Data Structures
Chapter 5: Heap and Priority Queue

1. Heap & Priority Queue
2. **Heapsort**
3. Heap & PQ Coding

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

내가 그리스도와 함께 십자가에 못 박혔나니 그런즉 이제는 내가 사는 것이 아니요 오직 내 안에 그리스도께서 사시는 것이라 이제 내가 육체 가운데 사는 것은 나를 사랑하사 나를 위하여 자기 자신을 버리신 하나님의 아들을 믿는 믿음 안에서 사는 것이라 (갈라디아서 2:20)

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm304, Handong Global University

Data Structures
Chapter 5: Heap and Priority Queue

1. Heap & Priority Queue
2. **Heapsort**
   - Heap Construction – Heapify
   - Heapsort
   - Time Complexity
3. Heap & PQ Coding

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

Heapsort

Basic plan for in-place sort

- **1st Pass**: Build maxheap with all **N** keys.
- **2nd Pass**: Repeatedly remove the maximum key.

unsorted	C	H	R	I	S	T	A	L	O	N	E

sorted	A	C	E	H	I	L	N	O	R	S	T
	1	2	3	4	5	6	7	8	9	10	11

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

4

Heapsort

Basic plan for in-place sort

- **1st Pass**: Build maxheap with all **N** keys.

- **2nd Pass**: Repeatedly remove the maximum key.

an array of **N** keys
in arbitrary order

1st Pass ⇩

build a maxheap
(in place)

2nd Pass ⇩

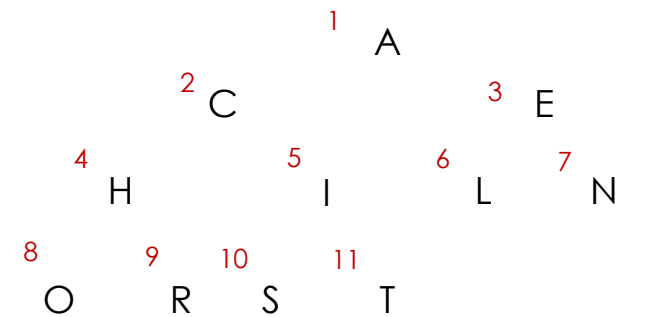sorted in place
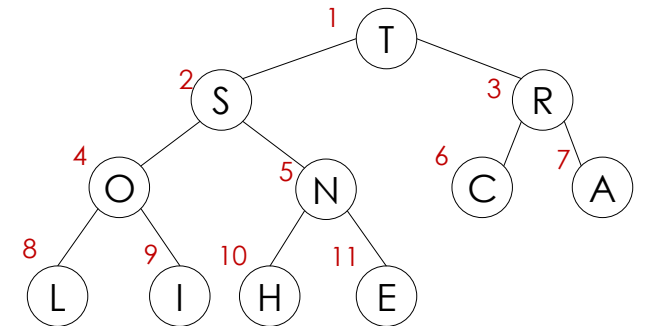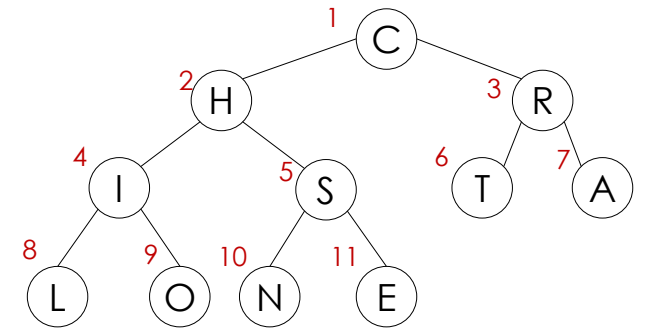
unsorted	C	H	R	I	S	T	A	L	O	N	E
heap-ordered	T	S	R	O	N	C	A	L	I	H	E
sorted	A	C	E	H	I	L	N	O	R	S	T
	1	2	3	4	5	6	7	8	9	10	11

Heapsort

Basic plan for in-place sort

- **1st Pass**: Build maxheap with all **N** keys.

- **2nd Pass**: Repeatedly remove the maximum key.

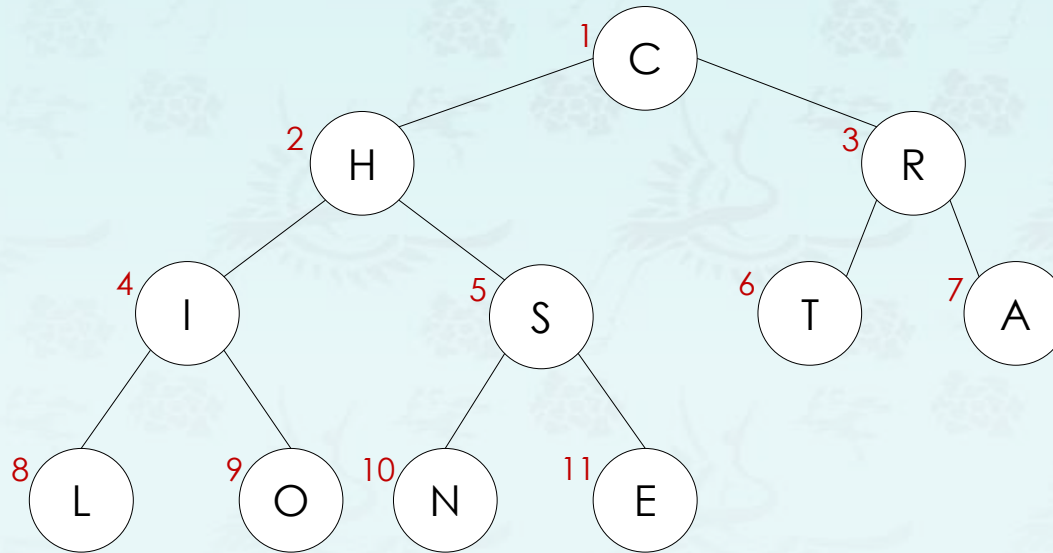|	C	H	R	I	S	T	A	L	O	N	E
0	1	2	3	4	5	6	7	8	9	10	11
											N

