

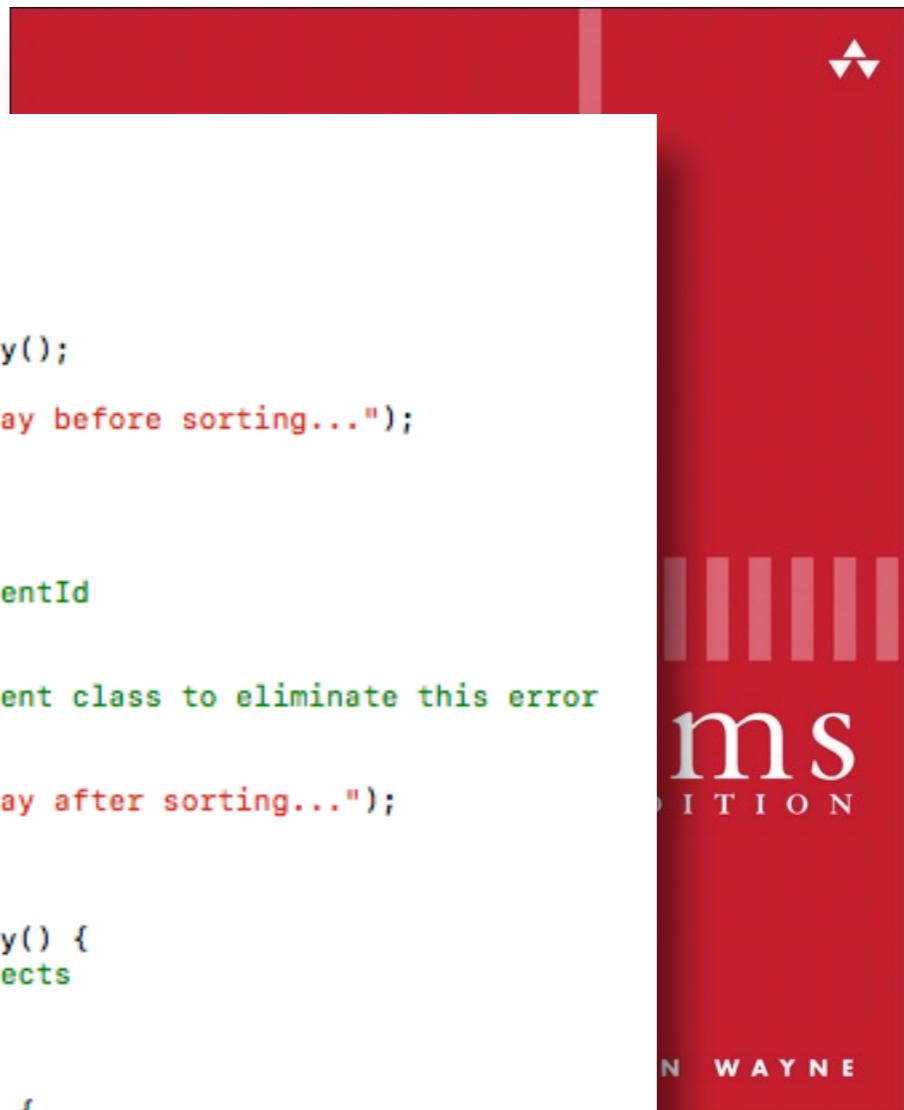
CSC 402

Data Structures I
Lecture 6

Homework 4

Mergesort

```
public class TestMergeSort {  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        Student[] students = initializeStudentsArray();  
  
        System.out.println("Displaying students array before sorting...");  
        display(students);  
  
        System.out.println("Being sorting...");  
        /*  
         * The Student array will be sorted by studentId  
         * which is declared as a String  
         */  
        Merge.sort(students); // TODO: Fix the Student class to eliminate this error  
        System.out.println("End sorting...");  
  
        System.out.println("Displaying students array after sorting...");  
        display(students);  
    }  
  
    private static Student[] initializeStudentsArray() {  
        // TODO: Initialize an array of Student objects  
        return null;  
    }  
  
    private static void display(Student[] students) {  
        // TODO: Display the contents of the Students Array  
    }  
}
```



Assignment due next Monday at 11:59 PM

Homework 4

Mergesort

```
/*
 * This is an example of a successful run:
 *
 *   Displaying students array before sorting...
 *   Students [firstName=Joe, lastName=Jones, studentId=1001]
 *   Students [firstName=Adam, lastName=Ant, studentId=950]
 *   Students [firstName=Bill, lastName=Barnes, studentId=735]
 *   Students [firstName=Mark, lastName=Roth, studentId=1102]
 *   Students [firstName=Jerome, lastName=Howard, studentId=1050]
 *   Being sorting...
 *   End sorting...
 *   Displaying students array after sorting...
 *   Students [firstName=Adam, lastName=Ant, studentId=950]
 *   Students [firstName=Bill, lastName=Barnes, studentId=735]
 *   Students [firstName=Mark, lastName=Roth, studentId=1102]
 *   Students [firstName=Jerome, lastName=Howard, studentId=1050]
 *   Students [firstName=Joe, lastName=Jones, studentId=1001]
 *
 */
```



Assignment due next Monday at 11:59 PM

Homework 4

Mergesort

```
/*
 * This is an example of a successful run:
 *
 * Displaying students array before sorting...
 Students [firstName=Joe, lastName=Jones, studentId=1001]
 Students [firstName=Adam, lastName=Ant, studentId=950]
 Students [firstName=Bill, lastName=Barnes, studentId=735]
 @Override
 public int compareTo(Object object) {
     Student student = (Student)object;
     return student.getStudentId().compareTo(this.getStudentId());
 }
 *
 *
 Students [firstName=Bill, lastName=Barnes, studentId=735]
 Students [firstName=Mark, lastName=Roth, studentId=1102]
 Students [firstName=Jerome, lastName=Howard, studentId=1050]
 Students [firstName=Joe, lastName=Jones, studentId=1001]
```



Assignment due next Monday at 11:59 PM

Homework 4

Mergesort

```
private static Student[] initializeStudentsArray() {
    // TODO: Initialize an array of Student objects
    Student[] students = {
        new Student("Joe", "Jones", "1001"),
        new Student("Adam", "Ant", "950"),
        new Student("Bill", "Barnes", "735"),
        new Student("Mark", "Roth", "1102"),
        new Student("Jerome", "Howard", "1050")
    };
    return students;
}

private static void display(Student[] students) {
    // TODO: Display the contents of the Students Array
    for(Student student : students) {
        System.out.println(student);
    }
}
```



Assignment due next Monday at 11:59 PM

Homework 4

Selectionsort

```
public class TestSelectionSort {  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        Check[] checks = initializeChecksArray();  
  
        System.out.println("Displaying checks array before sorting...");  
        display(checks);  
  
        System.out.println("Being sorting...");  
        /*  
         * The Checks array will be sorted by checkNumber  
         * which is declared as a Integer  
         */  
        Selection.sort(checks); // TODO: Fix the Check class to eliminate this error  
        System.out.println("End sorting...");  
  
        System.out.println("Displaying checks array after sorting...");  
        display(checks);  
    }  
  
    private static Check[] initializeChecksArray() {  
        // TODO: Initialize an array of Check objects  
        return null;  
    }  
  
    private static void display(Check[] checks) {  
        // TODO: Display the contents of the Check Array  
    }  
}
```



Assignment due next Monday at 11:59 PM

Homework 4

Selectionsort

```
/*
 * This is an example of a successful run:
 *
 *   Displaying checks array before sorting...
 *   Check [checkNumber=1001, payTo=Joe Jones, date=Mon Feb 06 15:36:40 CST 2017]
 *   Check [checkNumber=950, payTo=Adam Ant, date=Mon Feb 06 15:36:40 CST 2017]
 *   Check [checkNumber=735, payTo=Bill Barnes, date=Mon Feb 06 15:36:40 CST 2017]
 *   Check [checkNumber=1102, payTo=Mark Roth, date=Mon Feb 06 15:36:40 CST 2017]
 *   Check [checkNumber=1050, payTo=Jerome Howard, date=Mon Feb 06 15:36:40 CST 2017]
 *   Being sorting...
 *   End sorting...
 *   Displaying checks array after sorting...
 *   Check [checkNumber=735, payTo=Bill Barnes, date=Mon Feb 06 15:36:40 CST 2017]
 *   Check [checkNumber=950, payTo=Adam Ant, date=Mon Feb 06 15:36:40 CST 2017]
 *   Check [checkNumber=1001, payTo=Joe Jones, date=Mon Feb 06 15:36:40 CST 2017]
 *   Check [checkNumber=1050, payTo=Jerome Howard, date=Mon Feb 06 15:36:40 CST 2017]
 *   Check [checkNumber=1102, payTo=Mark Roth, date=Mon Feb 06 15:36:40 CST 2017]
 *
 */
```

ROBERT SEDGEWICK | KEVIN WAYNE

Assignment due next Monday at 11:59 PM

Homework 4

Selectionsort

```
/*
 * This is an example of a successful run:
 *
 * Displaying checks array before sorting...
Check [checkNumber=1001, payTo=Joe Jones, date=Mon Feb 06 15:36:40 CST 2017]
Check [checkNumber=950, payTo=Adam Ant, date=Mon Feb 06 15:36:40 CST 2017]
Check [checkNumber=725, payTo=Rill Barnes, date=Mon Feb 06 15:36:40 CST 2017]
C:
@Override
public int compareTo(Object object) {
B:    Check check = (Check)object;
E:    return this.getCheckNumber().compareTo(check.getCheckNumber());
D: }
C:
Check [checkNumber=950, payTo=Adam Ant, date=Mon Feb 06 15:36:40 CST 2017]
Check [checkNumber=1001, payTo=Joe Jones, date=Mon Feb 06 15:36:40 CST 2017]
Check [checkNumber=1050, payTo=Jerome Howard, date=Mon Feb 06 15:36:40 CST 2017]
Check [checkNumber=1102, payTo=Mark Roth, date=Mon Feb 06 15:36:40 CST 2017]
*
*/
```



ROBERT SEDGEWICK | KEVIN WAYNE

Assignment due next Monday at 11:59 PM

Homework 4

Selectionsort

```
/*
 * This is
 *
 * Display
 * Check [
 * Being s
 * End so
 * Display
 * Check [
 * *
 */
private static Check[] initializeChecksArray() {
    // TODO: Initialize an array of Check objects
    Check[] checks = {
        new Check(1001, "Joe Jones", new Date()),
        new Check(950, "Adam Ant", new Date()),
        new Check(735, "Bill Barnes", new Date()),
        new Check(1102, "Mark Roth", new Date()),
        new Check(1050, "Jerome Howard", new Date())
    };
    return checks;
}

private static void display(Check[] checks) {
    // TODO: Display the contents of the Check Array
    for (Check check : checks) {
        System.out.println(check);
    }
}
```



Assignment due next Monday at 11:59 PM

Lecture Overview



2.4 Priority Queues

- API and elementary implementations
- Binary heaps
- Heapsort
- Event-driven simulation

Lecture Overview



2.4 Priority Queues

- API and elementary implementations
- Binary heaps
- Heapsort
- Event-driven simulation

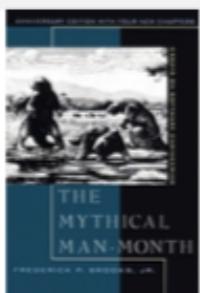
API & Elementary Implementations

Collections

A **collection** is a data types that store groups of items.

data type	key operations	data structure
stack	PUSH, POP	<i>linked list, resizing array</i>
queue	ENQUEUE, DEQUEUE	<i>linked list, resizing array</i>
priority queue	INSERT, DELETE-MAX	<i>binary heap</i>
symbol table	PUT, GET, DELETE	<i>BST, hash table</i>
set	ADD, CONTAINS, DELETE	<i>BST, hash table</i>

“Show me your code and conceal your data structures, and I shall continue to be mystified. Show me your data structures, and I won’t usually need your code; it’ll be obvious.” — Fred Brooks



API & Elementary Implementations

Priority queue

Collections. Insert and delete items. Which item to delete?

Stack. Remove the item most recently added.

Queue. Remove the item least recently added.

Randomized queue. Remove a random item.

Priority queue. Remove the **largest** (or **smallest**) item.

<i>operation</i>	<i>argument</i>	<i>return value</i>
<i>insert</i>	P	
<i>insert</i>	Q	
<i>insert</i>	E	
<i>remove max</i>		Q
<i>insert</i>	X	
<i>insert</i>	A	
<i>insert</i>	M	
<i>remove max</i>		X
<i>insert</i>	P	
<i>insert</i>	L	
<i>insert</i>	E	
<i>remove max</i>		P

API & Elementary Implementations

Priority queue API

Requirement. Generic items are Comparable.

public class MaxPQ<Key extends Comparable<Key>>	
MaxPQ()	<i>create an empty priority queue</i>
MaxPQ(Key[] a)	<i>create a priority queue with given keys</i>
void insert(Key v)	<i>insert a key into the priority queue</i>
Key delMax()	<i>return and remove the largest key</i>
boolean isEmpty()	<i>is the priority queue empty?</i>
Key max()	<i>return the largest key</i>
int size()	<i>number of entries in the priority queue</i>

Key must be Comparable
(bounded type parameter)

API & Elementary Implementations

Priority queue applications

- 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. [online median in data stream]
- Operating systems. [load balancing, interrupt handling]
- Computer networks. [web cache]
- Discrete optimization. [bin packing, scheduling]
- Spam filtering. [Bayesian spam filter]

Generalizes: stack, queue, randomized queue.

API & Elementary Implementations

Priority queue client example

Challenge. Find the largest M items in a stream of N items.

- Fraud detection: isolate \$\$ transactions.
- NSA monitoring: flag most suspicious documents.



N huge, M large

Constraint. Not enough memory to store N items.

```
% 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
↑

API & Elementary Implementations

Priority queue client example

Challenge. Find the largest M items in a stream of N items.

- Fraud detection: isolate \$\$ transactions.
- NSA monitoring: flag most suspicious documents.

N huge, M large

Constraint. Not enough memory to store N items.

```
use a min-oriented pq      MinPQ<Transaction> pq = new MinPQ<Transaction>();  
                          ↑  
                          while (StdIn.hasNextLine())  
                          {  
                            String line = StdIn.readLine();  
                            Transaction item = new Transaction(line);  
                            pq.insert(item);  
                            if (pq.size() > M) ← pq contains  
                                pq.delMin();  
                          }  
                          ↑  
                          Transaction data  
                          type is Comparable  
                          (ordered by $$)
```

API & Elementary Implementations

Priority queue client example

Challenge. Find the largest M items in a stream of N items.

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

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

API & Elementary Implementations

Priority queue: unordered and ordered array implementation

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 P

A sequence of operations on a priority queue

API & Elementary Implementations

Priority queue: unordered array implementation

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

    public UnorderedArrayMaxPQ(int capacity)
    {   pq = (Key[]) new Comparable[capacity]; } ← no generic array creation

    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]; ← should null out entry to prevent loitering
    }
}
```

no generic array creation

less() and exch()
similar to sorting methods
(but don't pass pq[])

API & Elementary Implementations

Priority queue elementary implementations

Challenge. Implement **all** operations efficiently.

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

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



2.4 Priority Queues

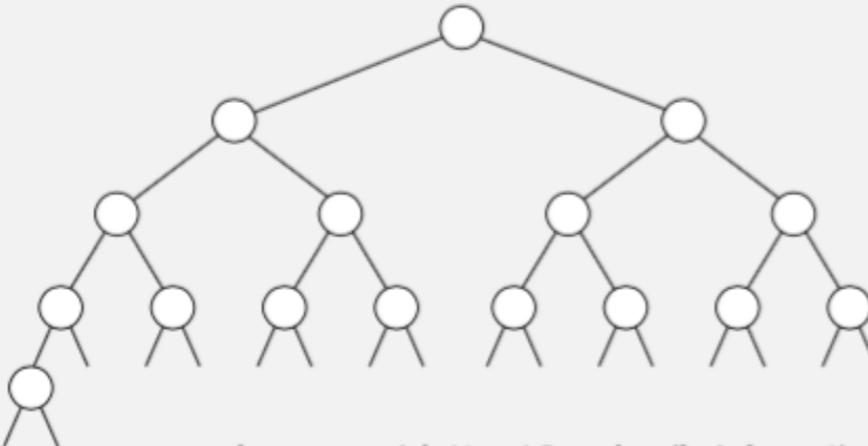
- API and elementary implementations
- Binary heaps
- Heapsort
- Event-driven simulation

Binary Heaps

Complete binary tree

Binary tree. Empty or node with links to left and right binary trees.

Complete tree. Perfectly balanced, except for bottom level.



Property. Height of complete tree with N nodes is $\lfloor \lg N \rfloor$.

Pf. Height increases only when N is a power of 2.

Binary Heaps

Binary heap representations

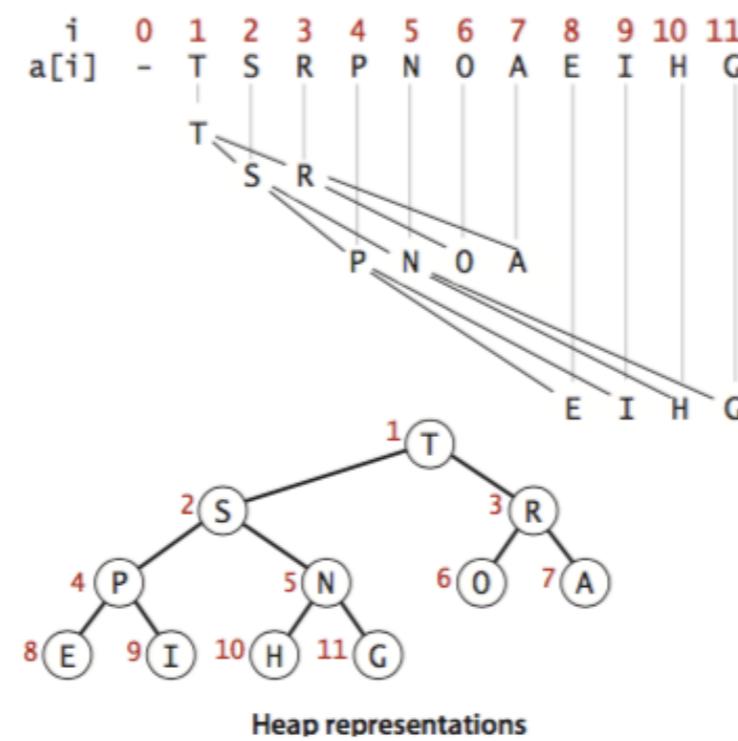
Binary heap. Array representation of a heap-ordered complete binary tree.

Heap-ordered binary tree.

- Keys in nodes.
- Parent's key no smaller than children's keys.

Array representation.

- Indices start at 1.
- Take nodes in **level** order.
- No explicit links needed!



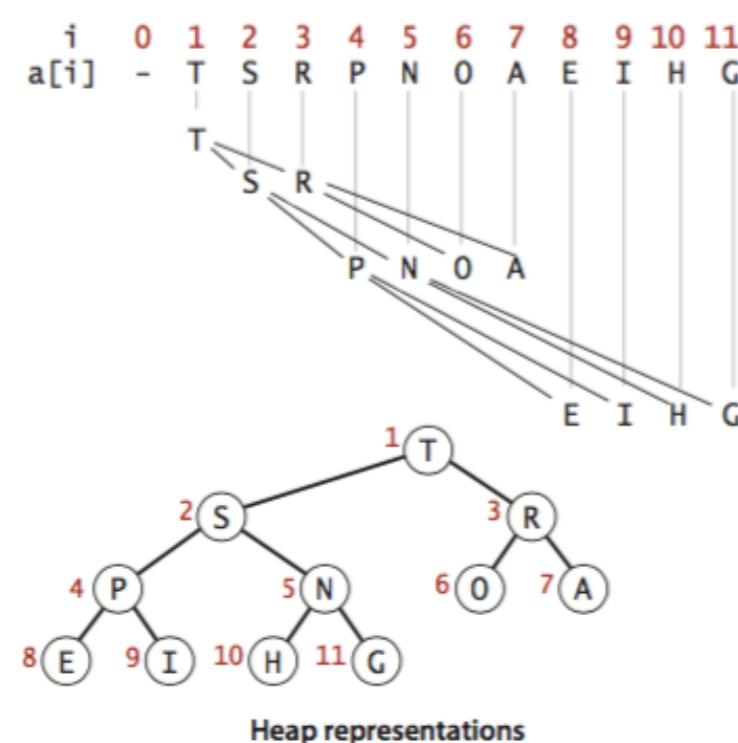
Binary Heaps

Binary heap properties

Proposition. Largest key is $a[1]$, which is root of binary tree.

Proposition. Can use array indices to move through tree.

- Parent of node at k is at $k/2$.
- Children of node at k are at $2k$ and $2k+1$.



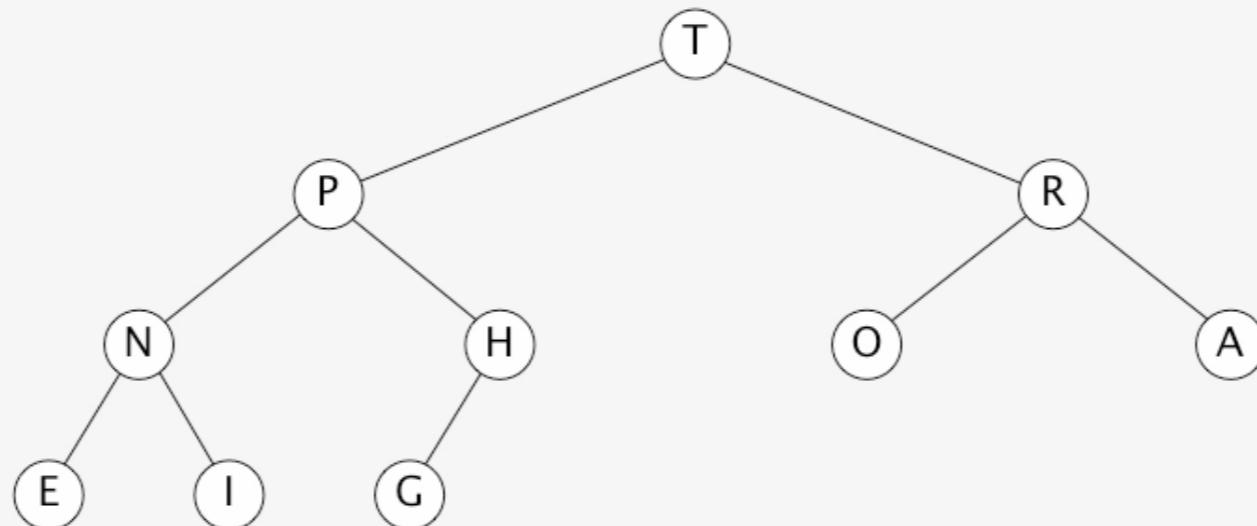
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

heap ordered



T	P	R	N	H	O	A	E	I	G
---	---	---	---	---	---	---	---	---	---

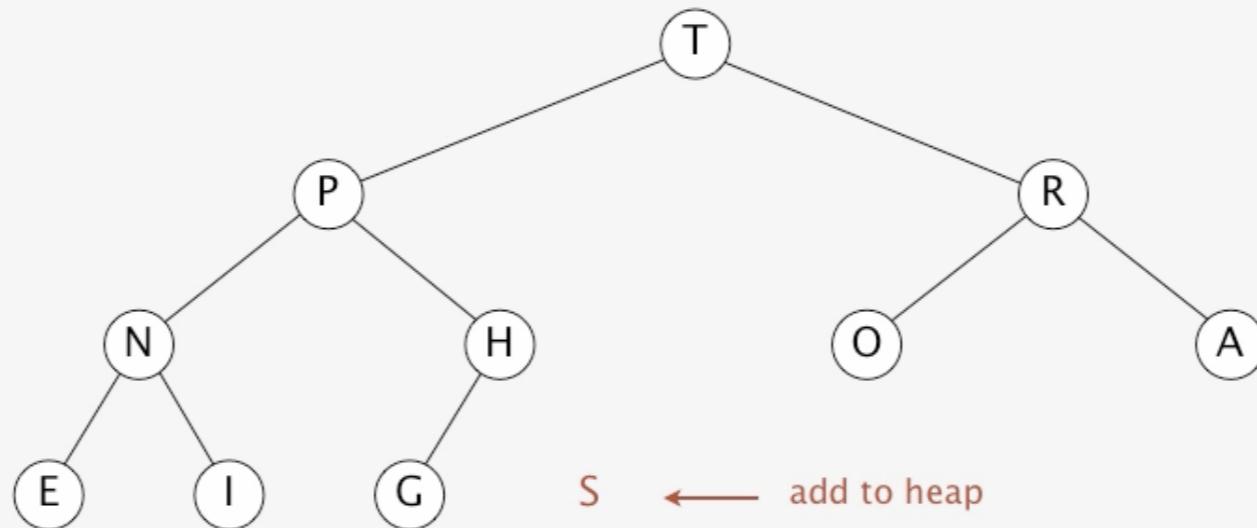
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



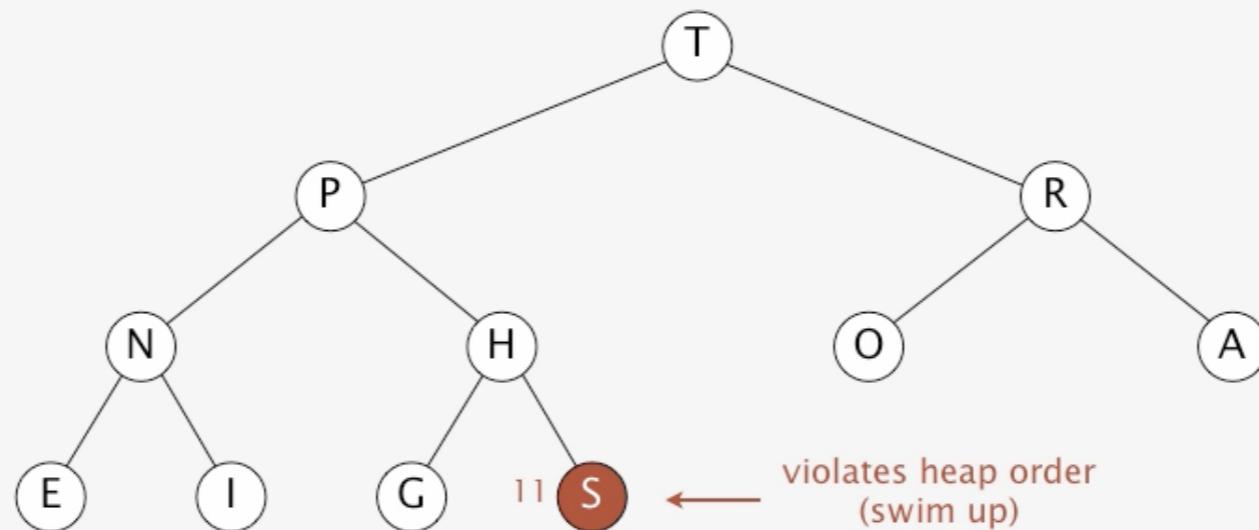
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



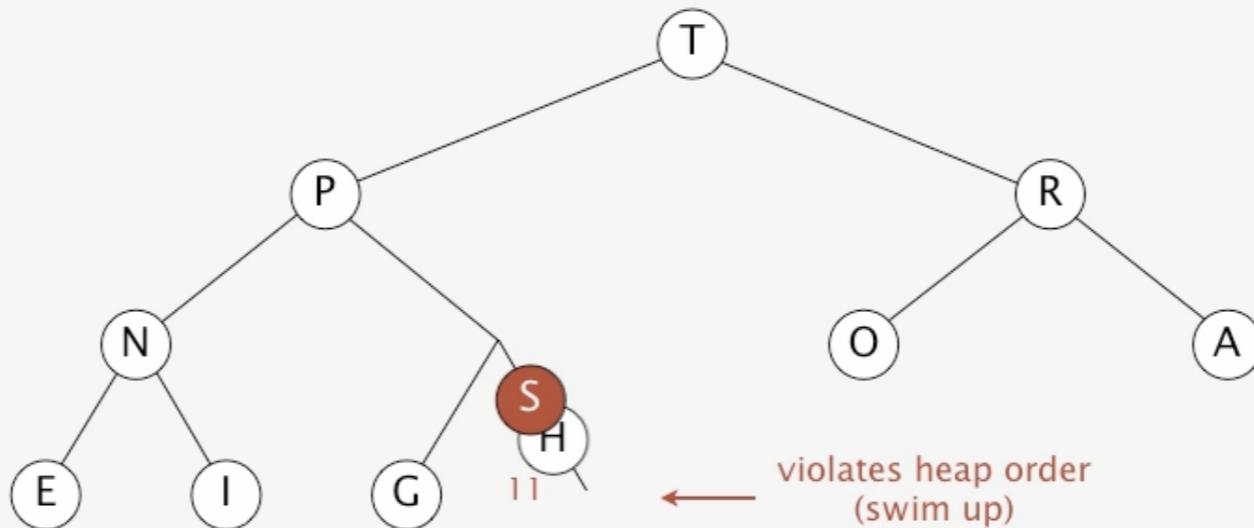
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



T	P	R	N	S	O	A	E	I	G	H
				11						

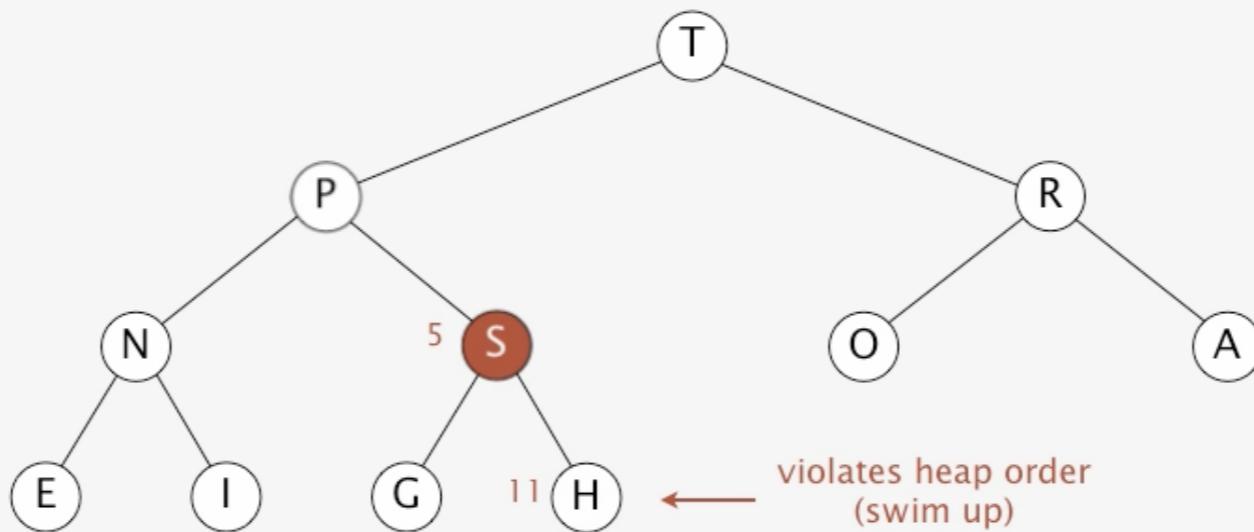
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



T	P	R	N	S	O	A	E	I	G	H
				5					11	

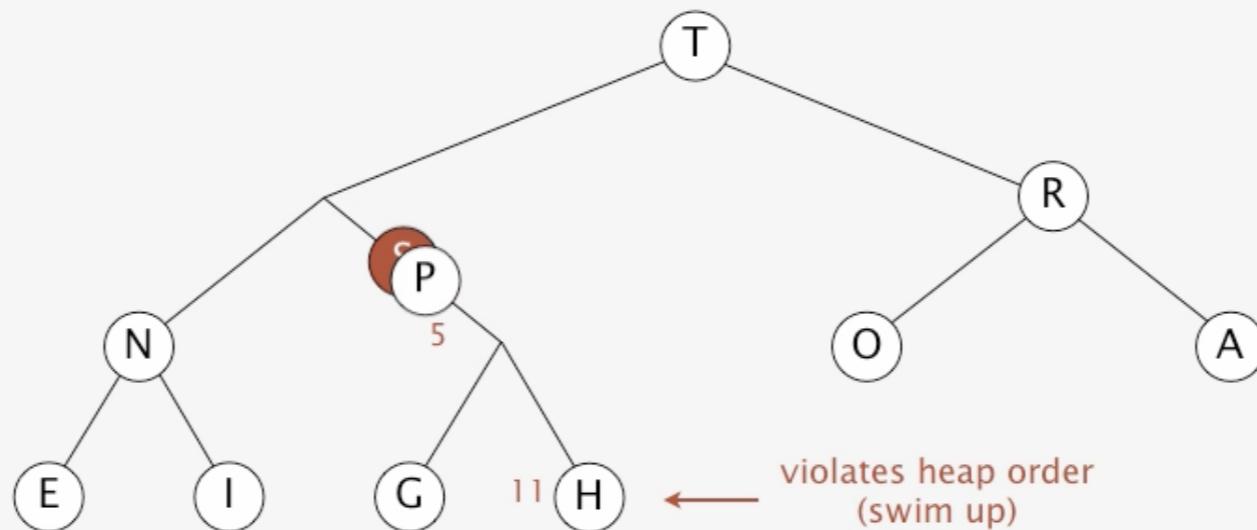
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



T	B	R	N	B	O	A	E	I	G	H
				5					11	

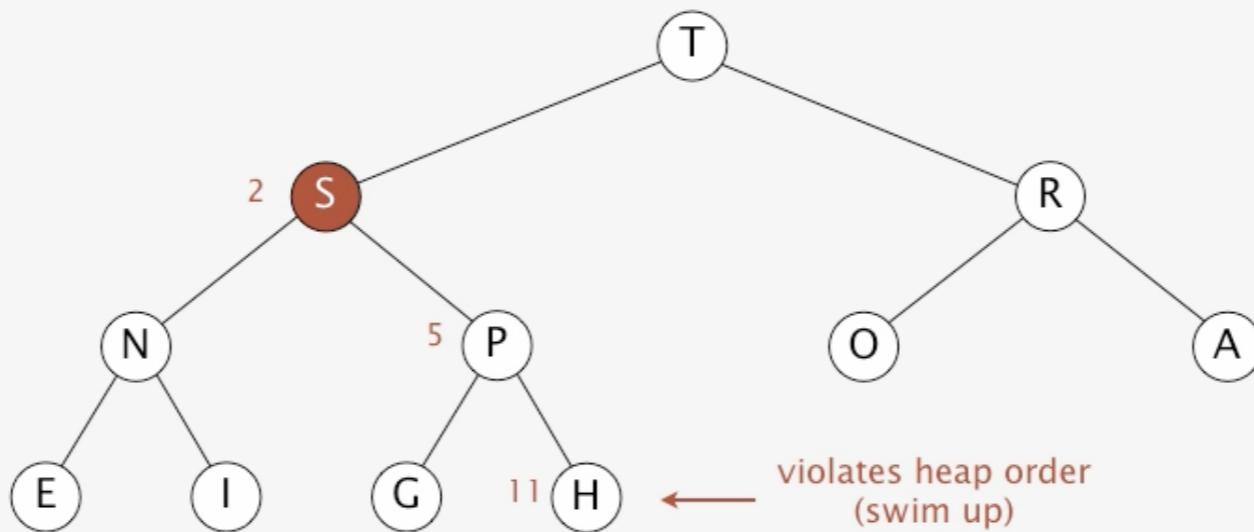
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



T	S	R	N	P	O	A	E	I	G	H
	2			5						11

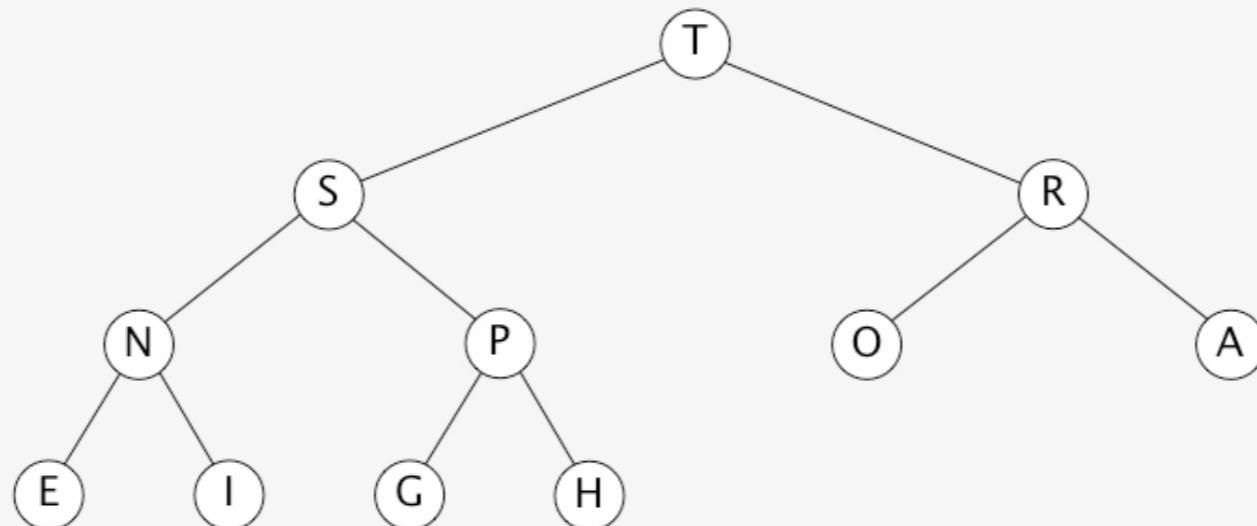
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

heap ordered



T	S	R	N	P	O	A	E	I	G	H
---	---	---	---	---	---	---	---	---	---	---

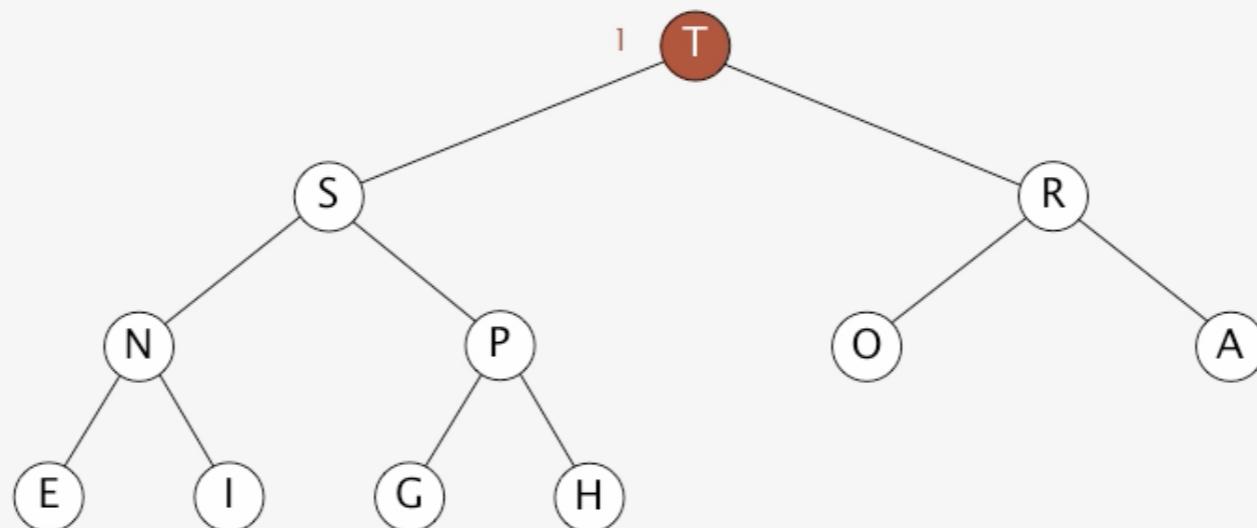
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



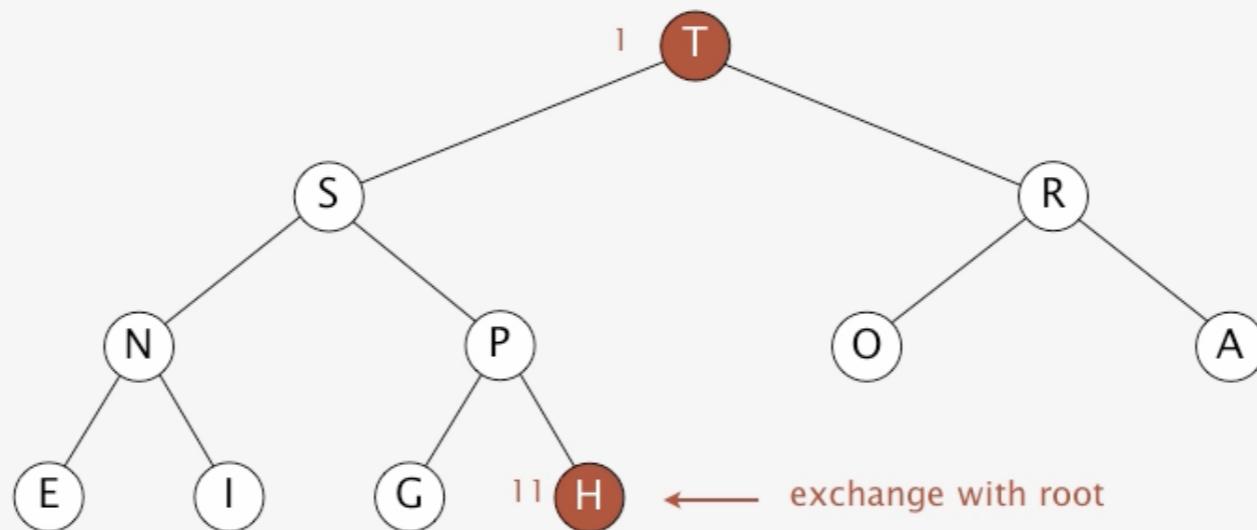
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



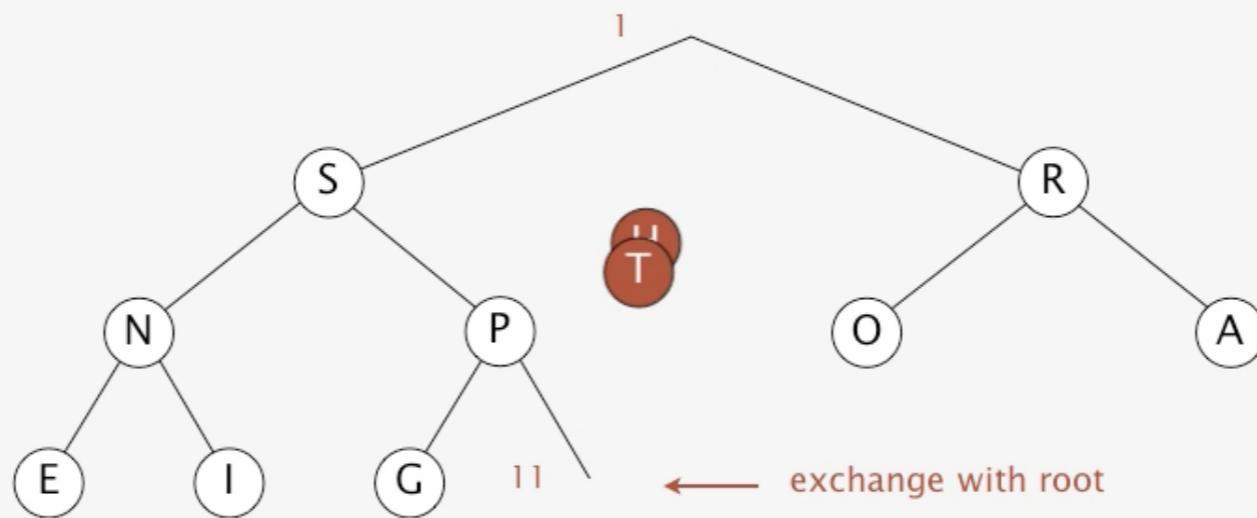
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



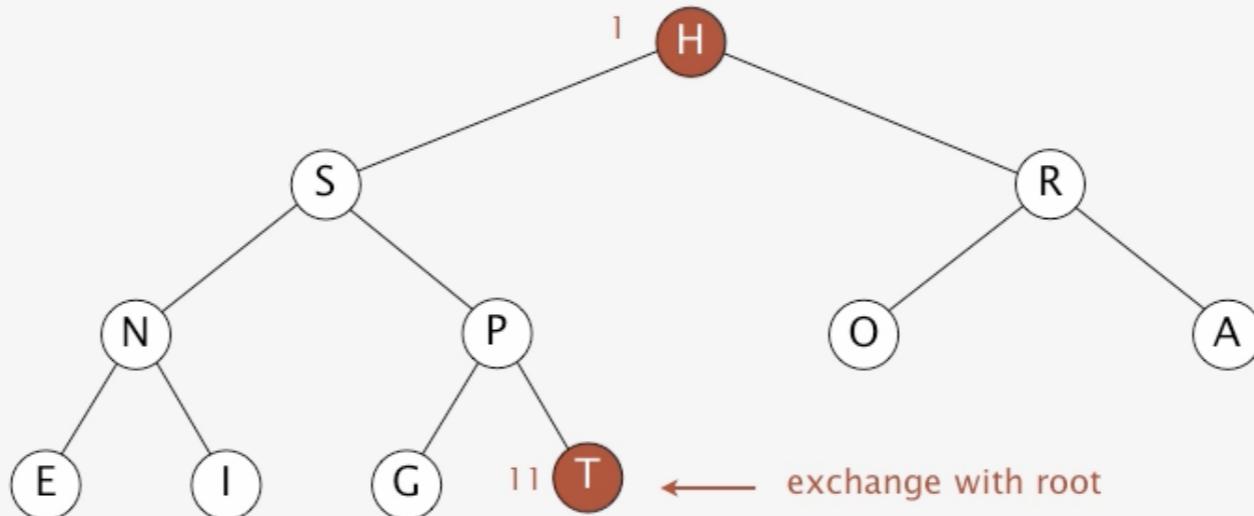
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



H	S	R	N	P	O	A	E	I	G	T
1				11						11

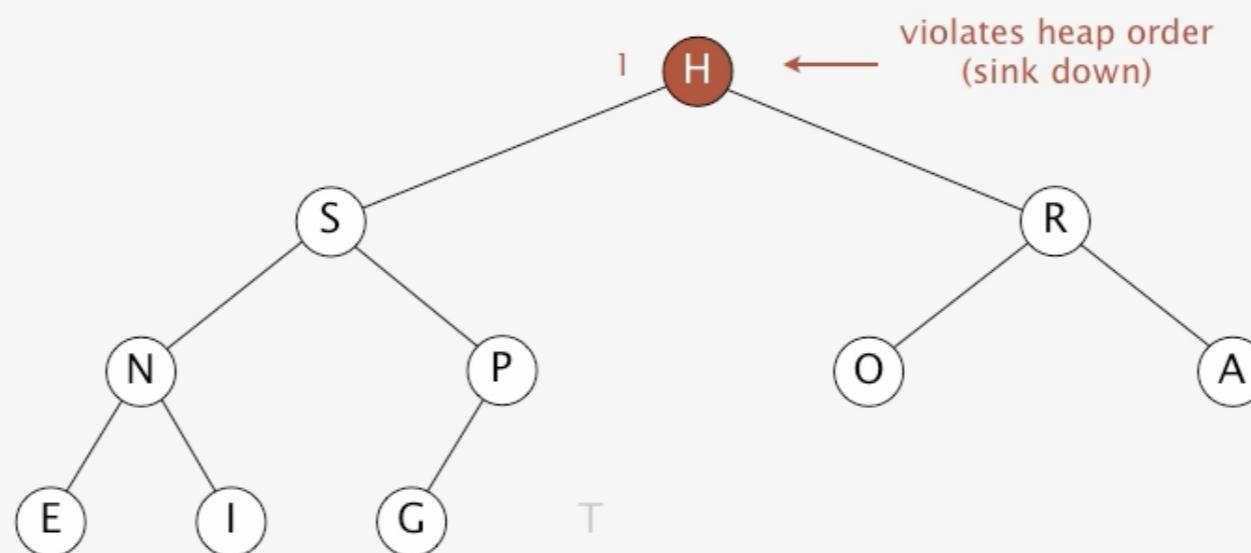
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



