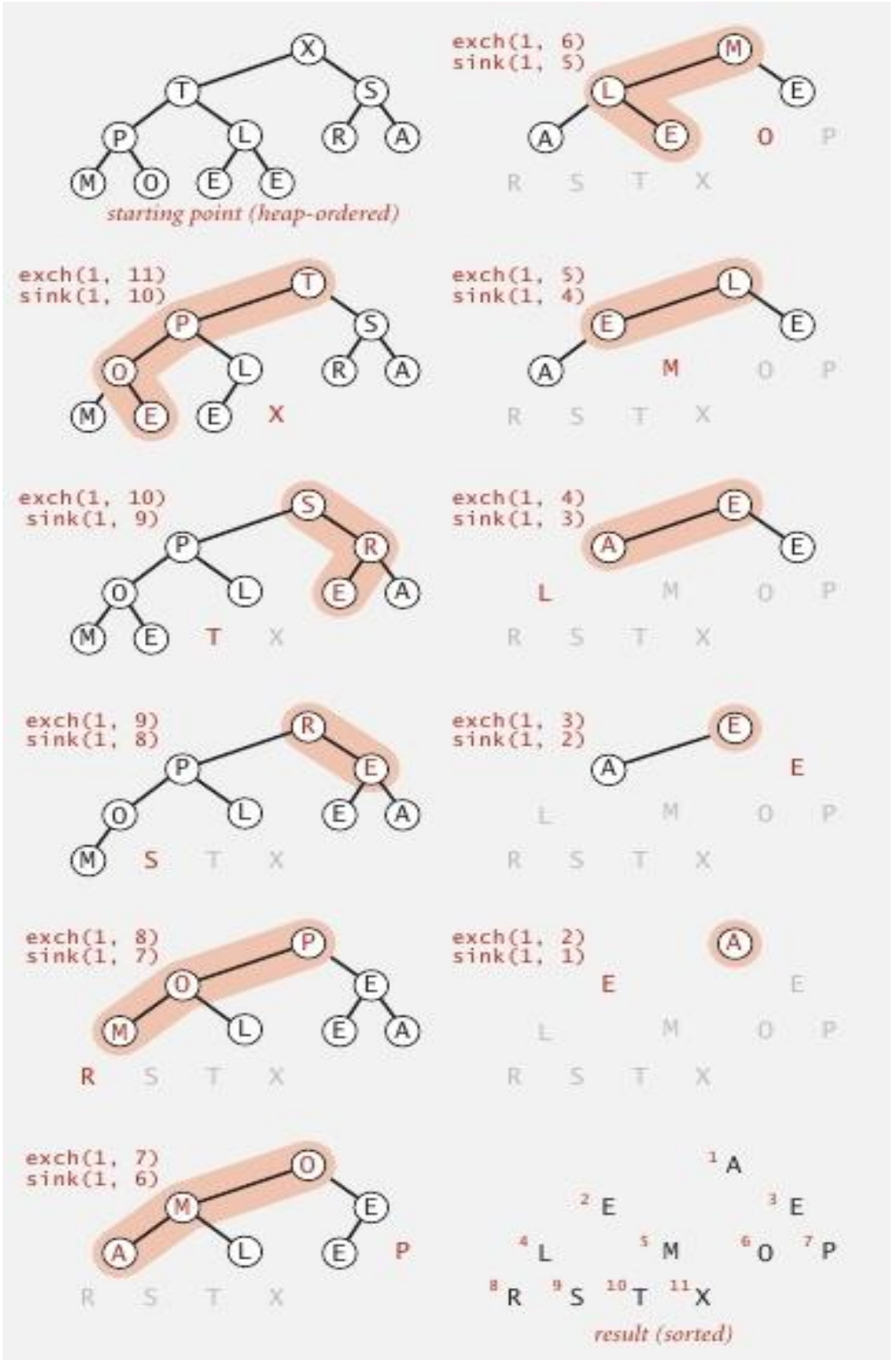


result (heap-ordered)