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

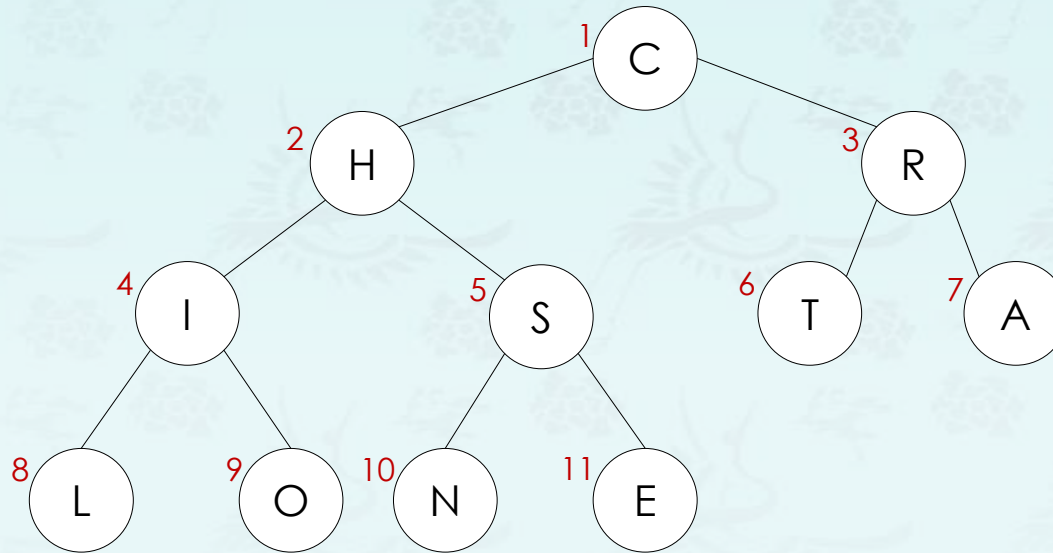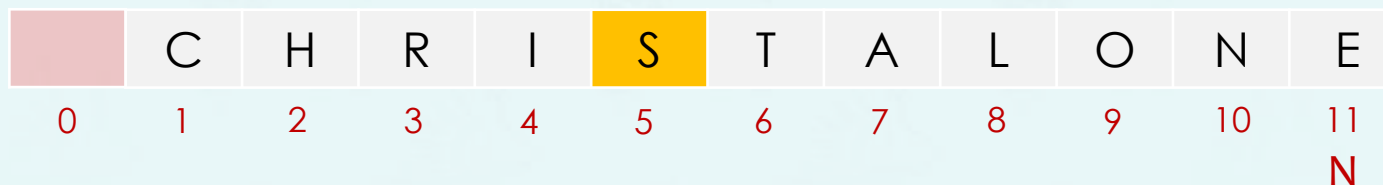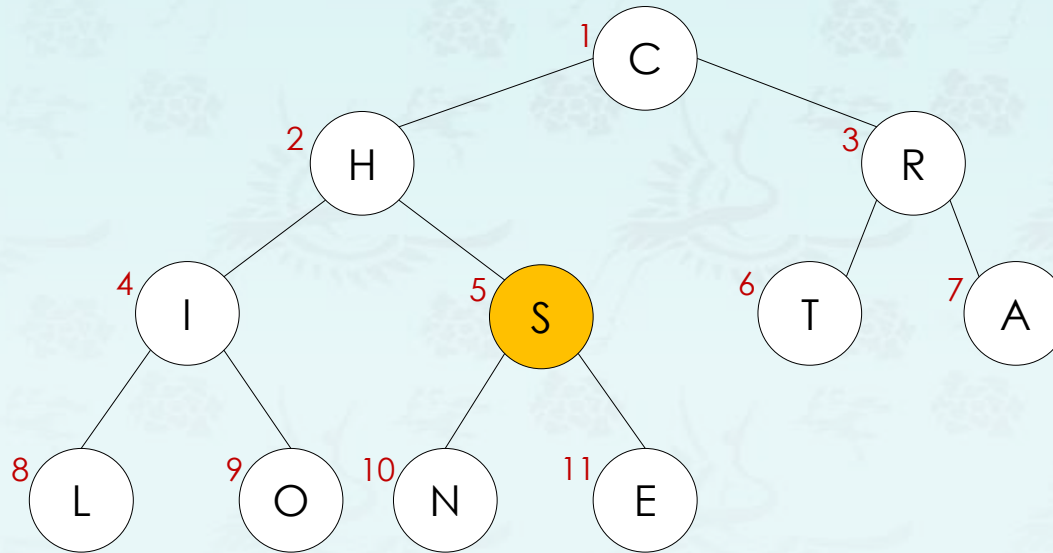|	C	H	R	I	S	T	A	L	O	N	E
0	1	2	3	4	5	6	7	8	9	10	11

N

Heapsort

1st Pass: Heap construction (heapify)

Where should we start from?

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

	C	H	R	I	S	T	A	L	O	N	E
0	1	2	3	4	5	6	7	8	9	10	11

N

Heapsort

1ˢᵗ Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

Where should we start from?

- root, leaf, or any particular node?
- Since leaf nodes are already 1-node heap, go up and to the left node which is not a leaf which is called _____ .
- How do you locate the **last internal node** for a tree of N nodes?

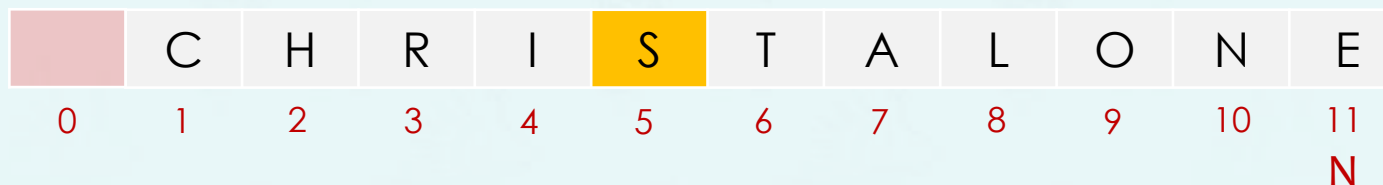*Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University*

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

> Where should we start from?

- root, leaf, or any particular node?
- Since leaf nodes are already 1-node heap, go up and to the left node which is not a leaf which is called _____ .
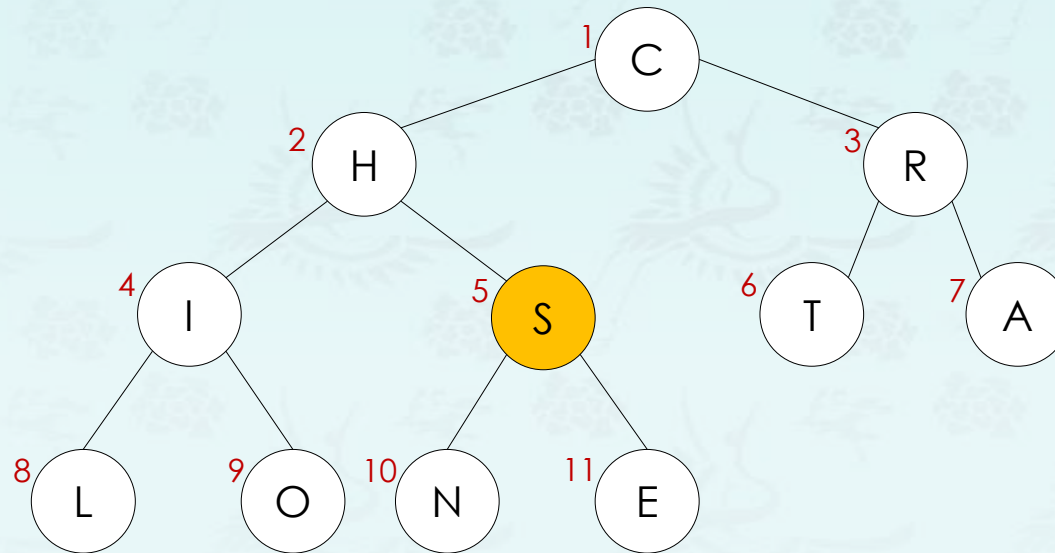- How do you locate the **last internal node** for a tree of N nodes? **floor(N / 2)**