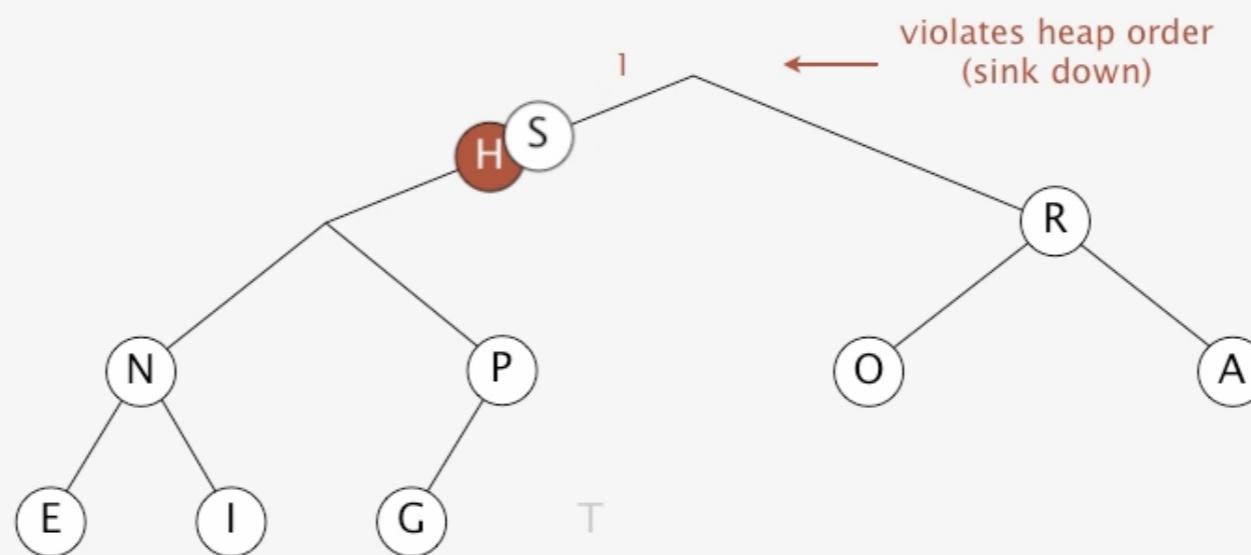
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



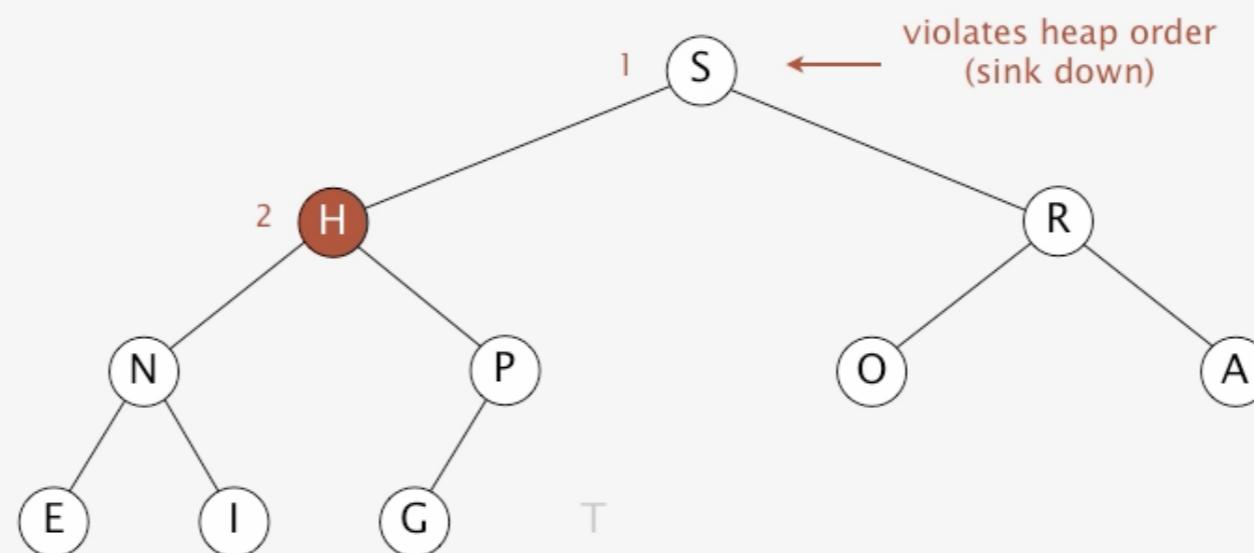
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



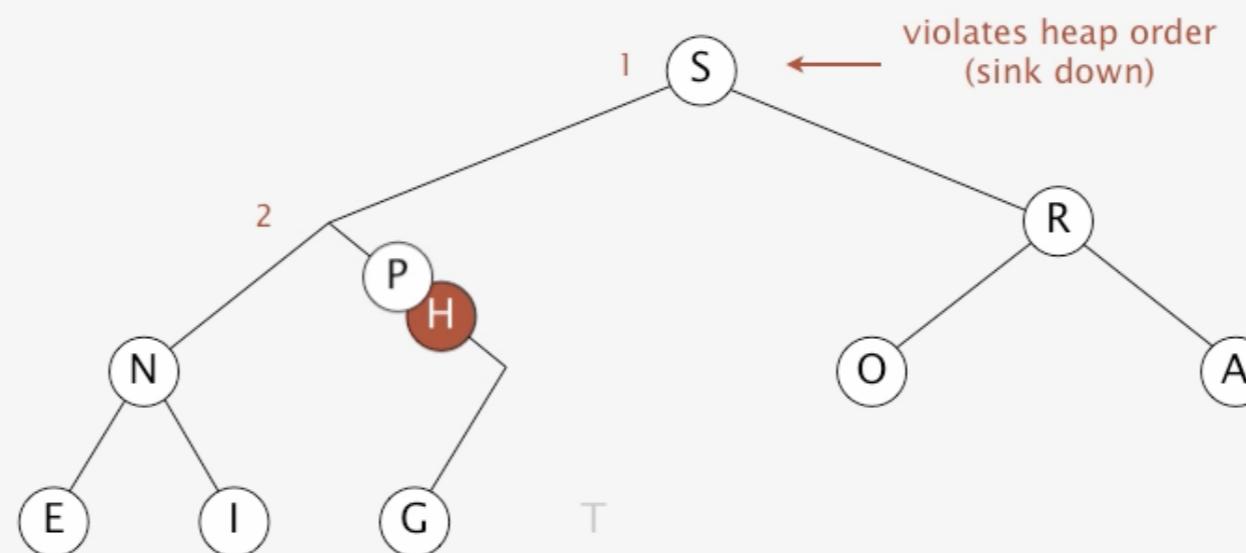
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

[Remove the maximum.](#) Exchange root with node at end, then sink it down.

remove the maximum



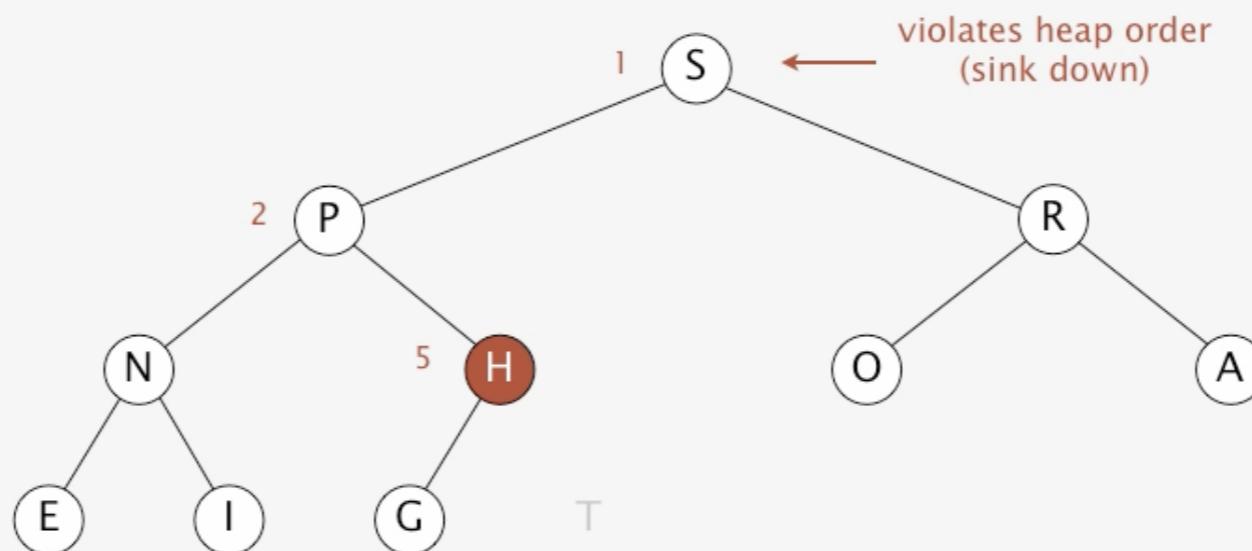
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



S	P	R	N	H	O	A	E	I	G	T
1	2			5						

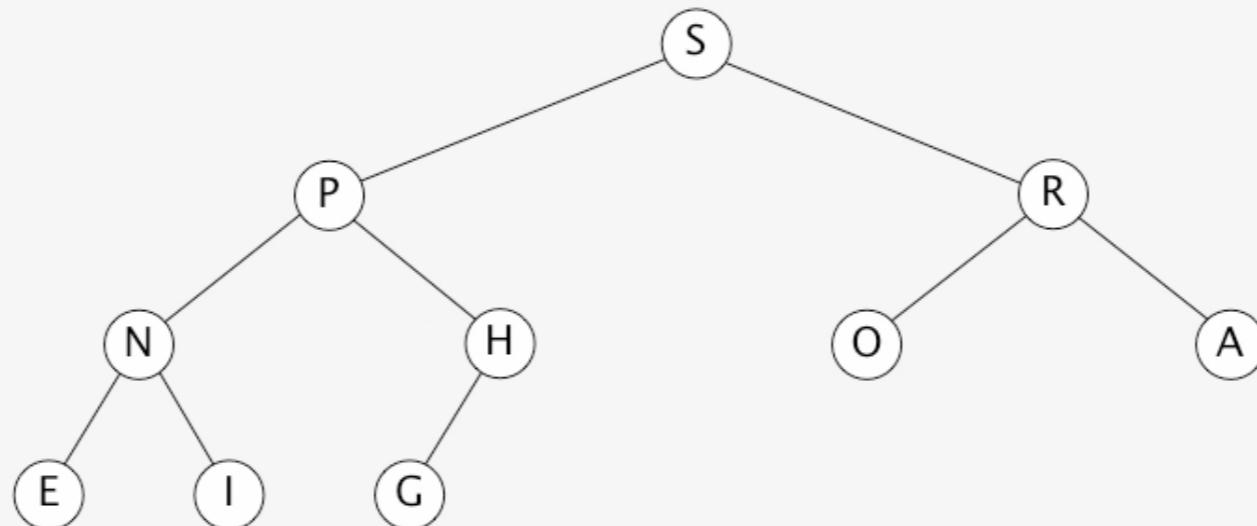
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

heap ordered



S	P	R	N	H	O	A	E	I	G
---	---	---	---	---	---	---	---	---	---

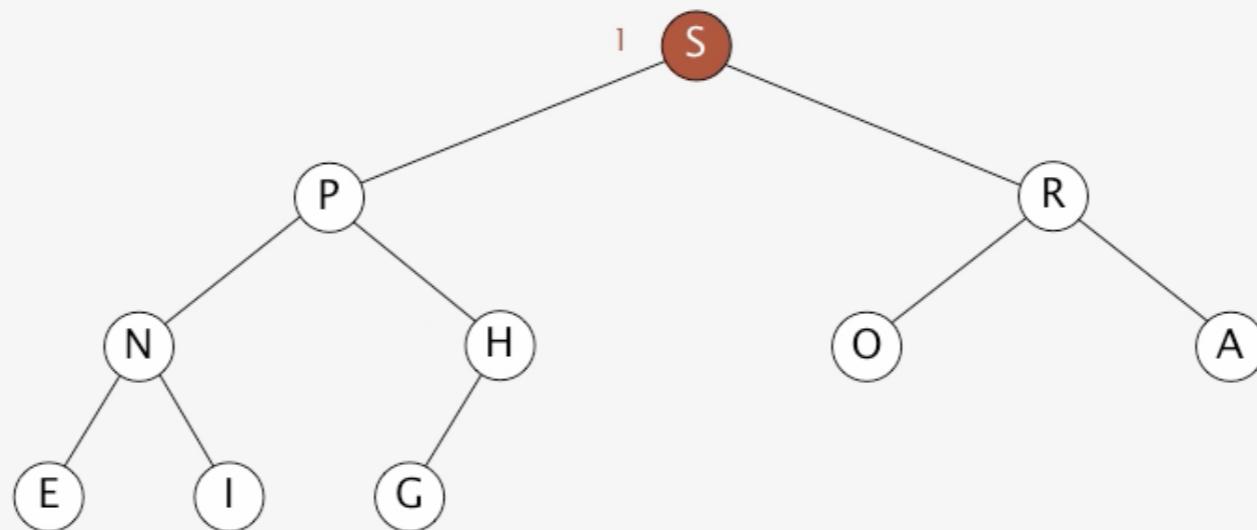
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



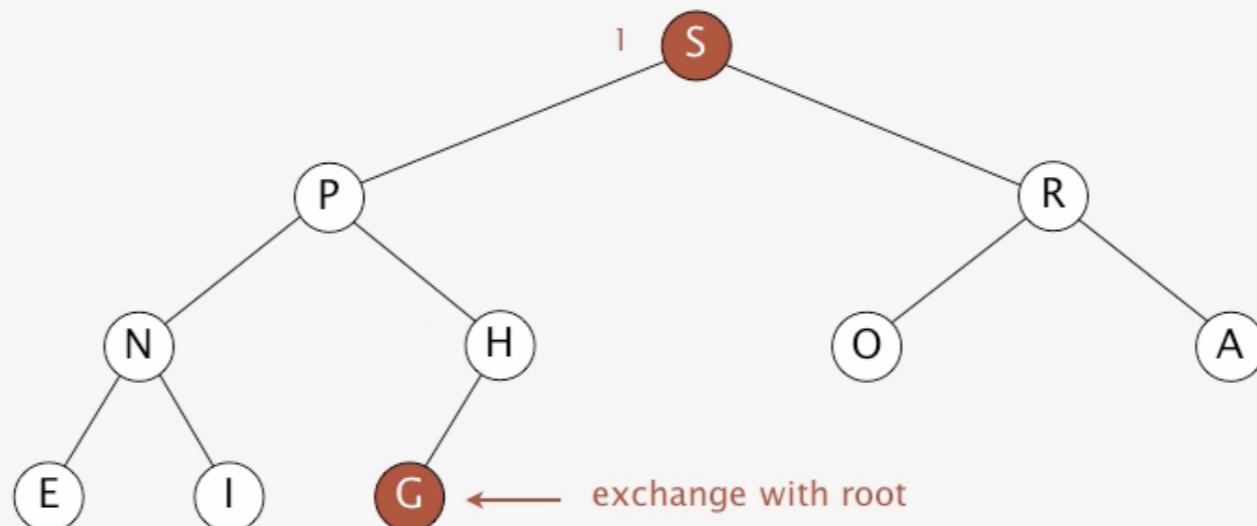
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



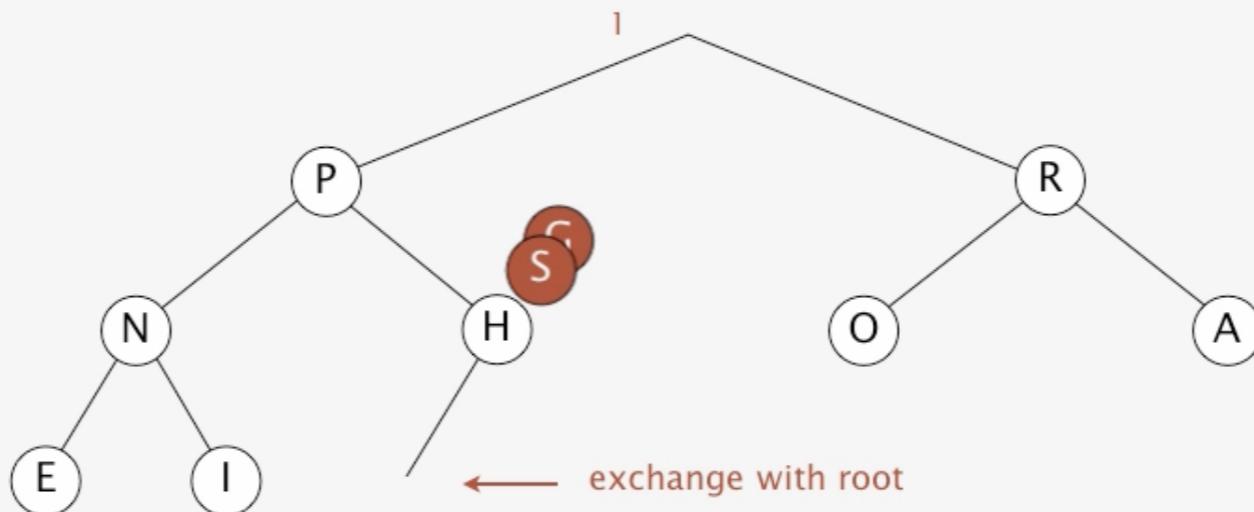
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



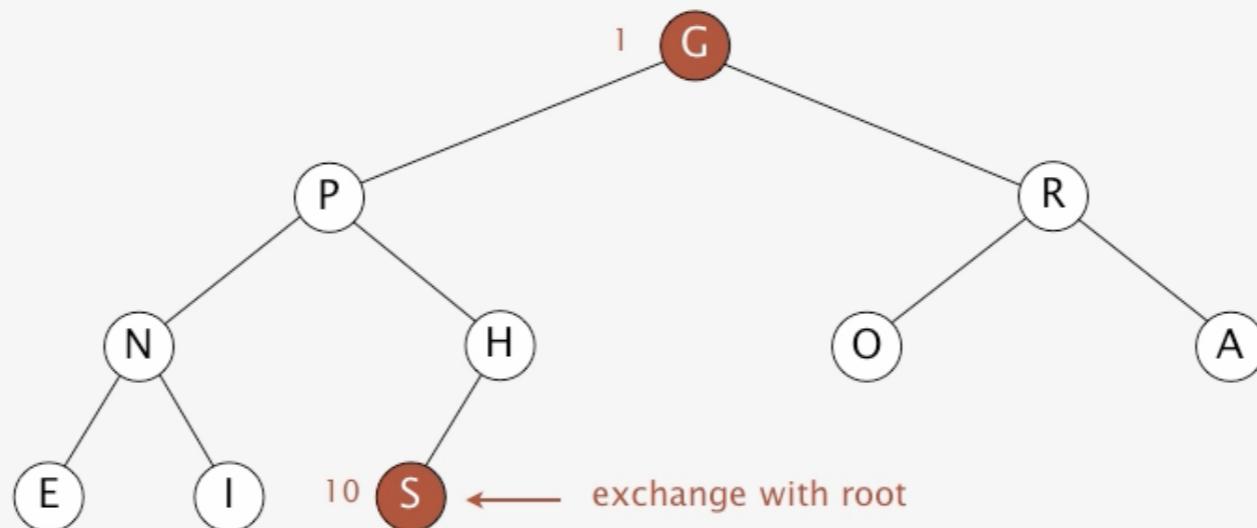
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



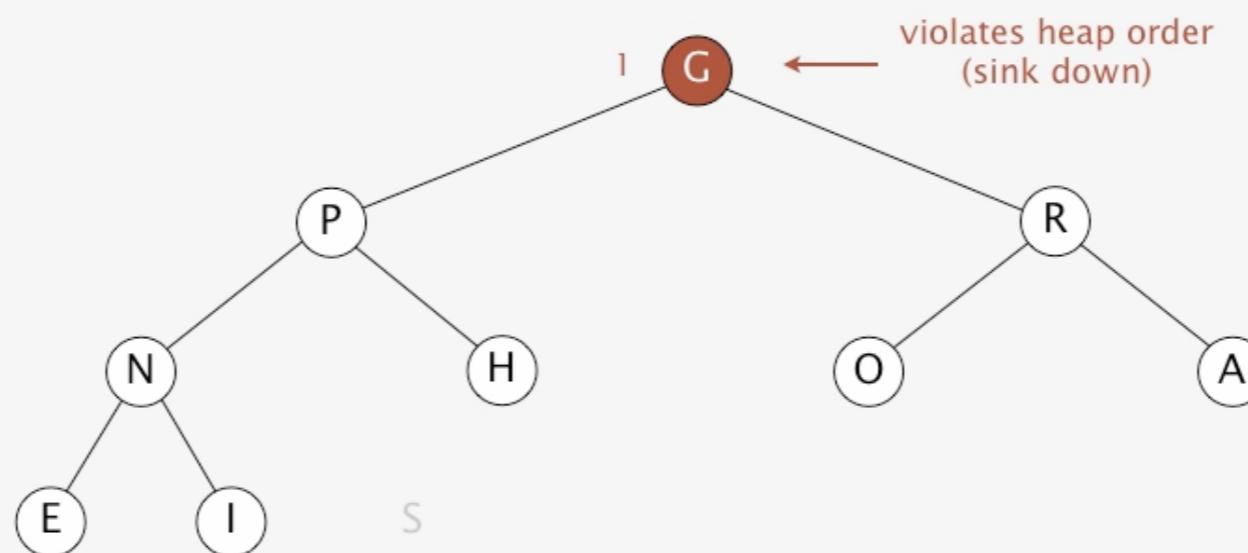
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



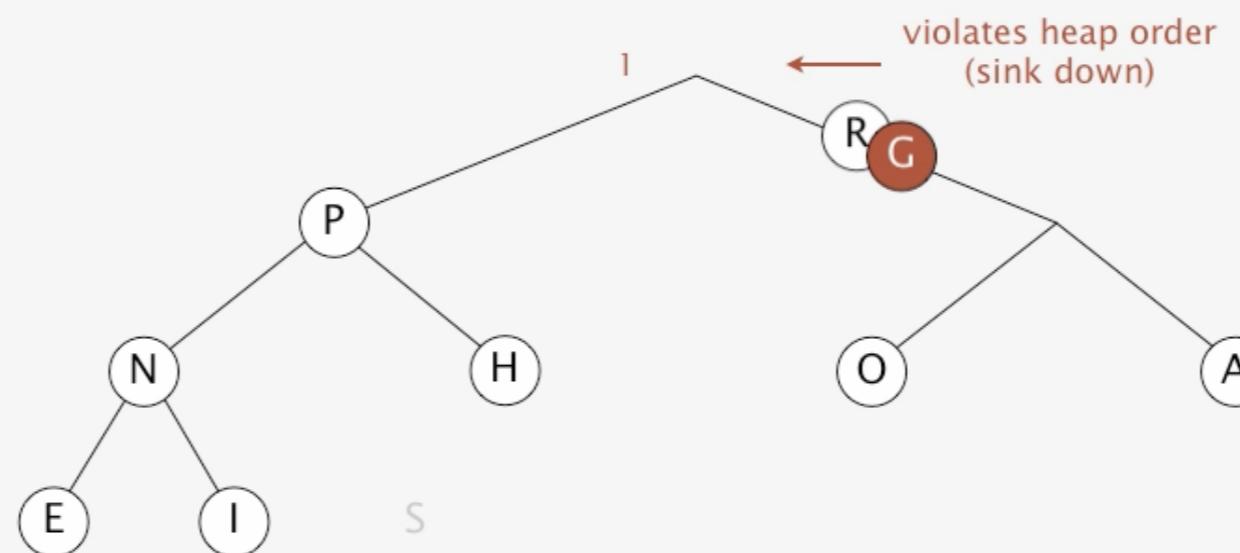
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

[Remove the maximum.](#) Exchange root with node at end, then sink it down.

remove the maximum



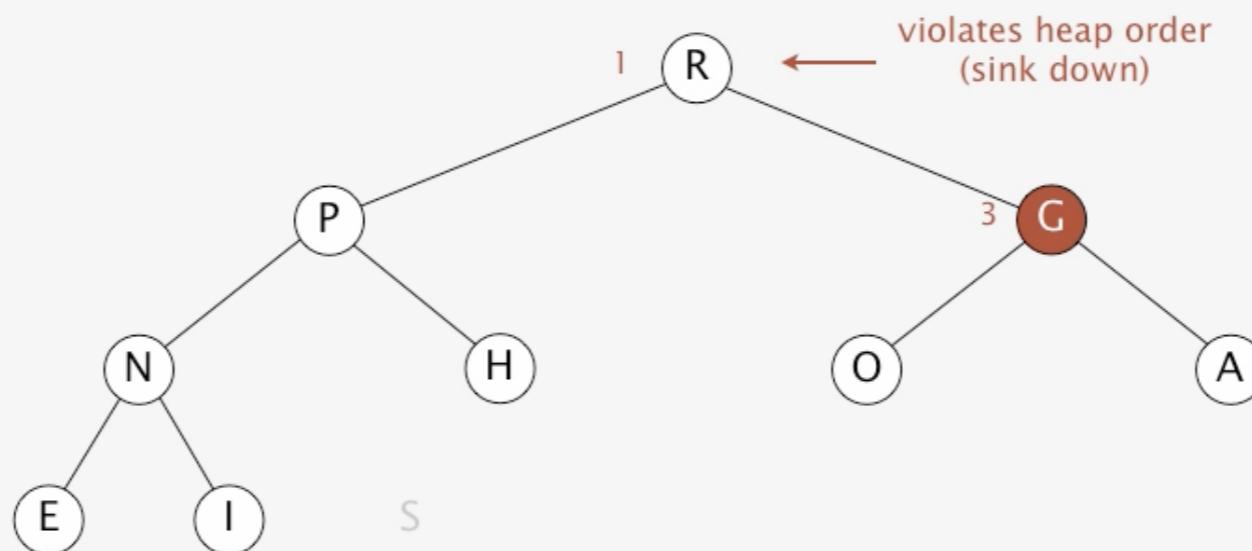
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



R	P	G	N	H	O	A	E	I	S
1	3								

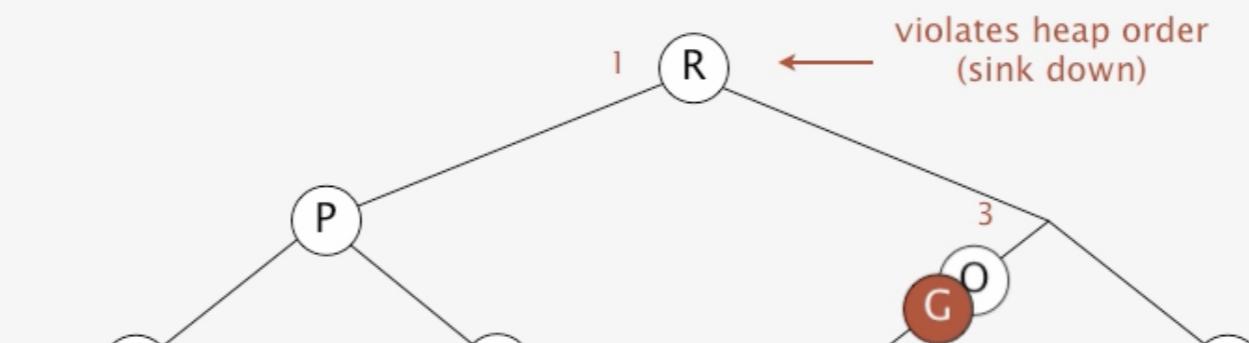
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



S

R	P	O	N	H	O	A	E	I	S
1	3								

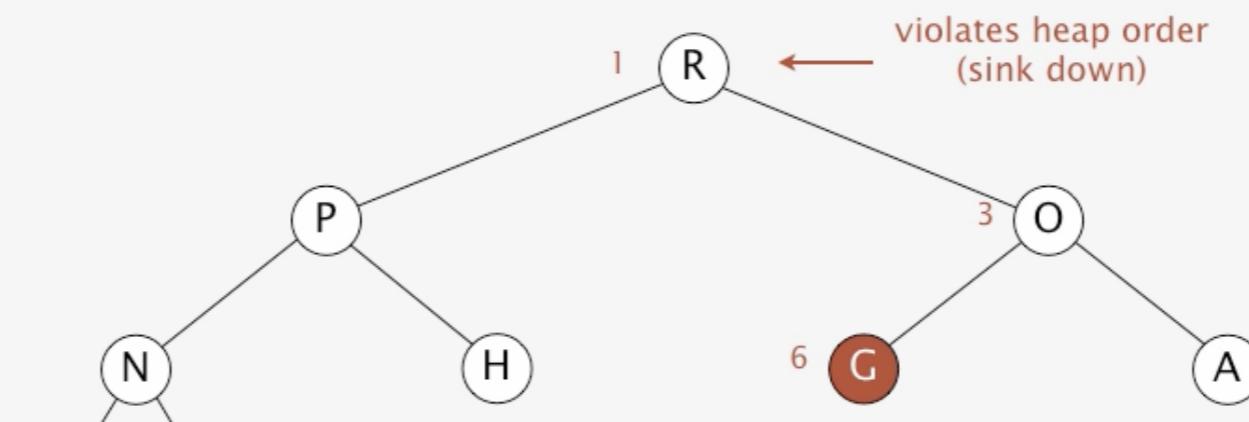
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

remove the maximum



R	P	O	N	H	G	A	E	I	S
1	3	6							

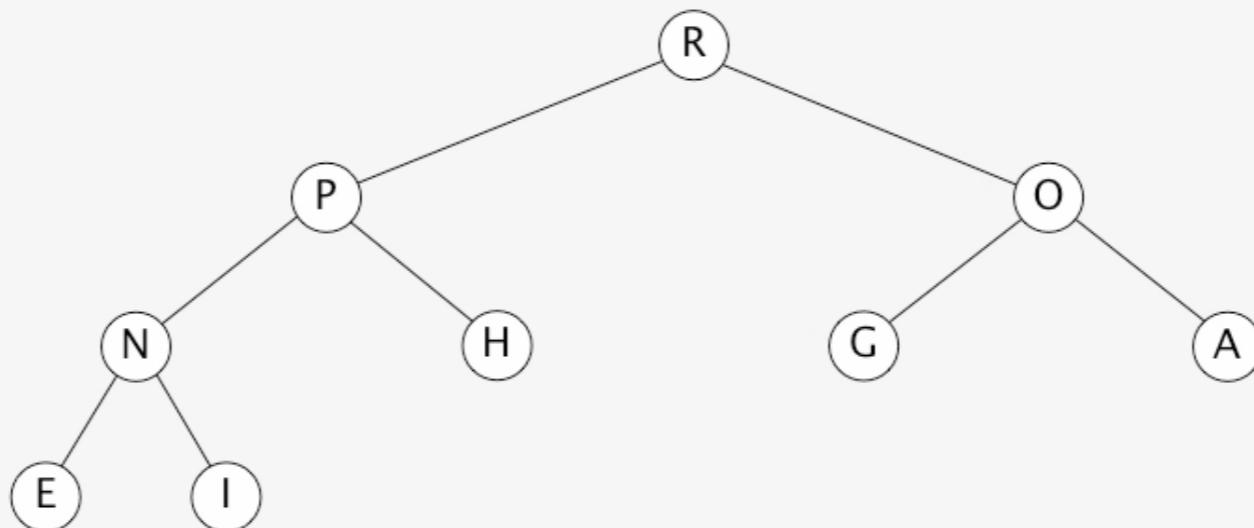
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

heap ordered



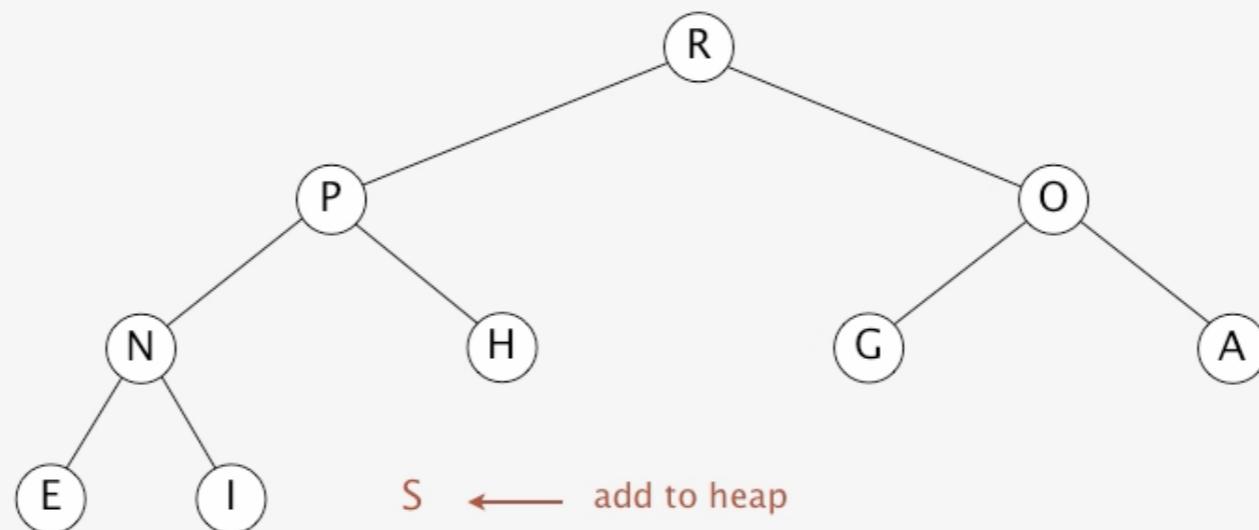
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



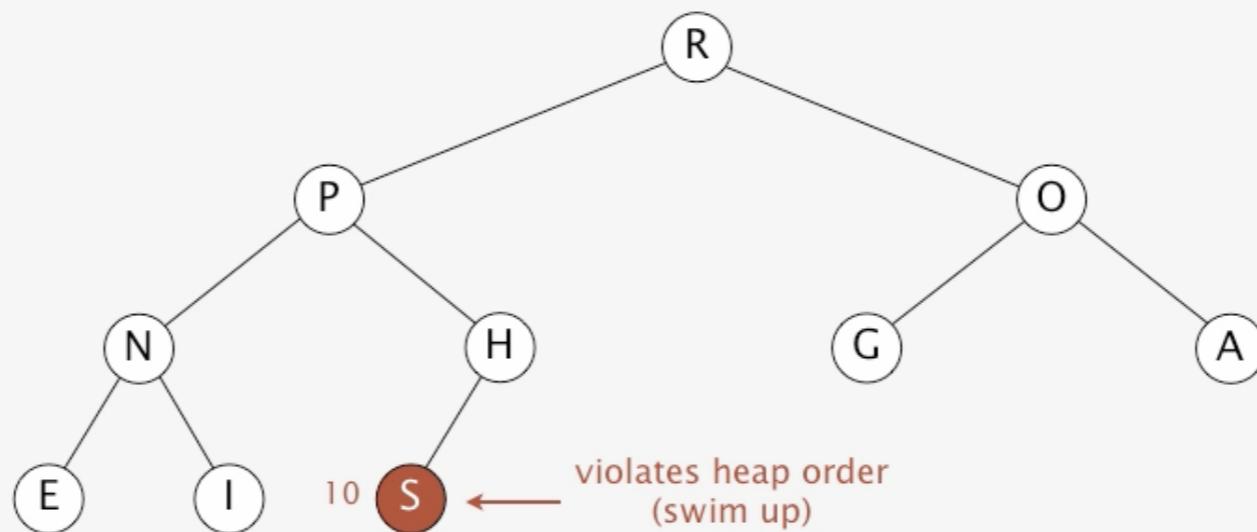
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



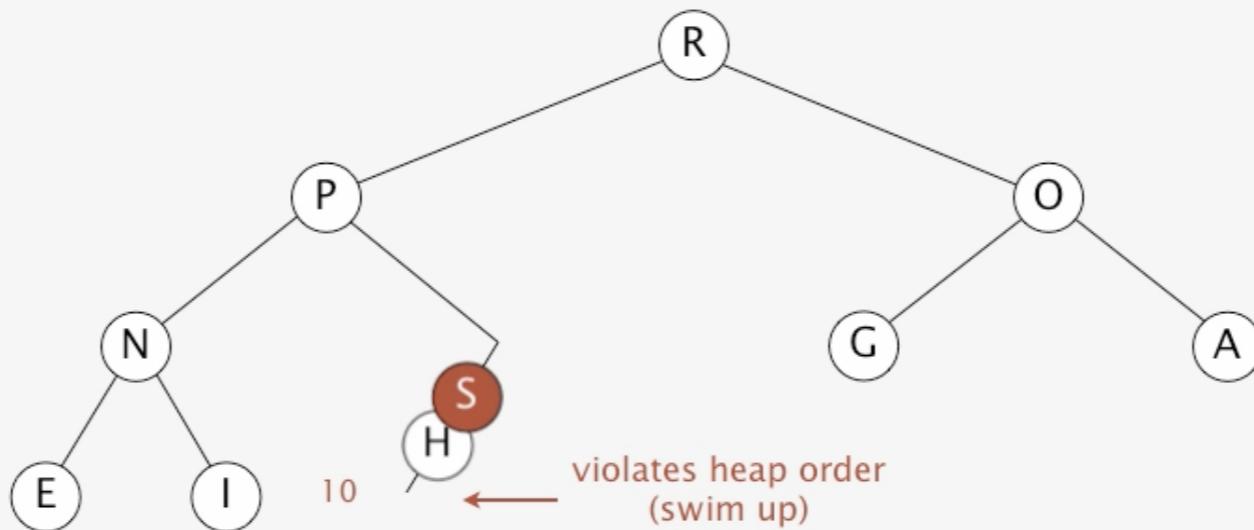
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



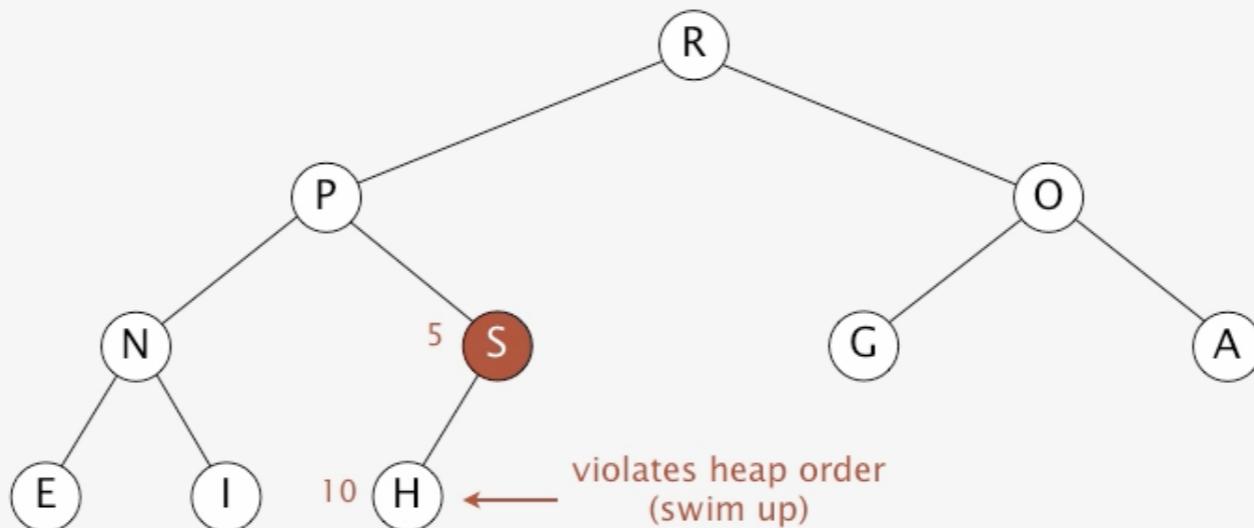
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



R	P	O	N	S	G	A	E	I	H
				5				10	

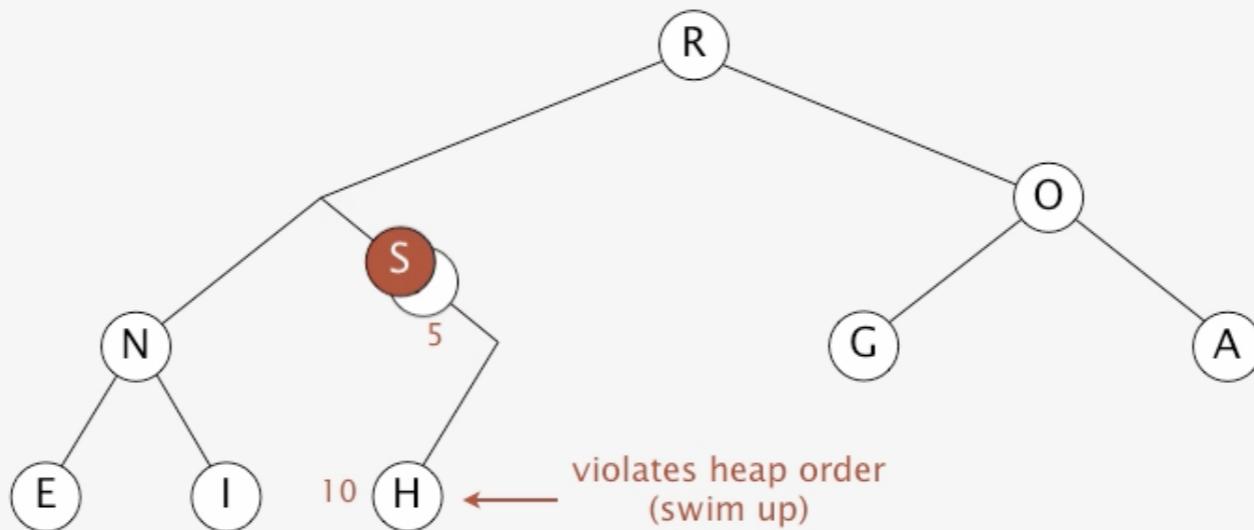
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



R	B	O	N	B	G	A	E	I	H
				5					10

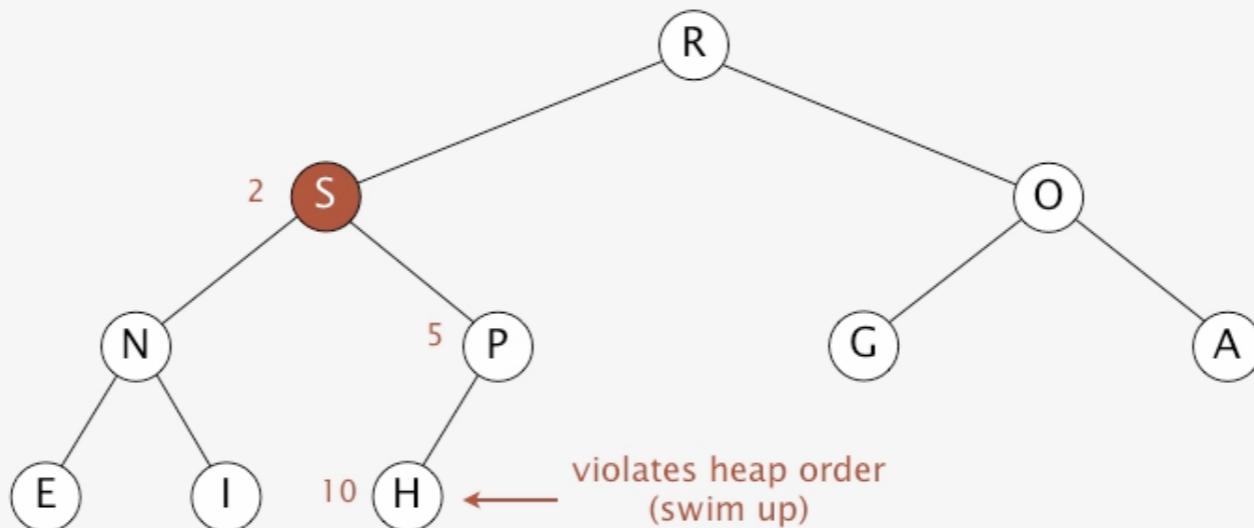
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



R	S	O	N	P	G	A	E	I	H
2	2	5	5	5	2	2	2	2	10

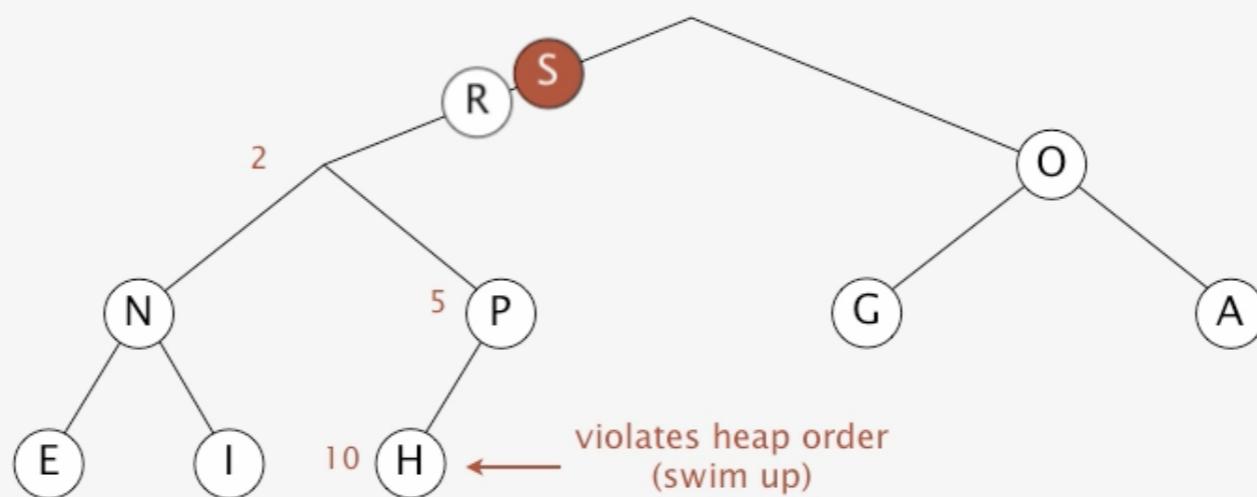
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



R	R	O	N	P	G	A	E	I	H
2	5	10							

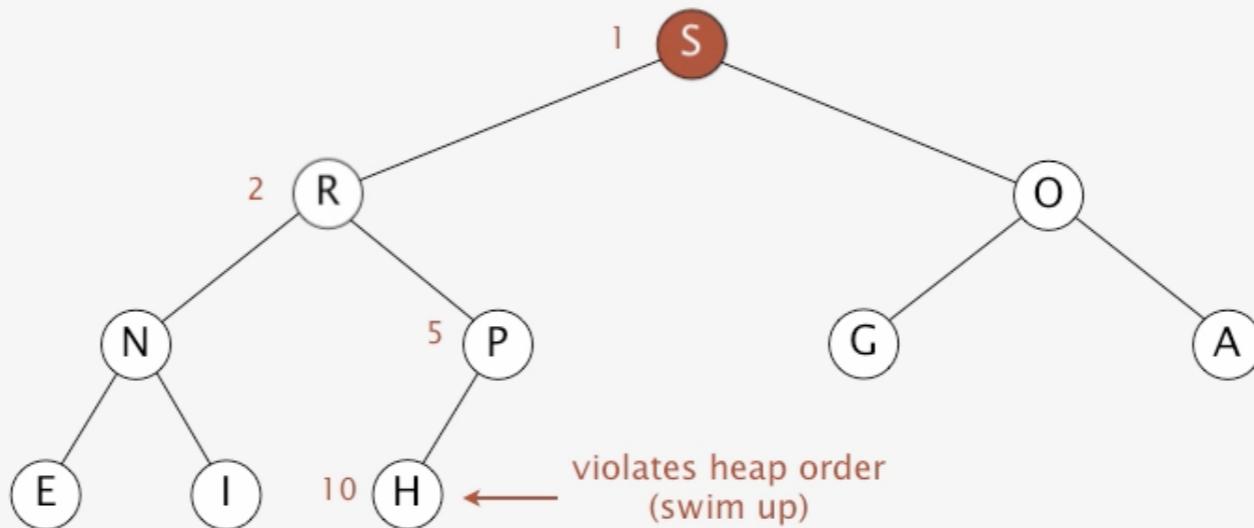
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

insert S



S	R	O	N	P	G	A	E	I	H
1	2	10	5	5					10

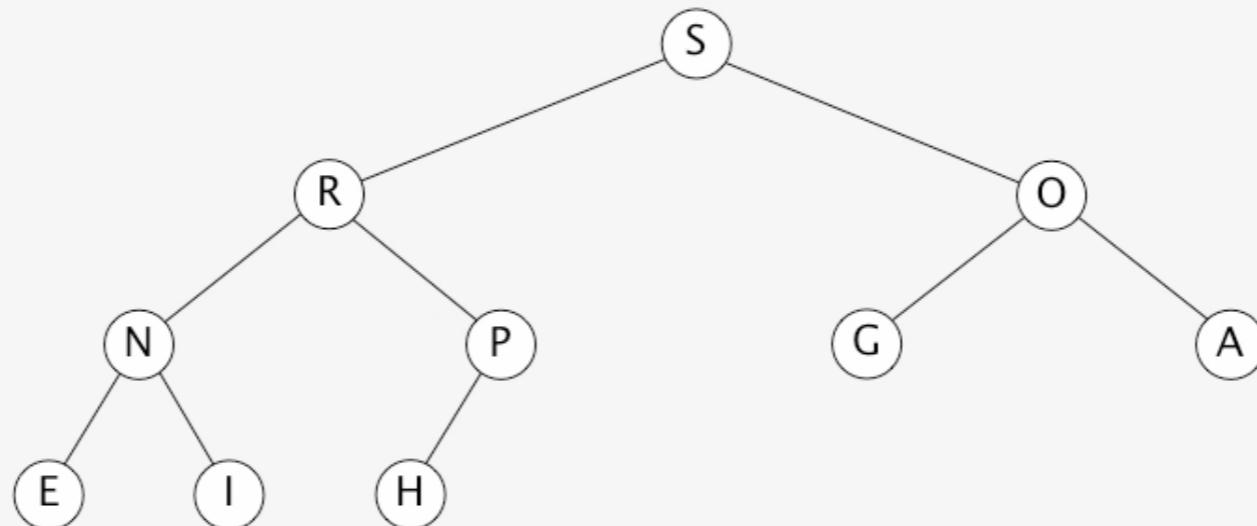
Binary Heaps

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

heap ordered



S	R	O	N	P	G	A	E	I	H
---	---	---	---	---	---	---	---	---	---

Binary Heaps

Promotion in a heap

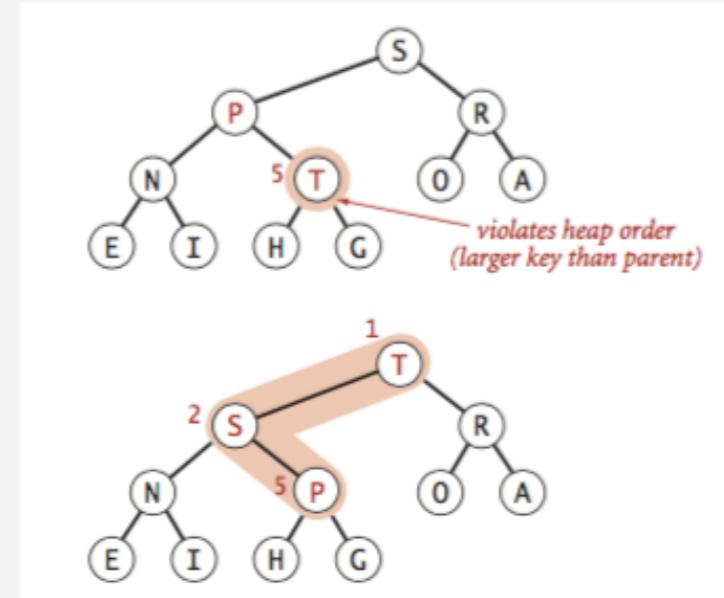
Scenario. Child's key becomes **larger** key than its parent's key.