Пирамидальная сортировка

- Второй проход
  - Удалять максимум поочередно
  - Восстановить порядок в пирамиде

```
while (N > 1)
{
   exch(a, 1, N--);
   sink(a, 1, N);
}
```


starting point (heap-ordered)
exch(1, 6)
sink(1, 5)
exch(1, 11)
sink(1, 10)
exch(1, 5)
sink(1, 4)
exch(1, 10)
sink(1, 9)
exch(1, 4)
sink(1, 3)
exch(1, 9)
sink(1, 8)
exch(1, 3)
sink(1, 2)
exch(1, 8)
sink(1, 7)
exch(1, 2)
sink(1, 1)
exch(1, 7)
sink(1, 6)
result (sorted)

Пирамидальная сортировка: реализация на Java

```
public class Heap
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int k = N/2; k >= 1; k--)
         sink(a, k, N);
      while (N > 1)
      {
         exch(a, 1, N);
         sink(a, 1, --N);
      }
   }

   private static void sink(Comparable[] a, int k, int N)
   {  /* as before */  }

   private static boolean less(Comparable[] a, int i, int j)
   {  /* as before */  }

   private static void exch(Comparable[] a, int i, int j)
   {  /* as before */  }

}
```

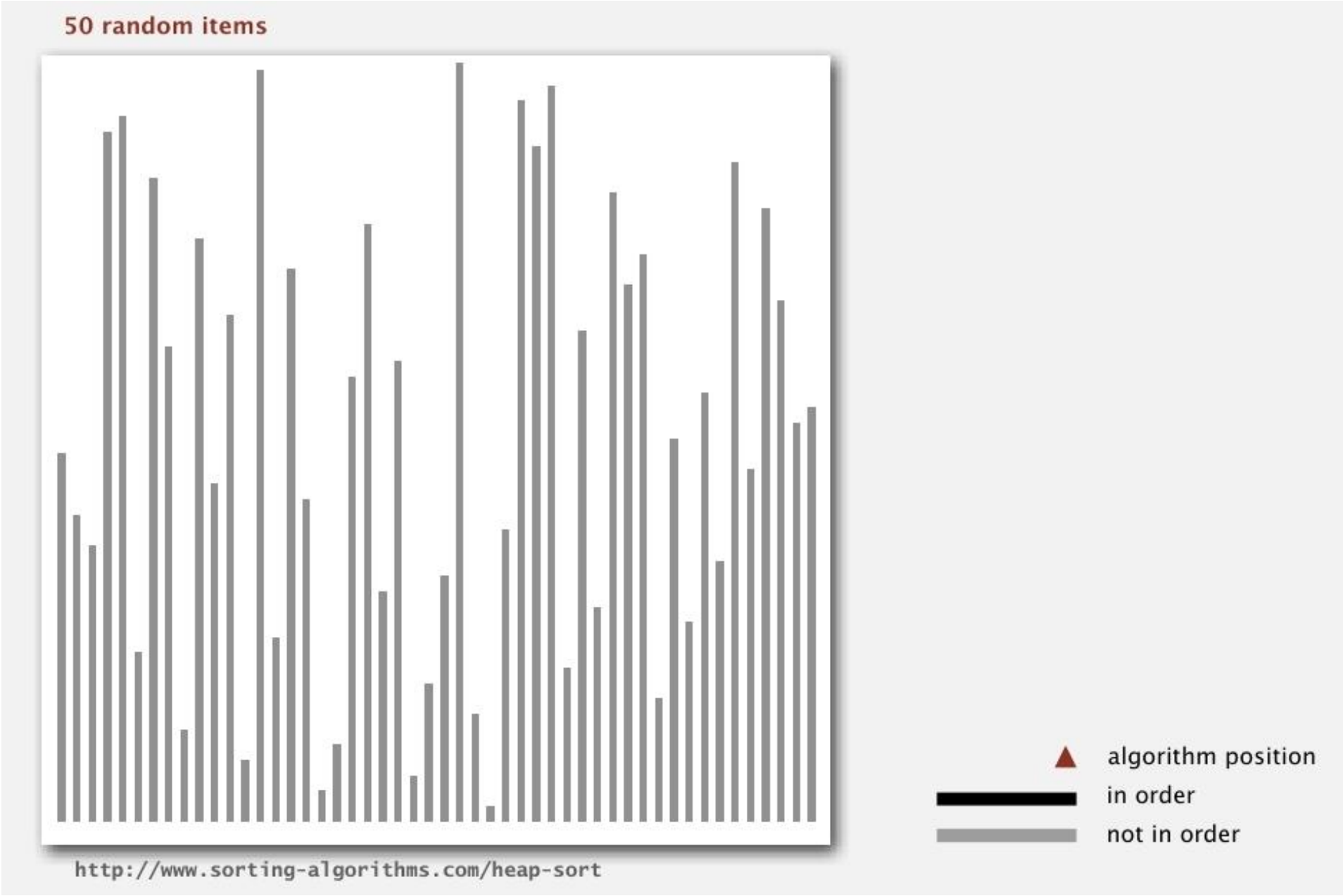
but convert from 1-based indexing to 0-base indexing