Heapsort

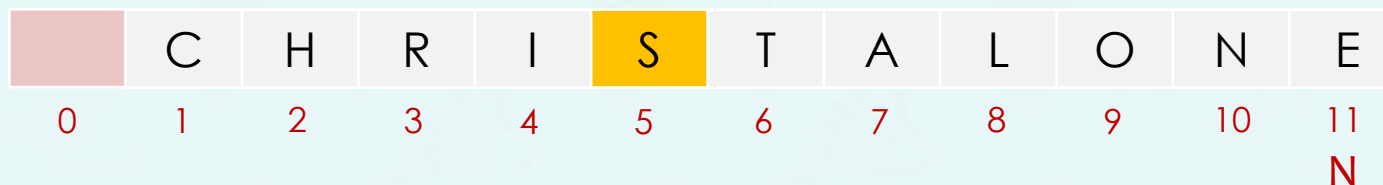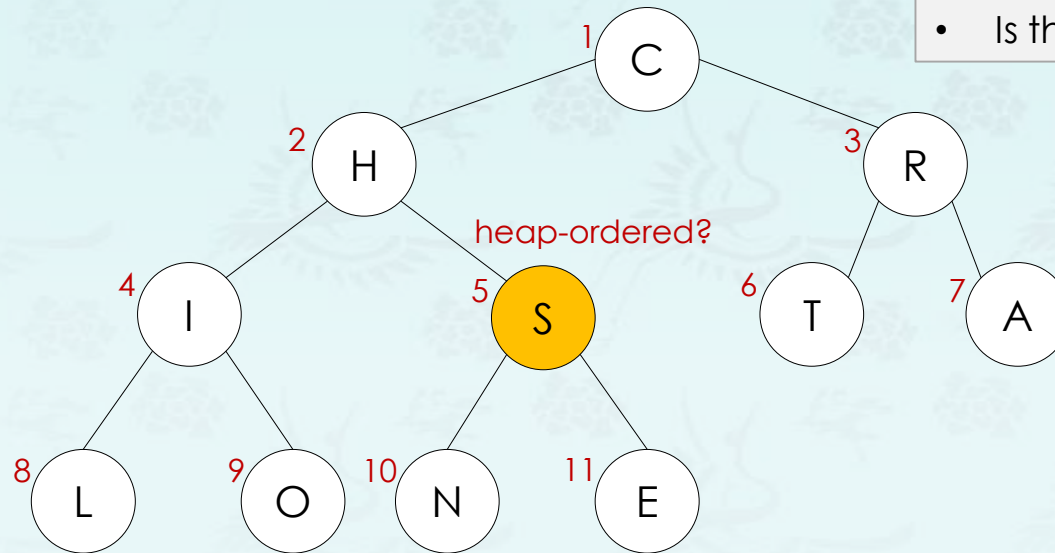## 1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

> Where should we start from?

- root, leaf, or any particular node?
- Since leaf nodes are already 1-node heap, go up and to the left node which is not a leaf which is called _____ .
- How do you locate the **last internal node** for a tree of N nodes? **floor(N / 2)**
- Is this 3-node heap at 5 heap-ordered?

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

Where should we start from?

- root, leaf, or any particular node?
- Since leaf nodes are already 1-node heap, go up and to the left node which is not a leaf which is called _____ .
- How do you locate the **last internal node** for a tree of N nodes? **floor(N / 2)**
- Is this 3-node heap at 5 heap-ordered?

Recall: `root >= max(left, right)`

Heapsort

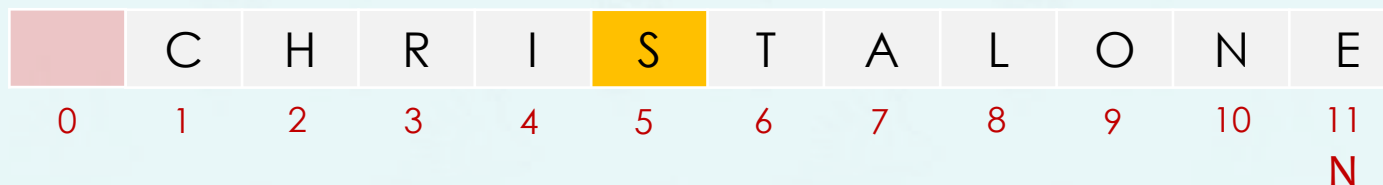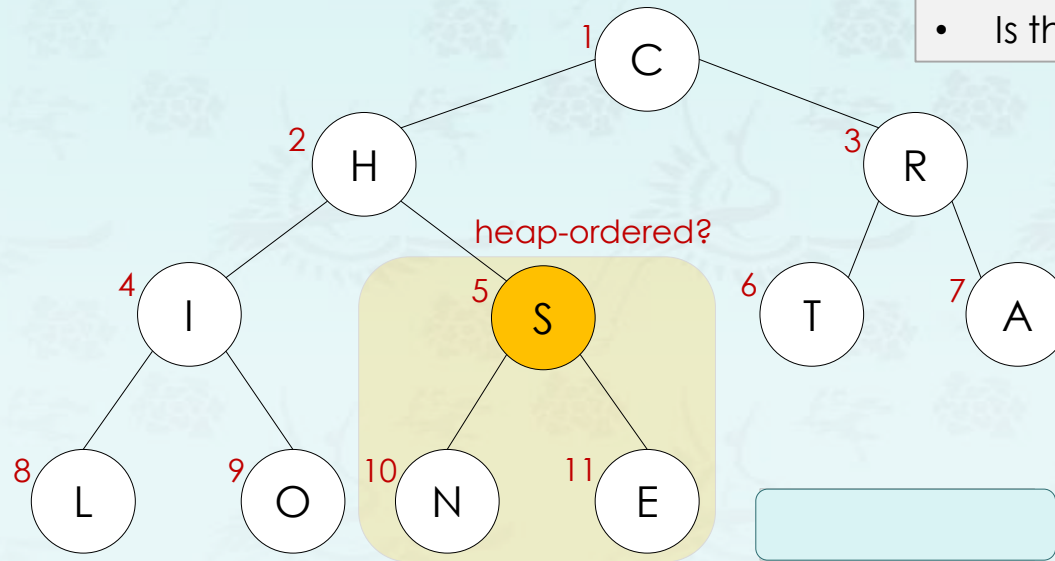## 1ˢᵗ Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

Where should we start from?

- root, leaf, or any particular node?
- Since leaf nodes are already 1-node heap, go up and to the left node which is not a leaf which is called _____.
- How do you locate the **last internal node** for a tree of N nodes? **floor(N / 2)**
- Is this 3-node heap at 5 heap-ordered?

Recall: `root >= max(left, right)`

heap-ordered?

```
1  C
2  H        3  R
4  I    5  S    6  T    7  A
8  L  9  O  10  N  11  E
```

	C	H	R	I	S	T	A	L	O	N	E
0	1	2	3	4	5	6	7	8	9	10	11

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University
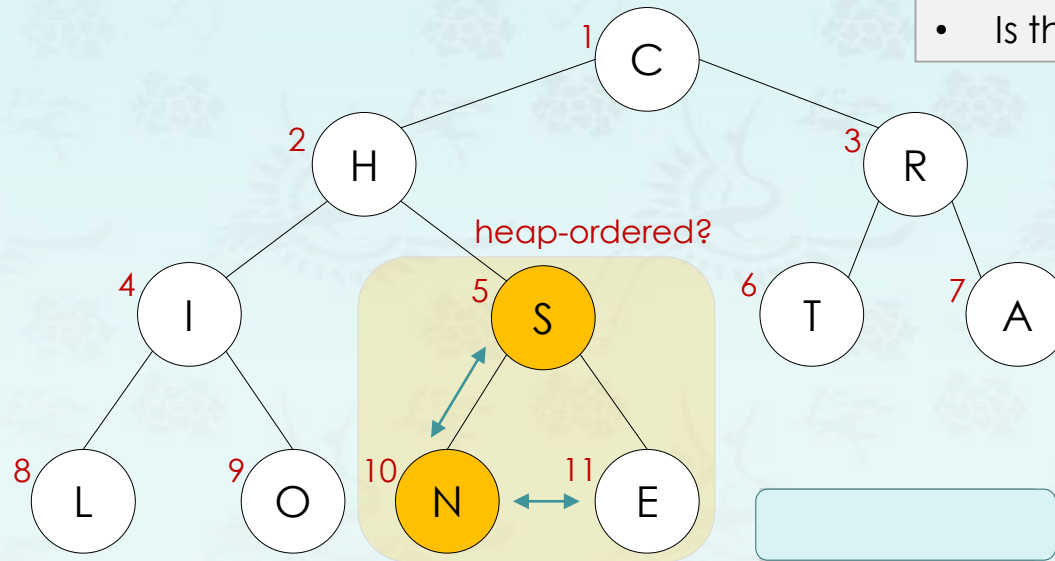
13

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

> Where should we start from?

- root, leaf, or any particular node?
- Since leaf nodes are already 1-node heap, go up and to the left node which is not a leaf which is called _____ .
- How do you locate the **last internal node** for a tree of N nodes? **floor(N / 2)**
- Is this 3-node heap at 5 heap-ordered?