To eliminate the violation:

- Exchange key in child with key in parent.
- Repeat until heap order restored.

```
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



Peter principle. Node promoted to level of incompetence.

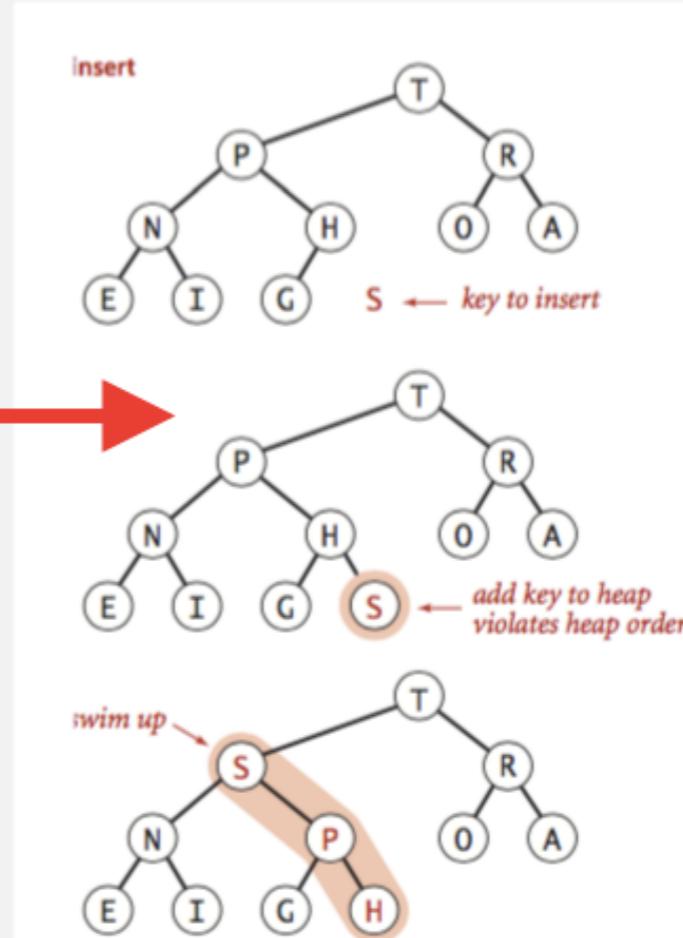
Binary Heaps

Insertion in a heap

Insert. Add node at end, then swim it up.

Cost. At most $1 + \lg N$ compares.

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



Binary Heaps

Demotion in a heap

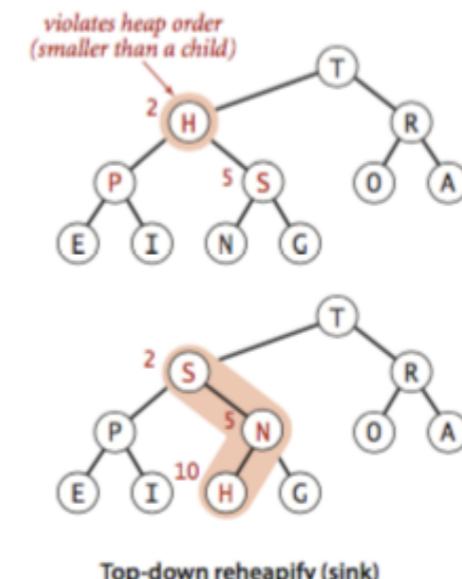
Scenario. Parent's key becomes **smaller** than one (or both) of its children's.

To eliminate the violation:

- Exchange key in parent with key in larger child.
- Repeat until heap order restored.

why not smaller child?

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



Power struggle. Better subordinate promoted.

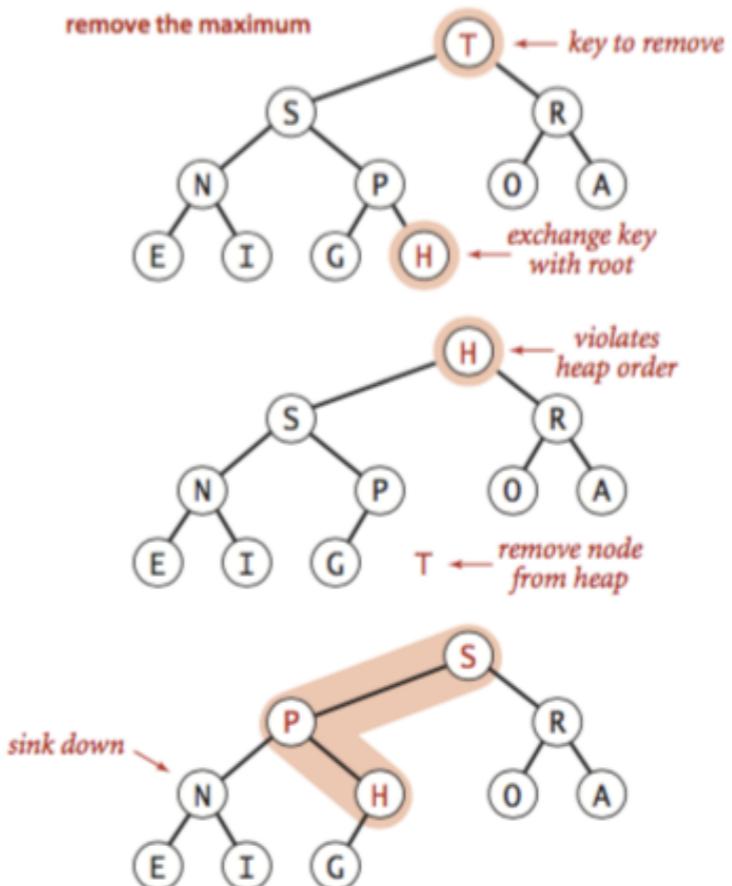
Binary Heaps

Delete the maximum in a heap

Delete max. Exchange root with node at end, then sink it down.

Cost. At most $2 \lg N$ compares.

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



Binary Heaps

Binary heap: Java implementation

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

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

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

    private void swim(int k)
    private void sink(int k)
    {   /* see previous code */ } ← heap helper functions

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

Binary Heaps

Priority queues implementation cost summary

implementation	insert	del max	max
unordered array	1	N	N
ordered array	N	1	1
binary heap	$\log N$	$\log N$	1

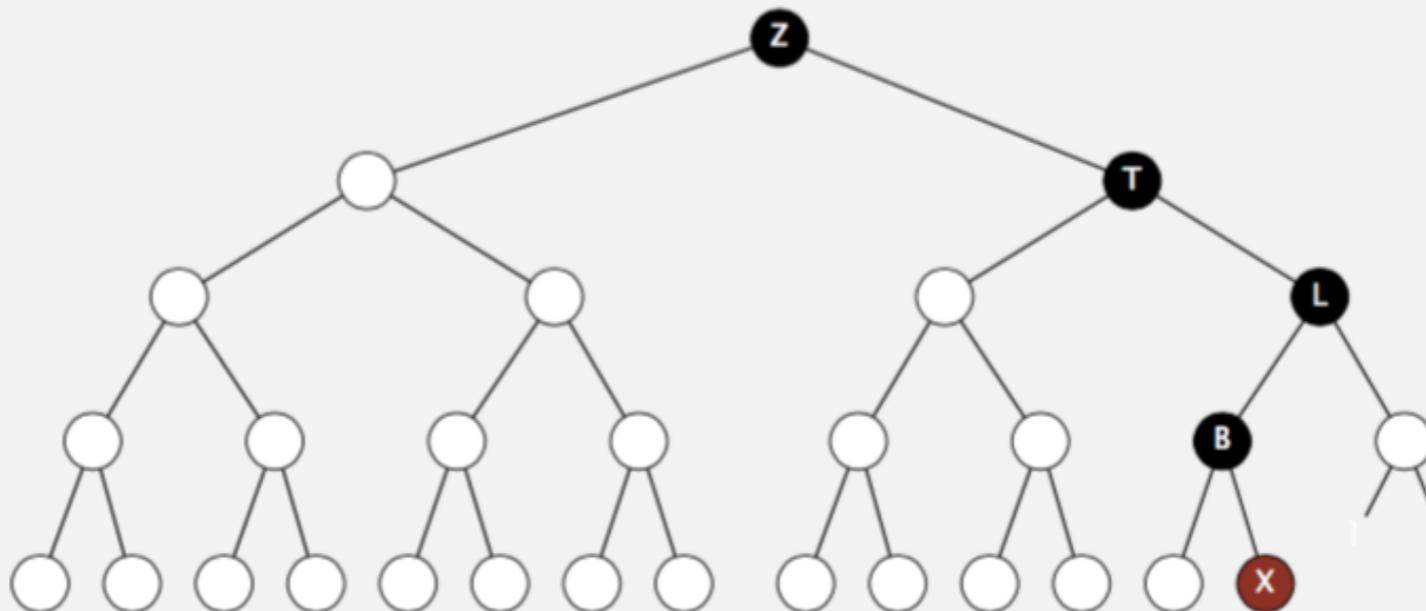
order-of-growth of running time for priority queue with N items

Binary Heaps

Binary heap: practical improvements

Half-exchanges in sink and swim.

- Reduces number of array accesses.
- Worth doing.



Binary Heaps

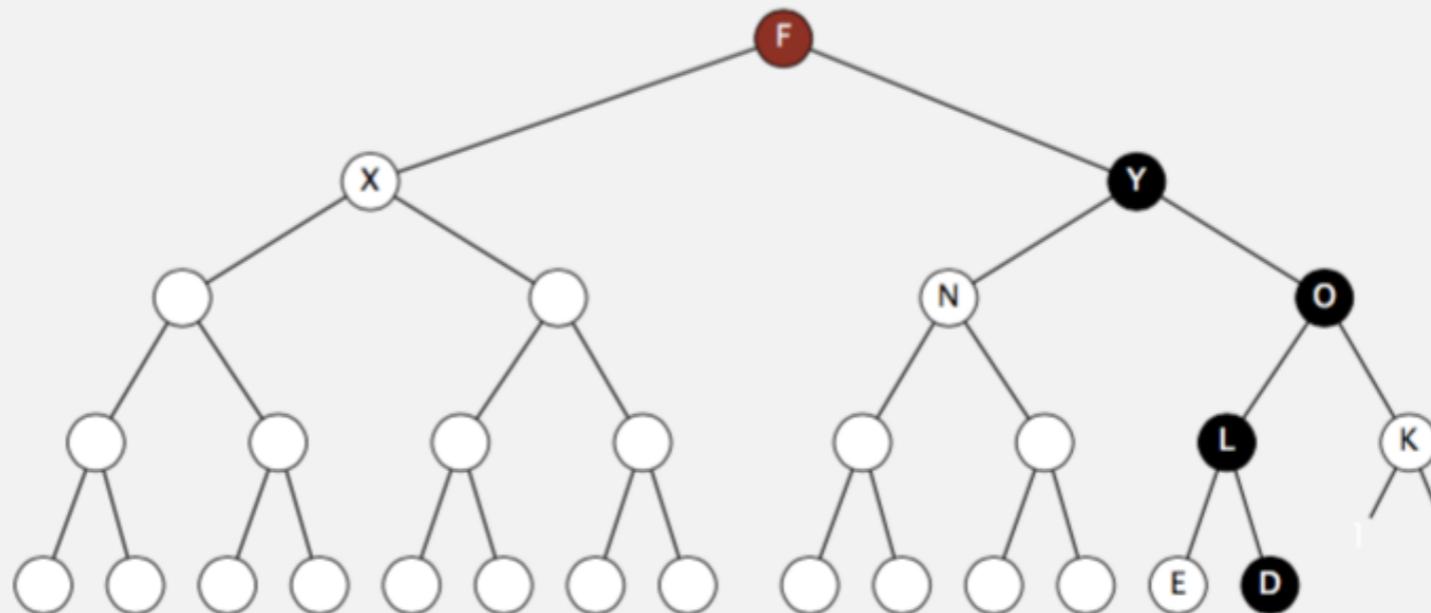
Binary heap: practical improvements

Floyd's sink-to-bottom trick.

- Sink key at root all the way to bottom. ← 1 compare per node
- Swim key back up. ← some extra compares and exchanges
- Fewer compares; more exchanges.
- Worthwhile depending on cost of compare and exchange.



R. W. Floyd
1978 Turing award

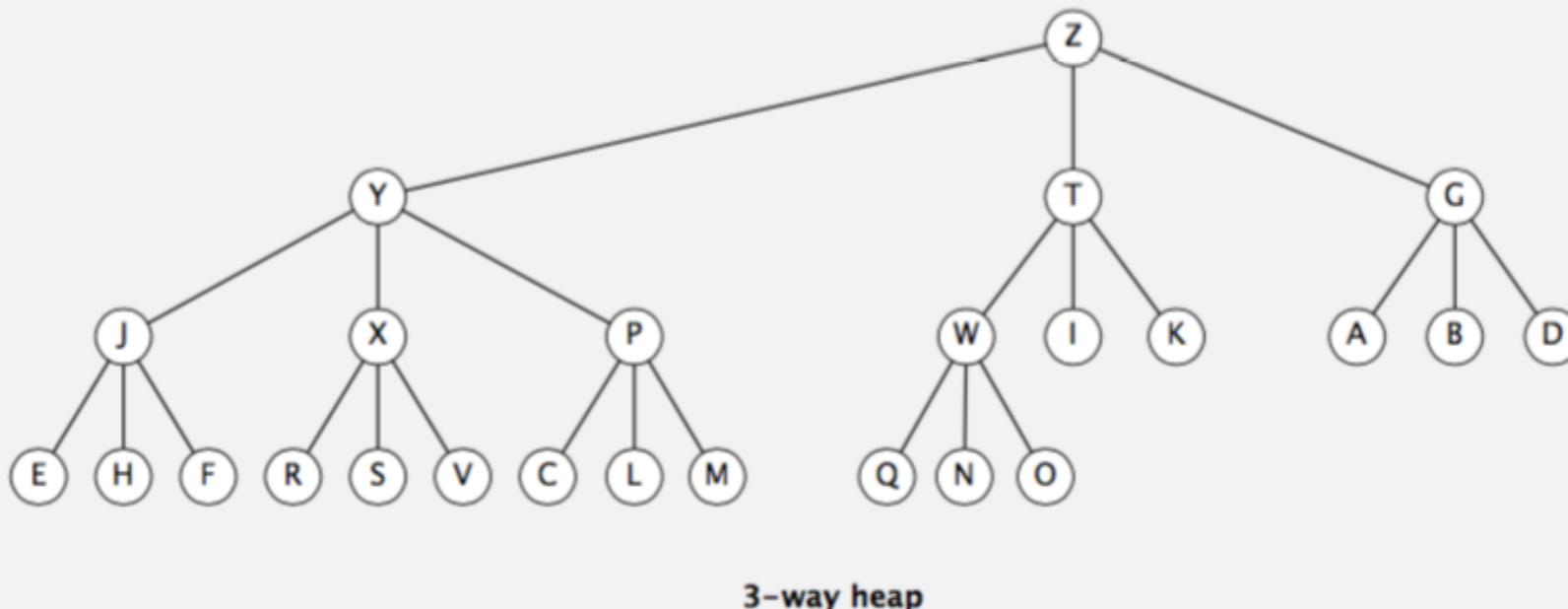


Binary Heaps

Binary heap: practical improvements

Multiway heaps.

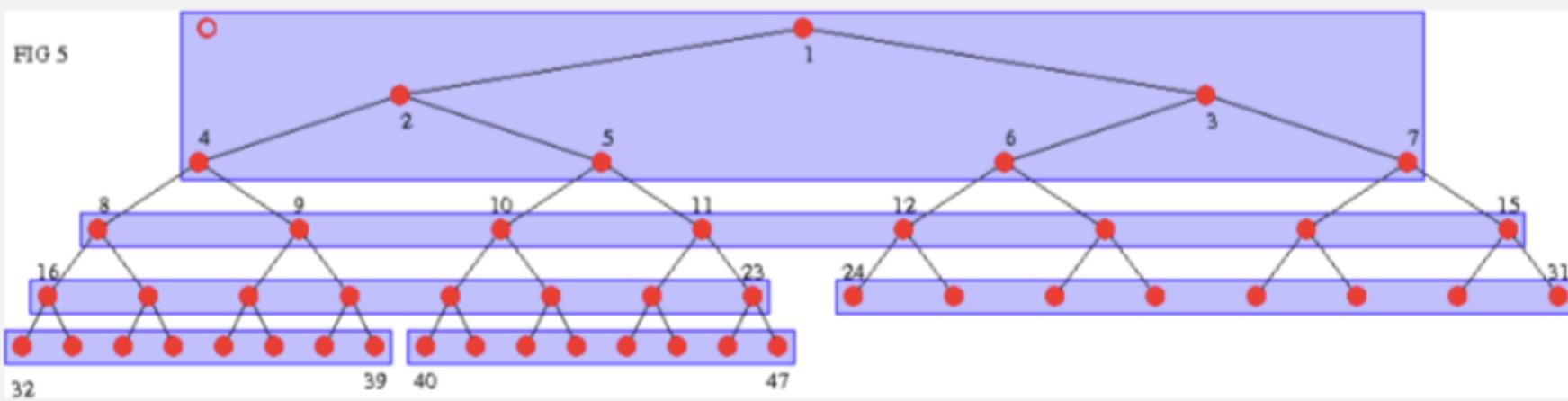
- Complete d -way tree.
- Parent's key no smaller than its children's keys.
- Swim takes $\log_d N$ compares; sink takes $d \log_d N$ compares.
- Sweet spot: $d = 4$.



Binary Heaps

Binary heap: practical improvements

Caching. Binary heap is not cache friendly.

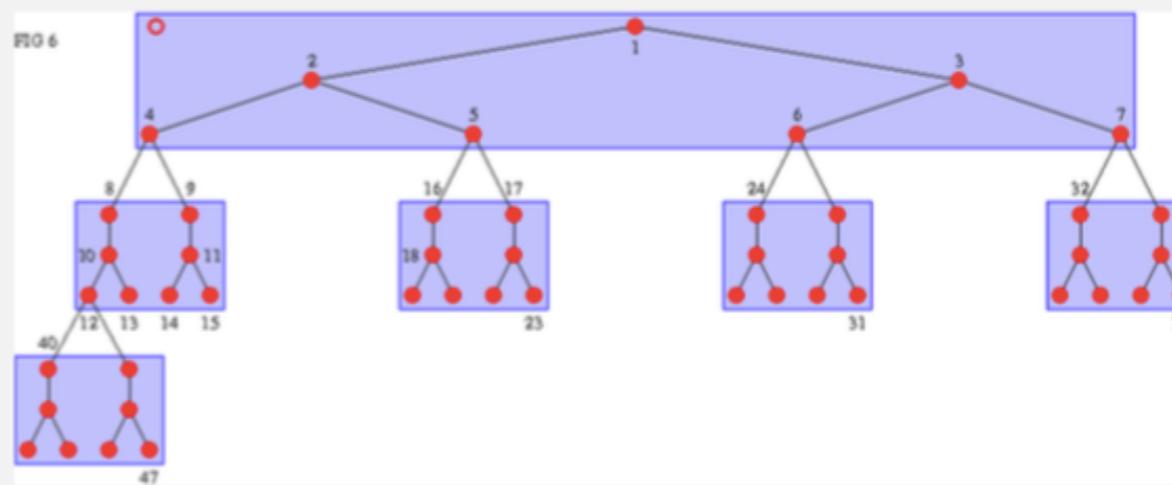
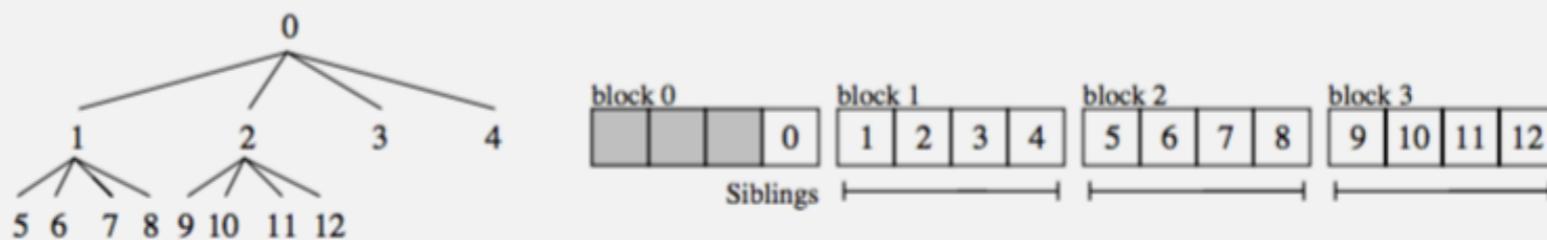


Binary Heaps

Binary heap: practical improvements

Caching. Binary heap is not cache friendly.

- Cache-aligned d -heap.
- Funnel heap.
- B-heap.
- ...



Binary Heaps

Priority queues implementation cost summary

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	$\log N^\dagger$	1
Brodal queue	1	$\log N$	1
impossible	1	1	1

← why impossible?

† amortized

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

Binary Heaps

Binary heap considerations

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 efficiently with `sink()` and `swim()`
[stay tuned for Prim/Dijkstra]

Immutability of keys.

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

Binary Heaps

Immutability: implementing in Java

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

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

```
public final class Vector {           ← can't override instance methods
    private final int N;
    private final double[] data;        ← instance variables private and final

    public Vector(double[] data) {
        this.N = data.length;
        this.data = new double[N];
        for (int i = 0; i < N; i++)
            this.data[i] = data[i];
    }
    ...
}
```

defensive copy of mutable
instance variables

instance methods don't change
instance variables

Immutable. String, Integer, Double, Color, Vector, Transaction, Point2D.

Mutable. StringBuilder, Stack, Counter, Java array.

Binary Heaps

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)



Lecture Overview



2.4 Priority Queues

- API and elementary implementations
- Binary heaps
- Heapsort
- Event-driven simulation

Sorting with a binary heap

Q. What is this sorting algorithm?