Пирамидальная сортировка

		a[i]											
N	k	0	1	2	3	4	5	6	7	8	9	10	11
initial values			S	O	R	T	E	X	A	M	P	L	E
11	5		S	O	R	T	L	X	A	M	P	E	E
11	4		S	O	R	T	L	X	A	M	P	E	E
11	3		S	O	X	T	L	R	A	M	P	E	E
11	2		S	T	X	P	L	R	A	M	O	E	E
11	1		X	T	S	P	L	R	A	M	O	E	E
heap-ordered			X	T	S	P	L	R	A	M	O	E	E
10	1		T	P	S	O	L	R	A	M	E	E	X
9	1		S	P	R	O	L	E	A	M	E	T	X
8	1		R	P	E	O	L	E	A	M	S	T	X
7	1		P	O	E	M	L	E	A	R	S	T	X
6	1		O	M	E	A	L	E	P	R	S	T	X
5	1		M	L	E	A	E	O	P	R	S	T	X
4	1		L	E	E	A	M	O	P	R	S	T	X
3	1		E	A	E	L	M	O	P	R	S	T	X
2	1		E	A	E	L	M	O	P	R	S	T	X
1	1		A	E	E	L	M	O	P	R	S	T	X
sorted result			A	E	E	L	M	O	P	R	S	T	X

Heapsort trace (array contents just after each sink)

Пирамидальная сортировка



Пирамидальная сортировка: математический анализ

- Первый проход <= 2N сравнений и перестановок
- Второй проход <= 2N $\log_2 N$ сравнений и перестановок
- Значение. Алгоритм, не требующий дополнительной памяти и работающий за NlogN в худшем случае
  - Быстрая сортировка
  - Сортировка слиянием
- Нижняя граница. Пирамидальная сортировка оптимальна по памяти и по времени
  - Внутренний цикл длиннее, чем у Q-sort

Алгоритмы сортировки

	inplace?	stable?	worst	average	best	remarks
selection	x		$N^2 / 2$	$N^2 / 2$	$N^2 / 2$	N exchanges
insertion	x	x	$N^2 / 2$	$N^2 / 4$	N	use for small N or partially ordered
shell	x		?	?	N	tight code, subquadratic
quick	x		$N^2 / 2$	2 N ln N	N lg N	N log N probabilistic guarantee fastest in practice
3-way quick	x		$N^2 / 2$	2 N ln N	N	improves quicksort in presence of duplicate keys
merge		x	N lg N	N lg N	N lg N	N log N guarantee, stable
heap	x		2 N lg N	2 N lg N	N lg N	N log N guarantee, in-place
???	x	x	N lg N	N lg N	N lg N	holy sorting grail