Recall: `root >= max(left, right)`

3-node heap

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University
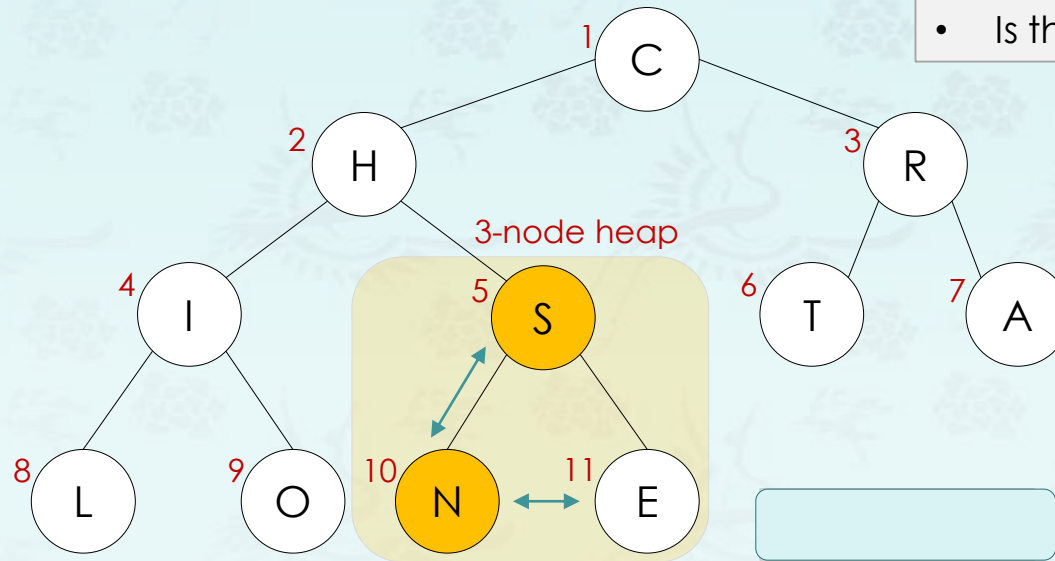
14

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;                              k=5, j=10
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;    k=5, j=10
    swap(h, k, j);
    k = j;
  }
}
```

3-node heap

sink(5, N)

	C	H	R	I	S	T	A	L	O	N	E
0	1	2	3	4	5	6	7	8	9	10	11

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University
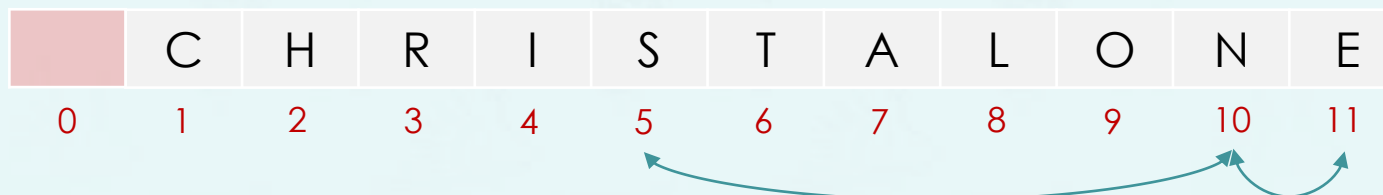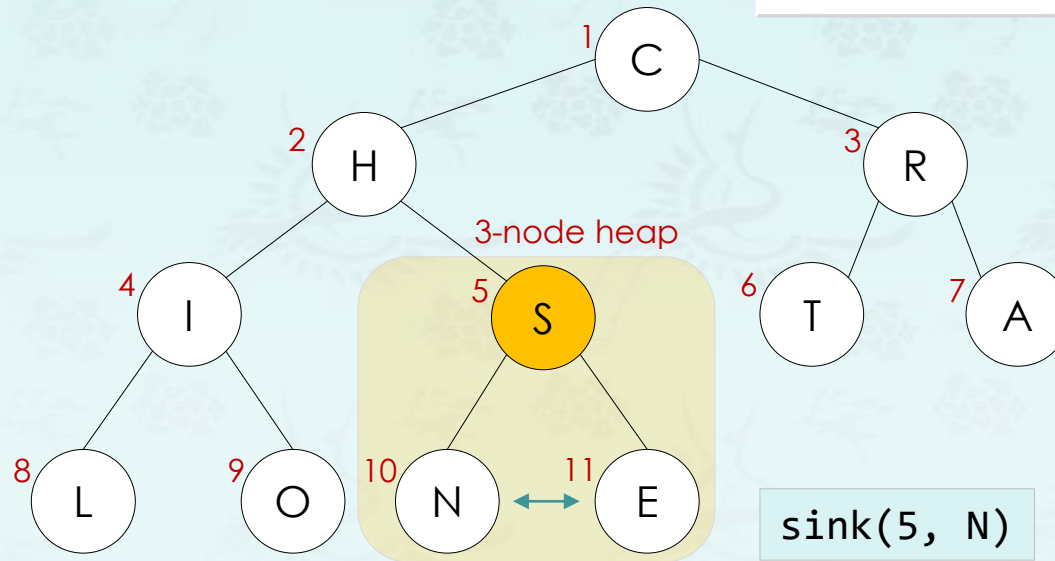
15

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;                              k=5, j=10
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;                  k=5, j=10
    swap(h, k, j);                              break
    k = j;
  }
}
```
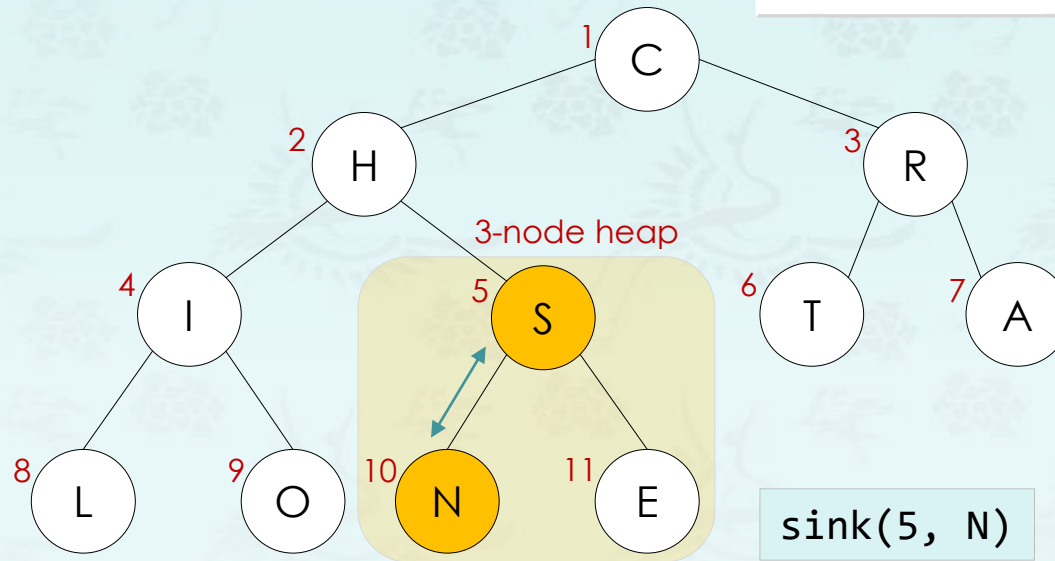
3-node heap

sink(5, N)