```
public void sort(String[] a)
{
    int N = a.length;
    MaxPQ<String> pq = new MaxPQ<String>();
    for (int i = 0; i < N; i++)
        pq.insert(a[i]);
    for (int i = N-1; i >= 0; i--)
        a[i] = pq.delMax();
}
```

Q. What are its properties?

A. $N \log N$, extra array of length N , not stable.

Heapsort intuition. A heap is an array; do sort in place.

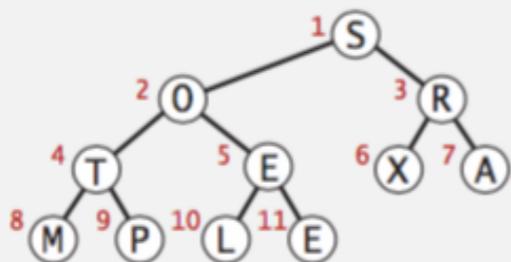
Heapsort

Heapsort

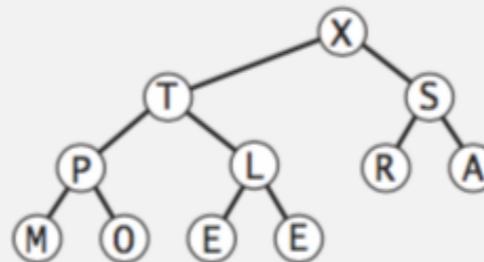
Basic plan for in-place sort.

- View input array as a complete binary tree.
- Heap construction: build a max-heap with all N keys.
- Sortdown: repeatedly remove the maximum key.

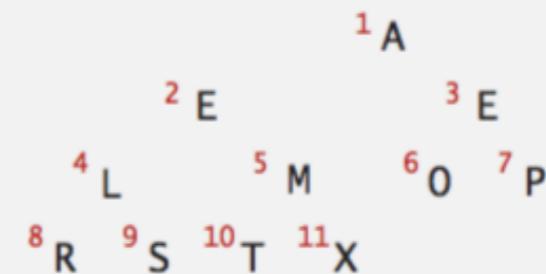
keys in arbitrary order



build max heap
(in place)



sorted result
(in place)



1 2 3 4 5 6 7 8 9 10 11
S O R T E X A M P L E

1 2 3 4 5 6 7 8 9 10 11
X T S P L R A M O E E

1 2 3 4 5 6 7 8 9 10 11
A E E L M O P R S T X

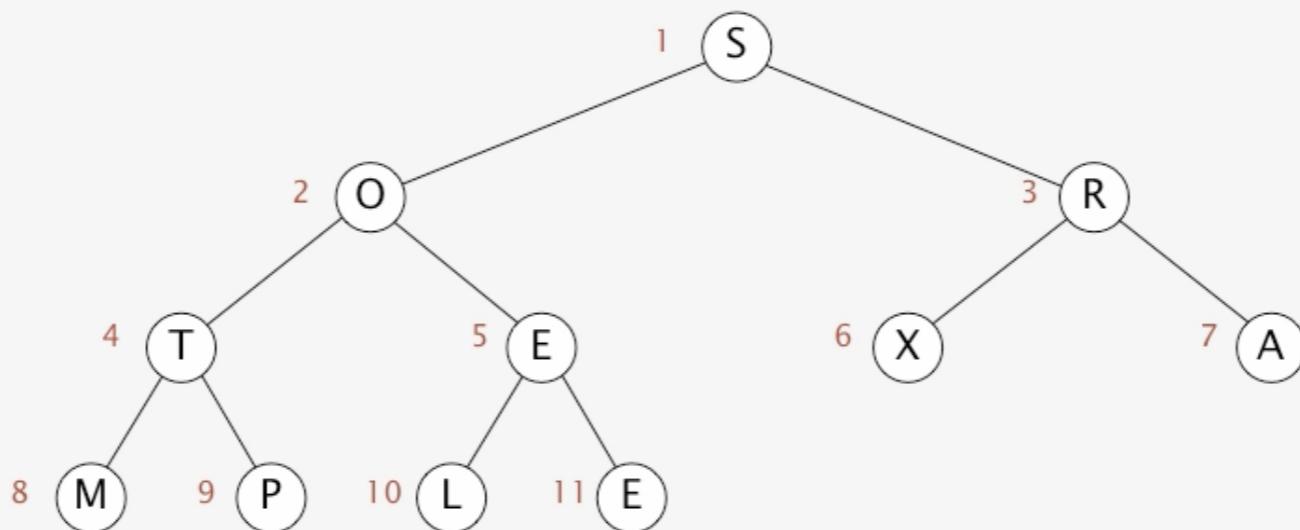
Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

we assume array entries are indexed 1 to N

array in arbitrary order

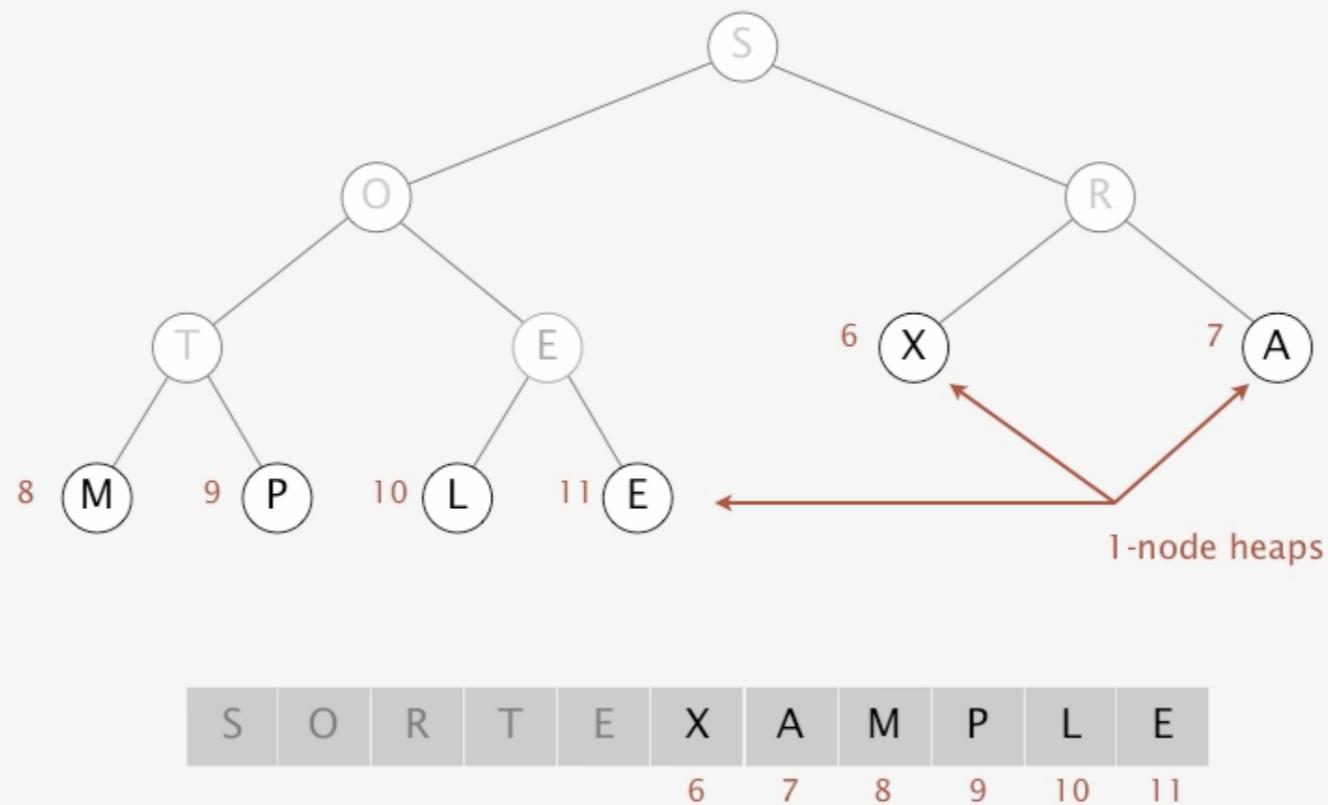


S	O	R	T	E	X	A	M	P	L	E
1	2	3	4	5	6	7	8	9	10	11

Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

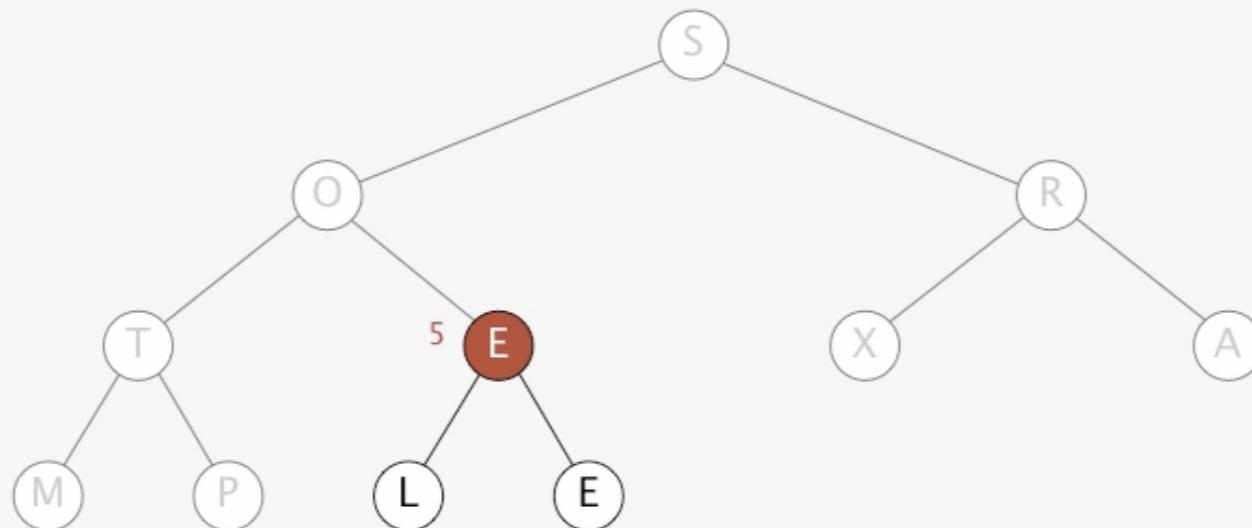


Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 5

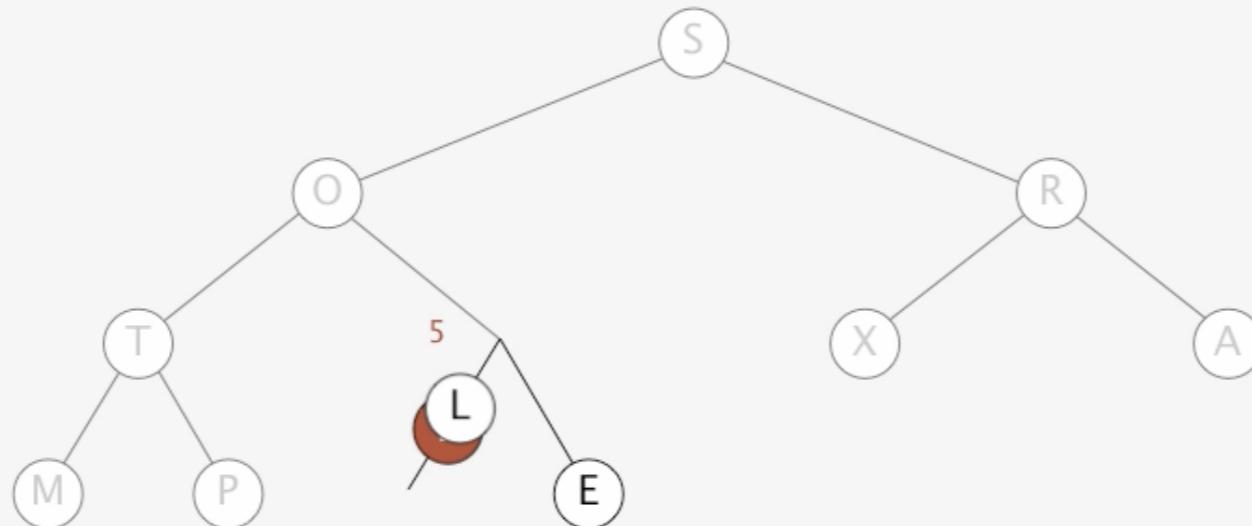


Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 5

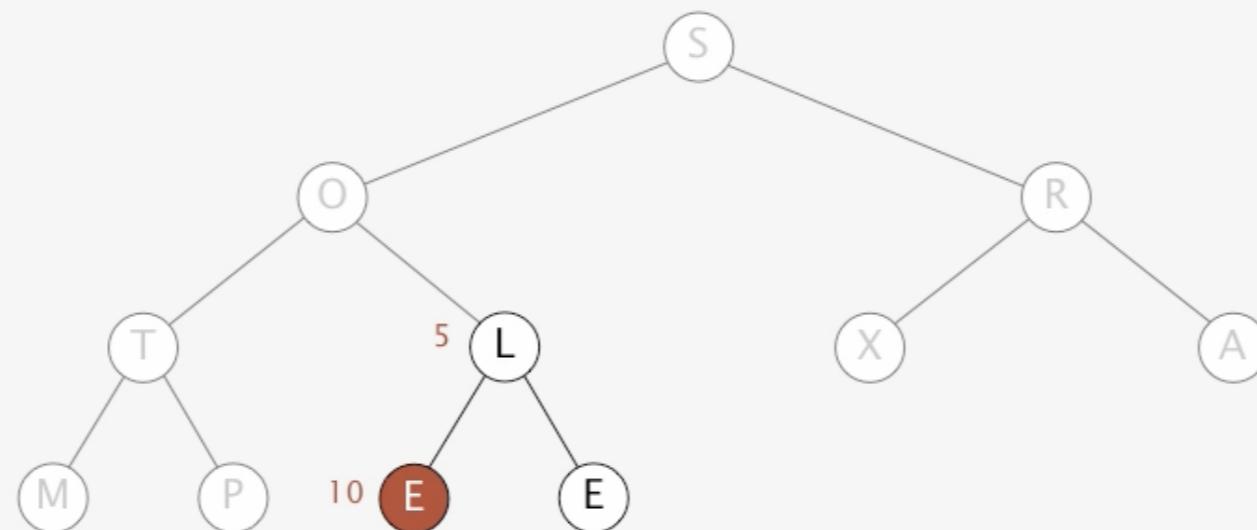


Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 5

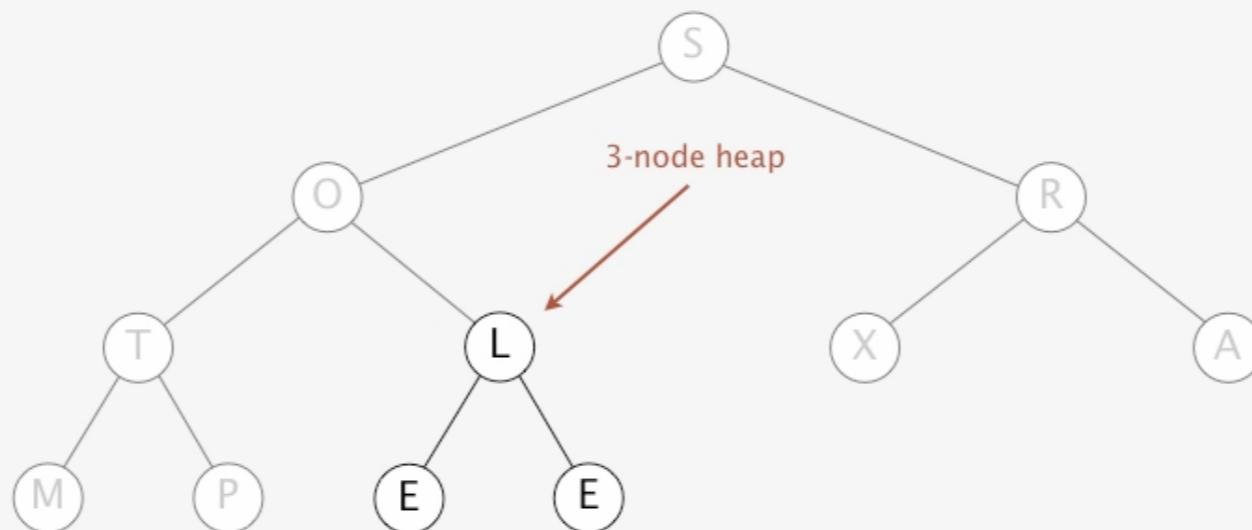


Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 5



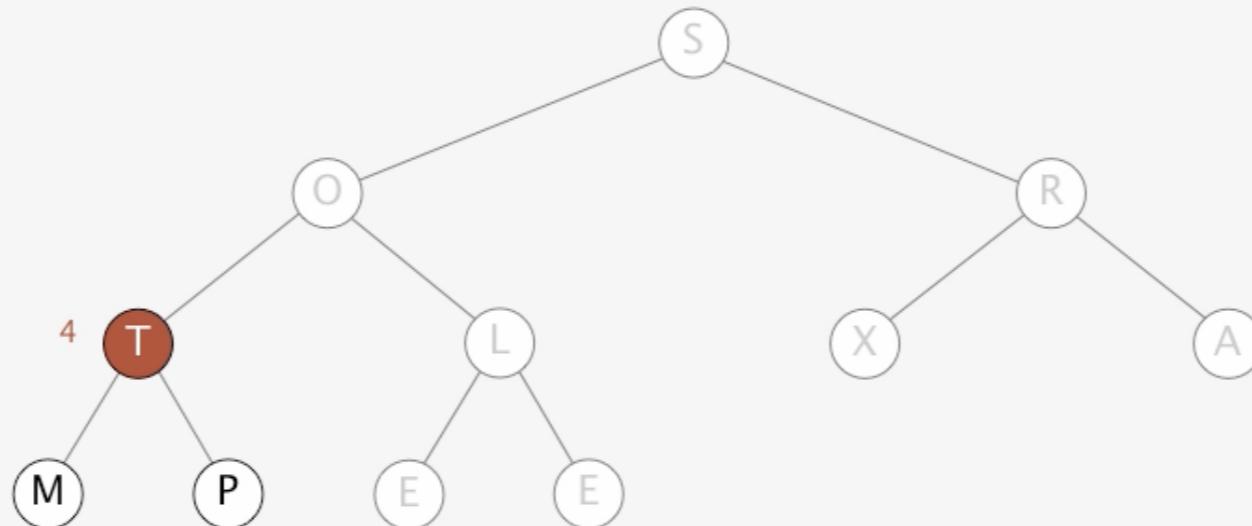
S	O	R	T	L	X	A	M	P	E	E
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 4

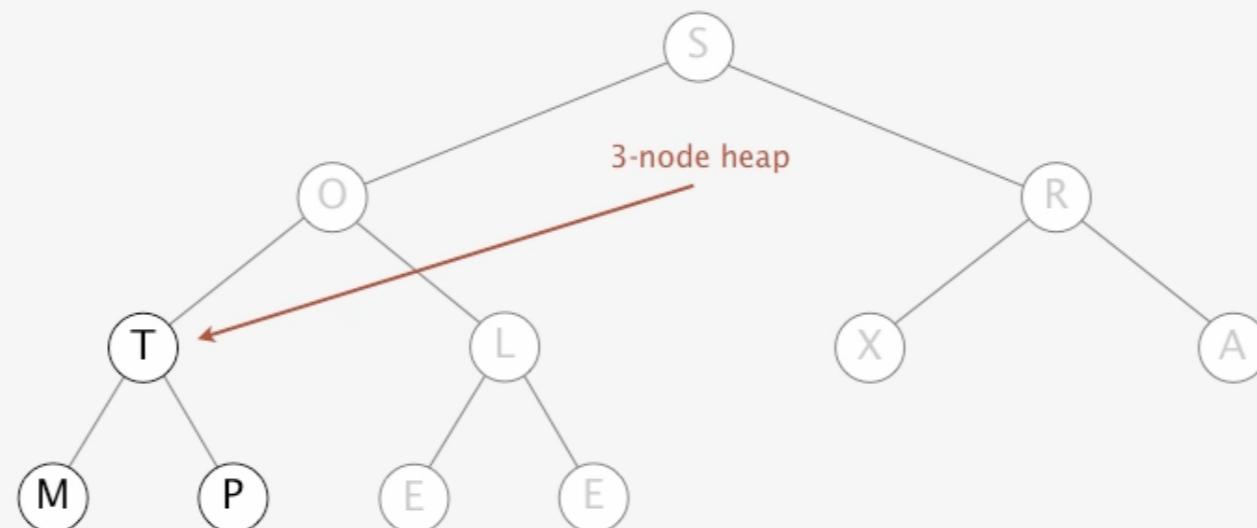


Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 4



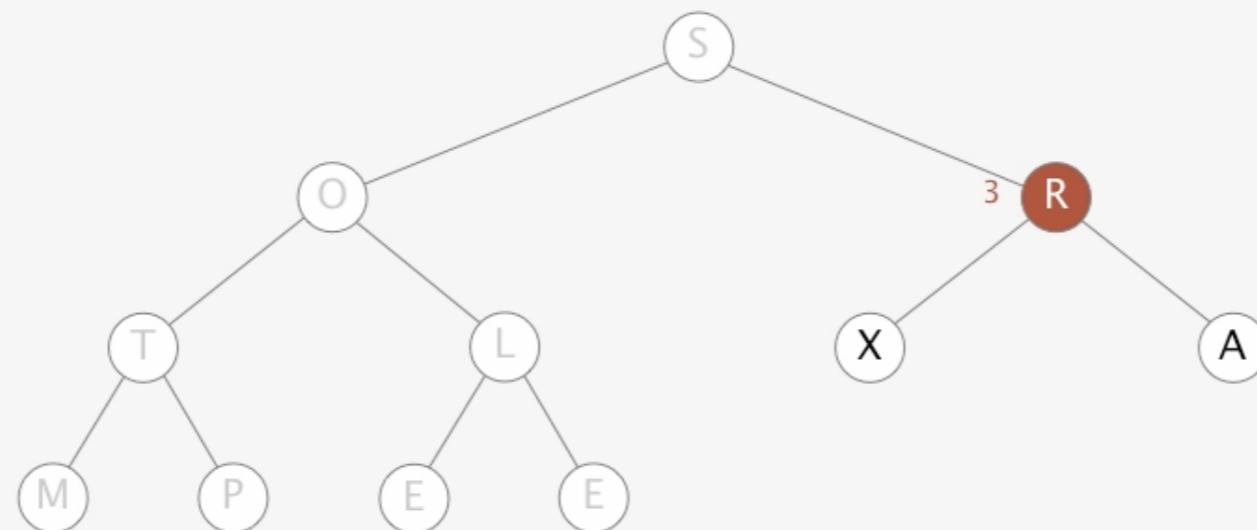
S	O	R	T	L	X	A	M	P	E	E
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 3

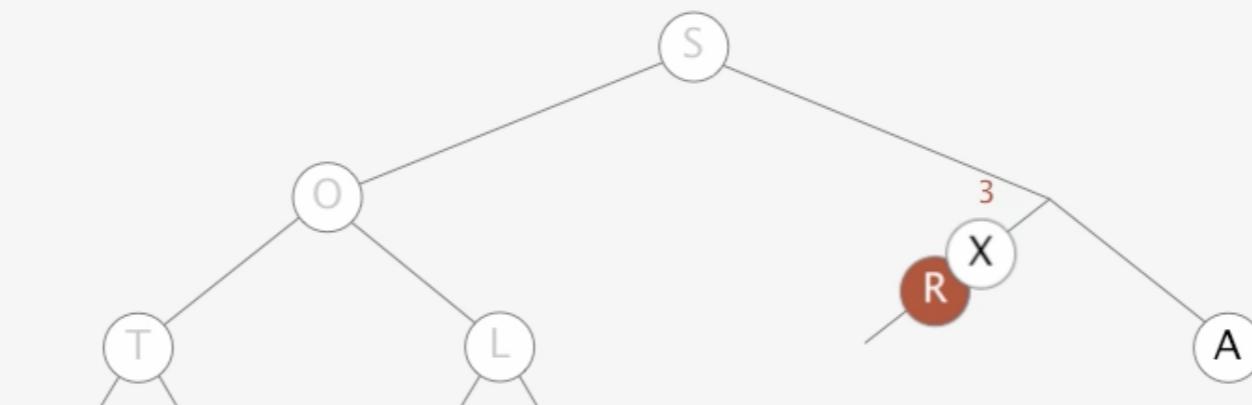


Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 3

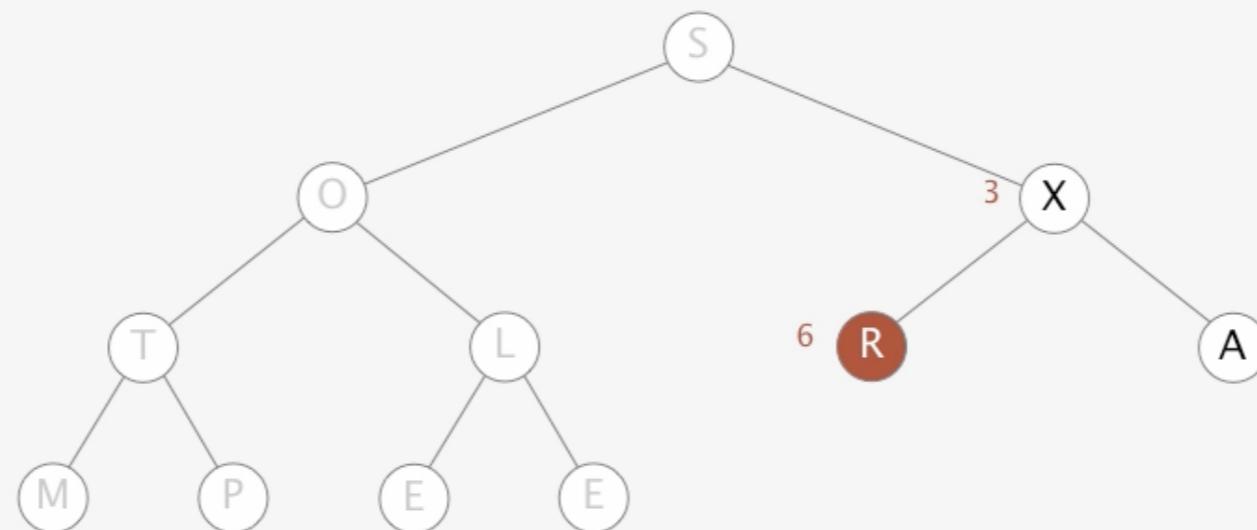


Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 3



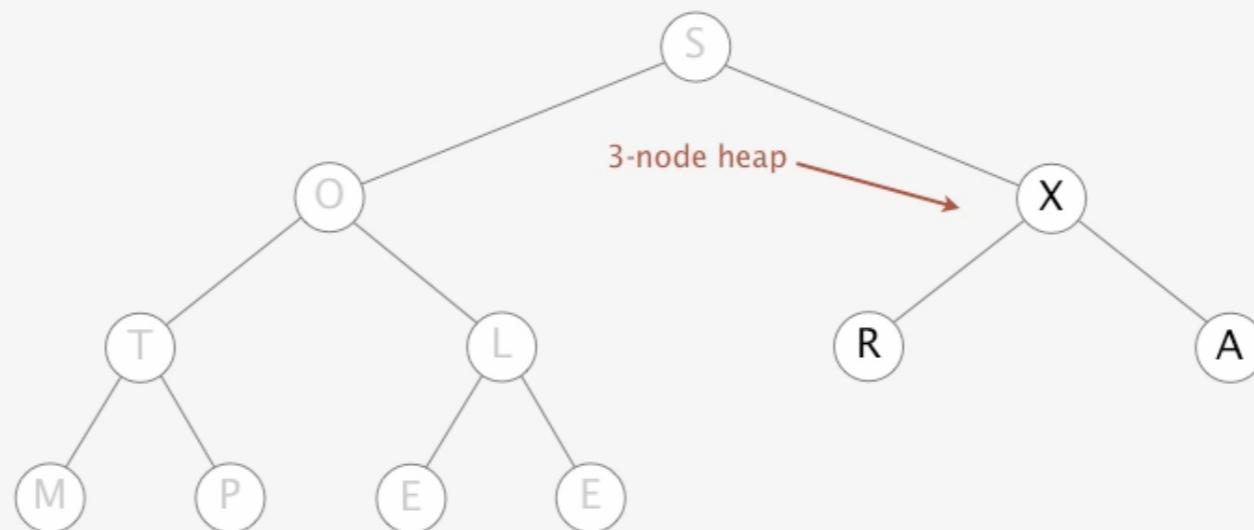
S	O	X	T	L	R	A	M	P	E	E
3					6					

Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 3



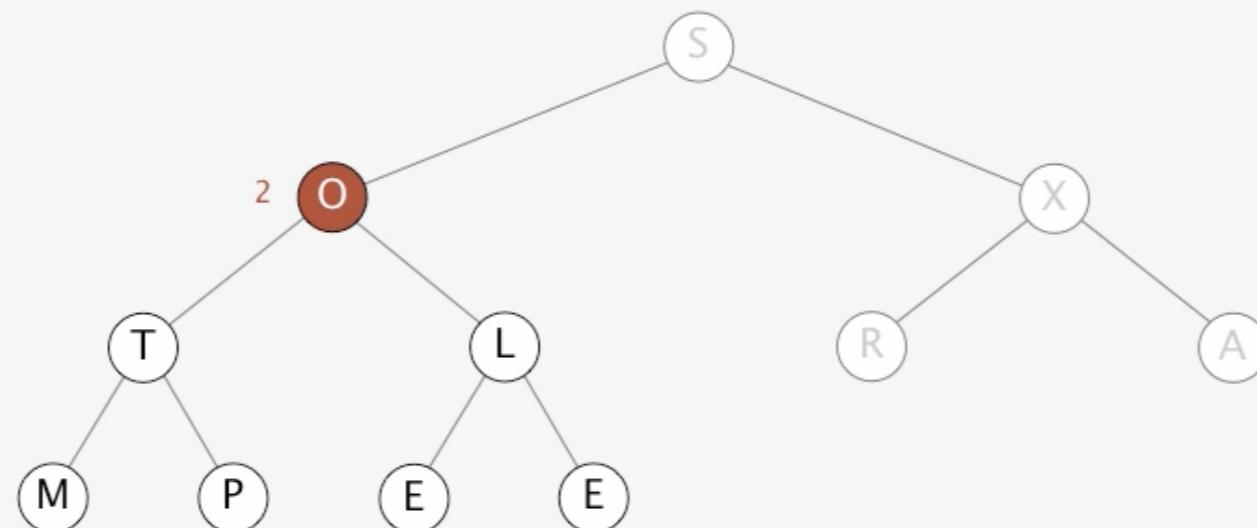
S	O	X	T	L	A	A	M	P	E	E
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 2



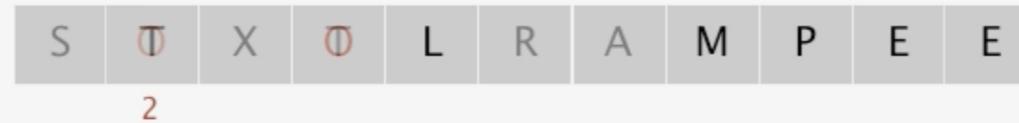
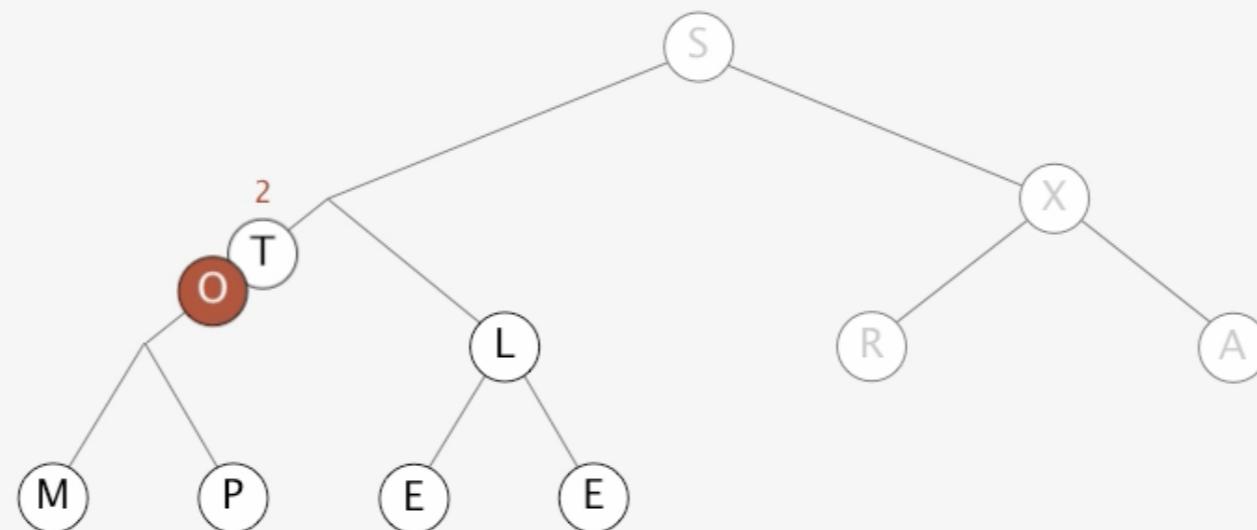
S O X T L R A M P E E
?

Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 2

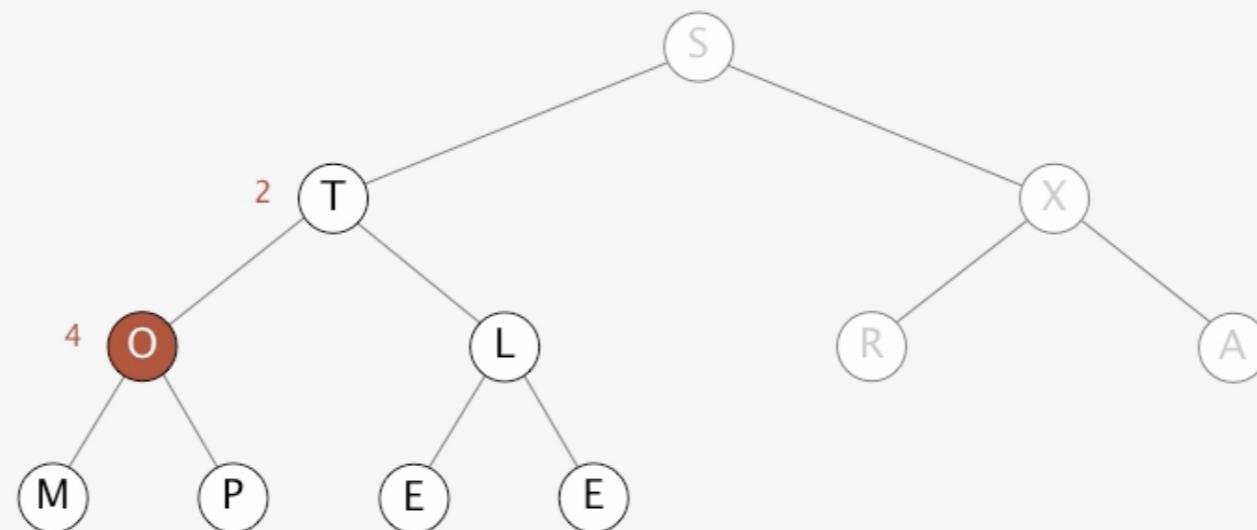


Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 2

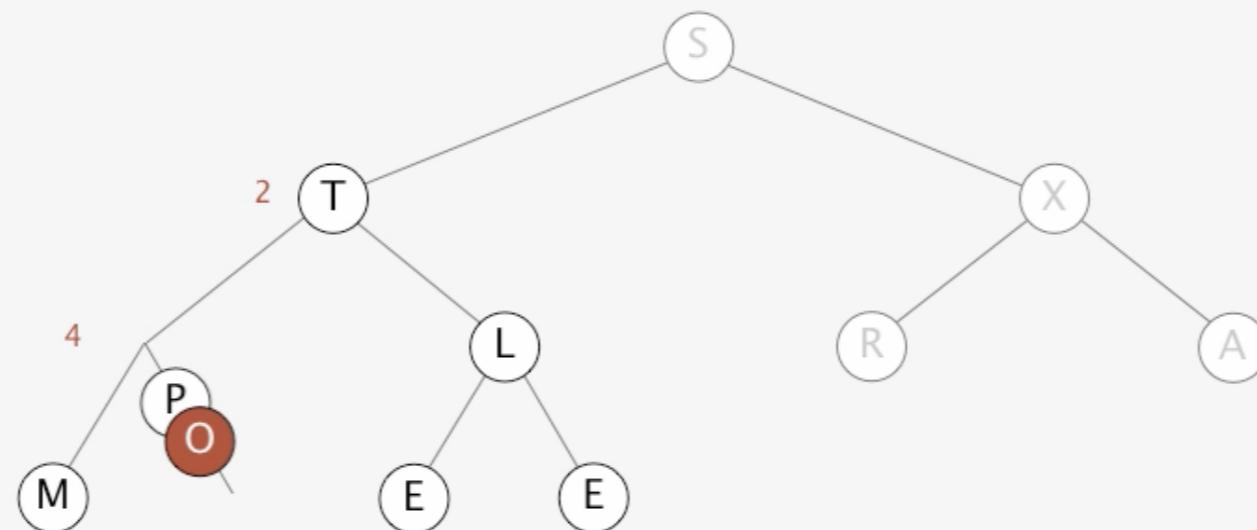


Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 2

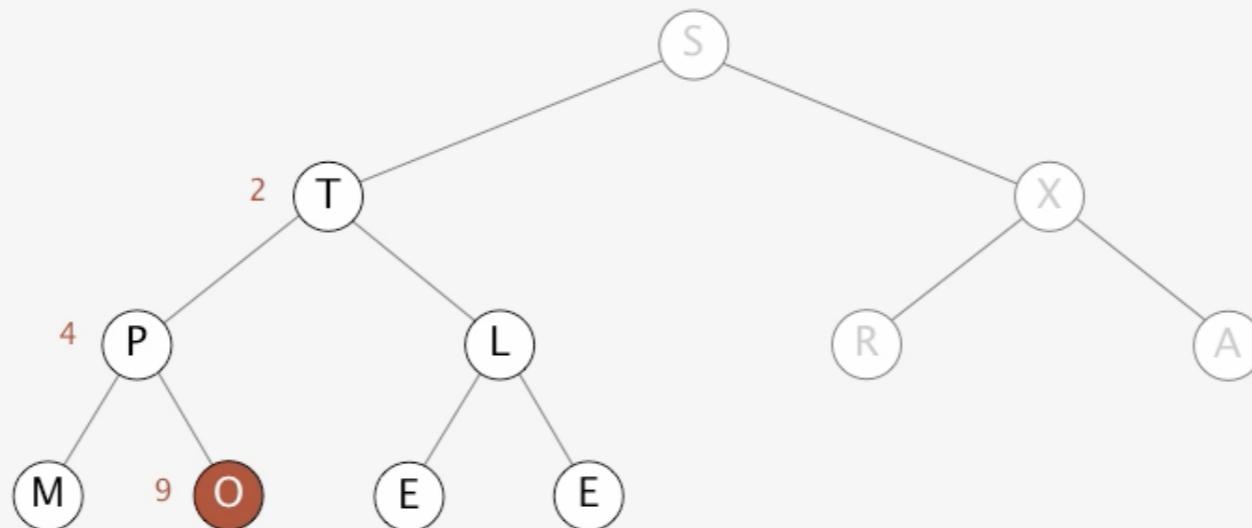


Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 2



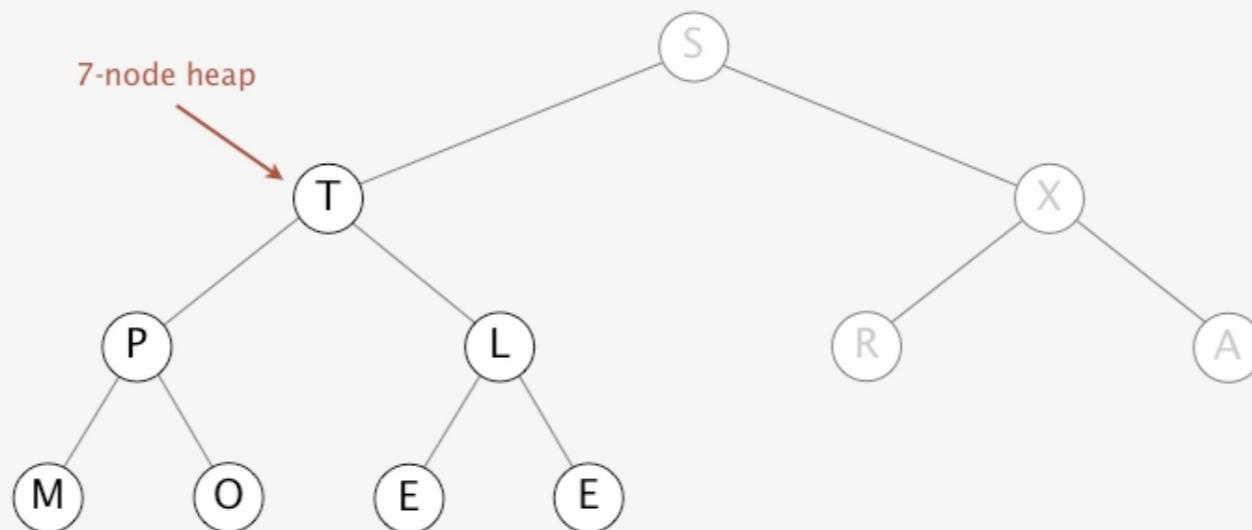
S	T	X	P	L	R	A	M	O	E	E
2	4						9			

Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 2



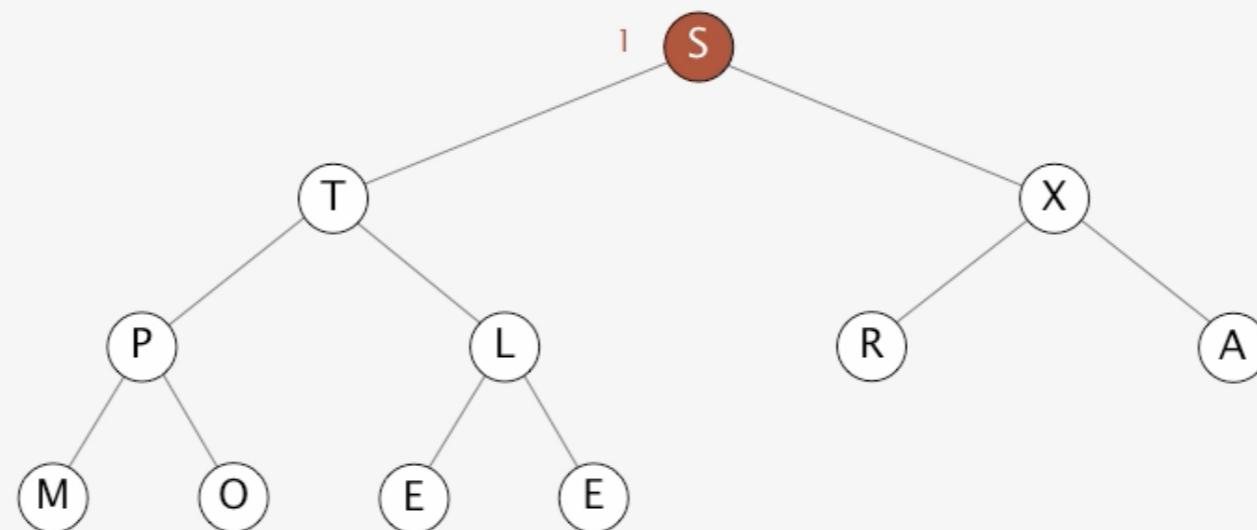
S	T	X	P	L	R	A	M	O	E	E
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 1

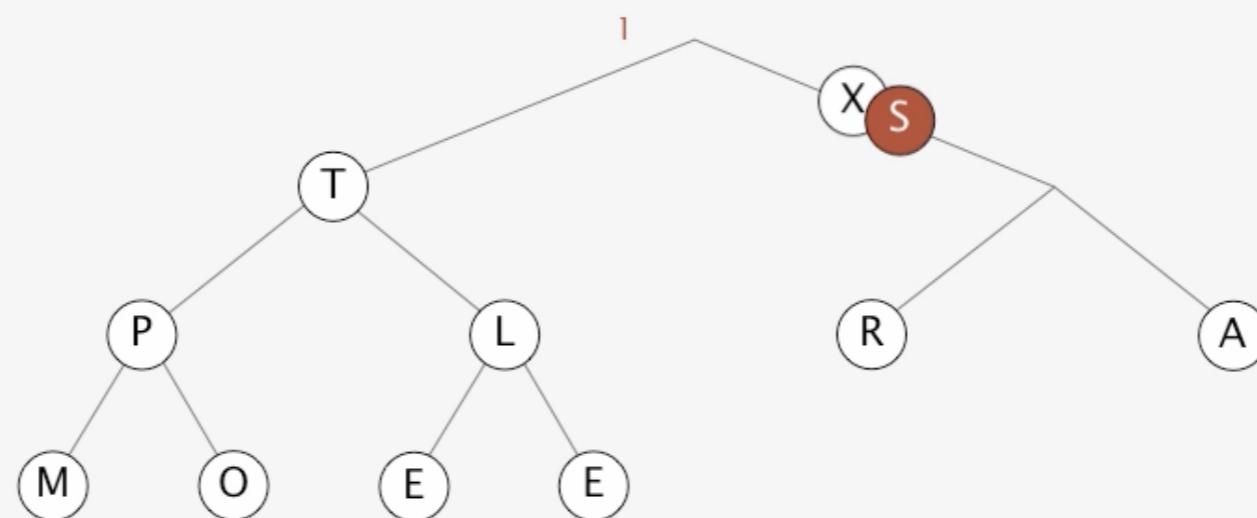


Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 1

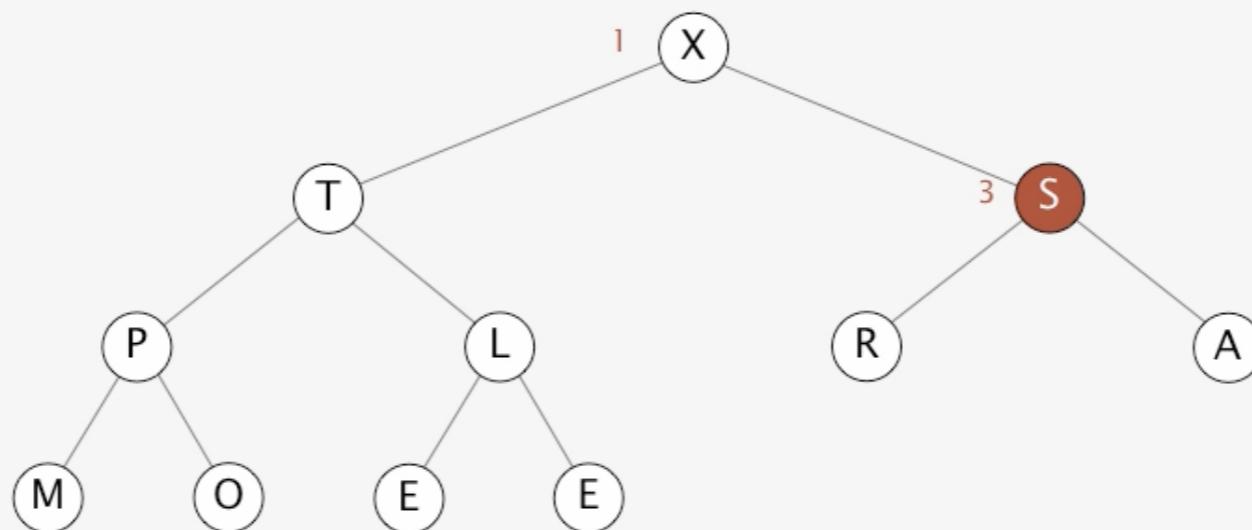


Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 1

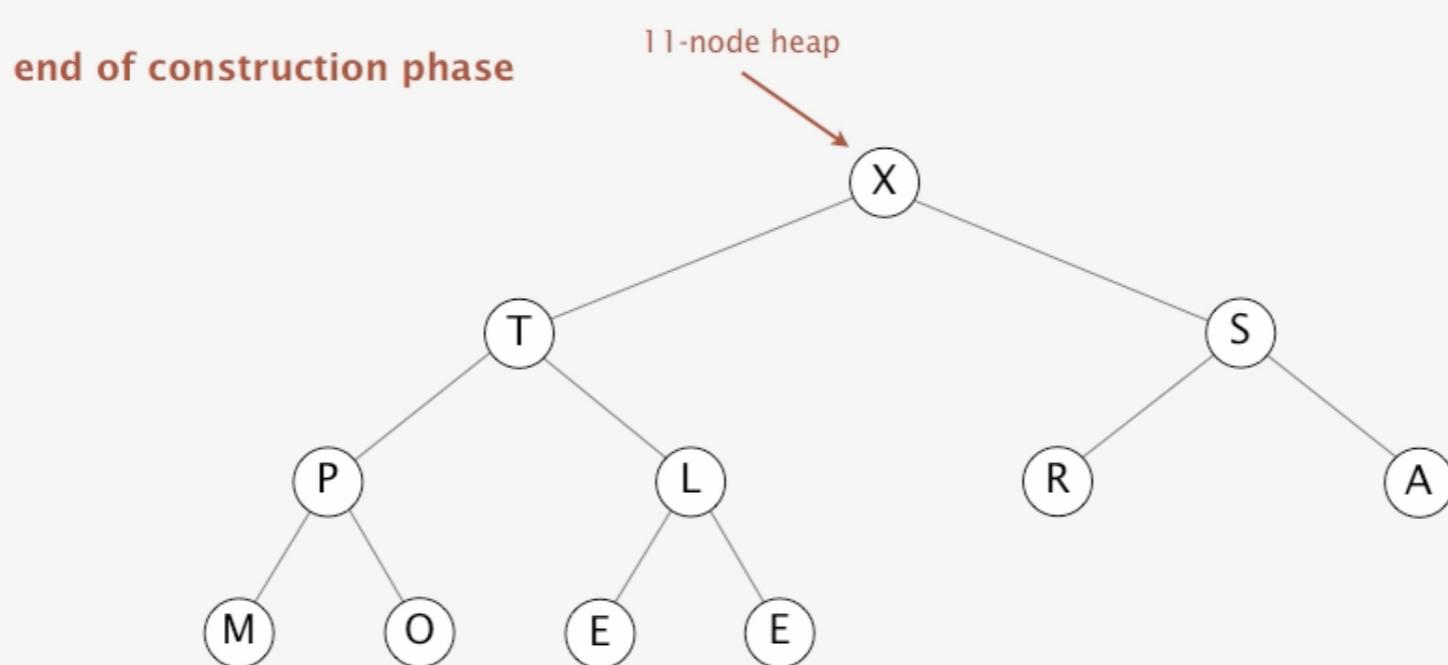


X	T	S	P	L	R	A	M	O	E	E
1		3								

Heapsort

Heapsort demo

Heap construction. Build max heap using bottom-up method.



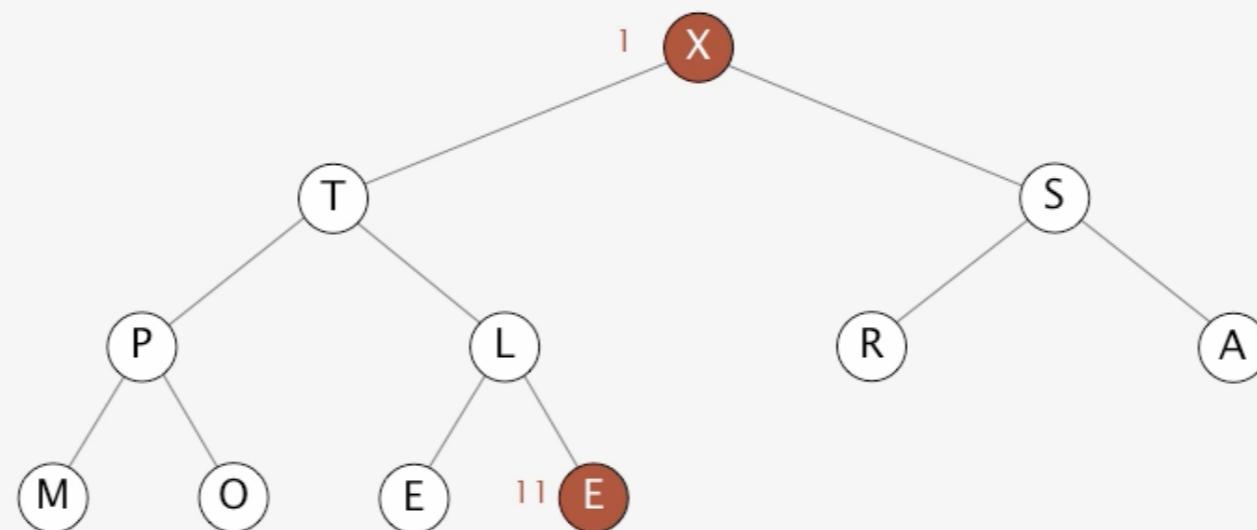
X	T	S	P	L	R	A	M	O	E	E
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 11

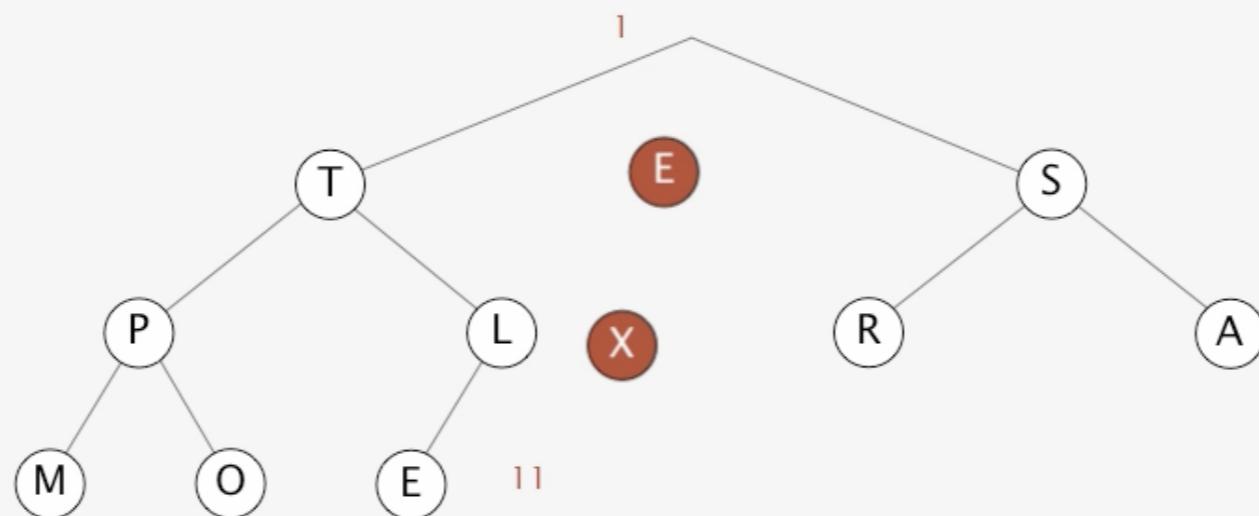


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 11



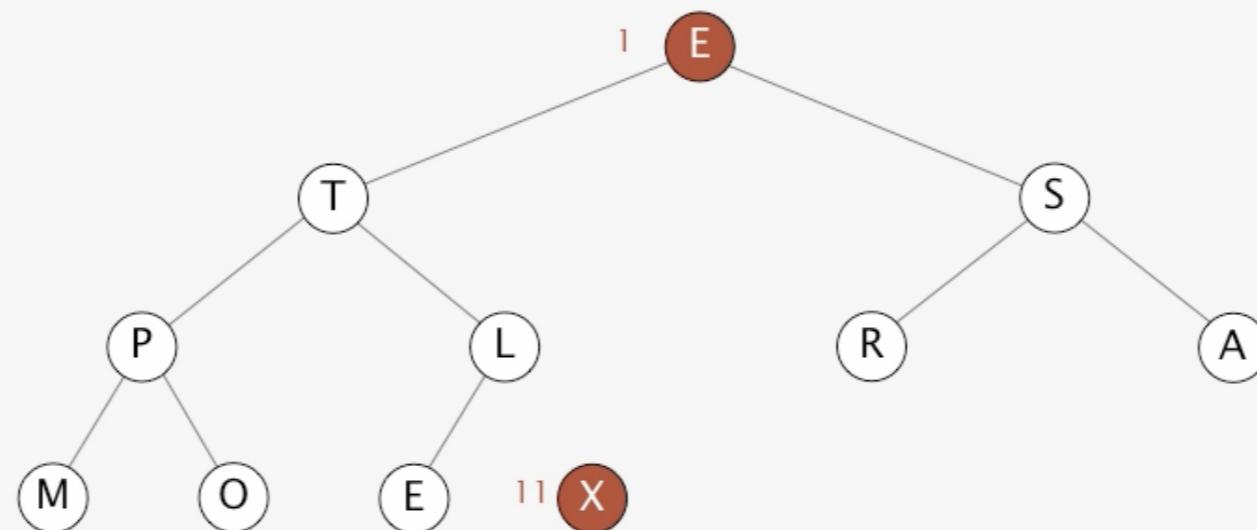
E	T	S	P	L	R	A	M	O	E	X
1				11						11

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 11

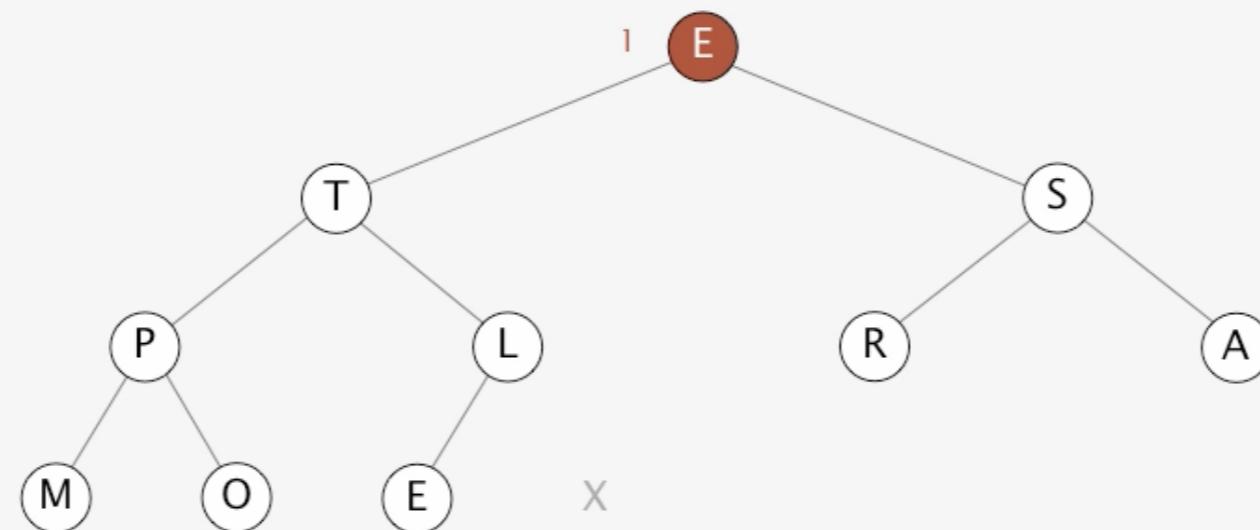


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

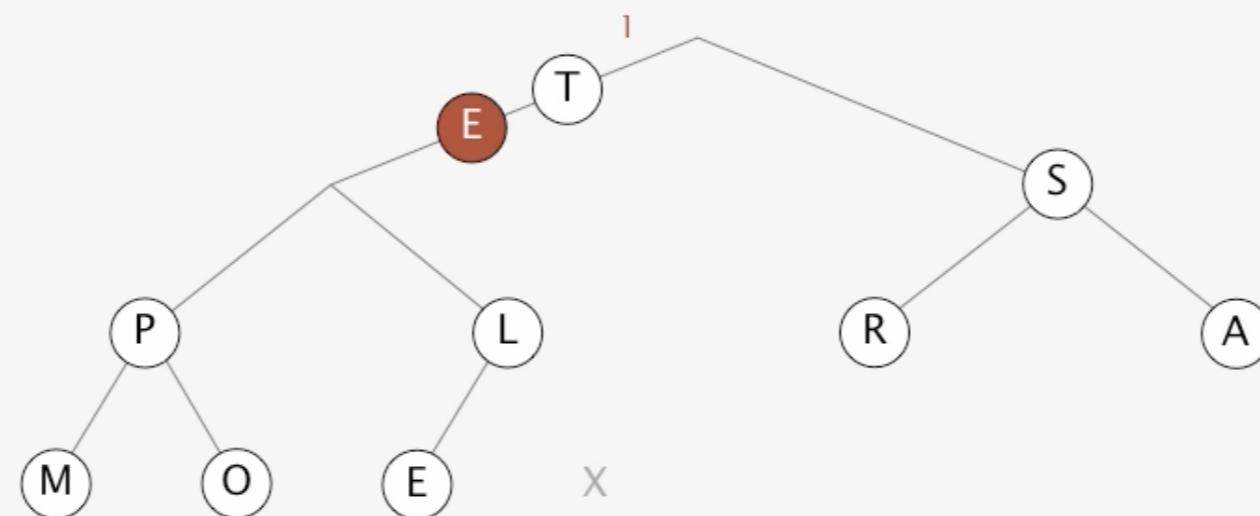


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

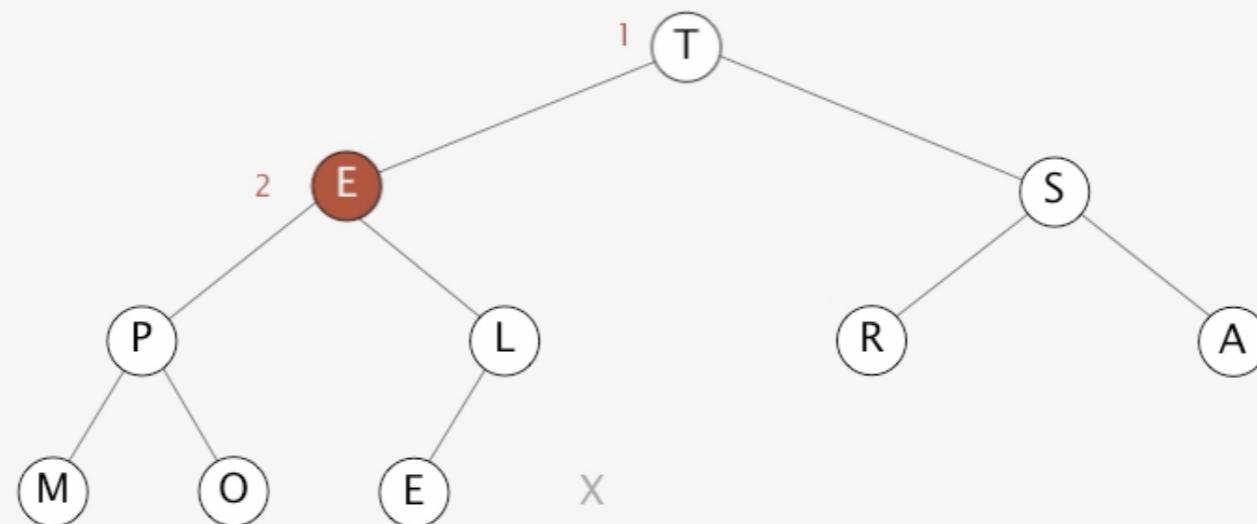


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

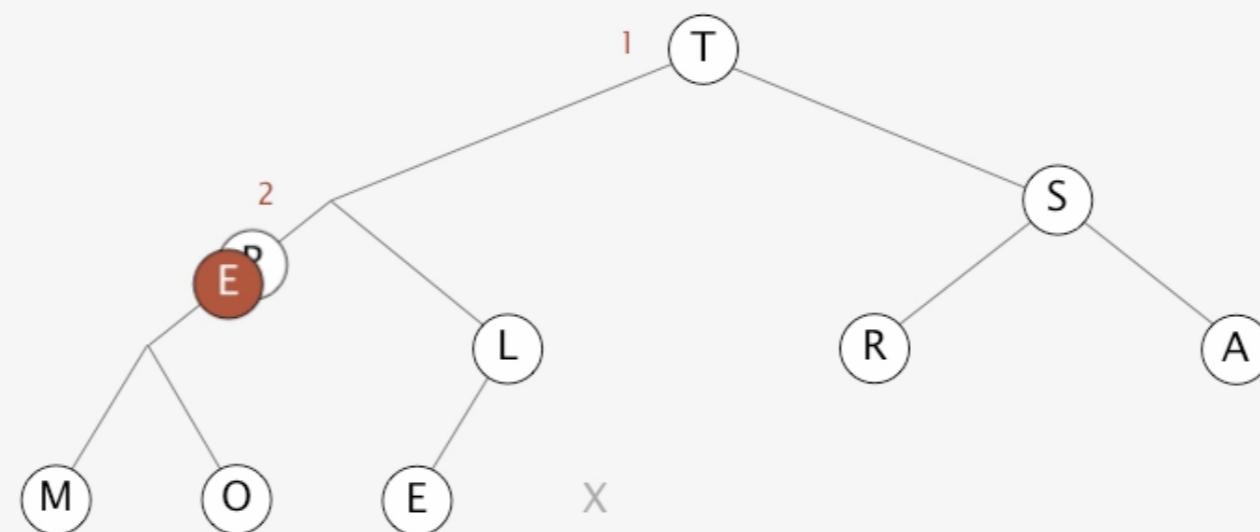


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

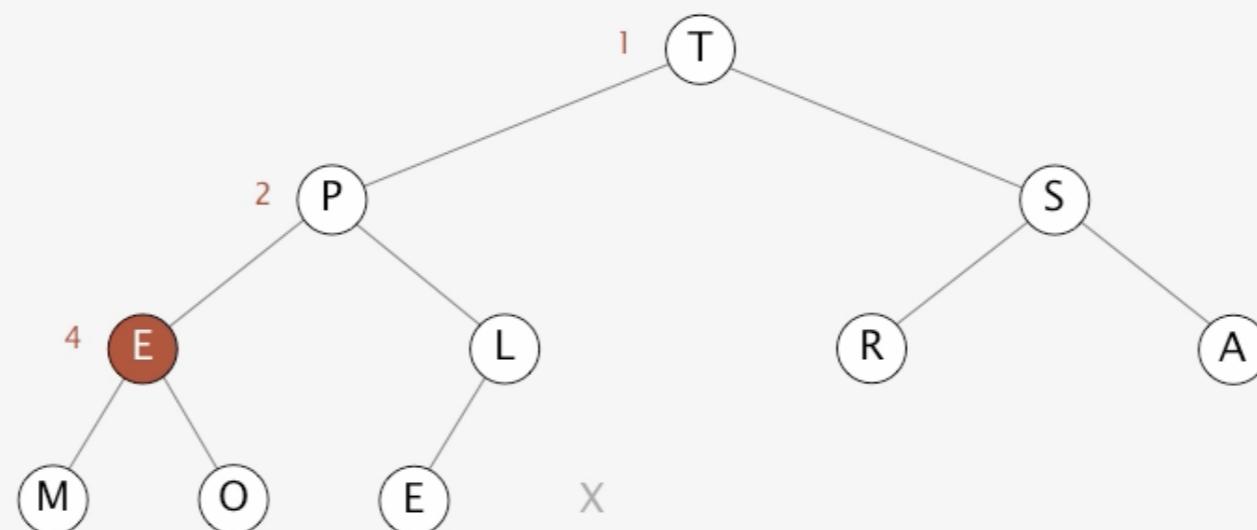


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



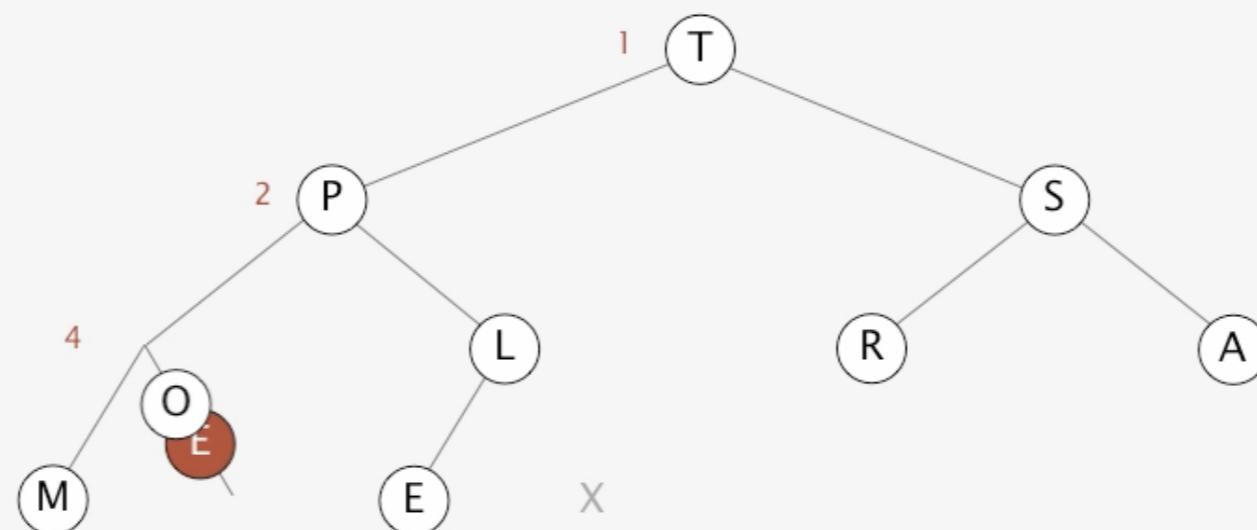
T	P	S	E	L	R	A	M	O	E	X
1	2	4								

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