	C	H	R	I	S	T	A	L	O	N	E
0	1	2	3	4	5	6	7	8	9	10	11

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

16

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

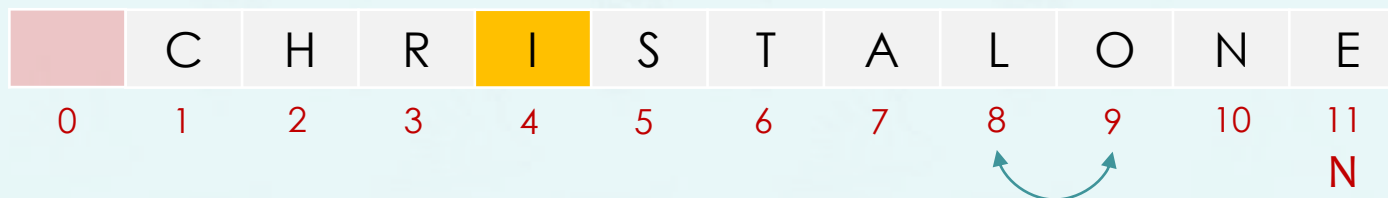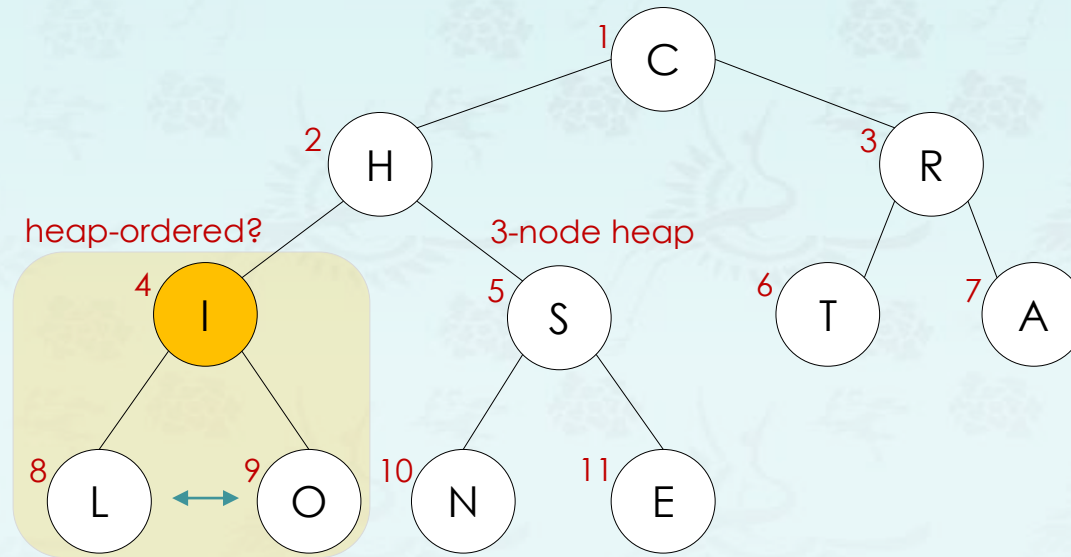# Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;
    swap(h, k, j);
    k = j;
  }
}
```

heap-ordered?   3-node heap

	C	H	R	I	S	T	A	L	O	N	E
0	1	2	3	4	5	6	7	8	9	10	11

N

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.
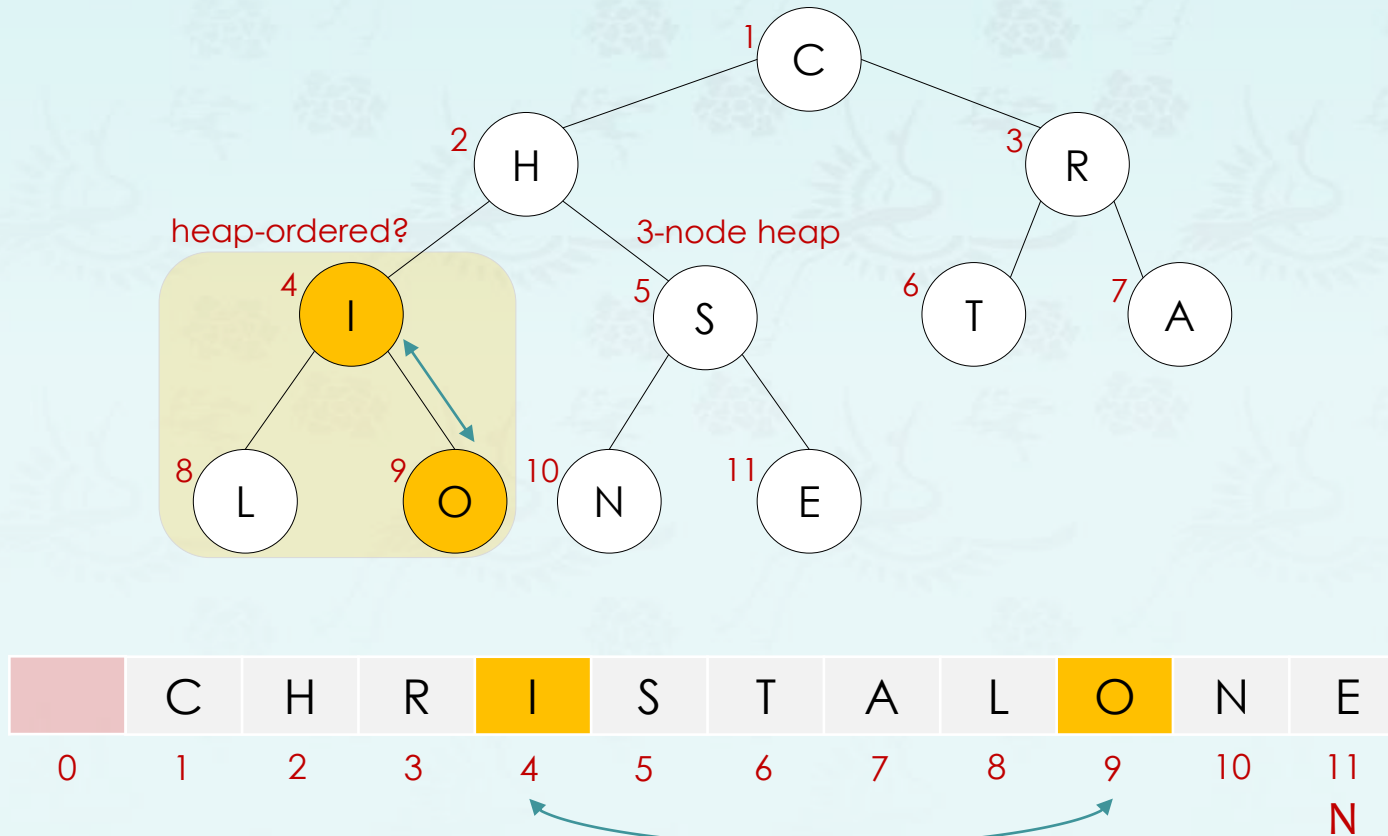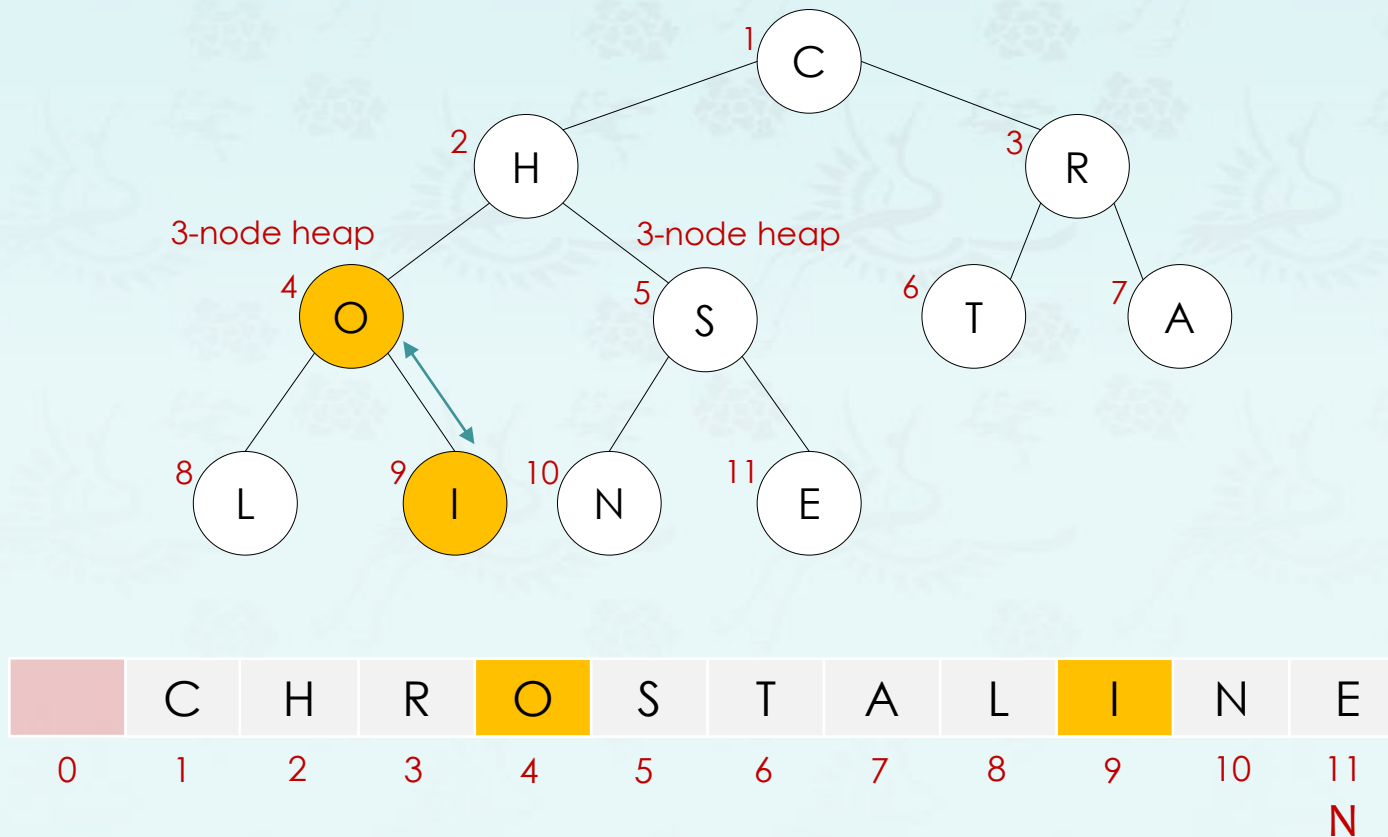
array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;                    k=4, j=8
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;   k=4, j=9
    swap(h, k, j);
    k = j;
  }
}
```