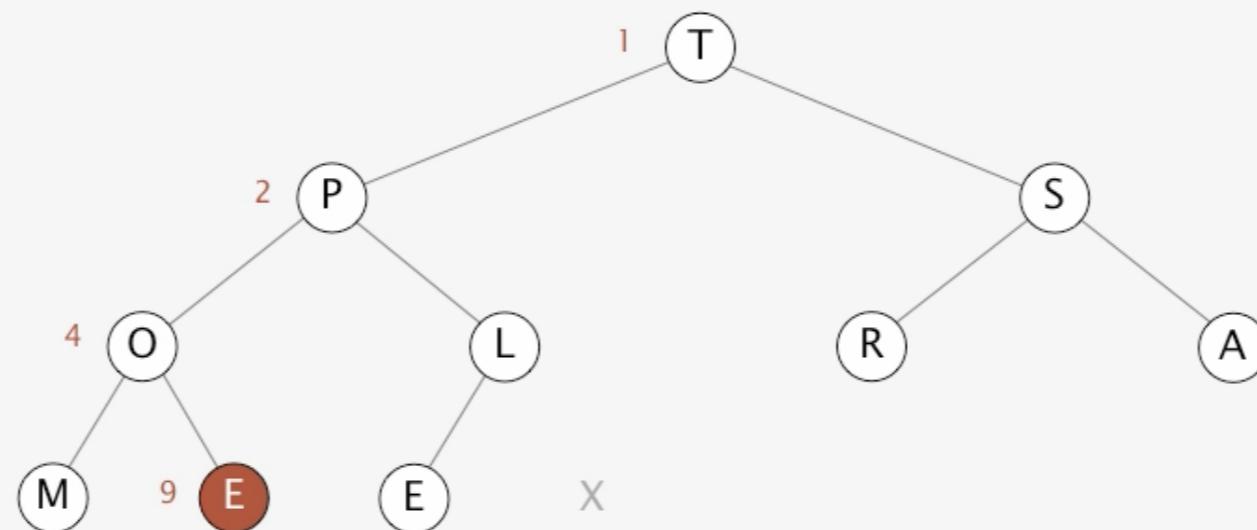
T	P	S	O	L	R	A	M	E	E	X
1	2	4	1	2	3	4	5	6	7	8

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

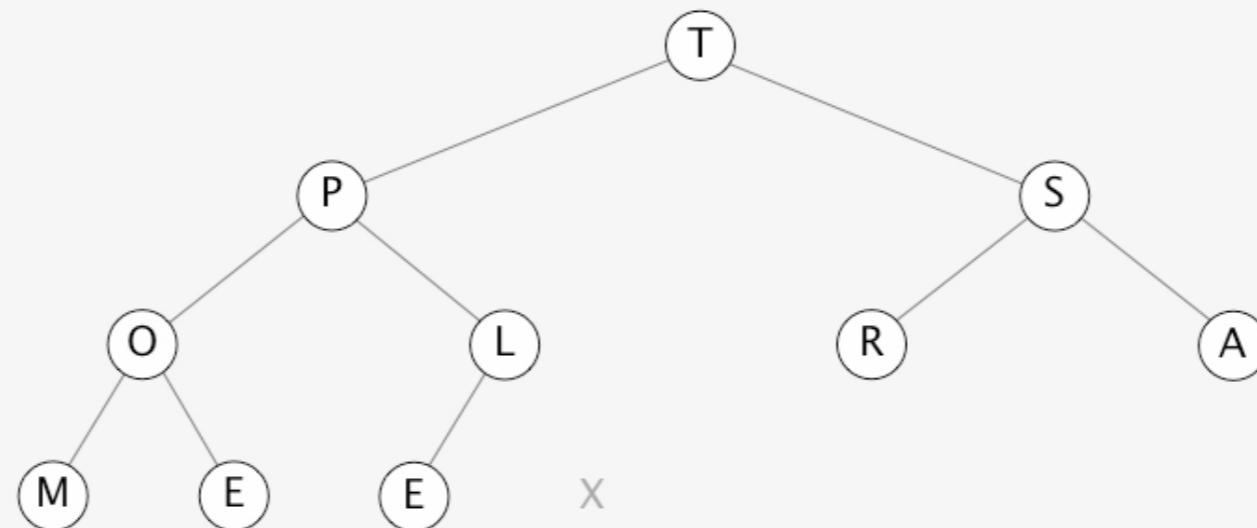


T	P	S	O	L	R	A	M	E	X
1	2	4					9		

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.



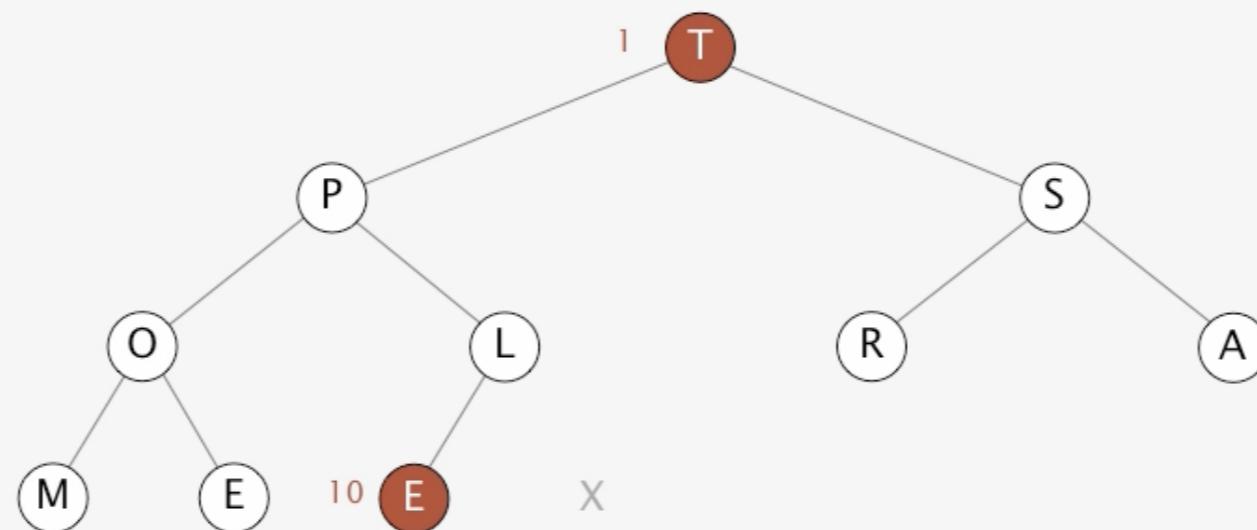
T	P	S	O	L	R	A	M	E	E	X
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 10

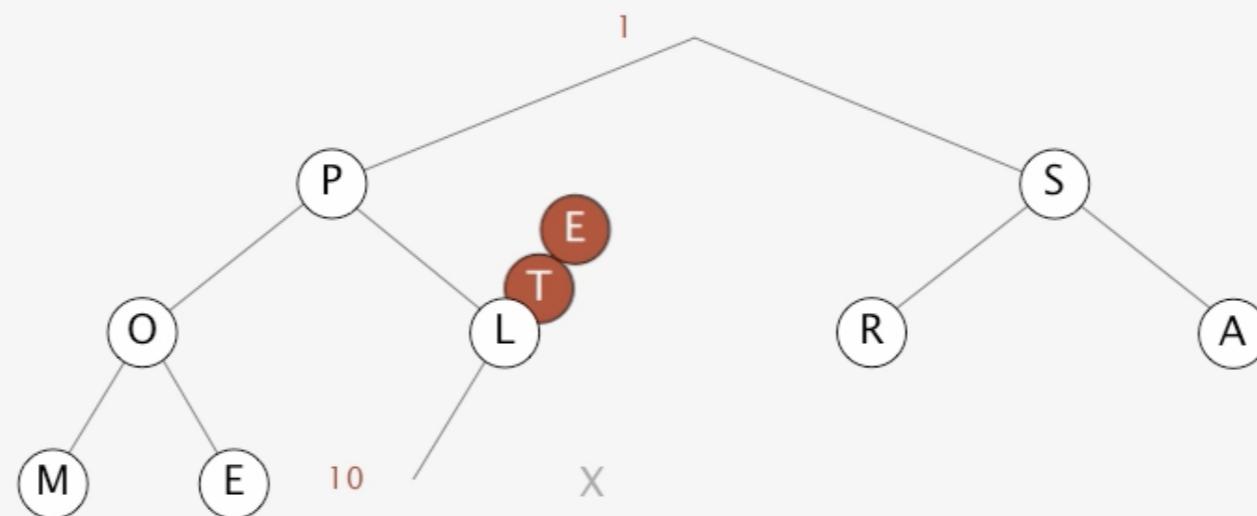


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 10

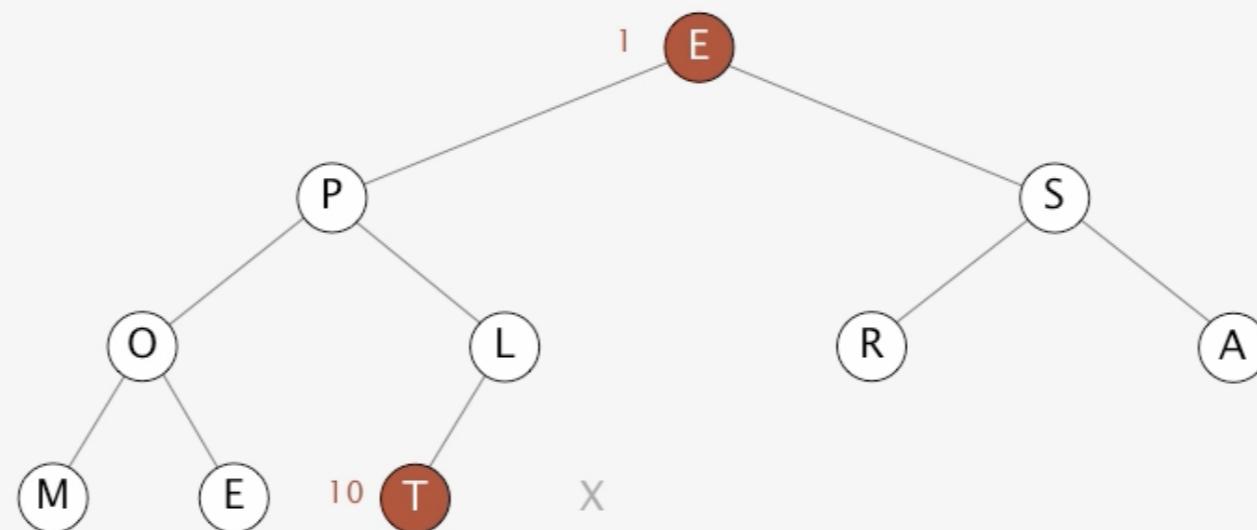


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 10

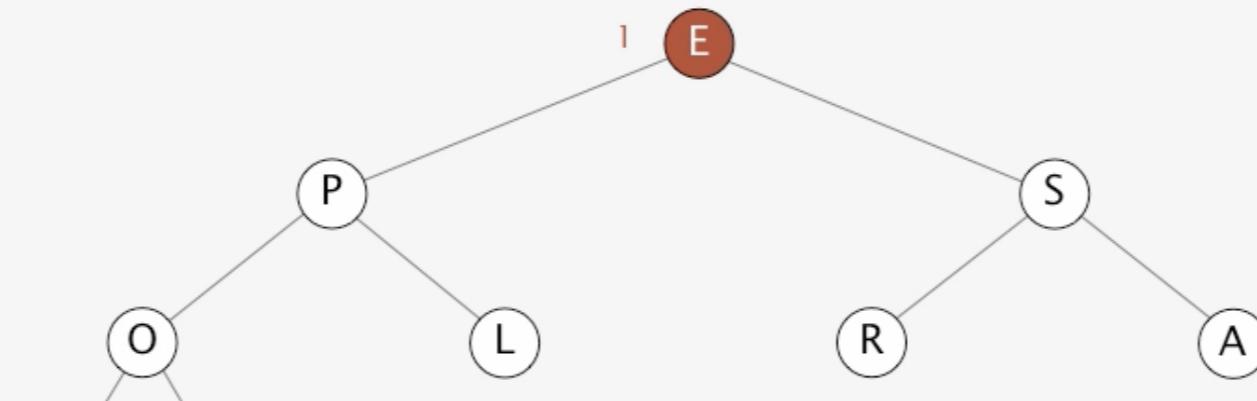


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

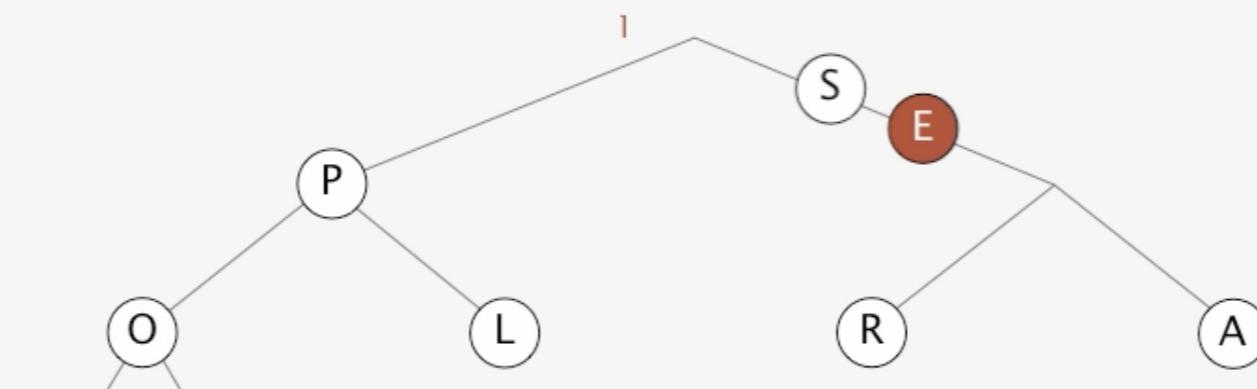


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

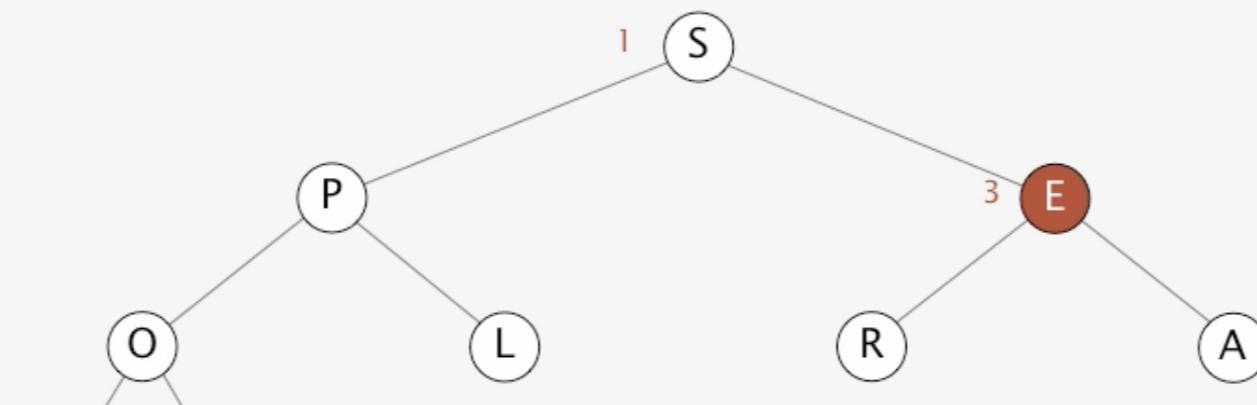


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



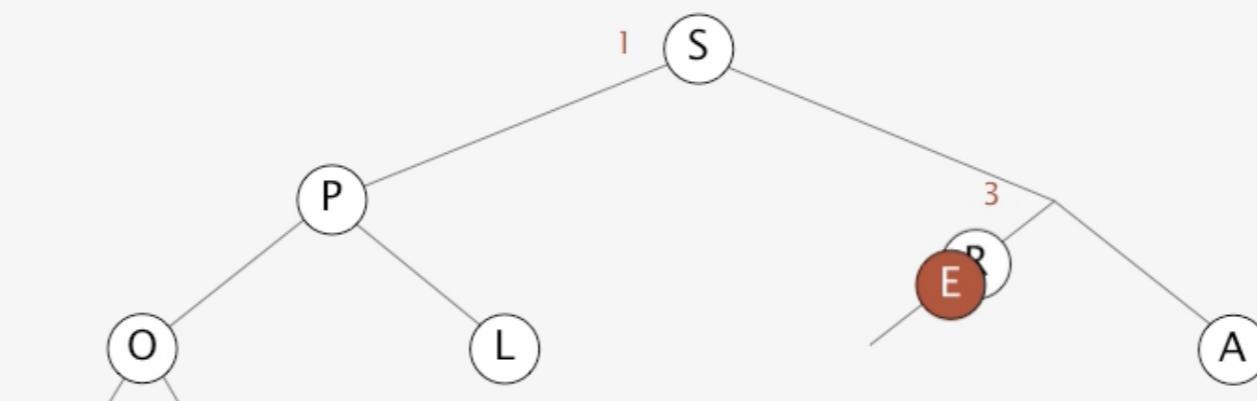
S	P	E	O	L	R	A	M	E	T	X
1		3								

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



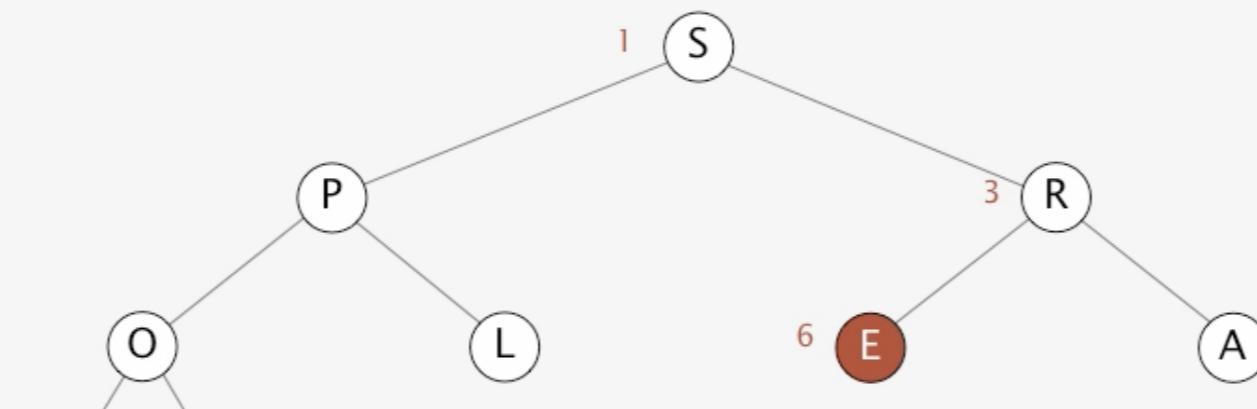
S	P	R	O	L	R	A	M	E	T	X
1		3								

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

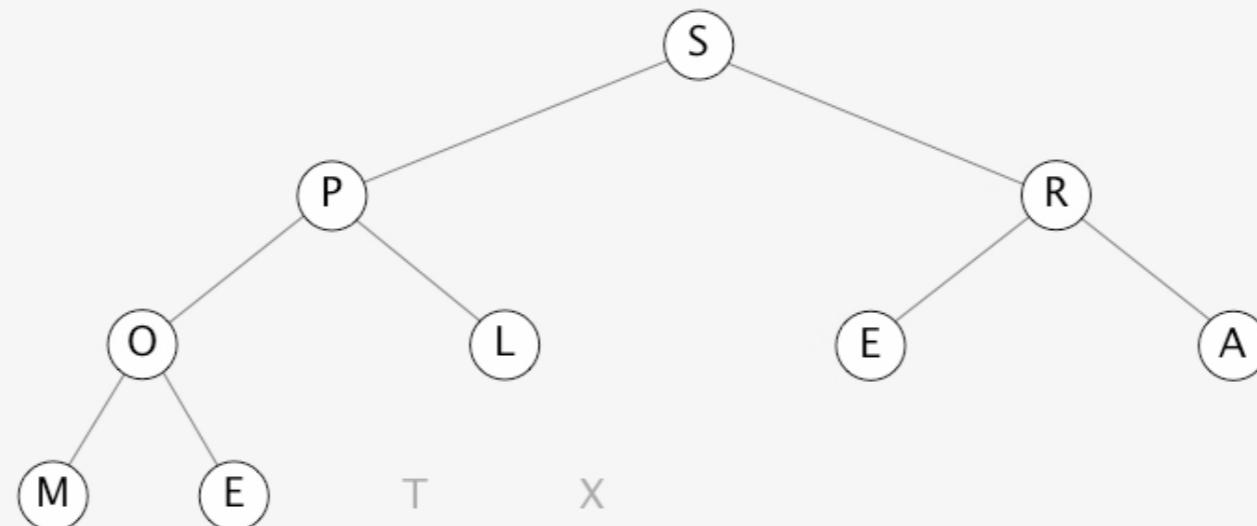


S	P	R	O	L	E	A	M	E	T	X
1	2	3	4	5	6	9	7	8		

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.



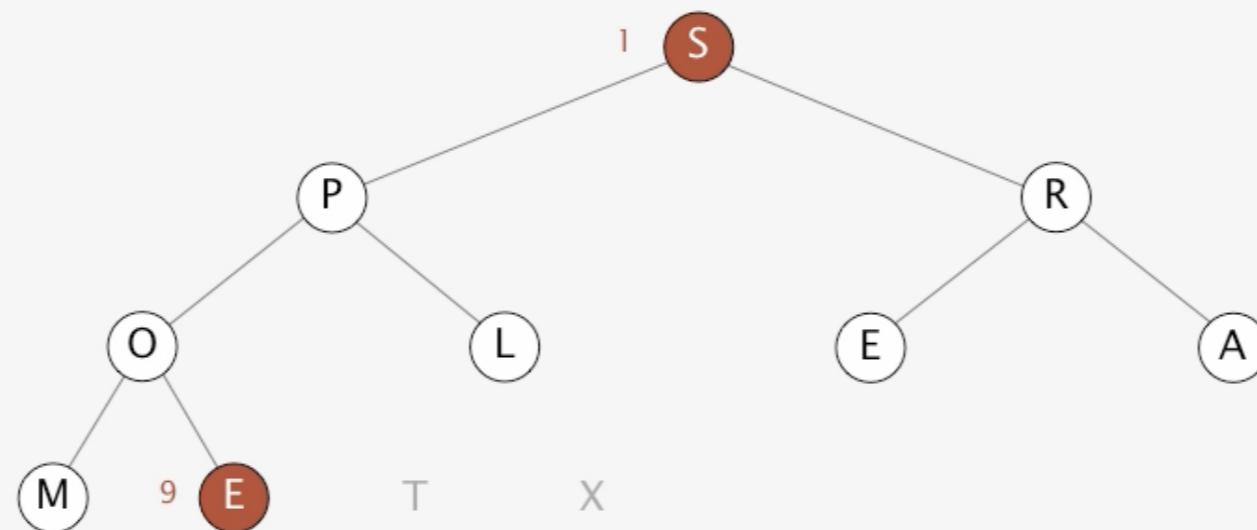
S	P	R	O	L	E	A	M	E	T	X
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 9



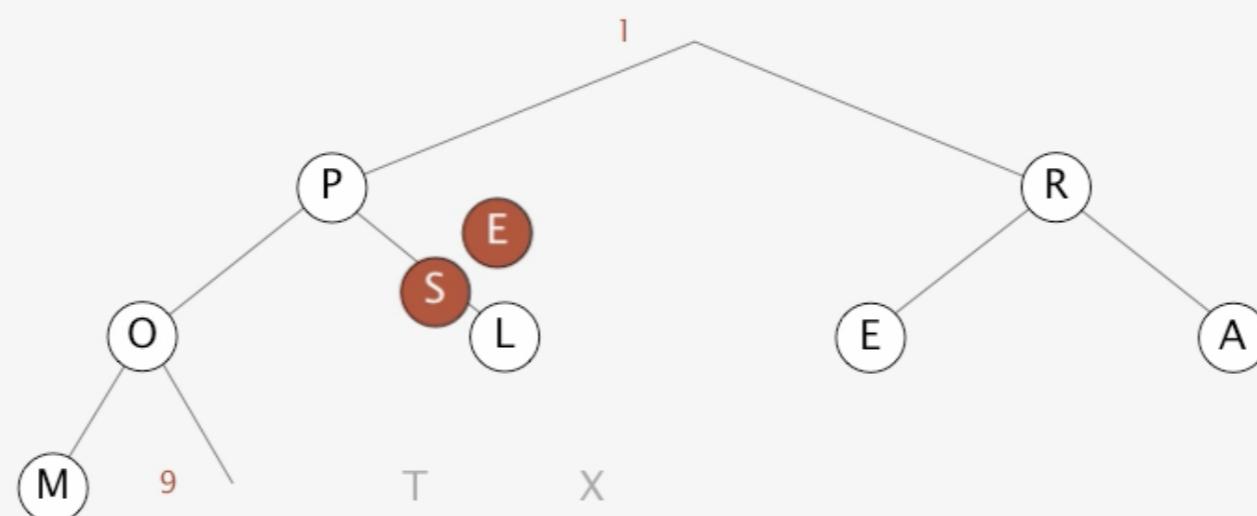
S	P	R	O	L	E	A	M	E	T	X
1					9					

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 9

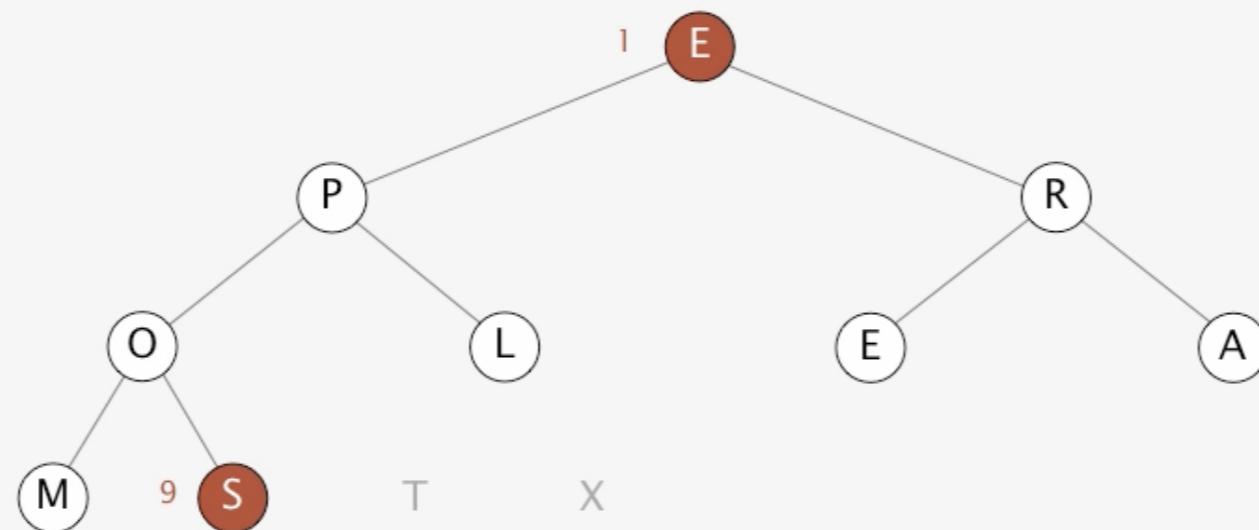


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 9

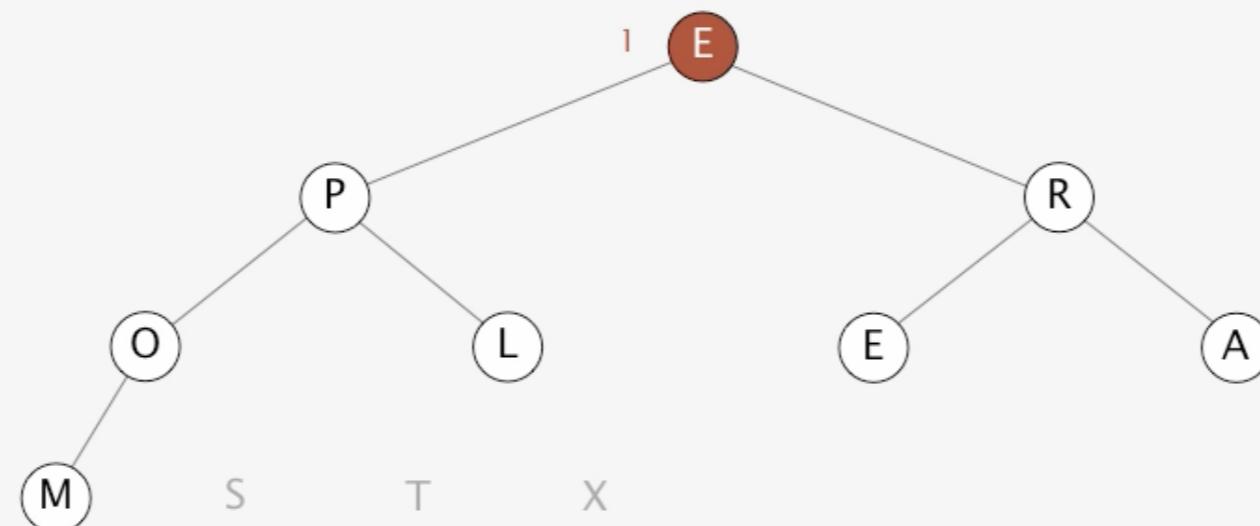


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

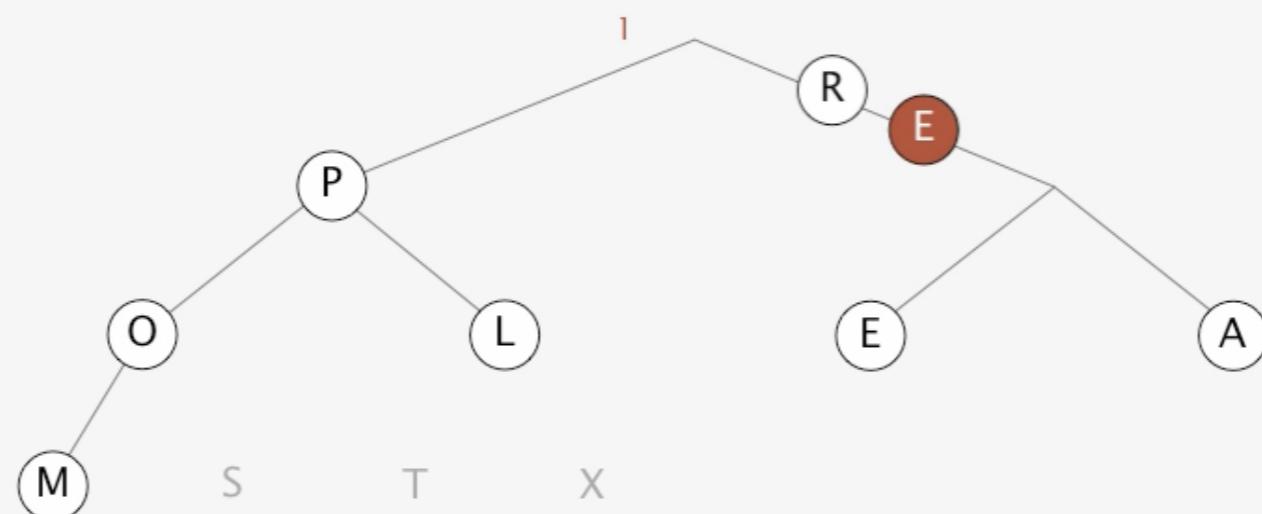


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

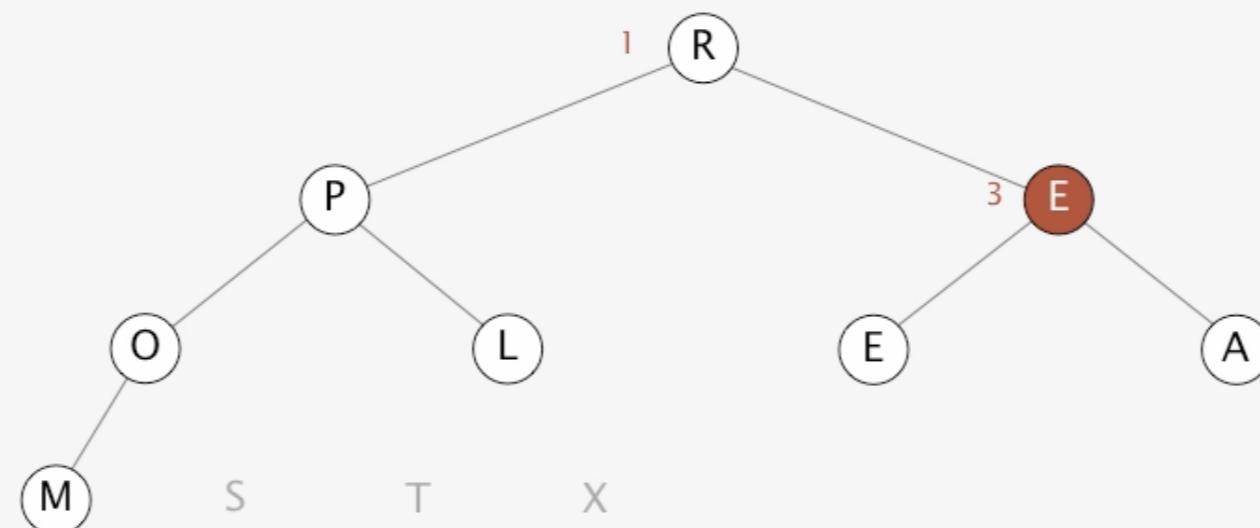


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

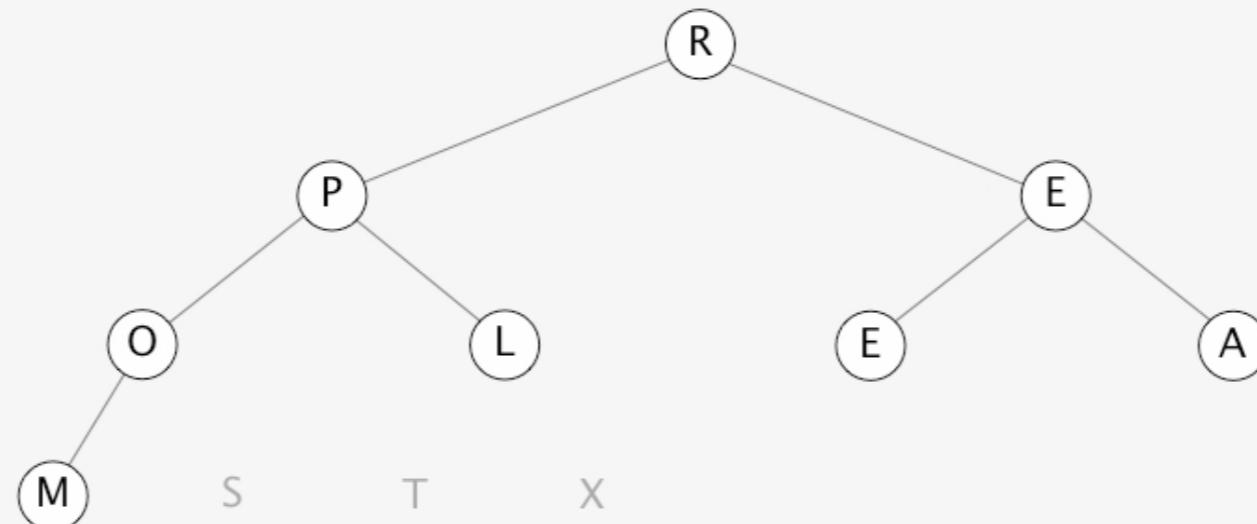
sink 1



Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.



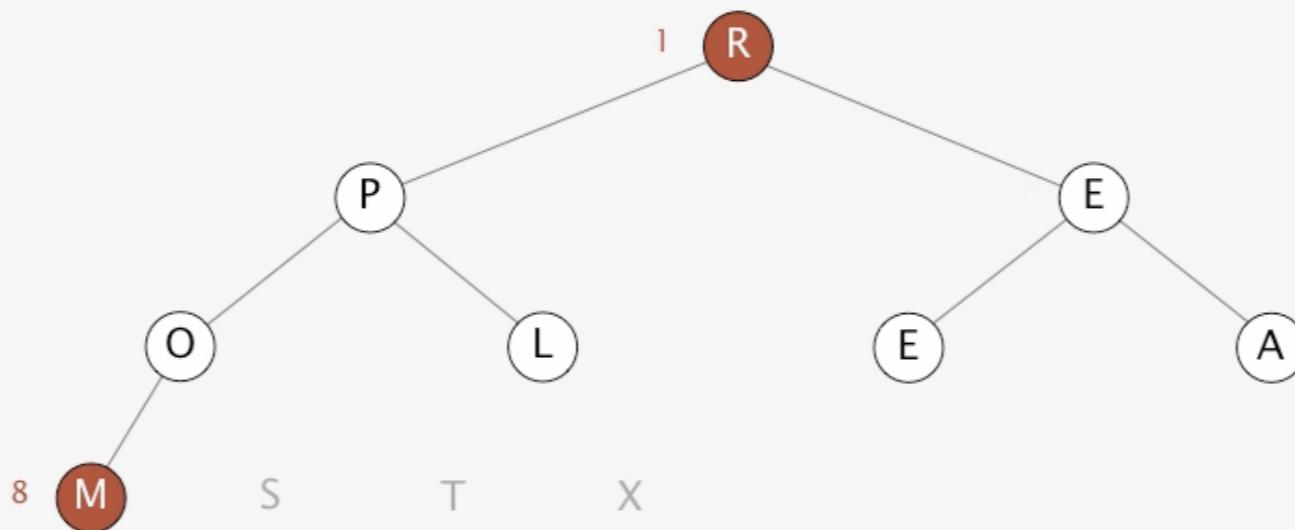
R	P	E	O	L	E	A	M	S	T	X
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 8



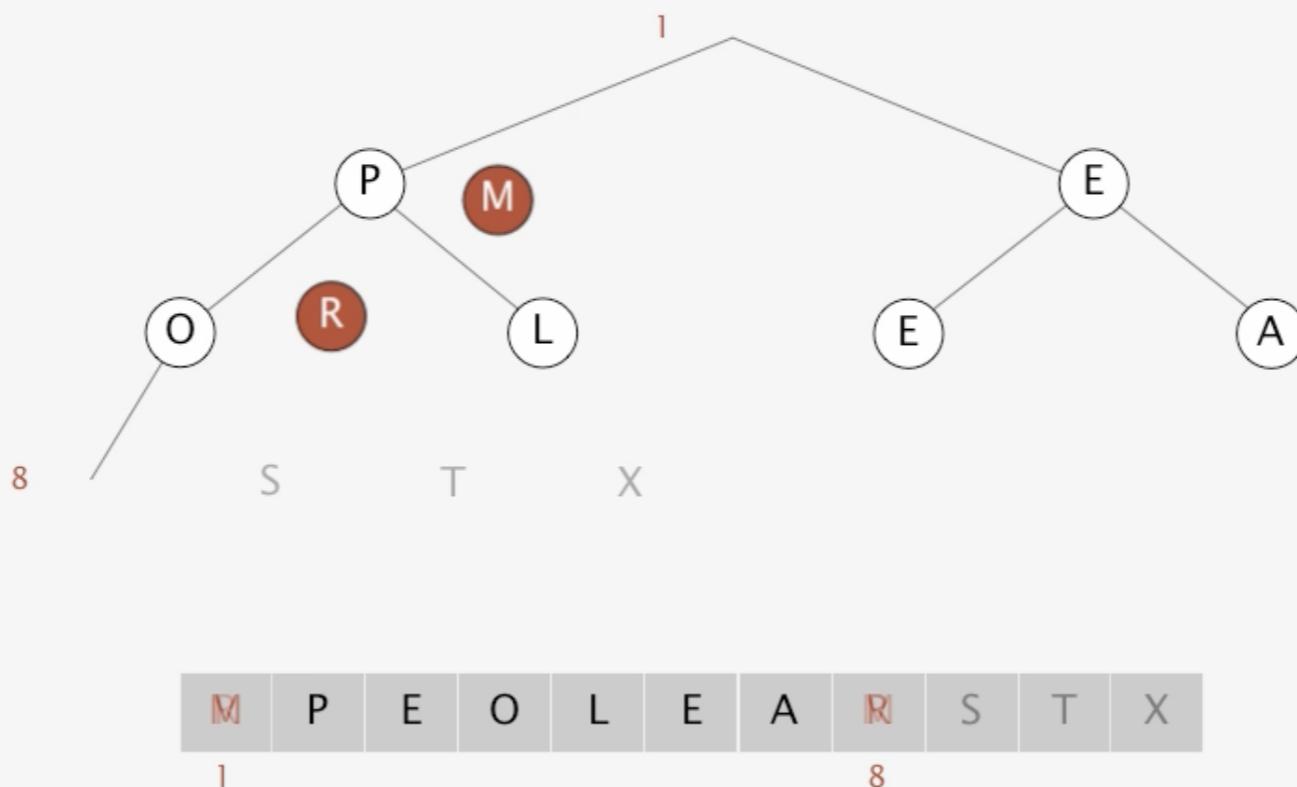
R	P	E	O	L	E	A	M	S	T	X
1							8			

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 8

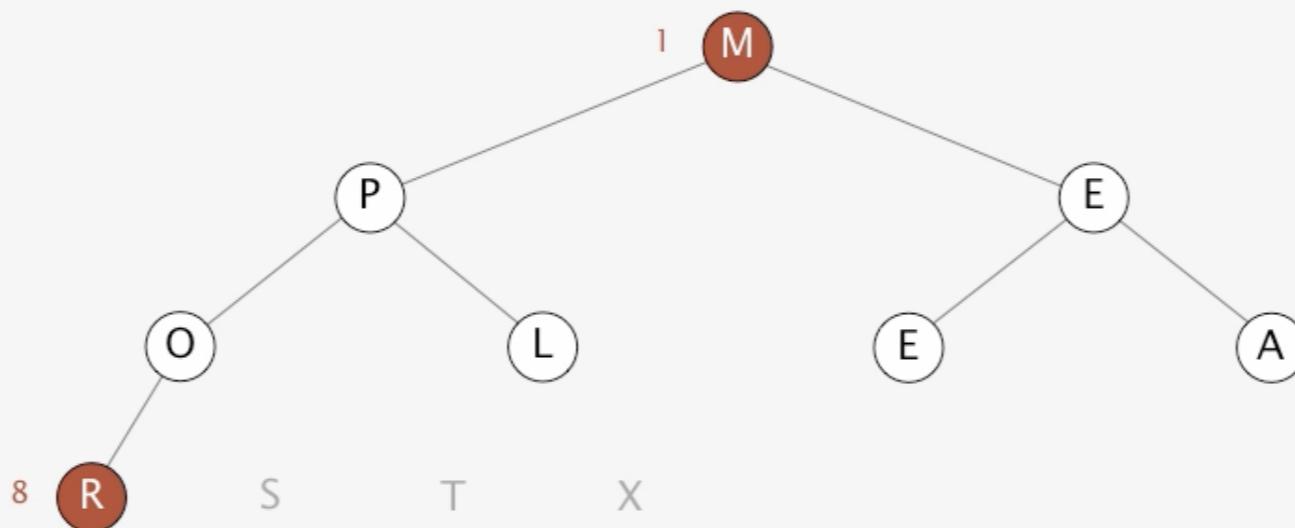


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 8

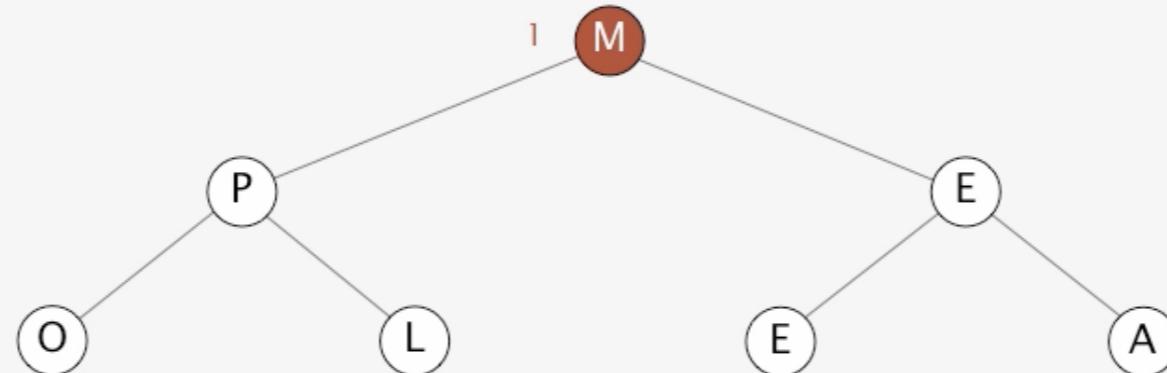


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

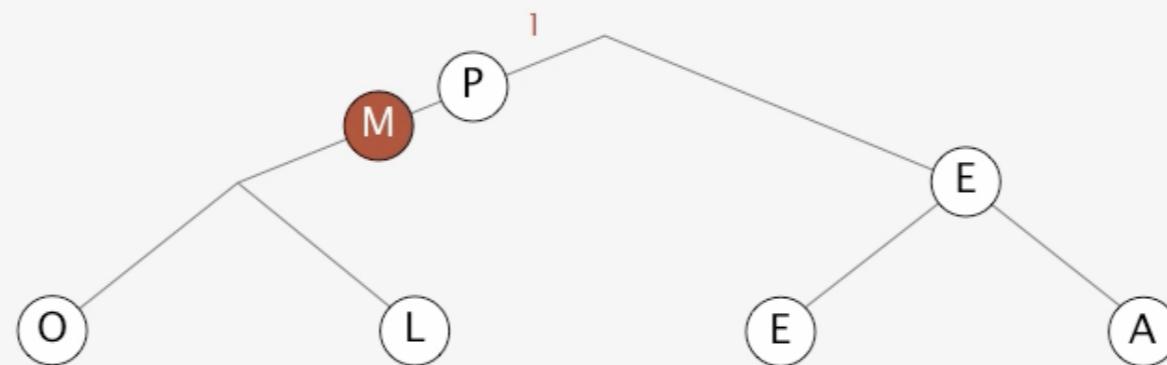


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

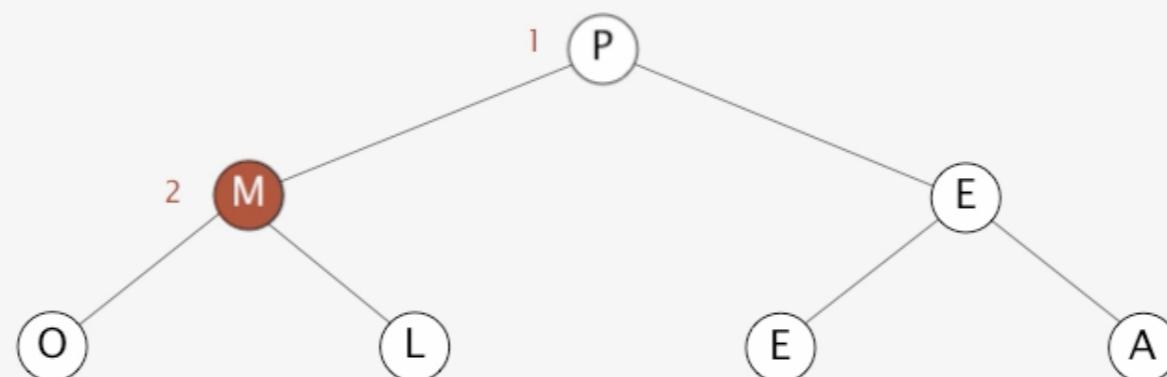
P | M | E | O | L | E | A | R | S | T | X
1

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

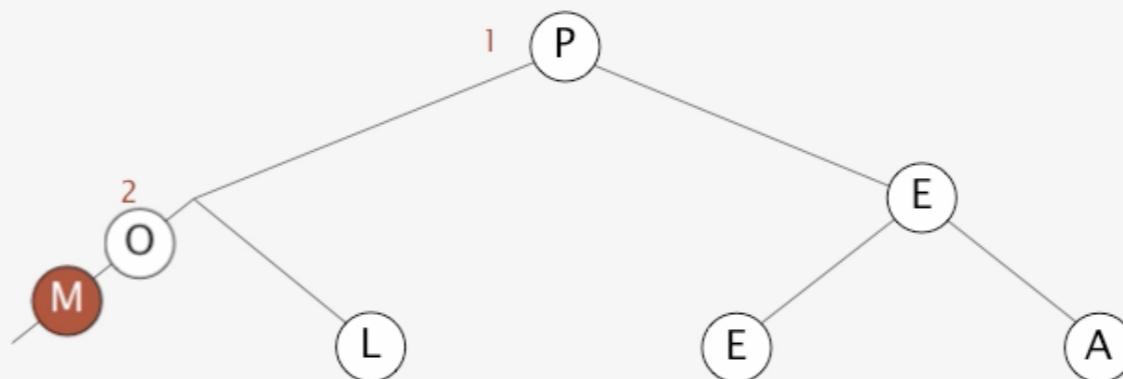


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

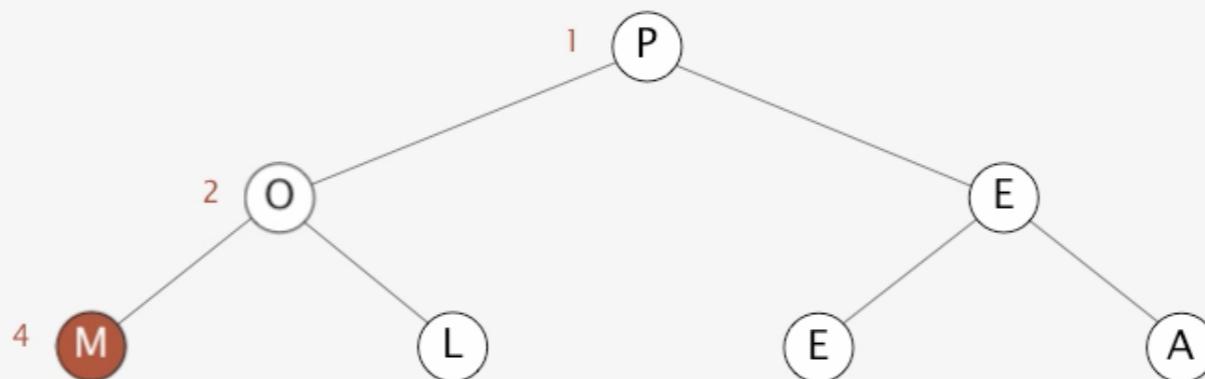


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



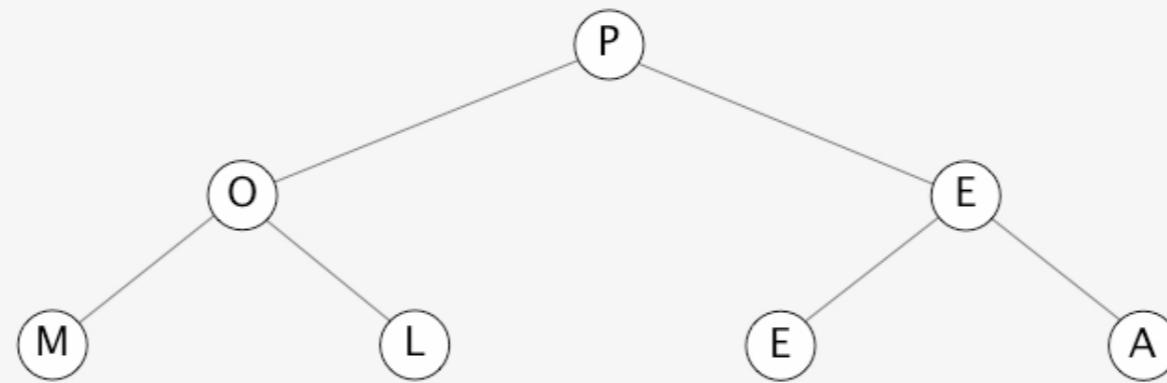
R S T X

P	O	E	M	L	E	A	R	S	T	X
1	2		4							

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.



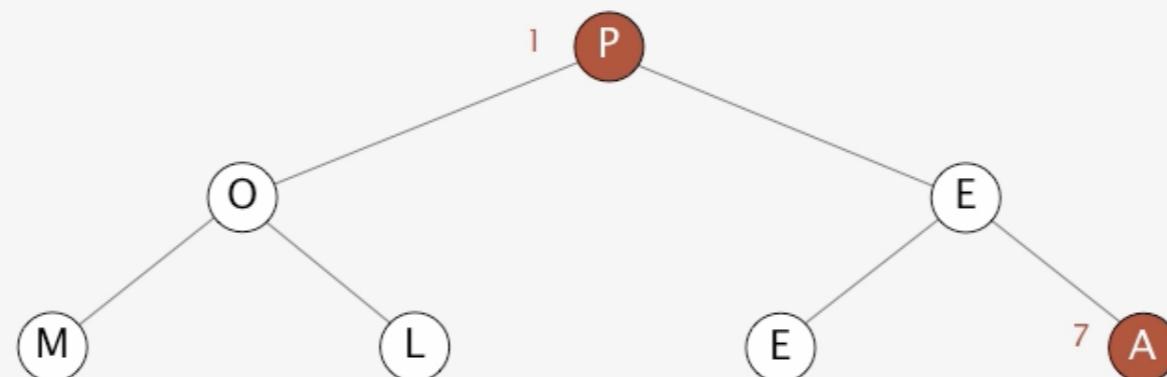
P	O	E	M	L	E	A	R	S	T	X
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 7



R S T X

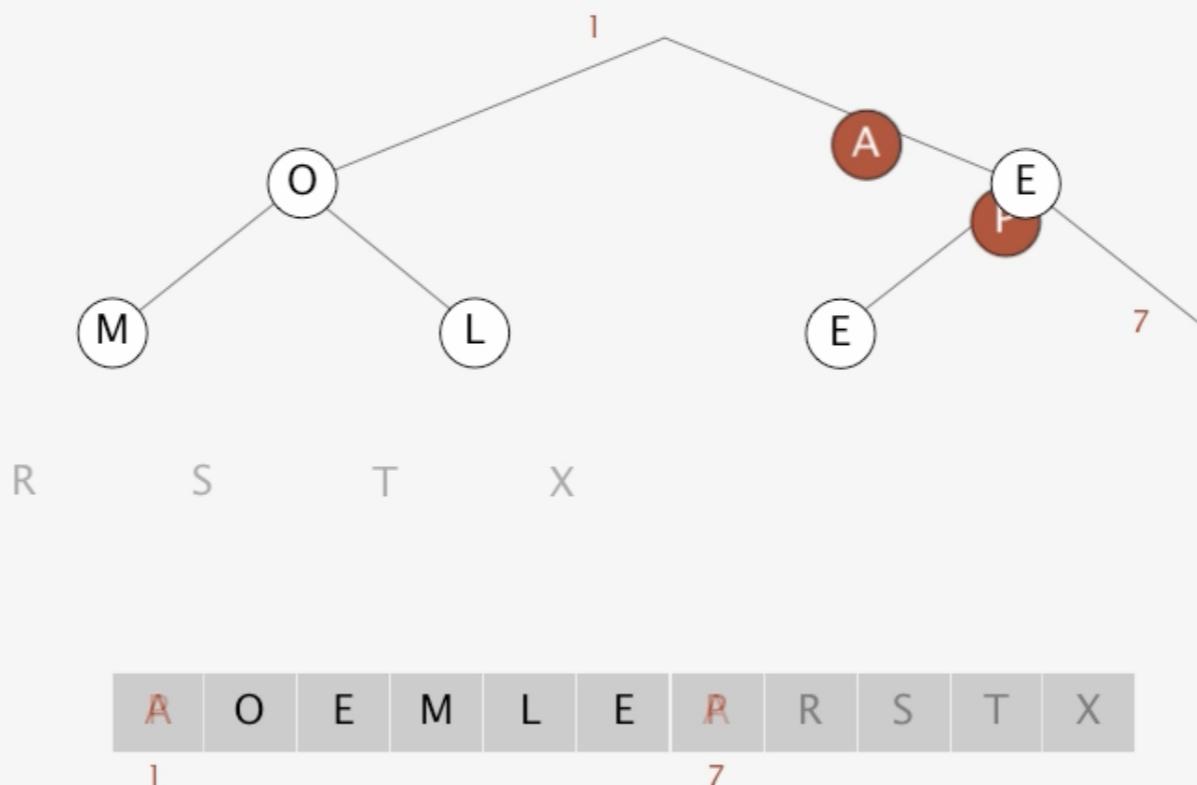
P	O	E	M	L	E	A	R	S	T	X
1						7				

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 7

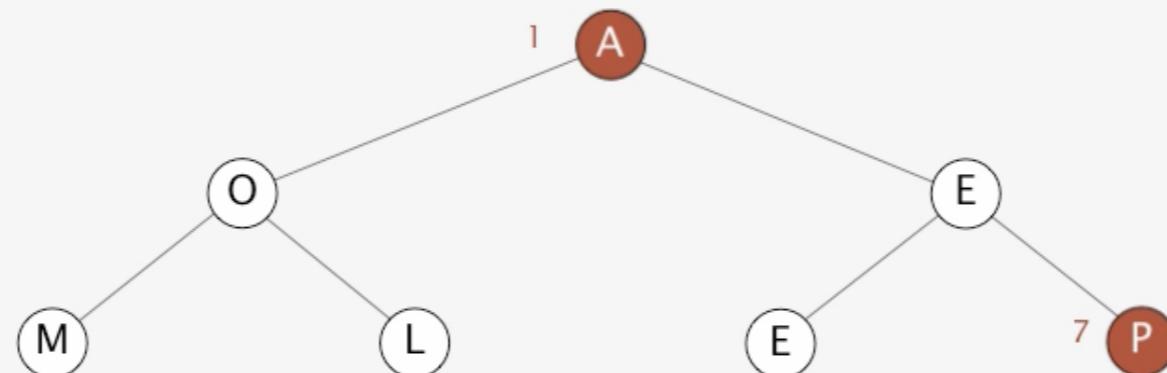


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 7



R S T X

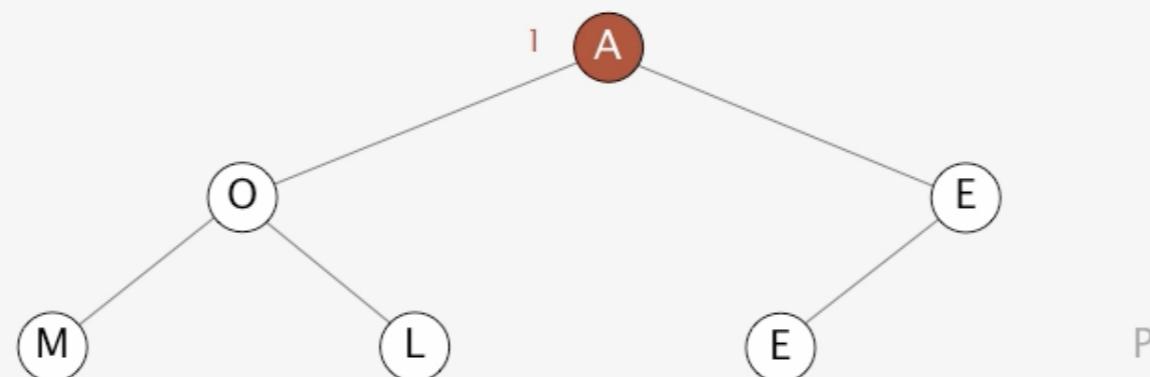
A	O	E	M	L	E	P	R	S	T	X
1						7				

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

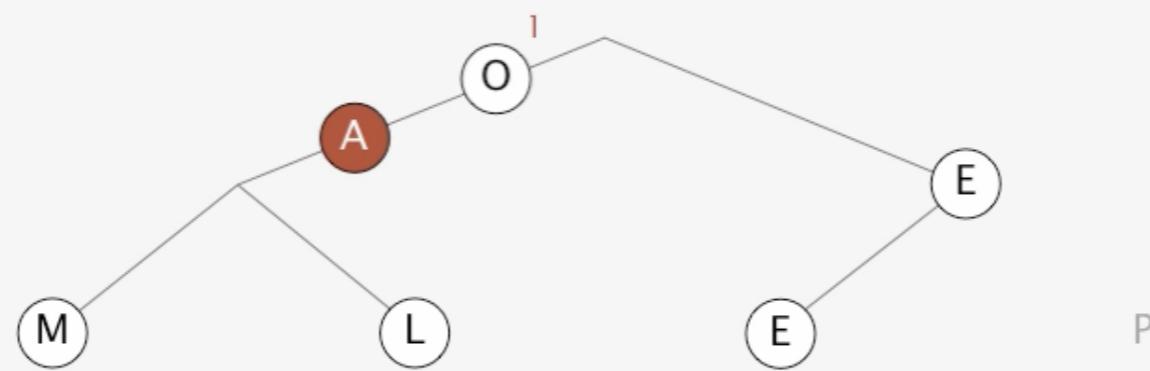


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

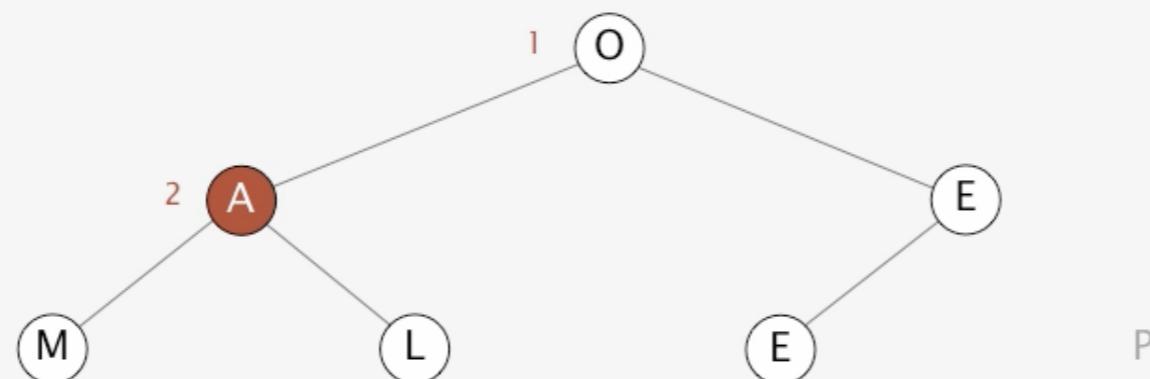


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

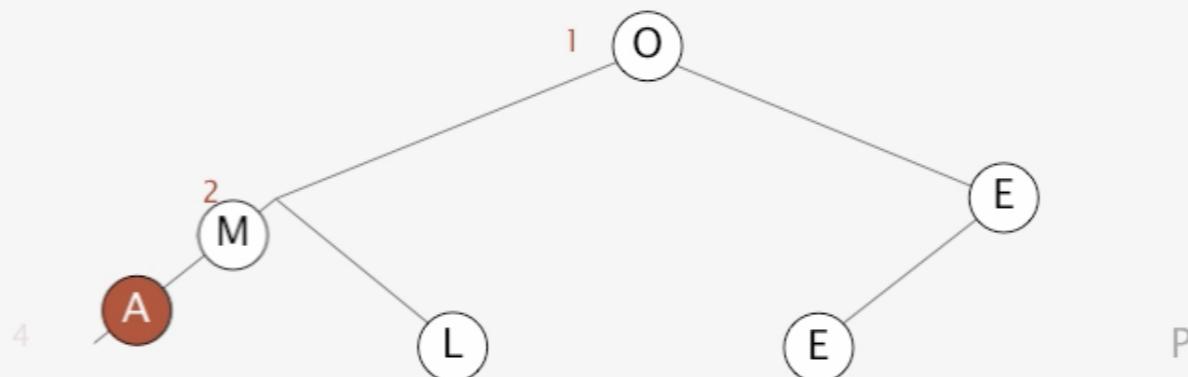


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

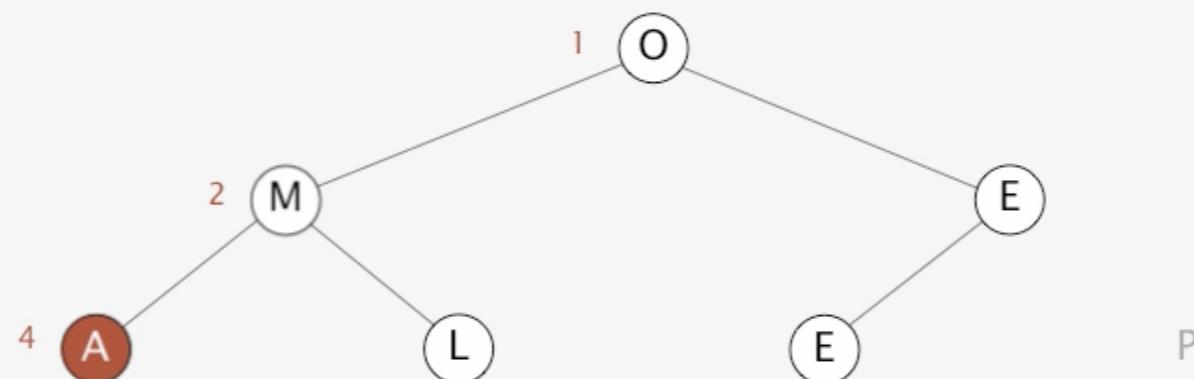
O	M	E	A	L	E	P	R	S	T	X
1	2		4							

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

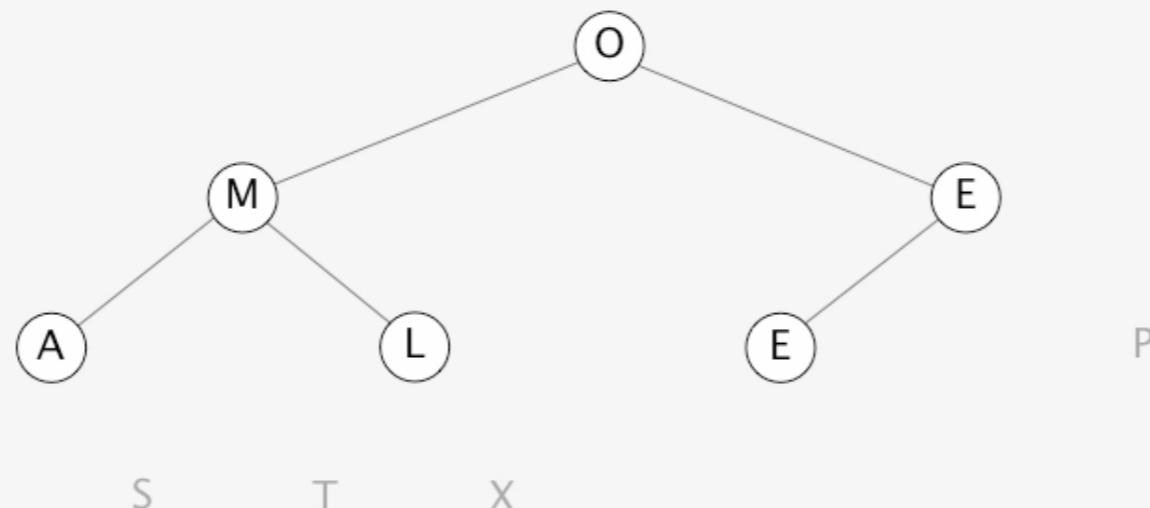
O	M	E	A	L	E	P	R	S	T	X
1	2		4							

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



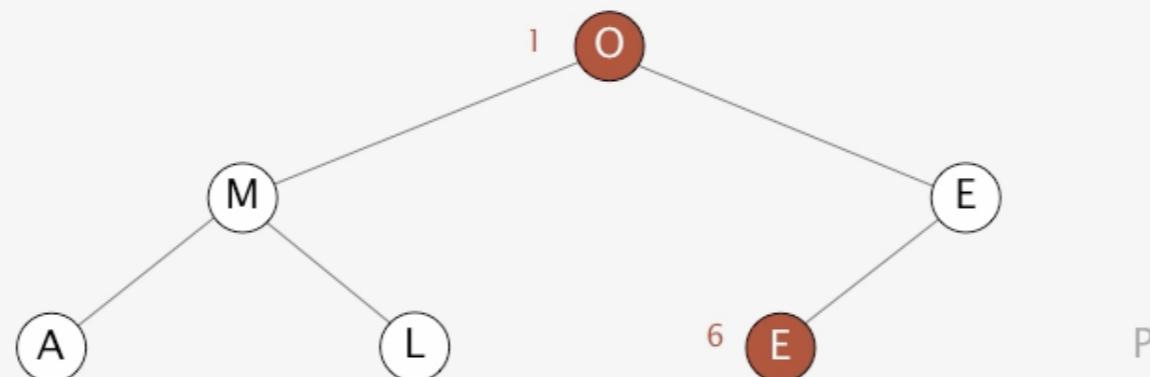
O	M	E	A	L	E	P	R	S	T	X
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 6



R S T X

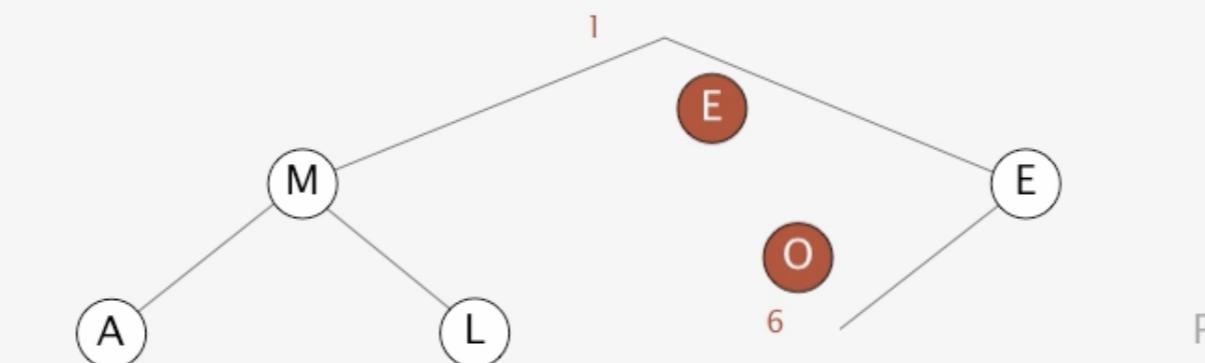
O	M	E	A	L	E	P	R	S	T	X
1					6					

Heapsort

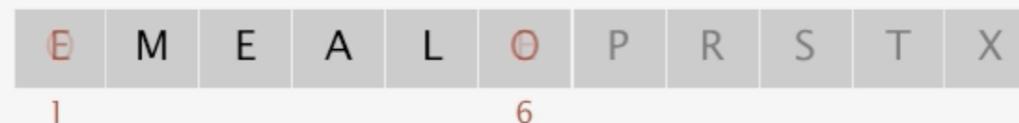
Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 6



R S T X

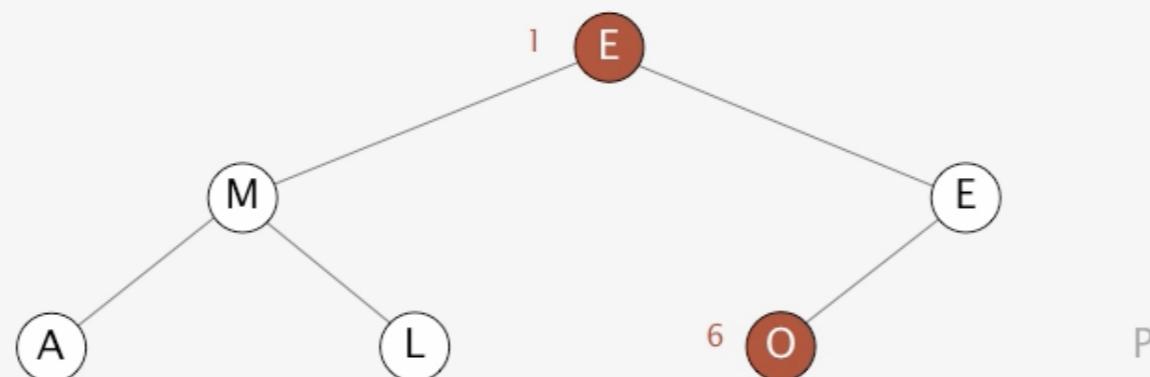


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 6



R S T X

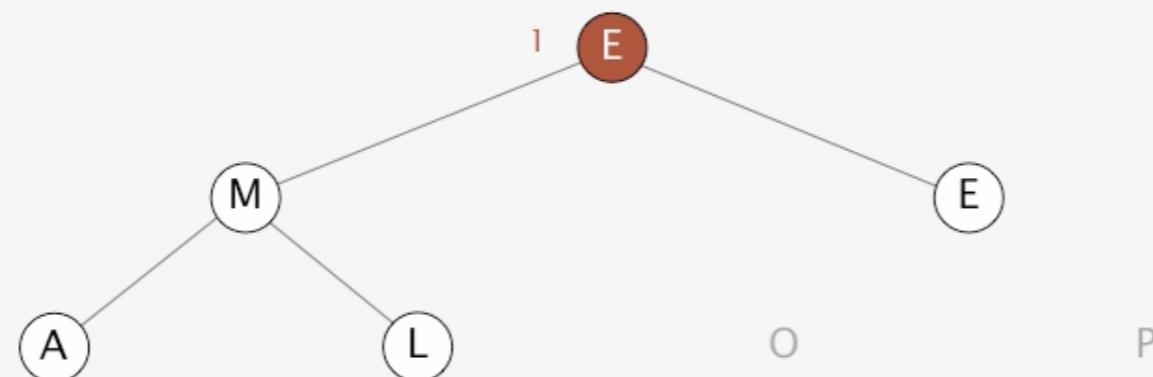
E	M	E	A	L	O	P	R	S	T	X
1					6					

Heapsort

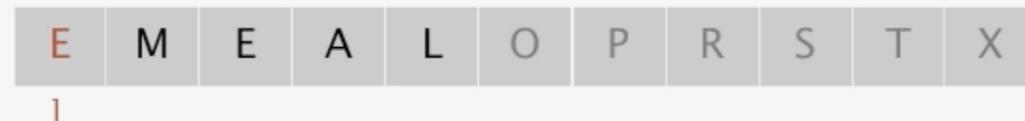
Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

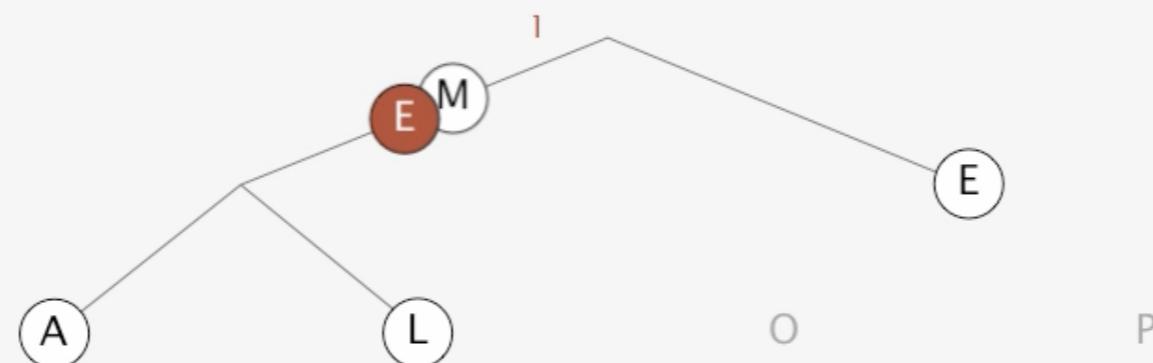


Heapsort

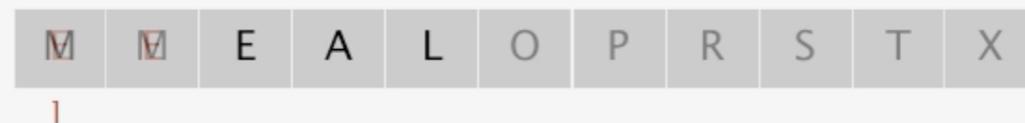
Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

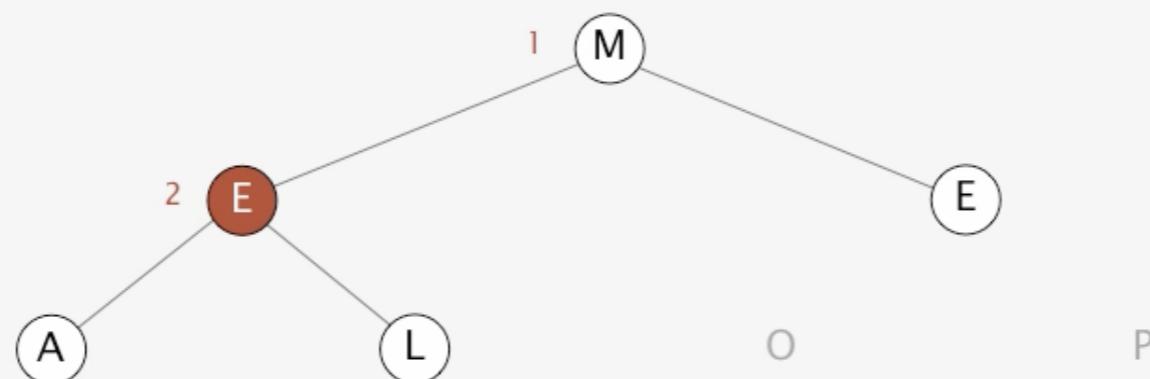


Heapsort

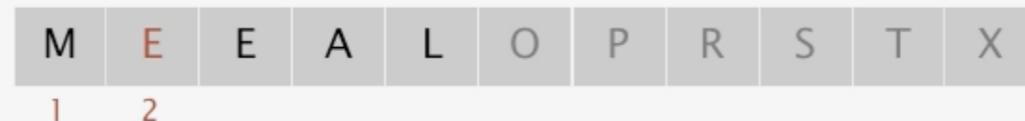
Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

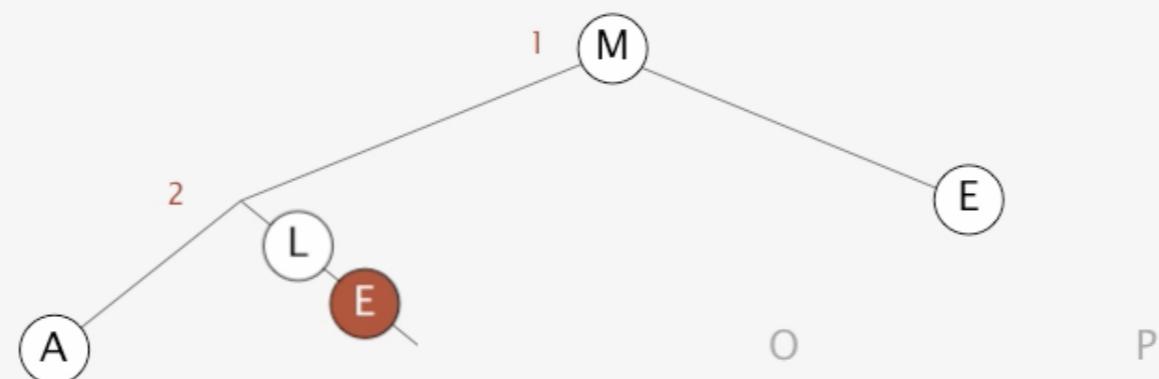


Heapsort

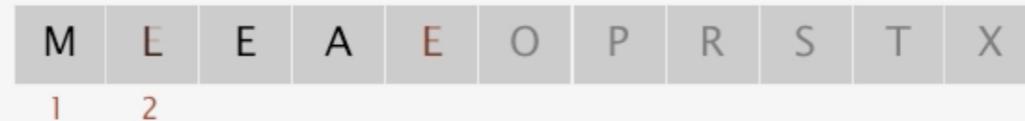
Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



R S T X

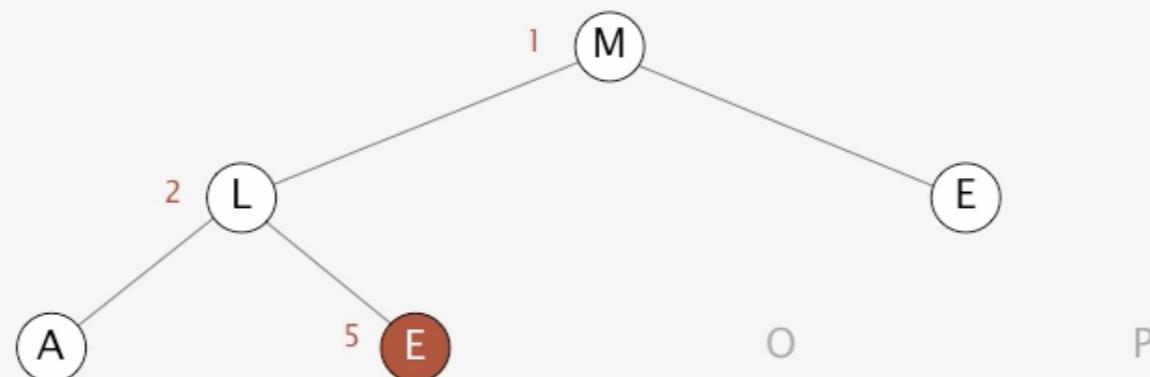


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



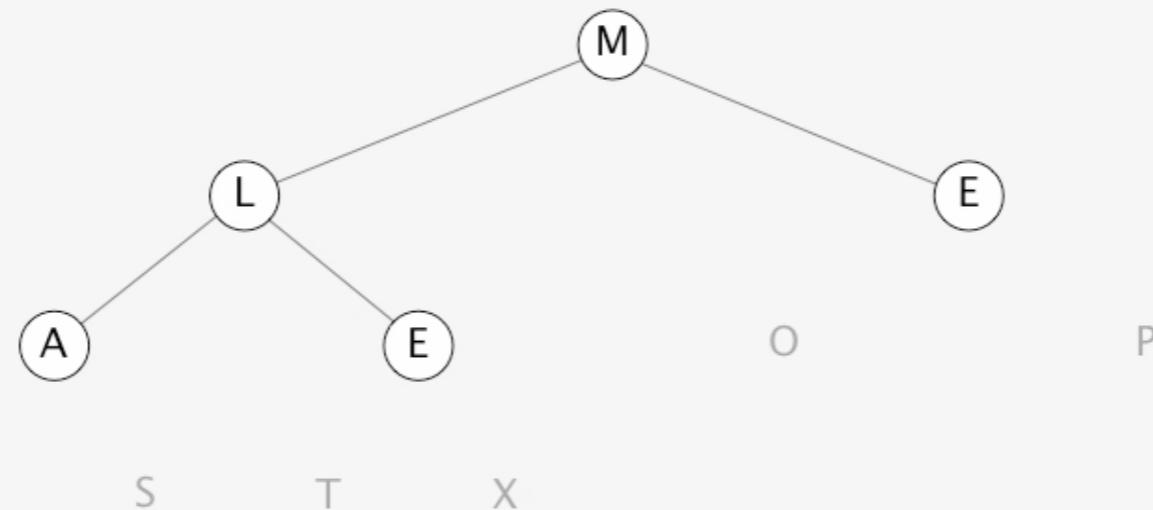
R S T X

M	L	E	A	E	O	P	R	S	T	X
1	2			5						

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.



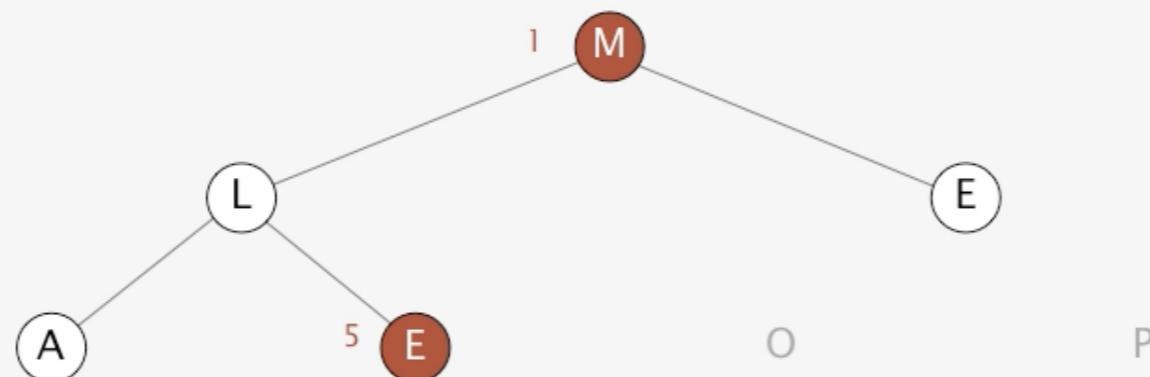
M	L	E	A	E	O	P	R	S	T	X
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 5



R S T X

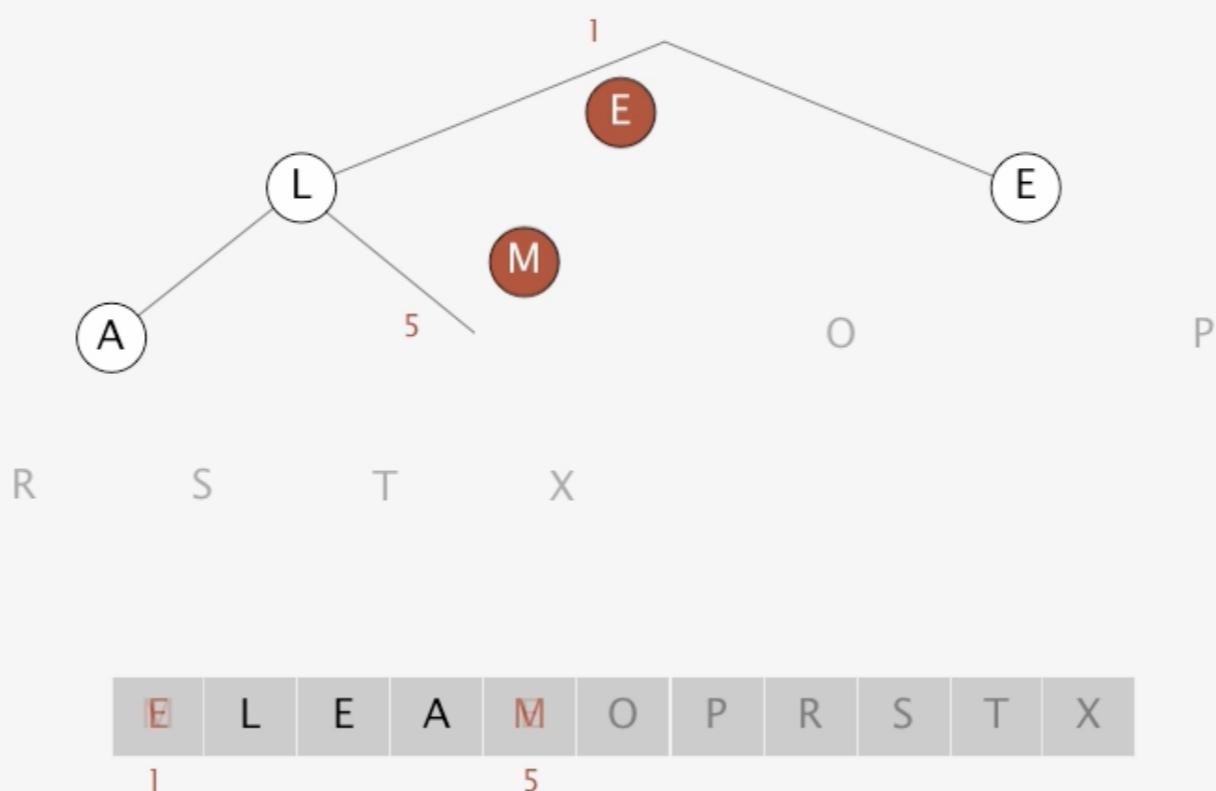
M	L	E	A	E	O	P	R	S	T	X
1				5						

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 5

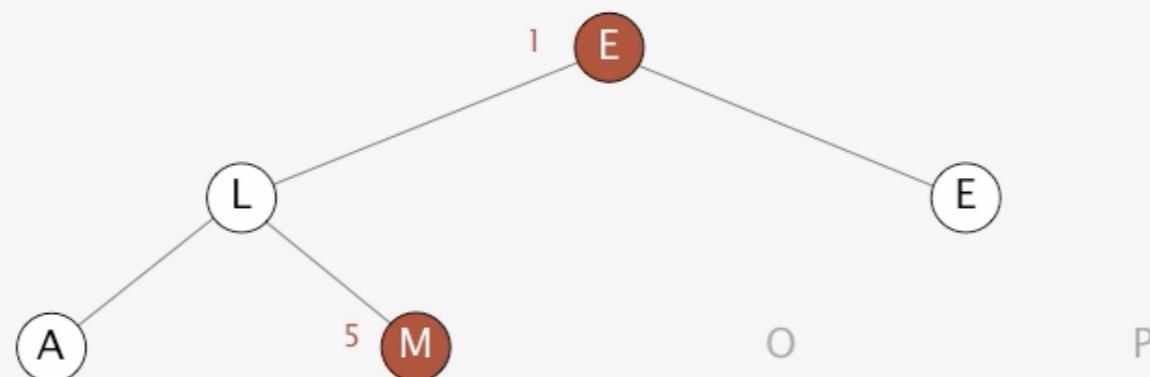


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 5



R S T X

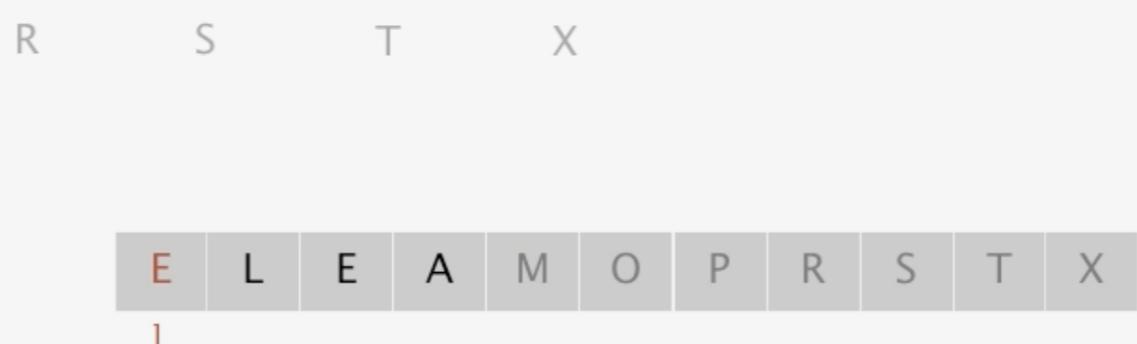
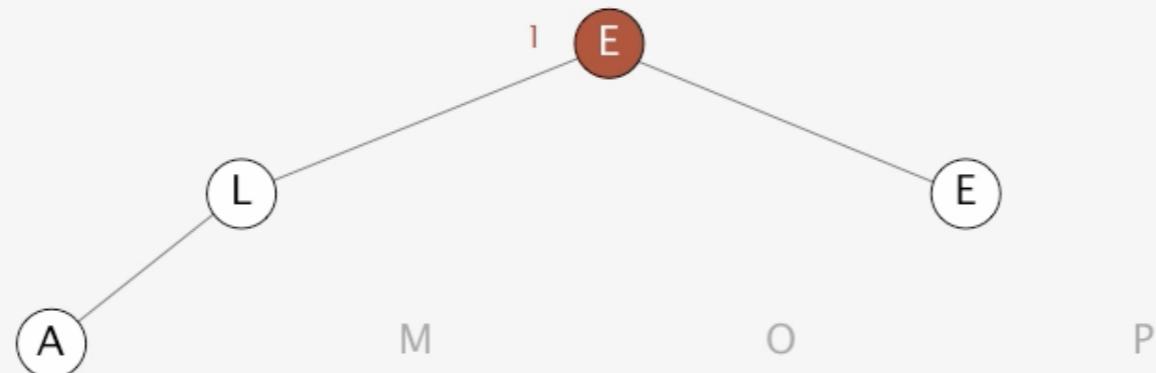
E	L	E	A	M	O	P	R	S	T	X
1				5						

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

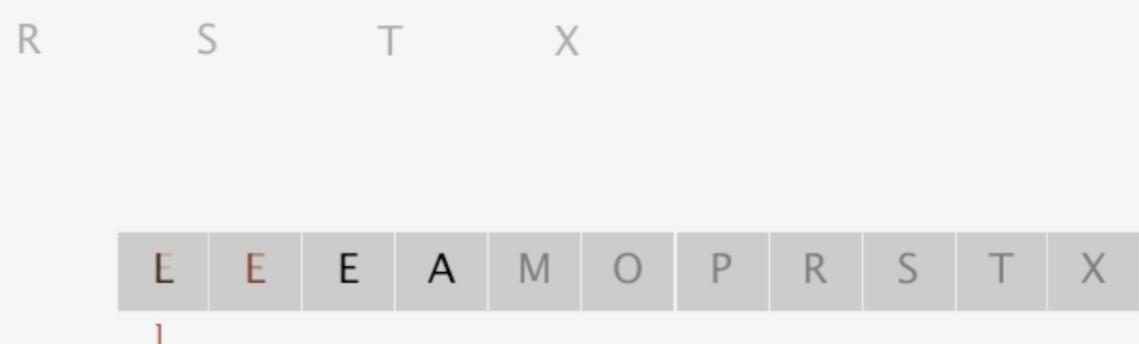
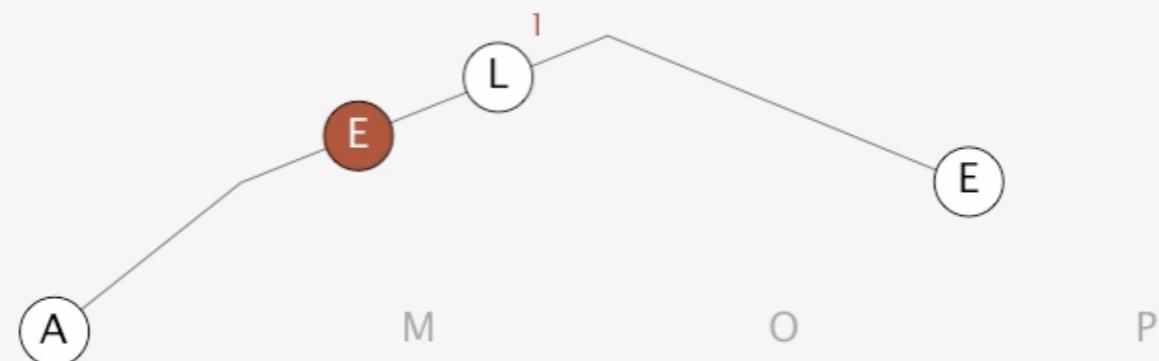


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

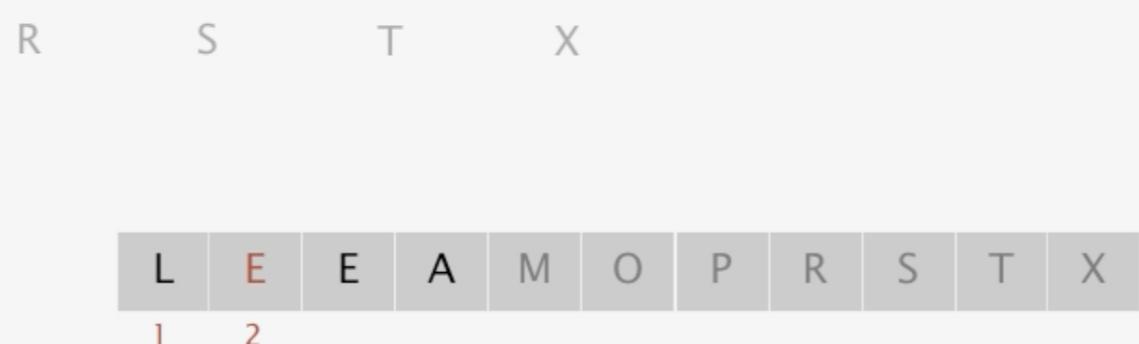
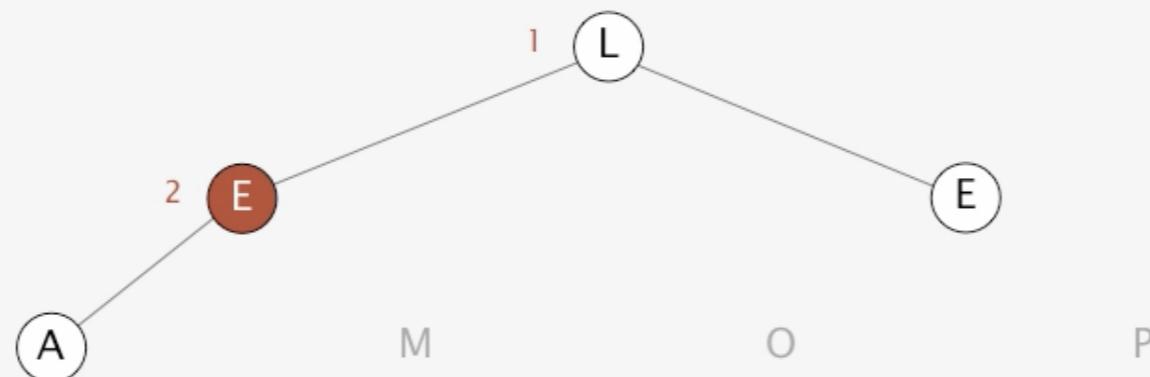


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

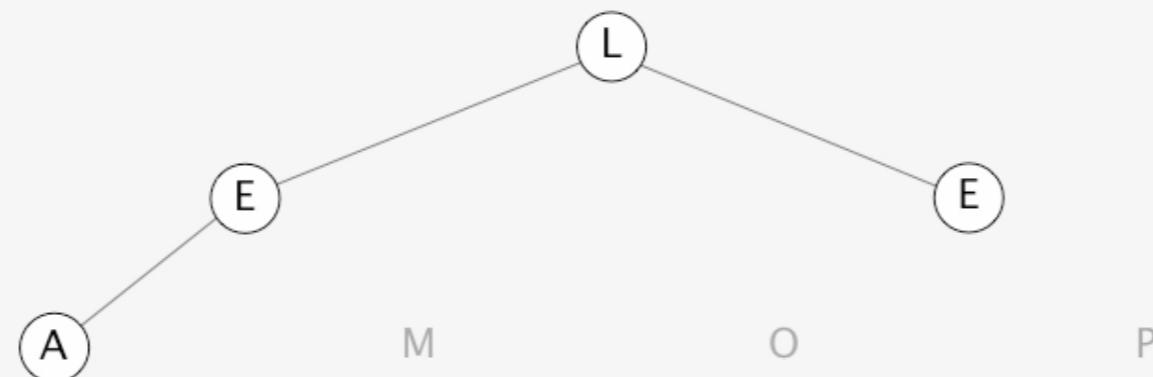
sink 1



Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.



R S T X

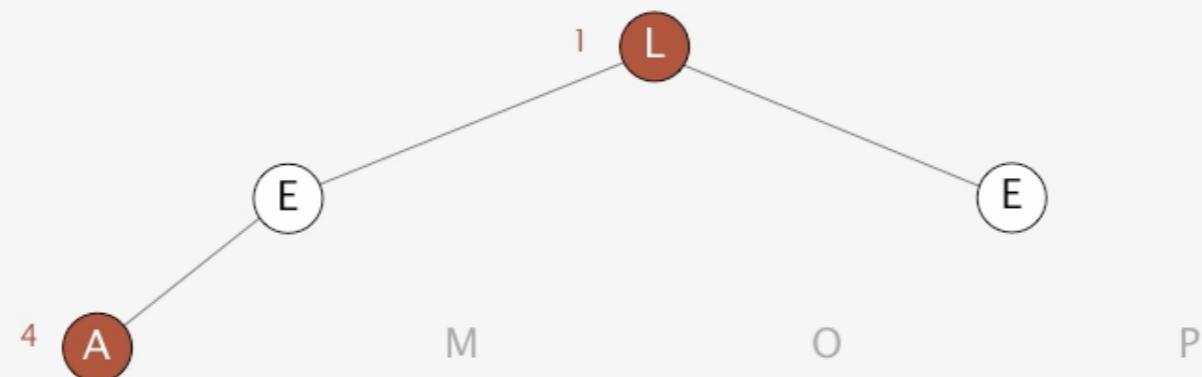
L	E	E	A	M	O	P	R	S	T	X
---	---	---	---	---	---	---	---	---	---	---

Heapsort

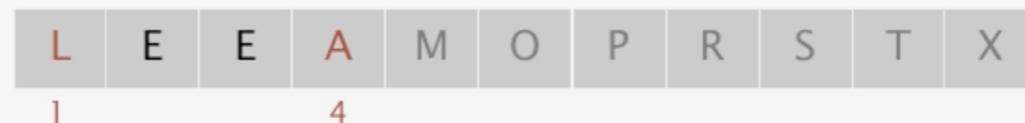
Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 4



R S T X

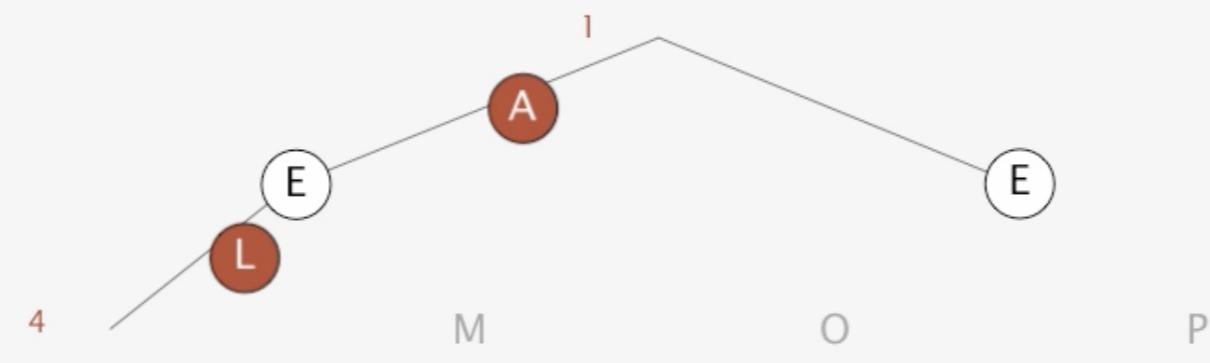


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 4



R S T X

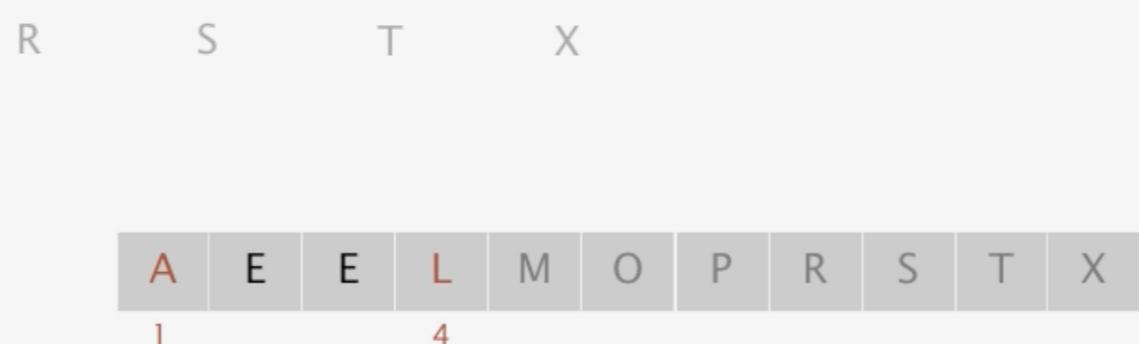
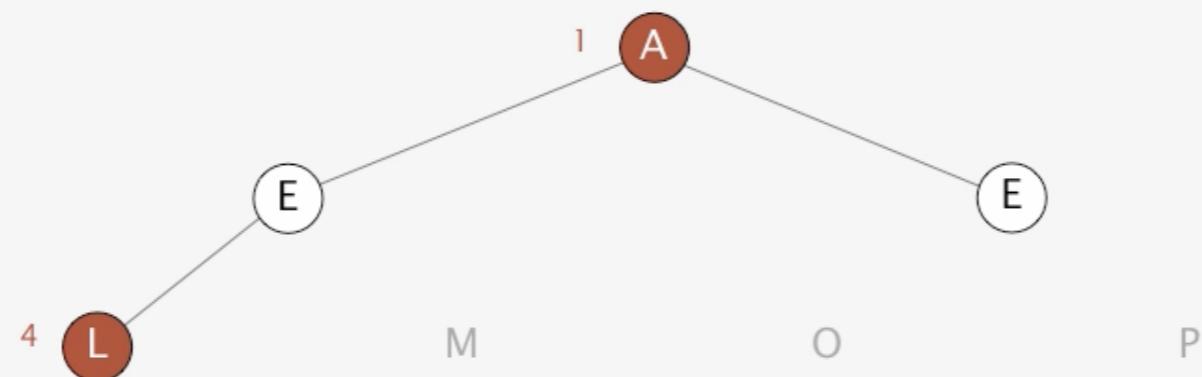
A	E	E	L	M	O	P	R	S	T	X
1			4							

Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 4

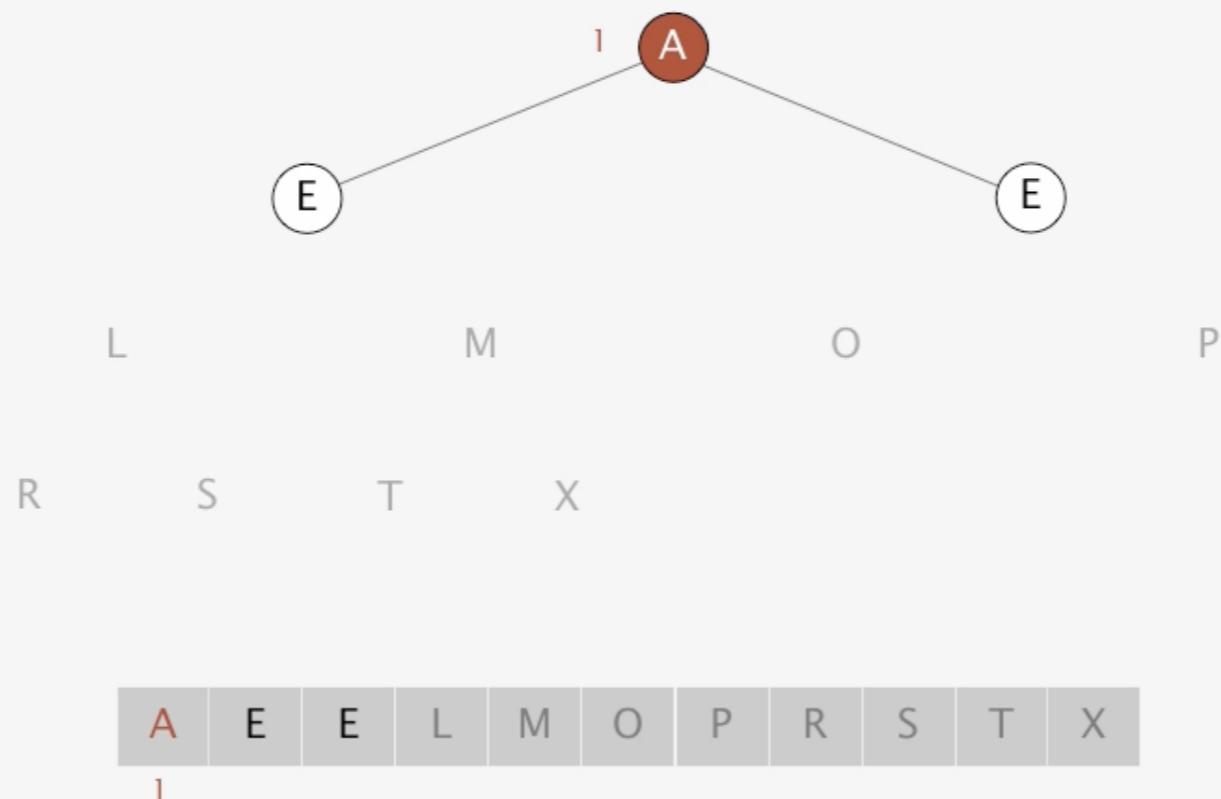


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

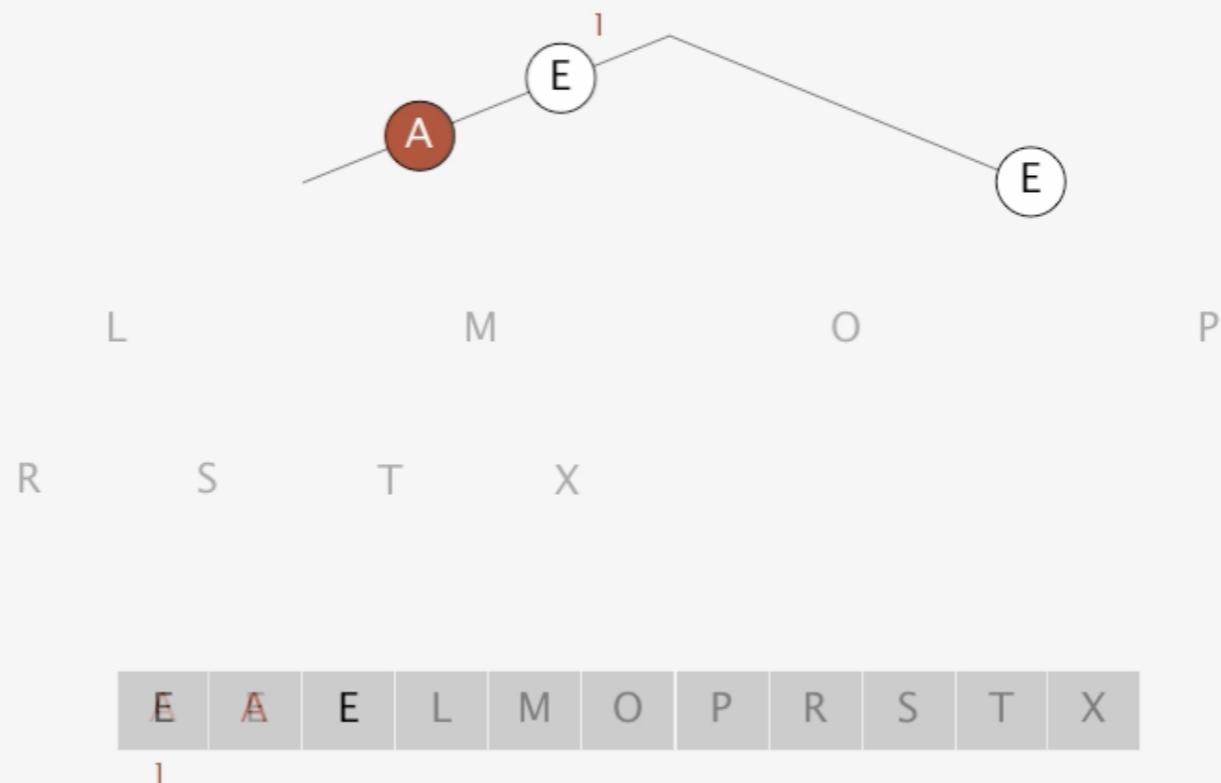


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1

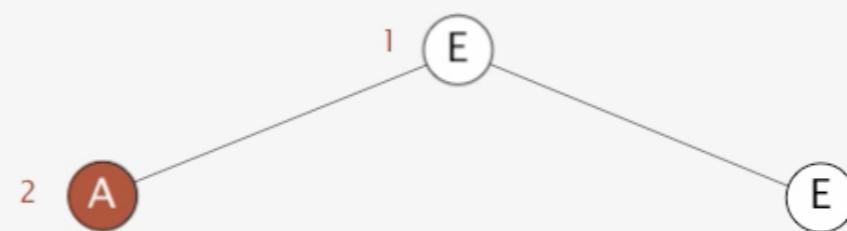


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

sink 1



L

M

0

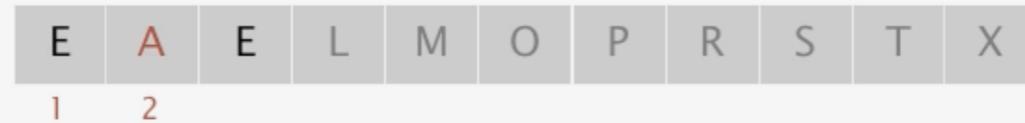
P

R

S

T

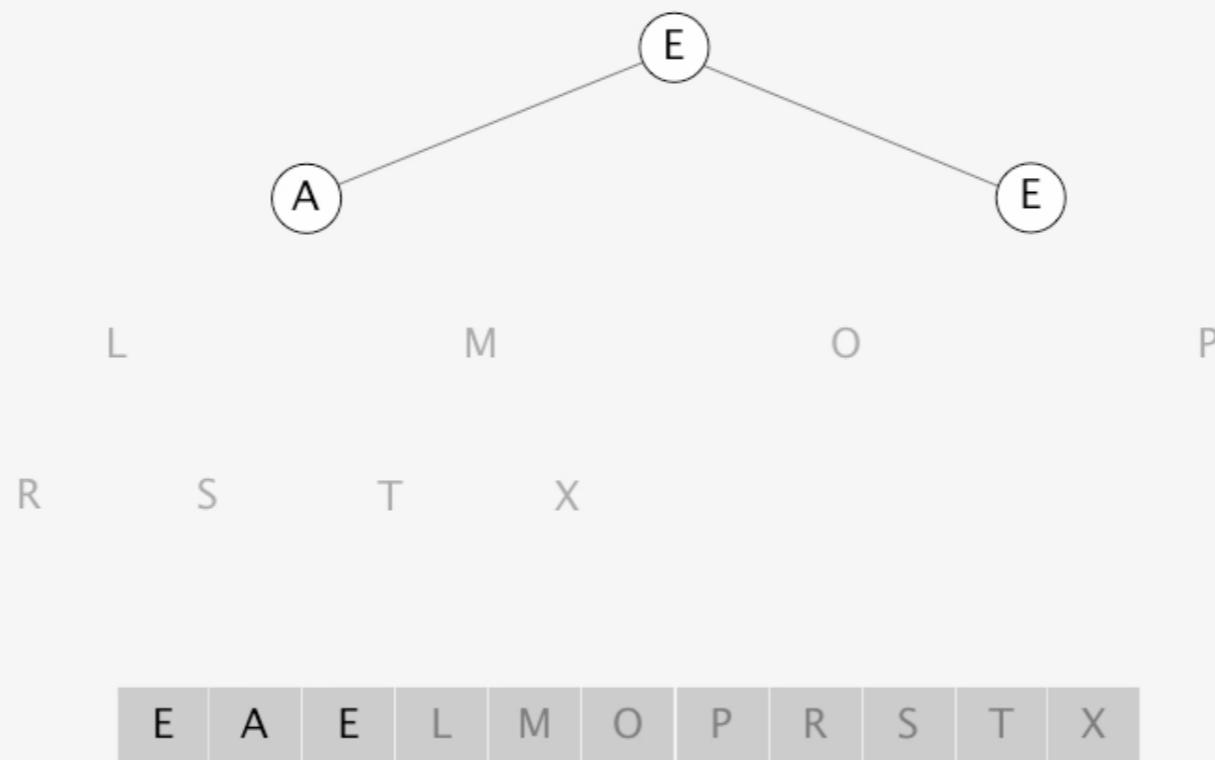
X



Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

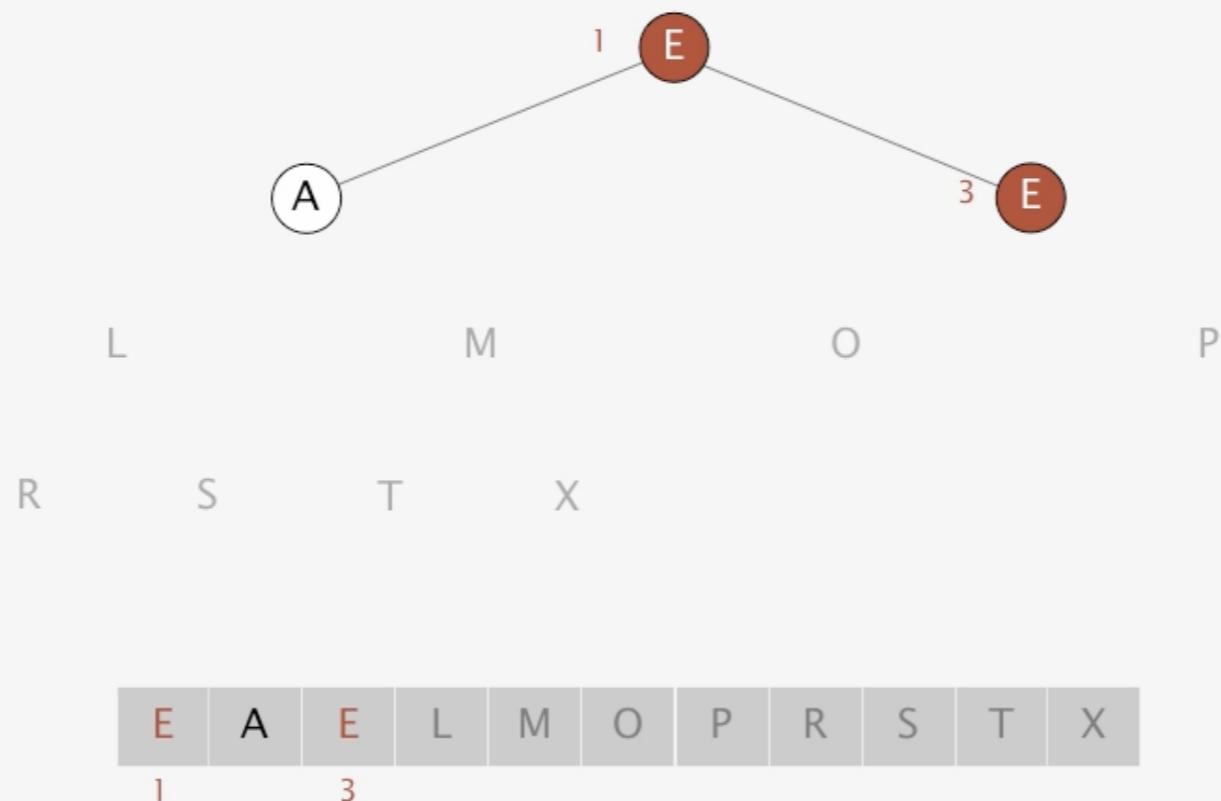


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 3

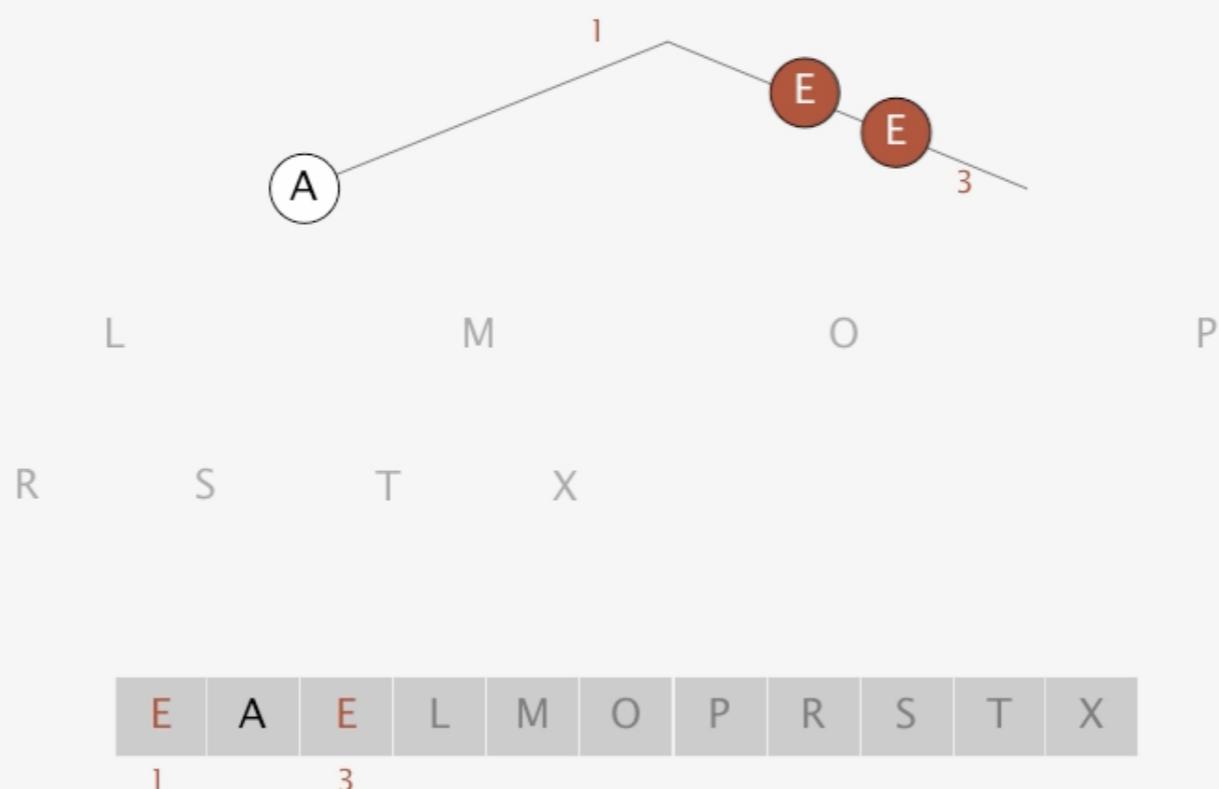


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 3

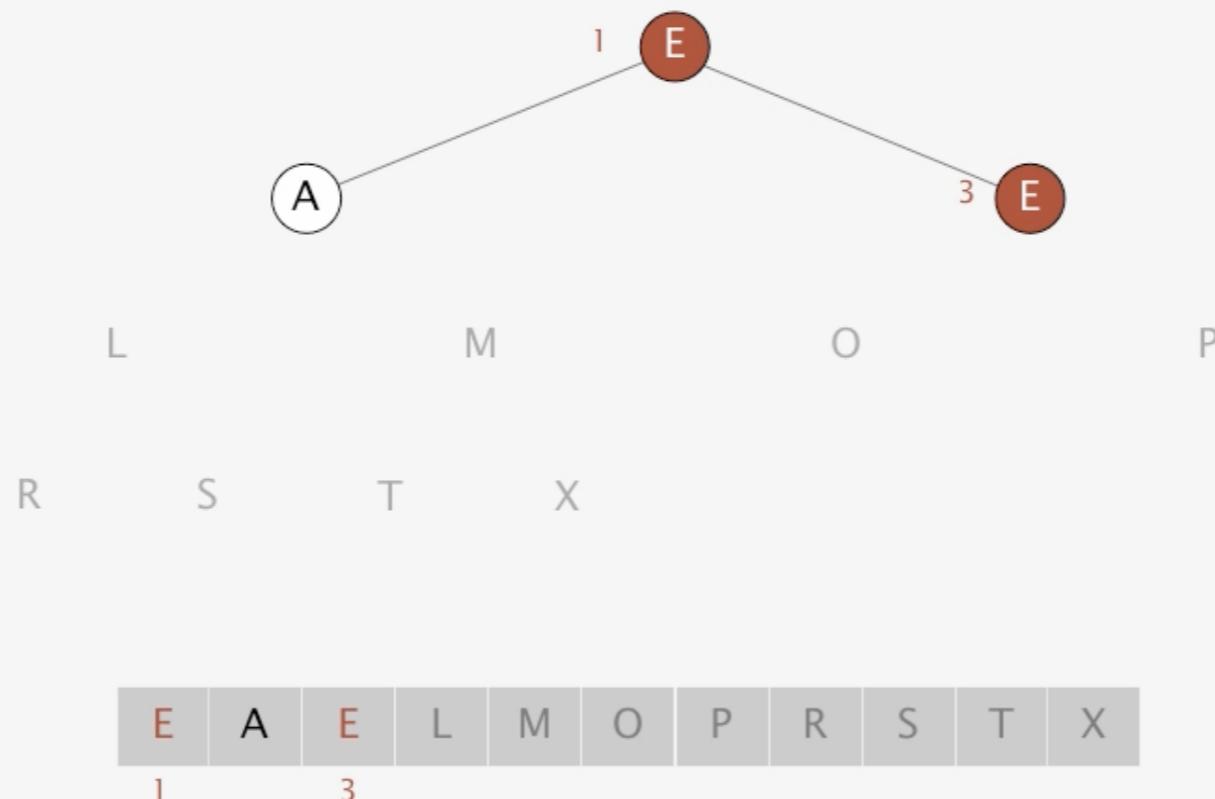


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 3

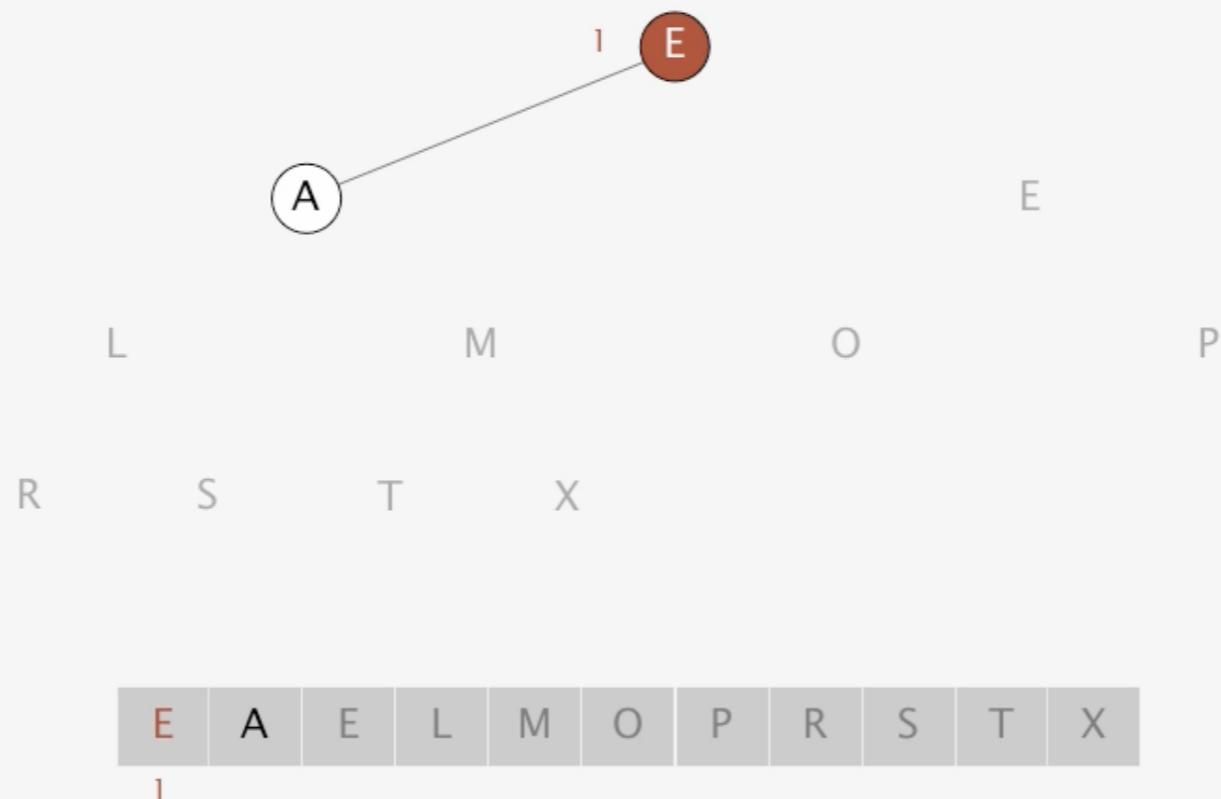


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

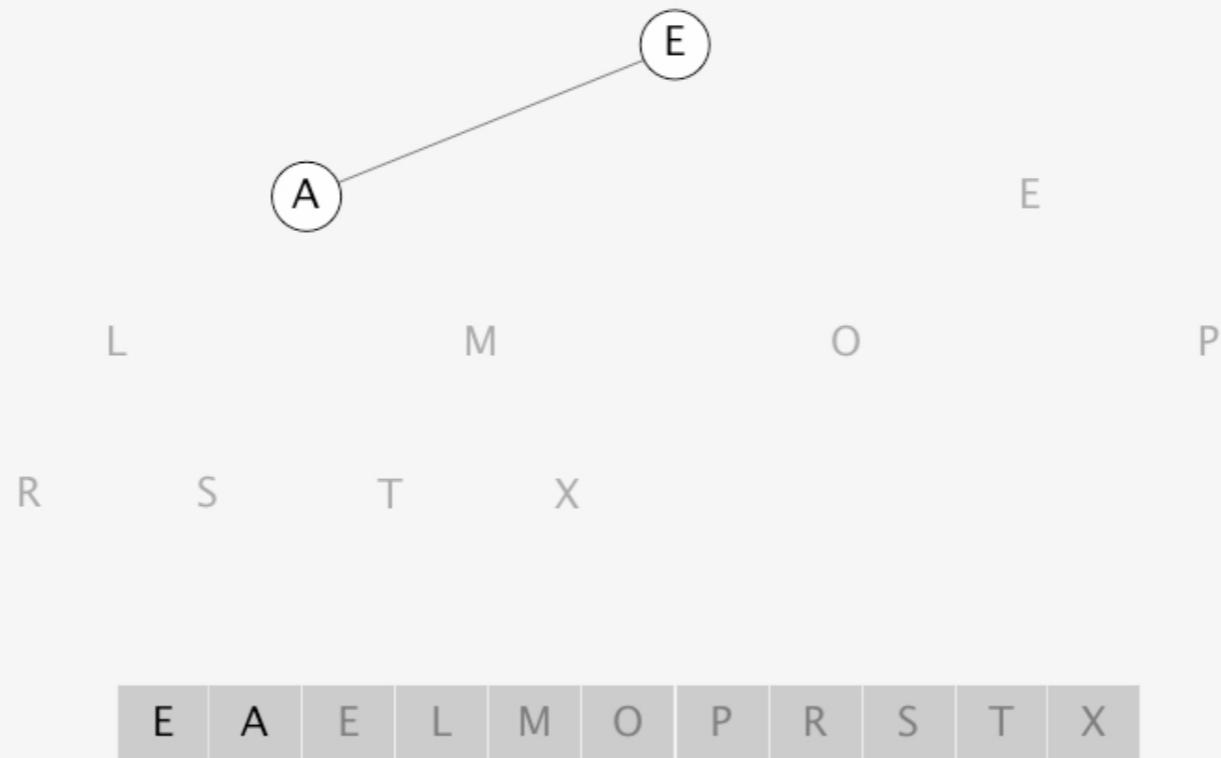
sink 1



Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

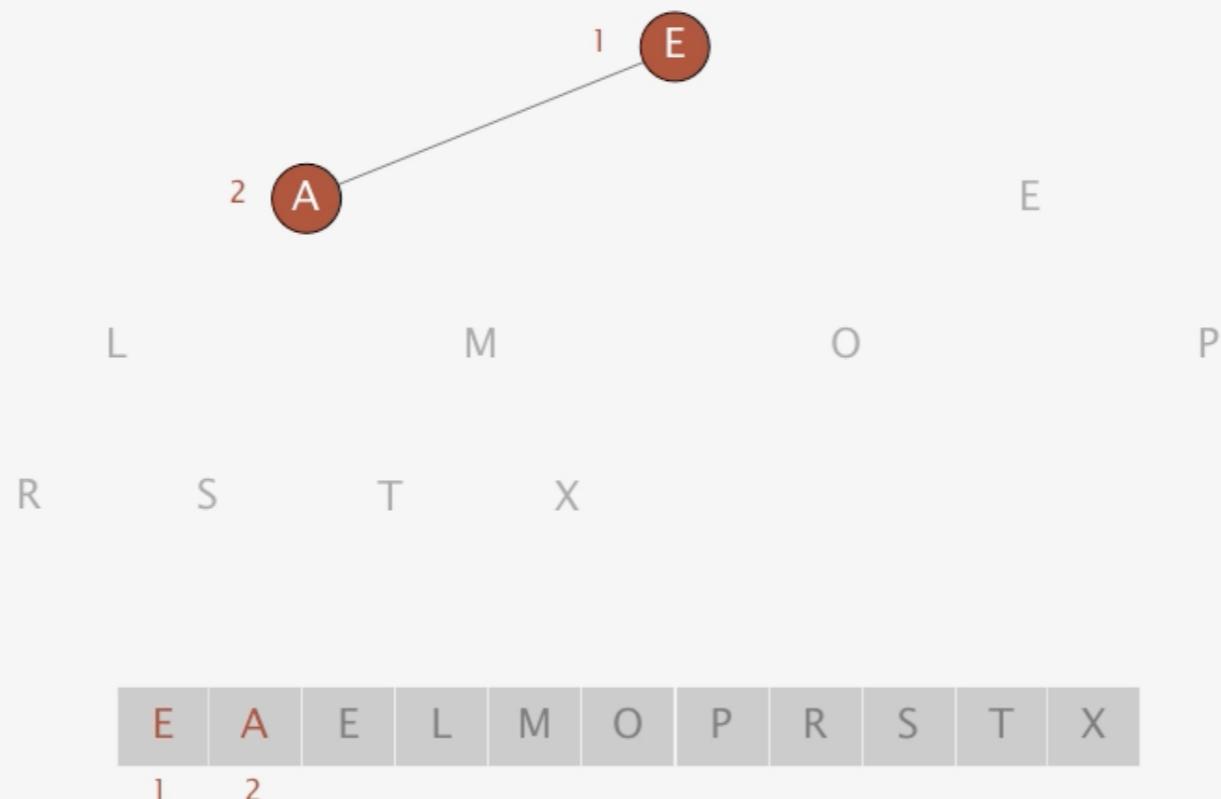


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 2

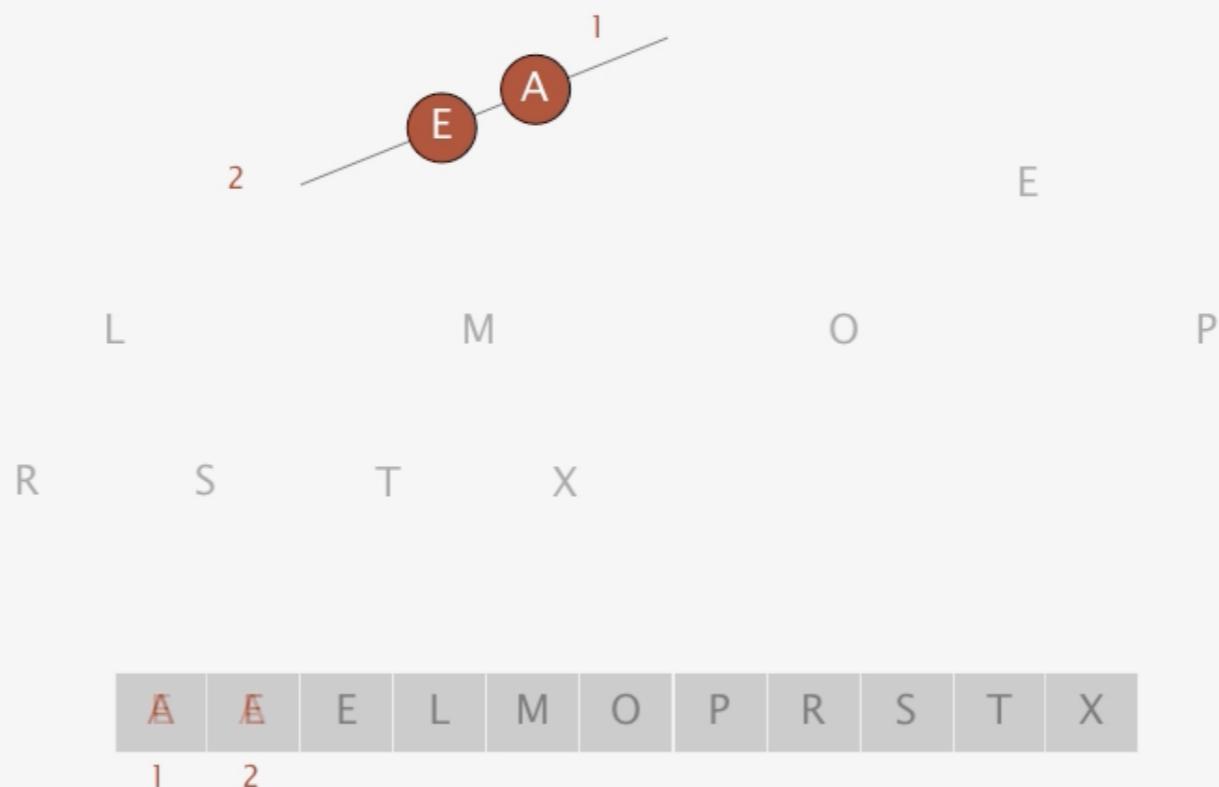


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 2

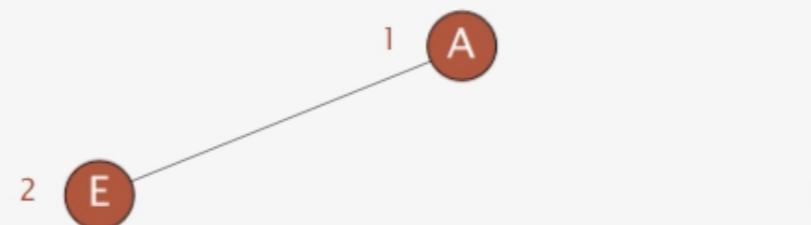


Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 2



L

M

0

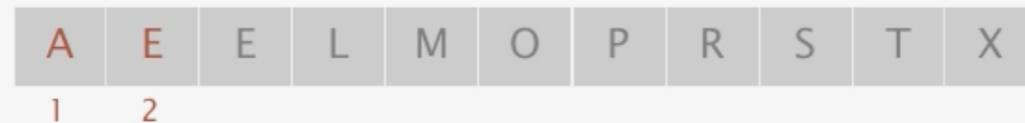
P

R

s

T

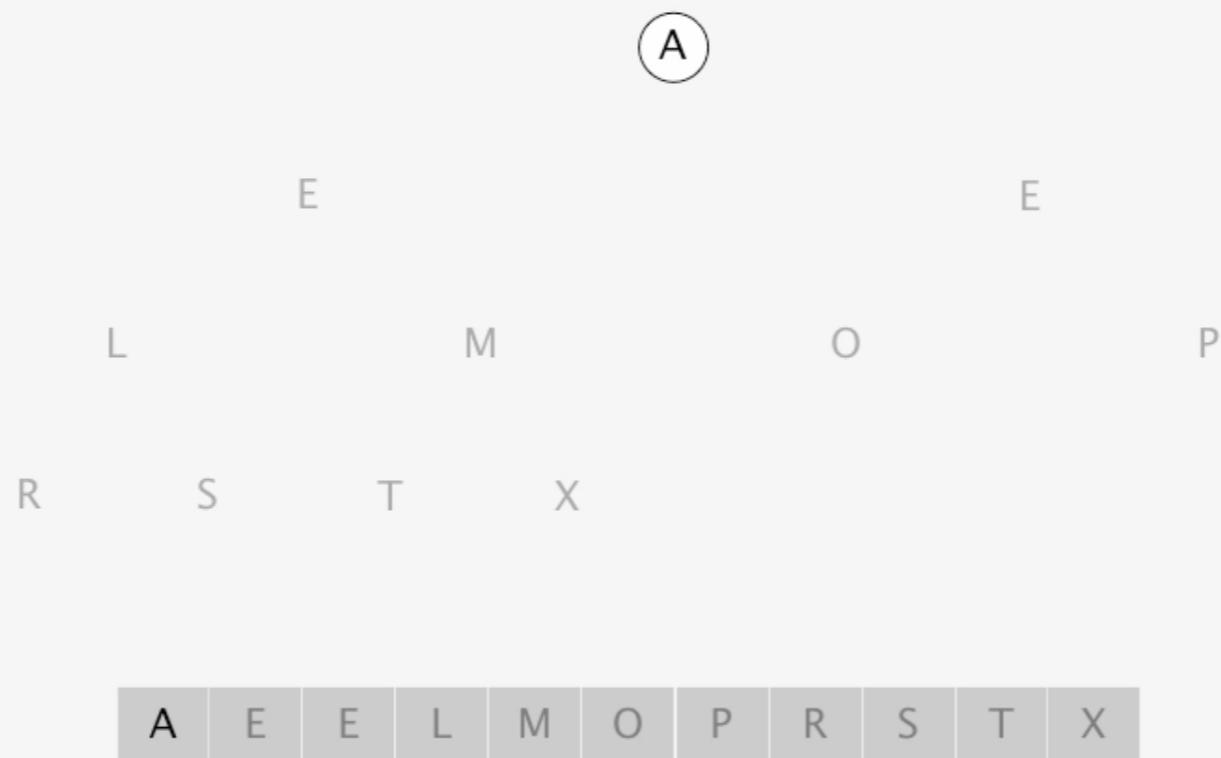
X



Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.



Heapsort

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

end of sortdown phase

A
E E
L O
R P
S T
T X

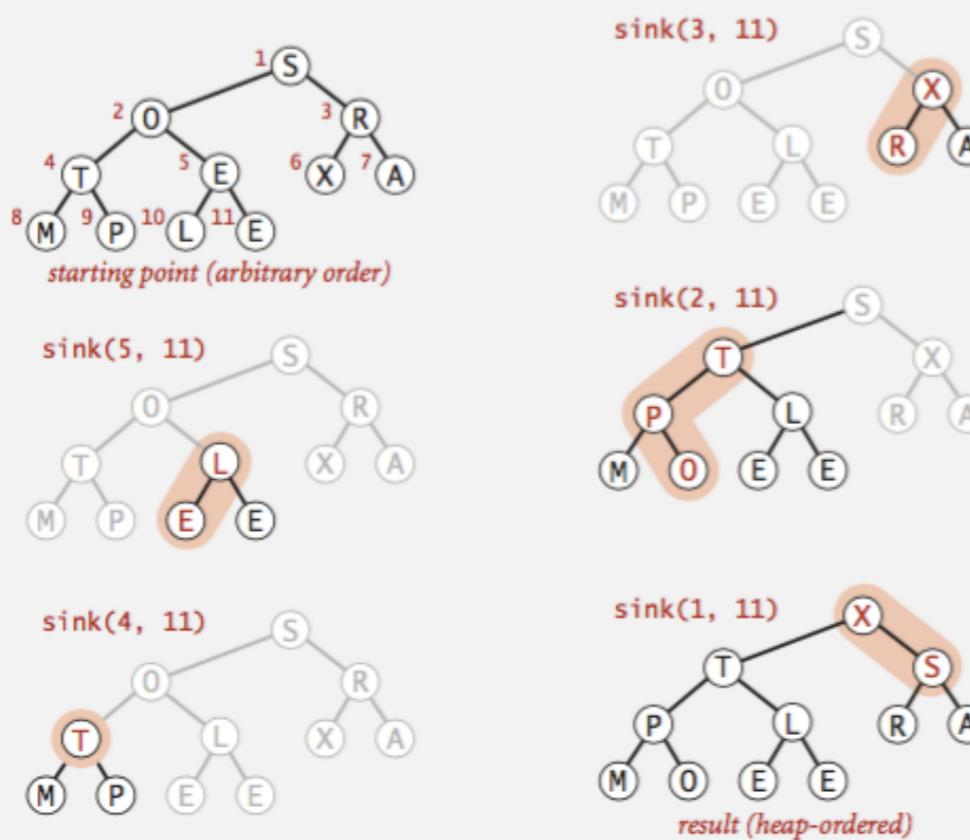
A	E	E	L	M	O	P	R	S	T	X
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heapsort: heap construction

First pass. Build heap using bottom-up method.

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



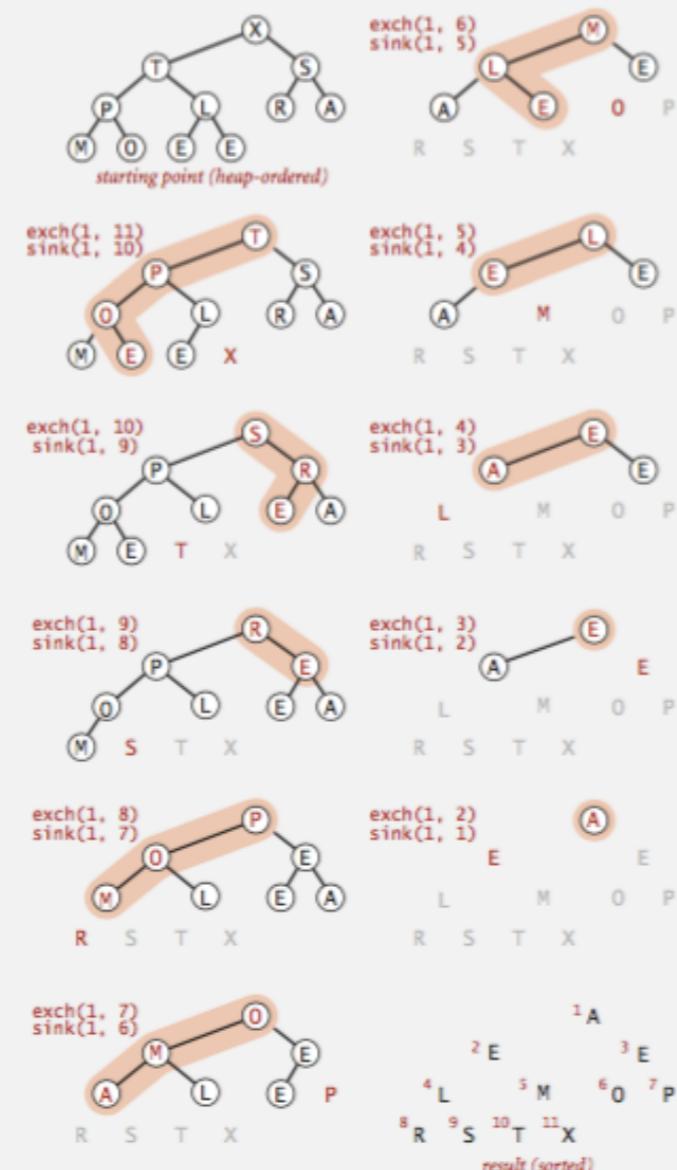
Heapsort

Heapsort: sortdown

Second pass.

- Remove the maximum, one at a time.
- Leave in array, instead of nulling out.

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



Heapsort

Heapsort: Java implementation