heap-ordered?   3-node heap

sink(4, N)

	C	H	R	I	S	T	A	L	O	N	E
0	1	2	3	4	5	6	7	8	9	10	11

N

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.
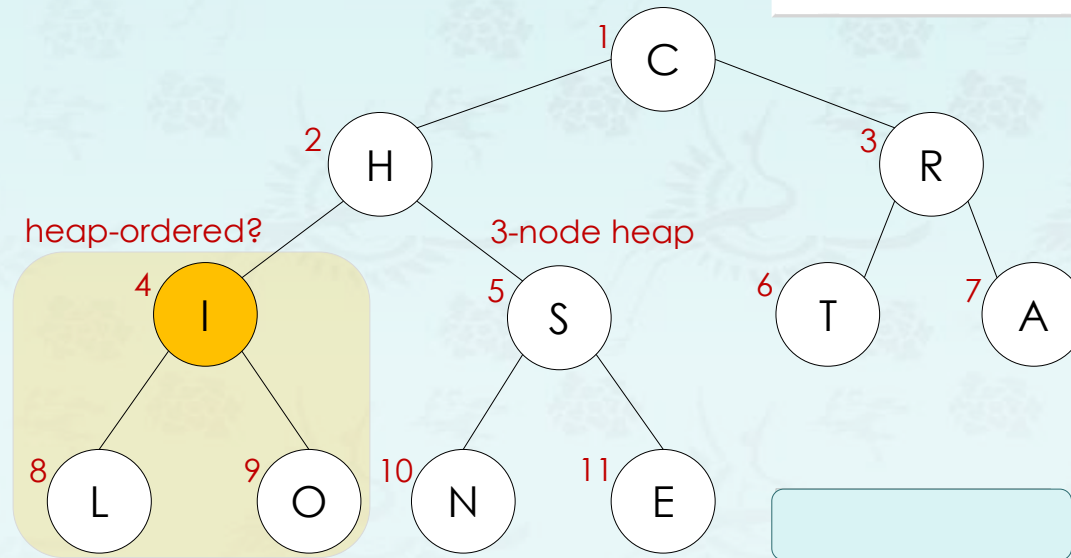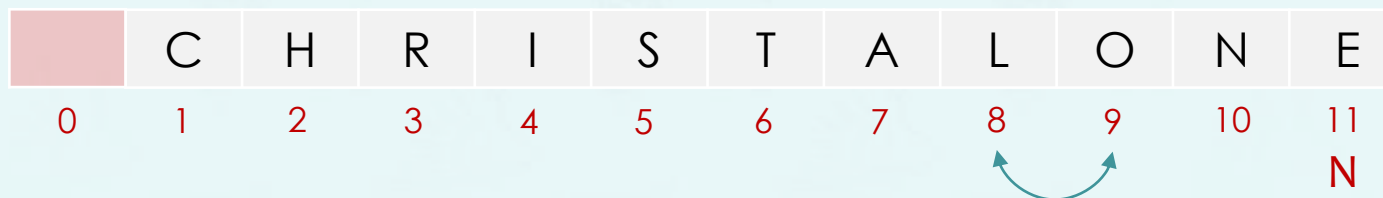
array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;            k=4, j=8
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;   k=4, j=9
    swap(h, k, j);
    k = j;
  }
}
```


heap-ordered?

3-node heap

sink(4, N)

	C	H	R	I	S	T	A	L	O	N	E
0	1	2	3	4	5	6	7	8	9	10	11

N

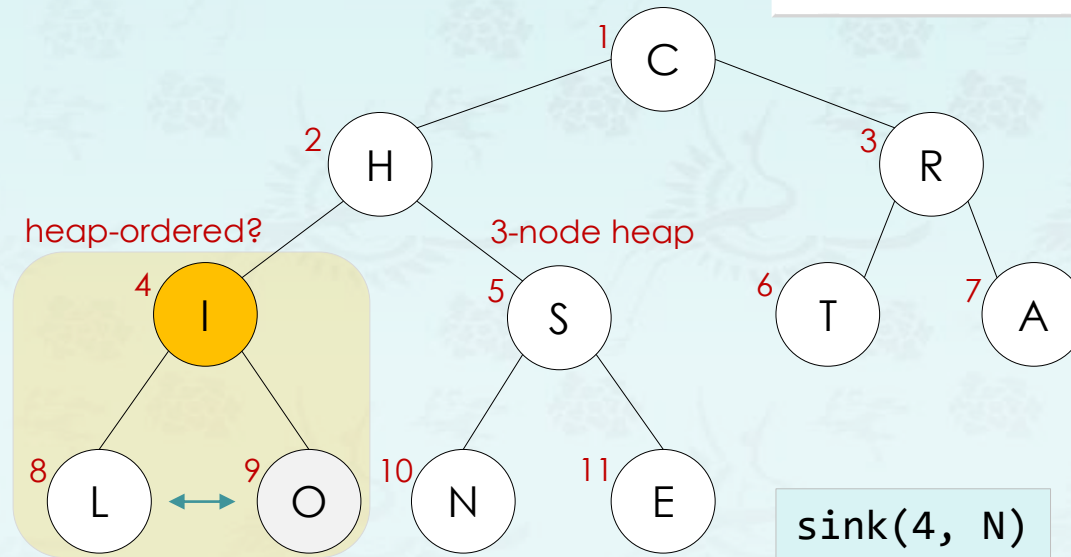# Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;                              k=4, j=8
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;   k=4, j=9
    swap(h, k, j);
    k = j;                                       k=9

  }                                              exit while()
}
```
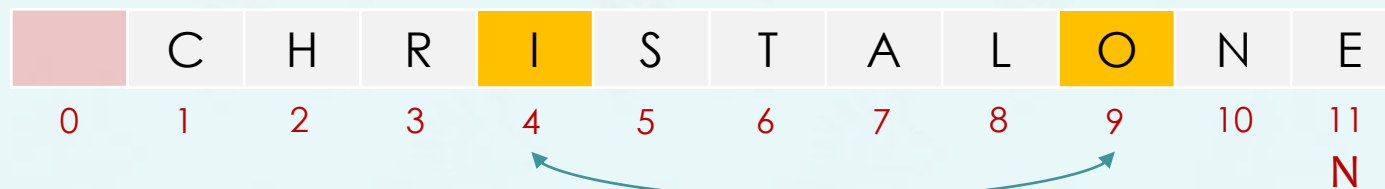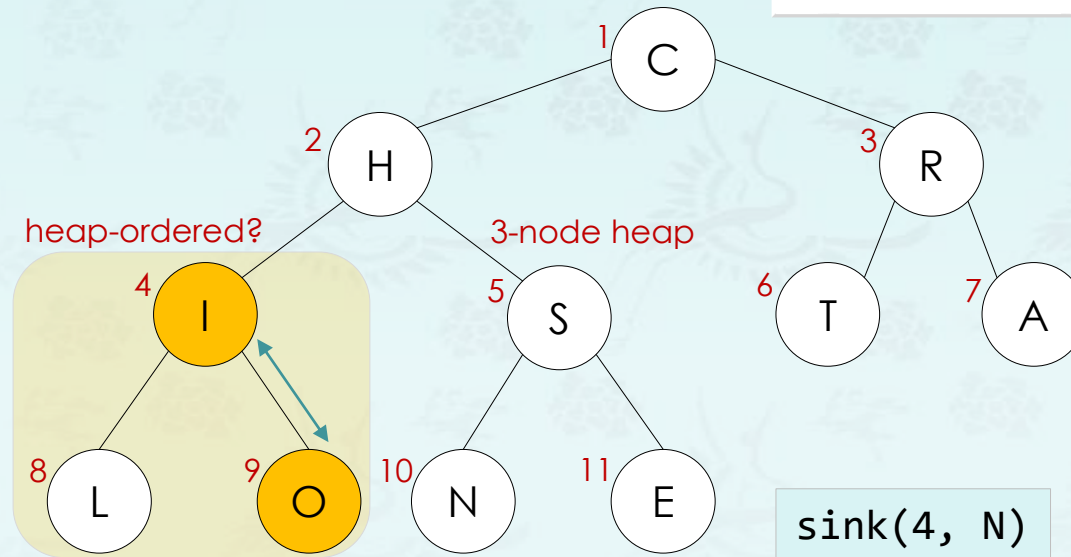
heap-ordered?

3-node heap

sink(4, N)

	C	H	R	O	S	T	A	L	I	N	E
0	1	2	3	4	5	6	7	8	9	10	11

N

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

23

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;                              k=4, j=8
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;   k=4, j=9
    swap(h, k, j);
    k = j;                       k=9
  }                             exit while()
}
```
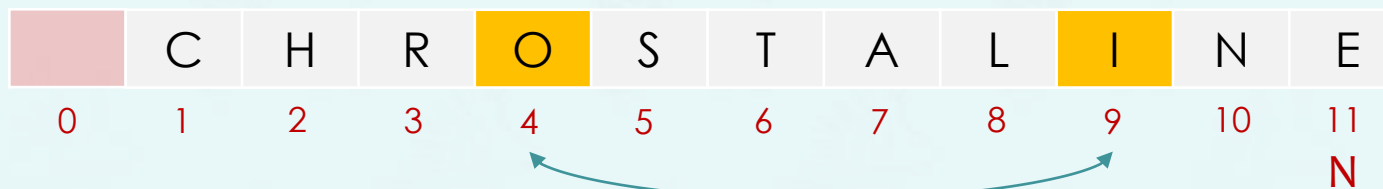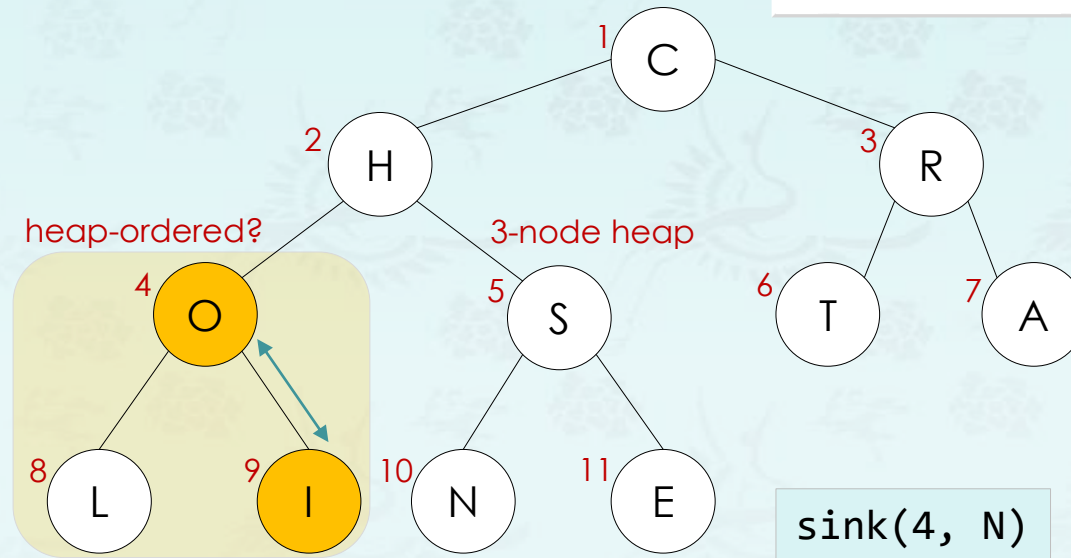

sink(4, N)

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

heap-ordered?

3-node heap   3-node heap

j or j++ ?

sink(3, N)

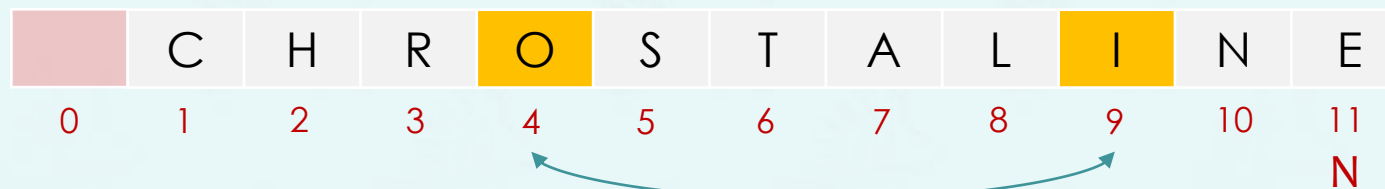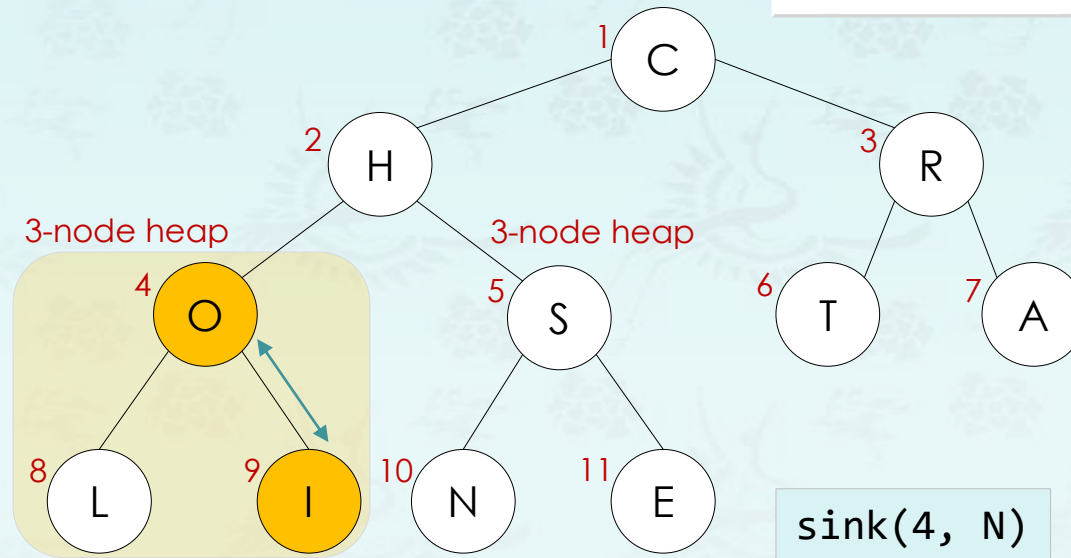*Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University*

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;                              k=3, j=6
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;   k=3, j=6
    swap(h, k, j);
    k = j;                       k=6
  }
                                 exit while()
}
```