```
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);
        }
    }
        but make static (and pass arguments)

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(Object[] a, int i, int j)
{ /* as before */ }
}

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

Heapsort

Heapsort: trace

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

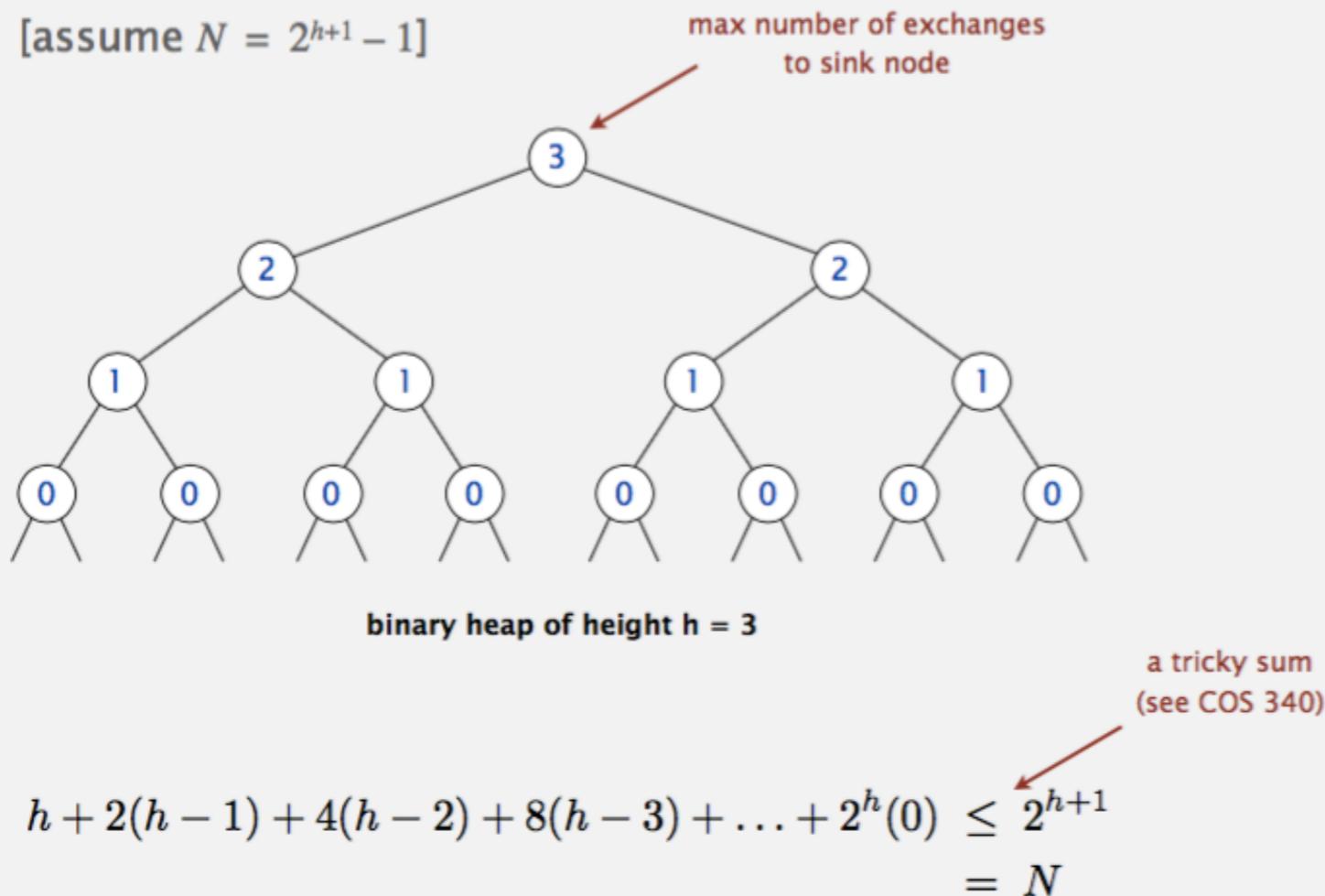
Heapsort trace (array contents just after each sink)

Heapsort

Heapsort: mathematical analysis

Proposition. Heap construction uses $\leq 2N$ compares and $\leq N$ exchanges.

Pf sketch. [assume $N = 2^{h+1} - 1$]



Heapsort: mathematical analysis

Proposition. Heap construction uses $\leq 2N$ compares and $\leq N$ exchanges.

Proposition. Heapsort uses $\leq 2N \lg N$ compares and exchanges.

algorithm can be improved to $\sim N \lg N$

Significance. In-place sorting algorithm with $N \log N$ worst-case.

- Mergesort: no, linear extra space. ← in-place merge possible, not practical
- Quicksort: no, quadratic time in worst case. ← $N \log N$ worst-case quicksort possible, not practical
- Heapsort: yes!

Bottom line. Heapsort is optimal for both time and space, but:

- Inner loop longer than quicksort's.
- Makes poor use of cache.
- Not stable.

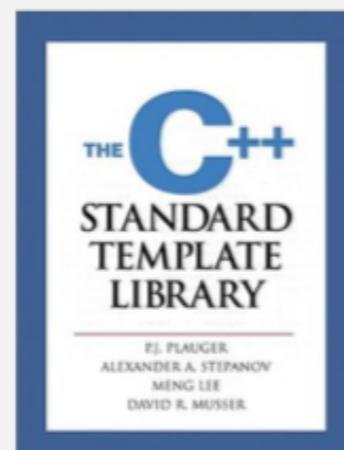
advanced tricks for improving

Introsort

Goal. As fast as quicksort in practice; $N \log N$ worst case, in place.

Introsort.

- Run quicksort.
- Cutoff to heapsort if stack depth exceeds $2 \lg N$.
- Cutoff to insertion sort for $N = 16$.



In the wild. C++ STL, Microsoft .NET Framework.

Heapsort

Sorting algorithms: summary

	inplace?	stable?	best	average	worst	remarks
selection	✓		$\frac{1}{2} N^2$	$\frac{1}{2} N^2$	$\frac{1}{2} N^2$	N exchanges
insertion	✓	✓	N	$\frac{1}{4} N^2$	$\frac{1}{2} N^2$	use for small N or partially ordered
shell	✓		$N \log_3 N$?	$c N^{3/2}$	tight code; subquadratic
merge		✓	$\frac{1}{2} N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee; stable
timsort		✓	N	$N \lg N$	$N \lg N$	improves mergesort when preexisting order
quick	✓		$N \lg N$	$2 N \ln N$	$\frac{1}{2} N^2$	$N \log N$ probabilistic guarantee; fastest in practice
3-way quick	✓		N	$2 N \ln N$	$\frac{1}{2} N^2$	improves quicksort when duplicate keys
heap	✓		N	$2 N \lg N$	$2 N \lg N$	$N \log N$ guarantee; in-place
?	✓	✓	N	$N \lg N$	$N \lg N$	holy sorting grail

Lecture Overview



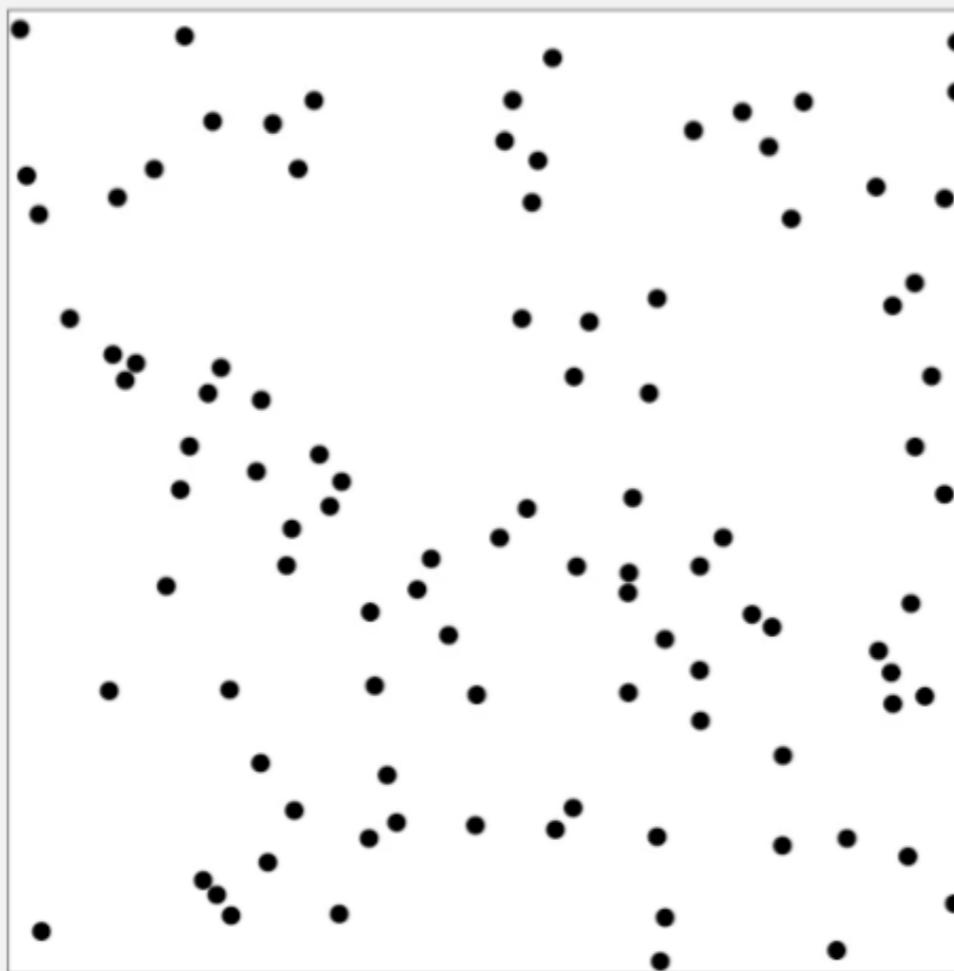
2.4 Priority Queues

- API and elementary implementations
- Binary heaps
- Heapsort
- Event-driven simulation

Event-Driven Simulation

Molecular dynamics simulation of hard discs

Goal. Simulate the motion of N moving particles that behave according to the laws of elastic collision.



Event-Driven Simulation

Molecular dynamics simulation of hard discs

Goal. Simulate the motion of N moving particles that behave according to the laws of elastic collision.

Hard disc model.

- Moving particles interact via elastic collisions with each other and walls.
- Each particle is a disc with known position, velocity, mass, and radius.
- No other forces.

temperature, pressure,
diffusion constant

motion of individual
atoms and molecules

Significance. Relates macroscopic observables to microscopic dynamics.

- Maxwell-Boltzmann: distribution of speeds as a function of temperature.
- Einstein: explain Brownian motion of pollen grains.

Event-Driven Simulation

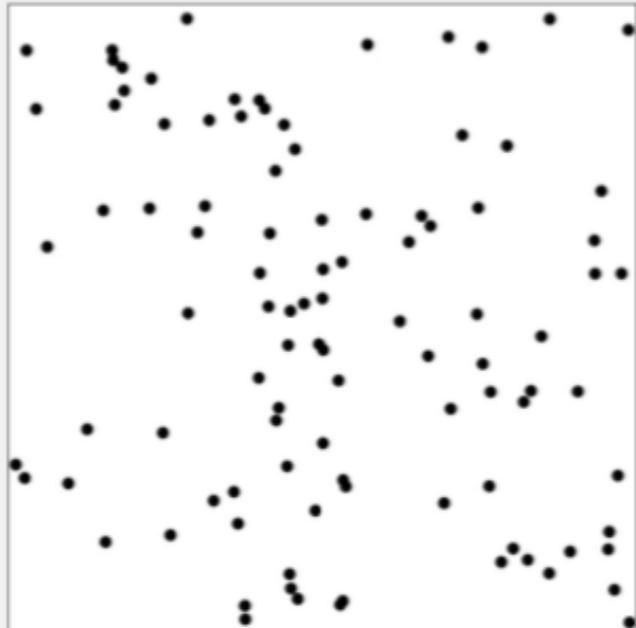
Warmup: bouncing balls

Time-driven simulation. N bouncing balls in the unit square.

```
public class BouncingBalls
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        Ball[] balls = new Ball[N];
        for (int i = 0; i < N; i++)
            balls[i] = new Ball();
        while(true)
        {
            StdDraw.clear();
            for (int i = 0; i < N; i++)
            {
                balls[i].move(0.5);
                balls[i].draw();
            }
            StdDraw.show(50);
        }
    }
}
```

main simulation loop

```
% java BouncingBalls 100
```



Event-Driven Simulation

Warmup: bouncing balls

```
public class Ball
{
    private double rx, ry;          // position
    private double vx, vy;          // velocity
    private final double radius;    // radius
    public Ball(...)
    { /* initialize position and velocity */ }

    public void move(double dt)
    {
        if ((rx + vx*dt < radius) || (rx + vx*dt > 1.0 - radius)) { vx = -vx; }      check for collision with walls
        if ((ry + vy*dt < radius) || (ry + vy*dt > 1.0 - radius)) { vy = -vy; }
        rx = rx + vx*dt;
        ry = ry + vy*dt;
    }
    public void draw()
    { StdDraw.filledCircle(rx, ry, radius); }
}
```



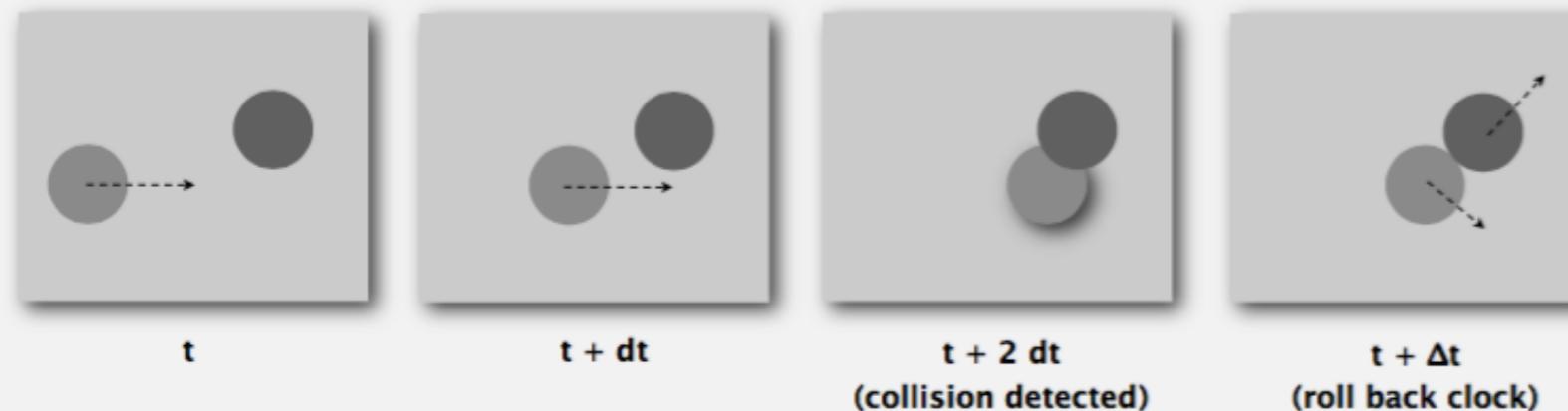
Missing. Check for balls colliding with each other.

- Physics problems: when? what effect?
- CS problems: which object does the check? too many checks?

Event-Driven Simulation

Time-driven simulation

- Discretize time in quanta of size dt .
- Update the position of each particle after every dt units of time, and check for overlaps.
- If overlap, roll back the clock to the time of the collision, update the velocities of the colliding particles, and continue the simulation.



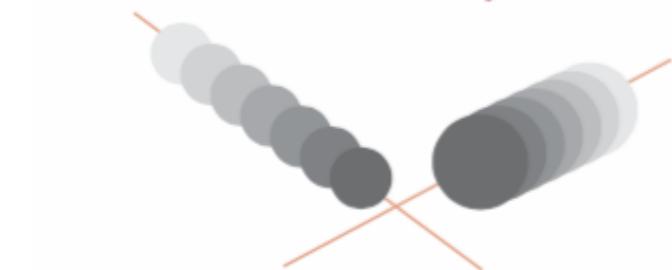
Event-Driven Simulation

Time-driven simulation

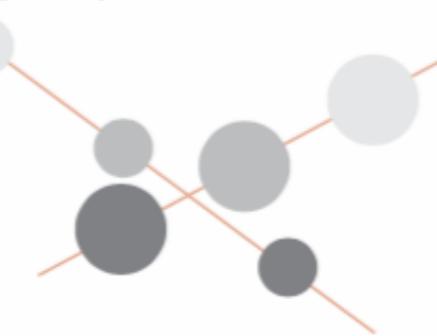
Main drawbacks.

- $\sim N^2 / 2$ overlap checks per time quantum.
- Simulation is too slow if dt is very small.
- May miss collisions if dt is too large.
(if colliding particles fail to overlap when we are looking)

dt too small: excessive computation



dt too large: may miss collisions



Event-Driven Simulation

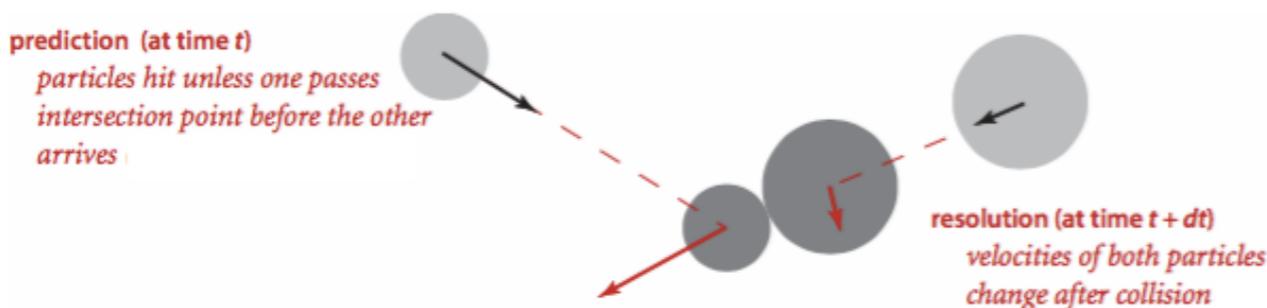
Event-driven simulation

Change state only when something happens.

- Between collisions, particles move in straight-line trajectories.
- Focus only on times when collisions occur.
- Maintain PQ of collision events, prioritized by time.
- Remove the min = get next collision.

Collision prediction. Given position, velocity, and radius of a particle, when will it collide next with a wall or another particle?

Collision resolution. If collision occurs, update colliding particle(s) according to laws of elastic collisions.

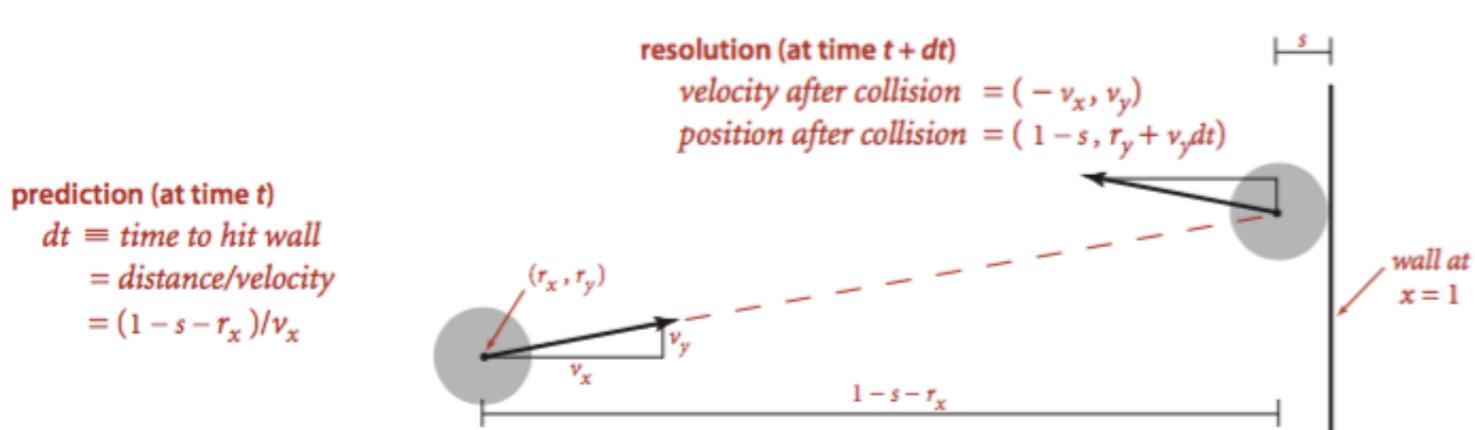


Event-Driven Simulation

Particle-wall collision

Collision prediction and resolution.

- Particle of radius s at position (r_x, r_y) .
- Particle moving in unit box with velocity (v_x, v_y) .
- Will it collide with a vertical wall? If so, when?



Predicting and resolving a particle-wall collision

Event-Driven Simulation

Particle-particle collision prediction

Collision prediction.

- Particle i : radius s_i , position (rx_i, ry_i) , velocity (vx_i, vy_i) .
- Particle j : radius s_j , position (rx_j, ry_j) , velocity (vx_j, vy_j) .
- Will particles i and j collide? If so, when?

$$\Delta t = \begin{cases} \infty & \text{if } \Delta v \cdot \Delta r \geq 0 \\ \infty & \text{if } d < 0 \\ -\frac{\Delta v \cdot \Delta r + \sqrt{d}}{\Delta v \cdot \Delta v} & \text{otherwise} \end{cases}$$

$$d = (\Delta v \cdot \Delta r)^2 - (\Delta v \cdot \Delta v) (\Delta r \cdot \Delta r - \sigma^2) \quad \sigma = \sigma_i + \sigma_j$$

$$\Delta v = (\Delta vx, \Delta vy) = (vx_i - vx_j, vy_i - vy_j)$$

$$\Delta r = (\Delta rx, \Delta ry) = (rx_i - rx_j, ry_i - ry_j)$$

$$\Delta v \cdot \Delta v = (\Delta vx)^2 + (\Delta vy)^2$$

$$\Delta r \cdot \Delta r = (\Delta rx)^2 + (\Delta ry)^2$$

$$\Delta v \cdot \Delta r = (\Delta vx)(\Delta rx) + (\Delta vy)(\Delta ry)$$

Important note: This is physics, so we won't be testing you on it!

Event-Driven Simulation

Particle-particle collision resolution

Collision resolution. When two particles collide, how does velocity change?

$$\begin{aligned} v_{x_i}' &= v_{x_i} + J_x / m_i \\ v_{y_i}' &= v_{y_i} + J_y / m_i \\ v_{x_j}' &= v_{x_j} - J_x / m_j \\ v_{y_j}' &= v_{y_j} - J_y / m_j \end{aligned}$$

Newton's second law
(momentum form)

$$J_x = \frac{J \Delta r_x}{\sigma}, \quad J_y = \frac{J \Delta r_y}{\sigma}, \quad J = \frac{2m_i m_j (\Delta v \cdot \Delta r)}{\sigma(m_i + m_j)}$$

impulse due to normal force
(conservation of energy, conservation of momentum)

Important note: This is physics, so we won't be testing you on it!

Event-Driven Simulation

Particle data type skeleton

```
public class Particle
{
    private double rx, ry;          // position
    private double vx, vy;          // velocity
    private final double radius;    // radius
    private final double mass;      // mass
    private int count;              // number of collisions

    public Particle(...) { }

    public void move(double dt) { }
    public void draw() { }

    public double timeToHit(Particle that) { }
    public double timeToHitVerticalWall() { }
    public double timeToHitHorizontalWall() { }

    public void bounceOff(Particle that) { }
    public void bounceOffVerticalWall() { }
    public void bounceOffHorizontalWall() { }

}
```

predict collision
with particle or wall

resolve collision
with particle or wall

Event-Driven Simulation

Particle-particle collision and resolution implementation

```
public double timeToHit(Particle that)
{
    if (this == that) return INFINITY;
    double dx = that.rx - this.rx, dy = that.ry - this.ry;
    double dvx = that.vx - this.vx; dvy = that.vy - this.vy;
    double dvdr = dx*dvx + dy*dvy;
    if( dvdr > 0) return INFINITY; ← no collision
    double dvdv = dvx*dvx + dvy*dvy;
    double drdr = dx*dx + dy*dy;
    double sigma = this.radius + that.radius;
    double d = (dvdr*dvdr) - dvdv * (drdr - sigma*sigma);
    if (d < 0) return INFINITY;
    return -(dvdr + Math.sqrt(d)) / dvdv;
}

public void bounceOff(Particle that)
{
    double dx = that.rx - this.rx, dy = that.ry - this.ry;
    double dvx = that.vx - this.vx, dvy = that.vy - this.vy;
    double dvdr = dx*dvx + dy*dvy;
    double dist = this.radius + that.radius;
    double J = 2 * this.mass * that.mass * dvdr / ((this.mass + that.mass) * dist);
    double Jx = J * dx / dist;
    double Jy = J * dy / dist;
    this.vx += Jx / this.mass;
    this.vy += Jy / this.mass;
    that.vx -= Jx / that.mass;
    that.vy -= Jy / that.mass;
    this.count++;
    that.count++;   Important note: This is physics, so we won't be testing you on it!
}
```

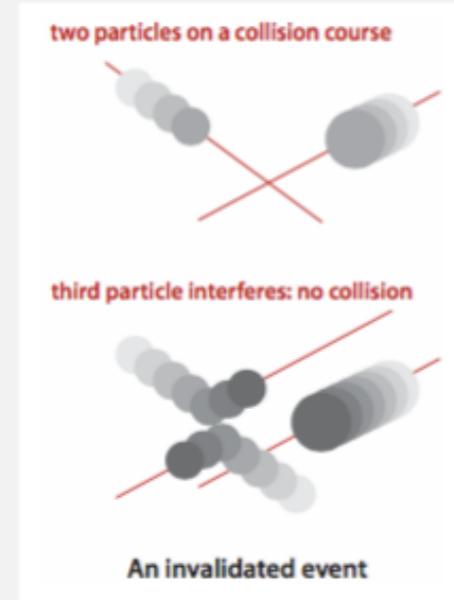
Event-Driven Simulation

Collision system: event-driven simulation main loop

Initialization.

- Fill PQ with all potential particle-wall collisions.
- Fill PQ with all potential particle-particle collisions.

"potential" since collision may not happen if some other collision intervenes



An invalidated event

Main loop.

- Delete the impending event from PQ (min priority = t).
- If the event has been invalidated, ignore it.
- Advance all particles to time t , on a straight-line trajectory.
- Update the velocities of the colliding particle(s).
- Predict future particle-wall and particle-particle collisions involving the colliding particle(s) and insert events onto PQ.

Event-Driven Simulation

Event data type

Conventions.

- Neither particle null \Rightarrow particle-particle collision.
- One particle null \Rightarrow particle-wall collision.
- Both particles null \Rightarrow redraw event.

```
private class Event implements Comparable<Event>
{
    private double time;          // time of event
    private Particle a, b;        // particles involved in event
    private int countA, countB;   // collision counts for a and b

    public Event(double t, Particle a, Particle b) { }           ← create event

    public int compareTo(Event that)
    {   return this.time - that.time; }                           ← ordered by time

    public boolean isValid()
    {   }
}
```

← invalid if intervening collision

Event-Driven Simulation

Collision system implementation: skeleton

```
public class CollisionSystem
{
    private MinPQ<Event> pq;          // the priority queue
    private double t = 0.0;              // simulation clock time
    private Particle[] particles;       // the array of particles

    public CollisionSystem(Particle[] particles) { }

    private void predict(Particle a)      add to PQ all particle-wall and particle-
    {                                     -particle collisions involving this particle
        if (a == null) return;
        for (int i = 0; i < N; i++)
        {
            double dt = a.timeToHit(particles[i]);
            pq.insert(new Event(t + dt, a, particles[i]));
        }
        pq.insert(new Event(t + a.timeToHitVerticalWall() , a, null));
        pq.insert(new Event(t + a.timeToHitHorizontalWall(), null, a));
    }

    private void redraw() { }

    public void simulate() { /* see next slide */ }
}
```

Event-Driven Simulation

Collision system implementation: main event-driven simulation loop

```
public void simulate()
{
    pq = new MinPQ<Event>();
    for(int i = 0; i < N; i++) predict(particles[i]);
    pq.insert(new Event(0, null, null));
```

initialize PQ with collision events and redraw event

```
    while(!pq.isEmpty())
    {
        Event event = pq.delMin();
        if(!event.isValid()) continue;
        Particle a = event.a;
        Particle b = event.b;
```

get next event

```
        for(int i = 0; i < N;
            particles[i].move(event.time - t);
        t = event.time;
```

update positions and time

```
        if      (a != null && b != null) a.bounceOff(b);
        else if (a != null && b == null) a.bounceOffVerticalWall();
        else if (a == null && b != null) b.bounceOffHorizontalWall();
        else if (a == null && b == null) redraw();
```

process event

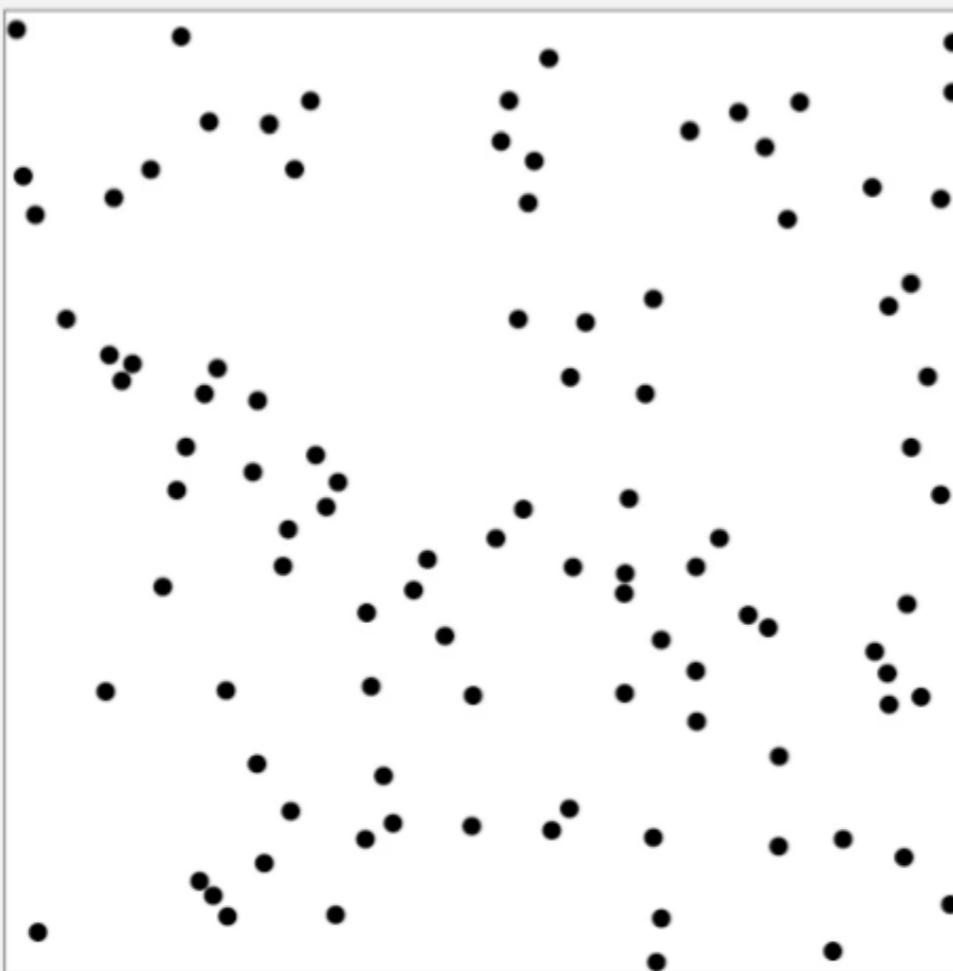
```
        predict(a);
        predict(b);
    }
}
```

predict new events based on changes

Event-Driven Simulation

Particle collision simulation example 1

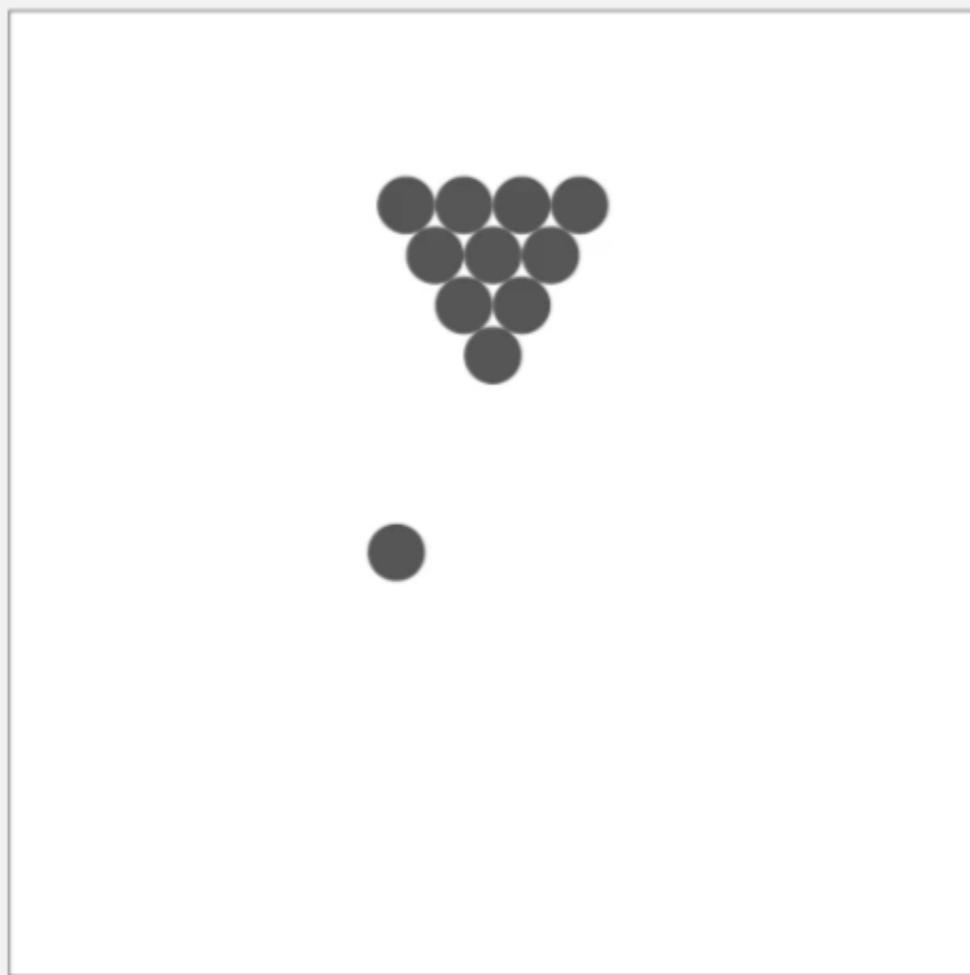
```
% java CollisionSystem 100
```



Event-Driven Simulation

Particle collision simulation example 2

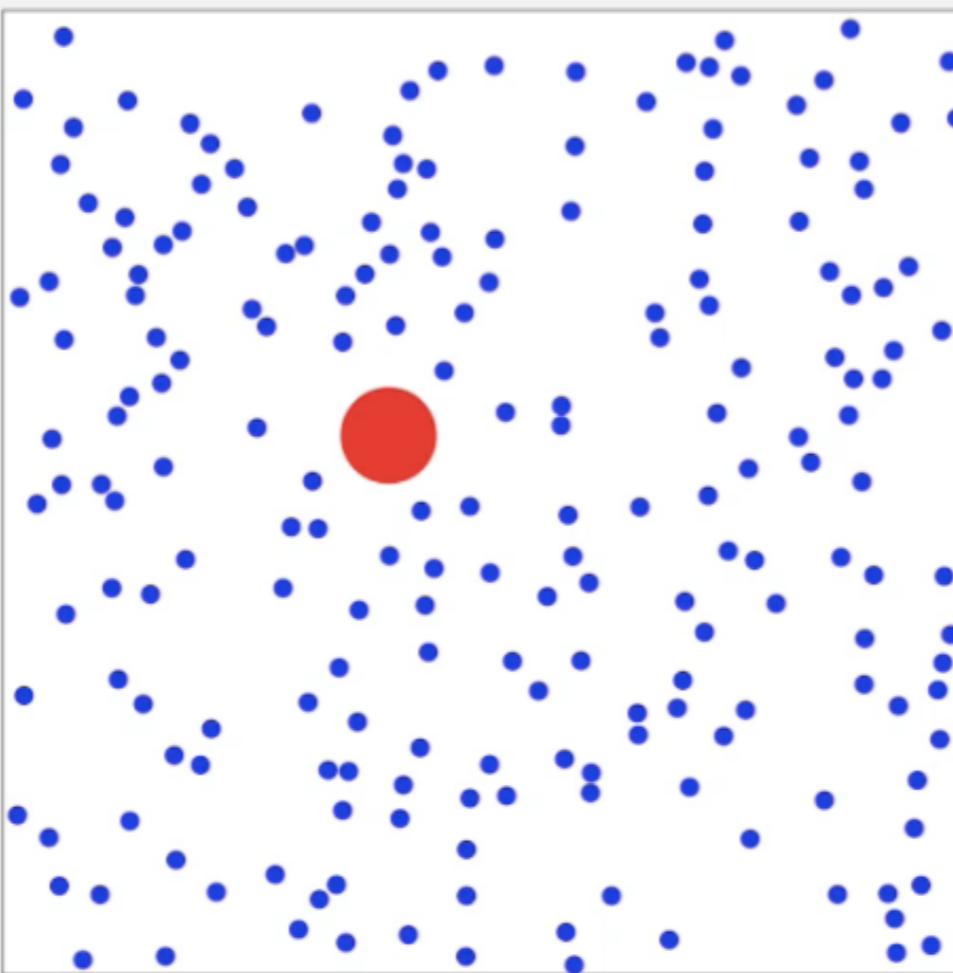
```
% java CollisionSystem < billiards.txt
```



Event-Driven Simulation

Particle collision simulation example 3

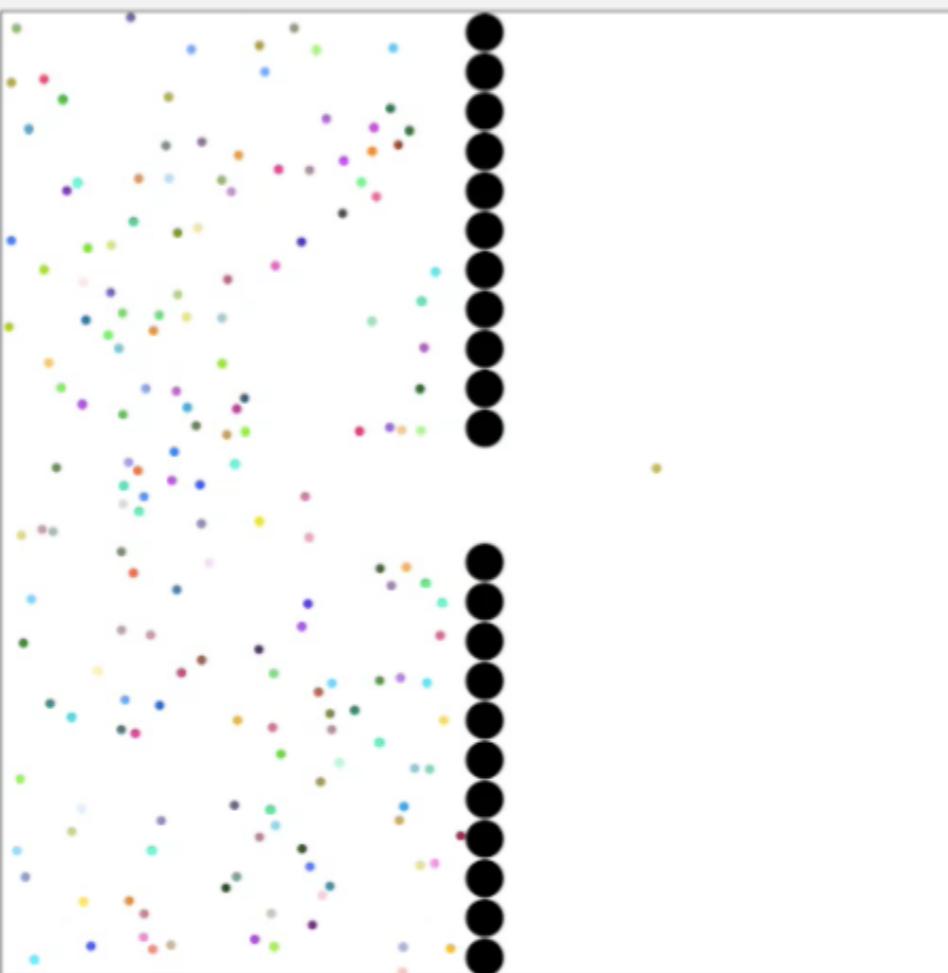
```
% java CollisionSystem < brownian.txt
```



Event-Driven Simulation

Particle collision simulation example 4

```
% java CollisionSystem < diffusion.txt
```



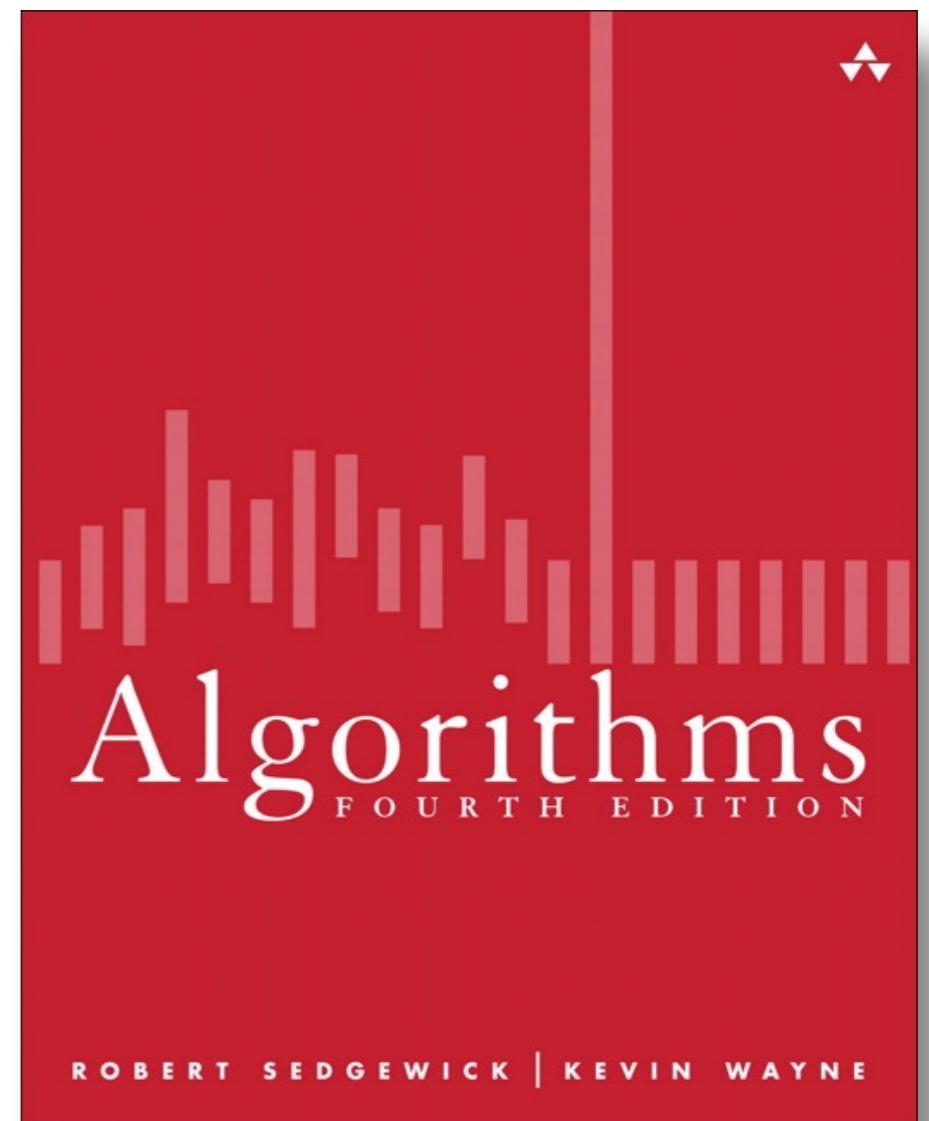
Homework 6

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the sort by options (with 0 being exit):

```
Sort by lastname  :[1]
Sort by home state :[2]
Sort by age      :[3]
Enter option or 0 to end imput: 0
Exiting...
```

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age
- Also recognize an invalid entry

```
Sort by lastname  :[1]
Sort by home state :[2]
Sort by age      :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```



Assignment due next Monday at 11:59 PM

Homework 6

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the sort by option (exit):

```
Sort by lastname :[1]
Sort by home state :[2]
Sort by age :[3]
Enter option or 0 to end input:
Exiting...
```

```
/*
 * This class will have 3 properties:
 *   - String
 *   - String
 *   - Integer
 */
public class Contact {
```

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age
- Also recognize an invalid entry

```
Sort by lastname :[1]
Sort by home state :[2]
Sort by age :[3]
Enter option or 0 to end input: 4
Invalid entry.
```



Assignment due next Monday at 11:59 PM

Homework 6

- Assignment - Write a program that will have; using the Contact class you created, sort at least 5 Contacts using Comparable interface and implement the requirements:
- A prompt for the user to enter their option (0 = exit):

```
Sort by last name
Sort by home state
Sort by age
Enter option (0 to exit)
Exiting..
```

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age
- Also recognize 0 = Exit

```
Sort by last name
Sort by home state
Sort by age
Enter option (1, 2, 3 or 0 to exit)
Invalid entry
```

```
import java.util.ArrayList;

public class TestSortOptions {

    public static void main(String[] args) {
        ArrayList<Contact> contacts = initializeContactsArray();
        promptForOption(contacts);
    }

    /*
     * Data Initialization
     */

    private static ArrayList<Contact> initializeContactsArray() {
        // TODO: Initialize an array of Student objects
        return null;
    }

    /*
     * Prompt for the user to enter their option from the keyboard
     *
     * 1 = Sort by last name
     * 2 = Sort by Home State
     * 3 = Sort by Age
     * 0 = End input and exit/terminate the application
     */
    private static void promptForOption(ArrayList<Contact> contacts) {
        // TODO: Prompt and accept option input
    }

    /*
     * Display the Contact information sorted by using the selected option from
     * the above "promptForOption" method result
     */
    private static void displayContacts(ArrayList<Contact> contacts) {
        // TODO: Display the contents of the Contacts Array
    }
}
```

ithms
THE EDITION

K | KEVIN WAYNE

Assignment due next Monday at 11:59 PM

Homework 6

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the sort by options (with 0 being exit):

```
Sort by lastname :[1]
Sort by home state :[2]
Sort by age :[3]
Enter option or 0 to end imput: 0
Exiting...
```

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age
- Also recognize an invalid entry

```
Sort by lastname :[1]
Sort by home state :[2]
Sort by age :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```



Assignment due next Monday at 11:59 PM

Homework 6

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the sort by options (with 0 being exit):

```
Sort by lastname :[1]
Sort by home state :[2]
Sort by age :[3]
Enter option or 0 to end imput: 0
Exiting...
```

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age
- Also recognize an invalid entry

```
Sort by lastname :[1] face [firstname=Bill, lastname=Barnes, homestate=OH, age=62]
Sort by home state :[2] face [firstname=Adam, lastname=Ant, homestate=MI, age=14]
Sort by age :[3] face [firstname=Jane, lastname=Doe, homestate=CA, age=41]
Enter option or 0 to end imput: 4
Invalid entry.
```



Assignment due next Monday at 11:59 PM

Homework 6

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the sort by options (with 0 being exit):

```
Sort by lastname :[1]
Sort by home state :[2]
Sort by age :[3]
Enter option or 0 to end imput: 0
Exiting...
```

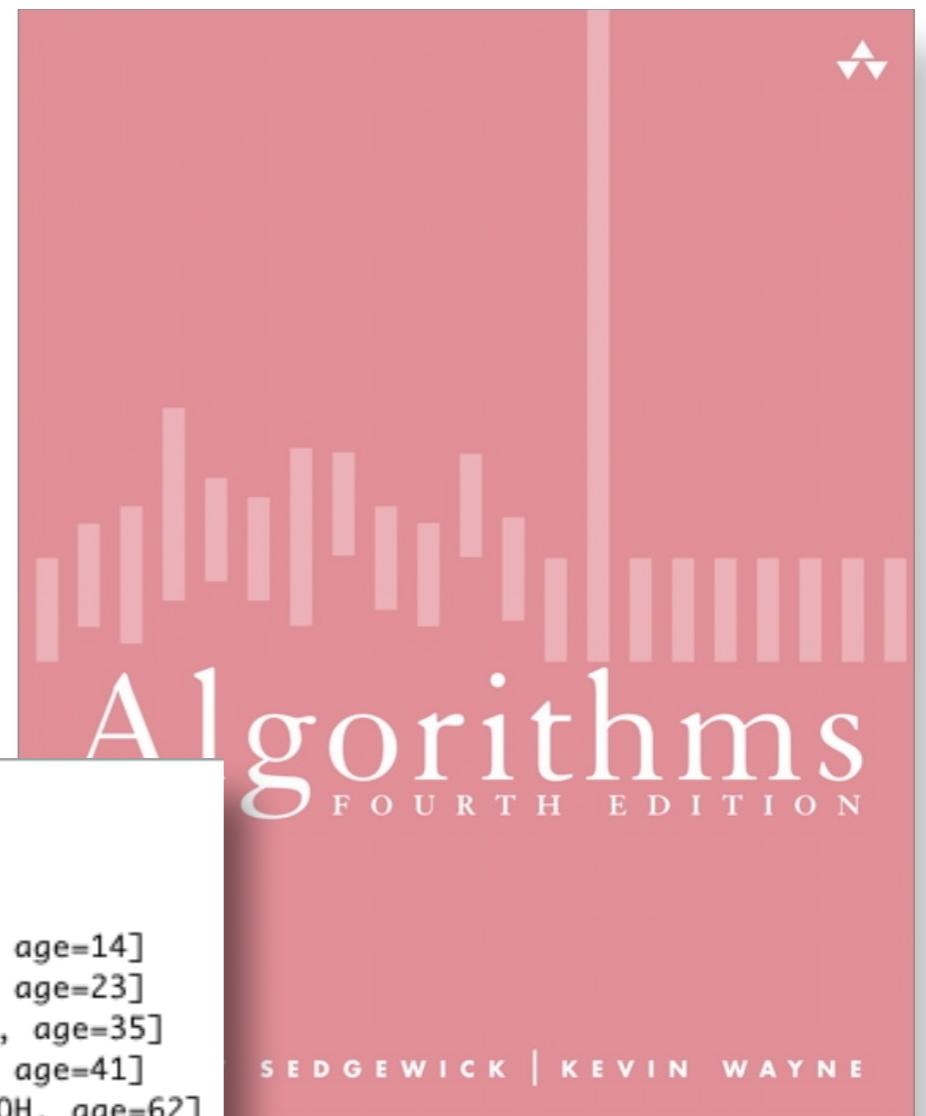
Exit now...

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age
- Also recognize a 0 to exit

```
Sort by lastname :[1]
Sort by home state :[2]
Sort by age :[3]
Enter option or 0 to end imput: 3
Contact [firstname=Adam, lastname=Ant, homestate=MI, age=14]
Contact [firstname=Ida, lastname=Know, homestate=FL, age=23]
Contact [firstname=Joe, lastname=Jones, homestate=IL, age=35]
Contact [firstname=Jane, lastname=Doe, homestate=CA, age=41]
Contact [firstname=Bill, lastname=Barnes, homestate=OH, age=62]
```

```
Sort by lastname :[1]
Sort by home state :[2]
Sort by age :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```

Enter a valid entry:



Assignment due next Monday at 11:59 PM