heap-ordered?

3-node heap   3-node heap

sink(3, N)

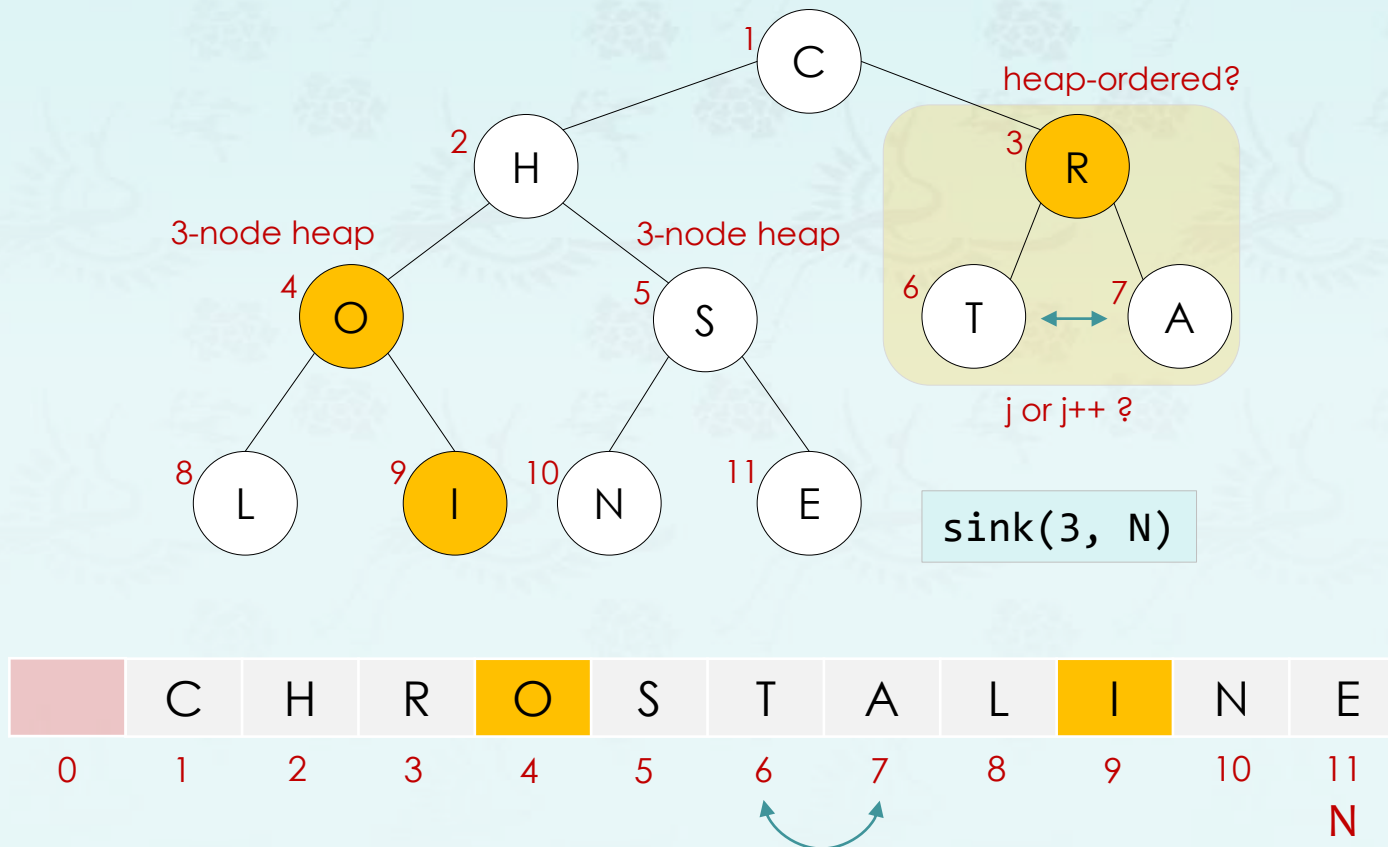|	C	H	R	O	S	T	A	L	I	N	E
0	1	2	3	4	5	6	7	8	9	10	11
											N

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;                          k=3, j=6
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;   k=3, j=6
    swap(h, k, j);
    k = j;                       k=6
  }
                                 exit while()
}
```


3-node heap

3-node heap

3-node heap

sink(3, N)

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
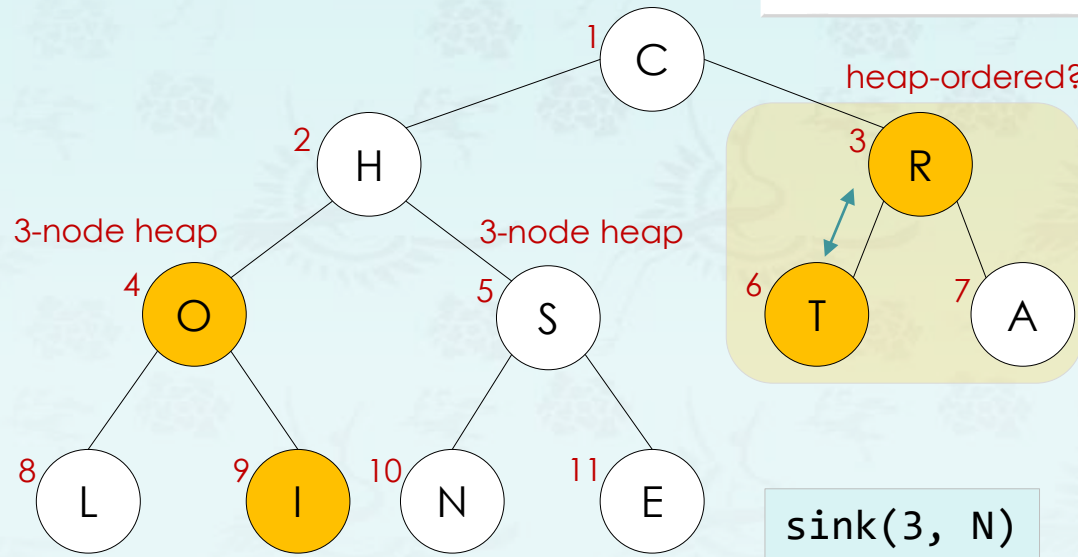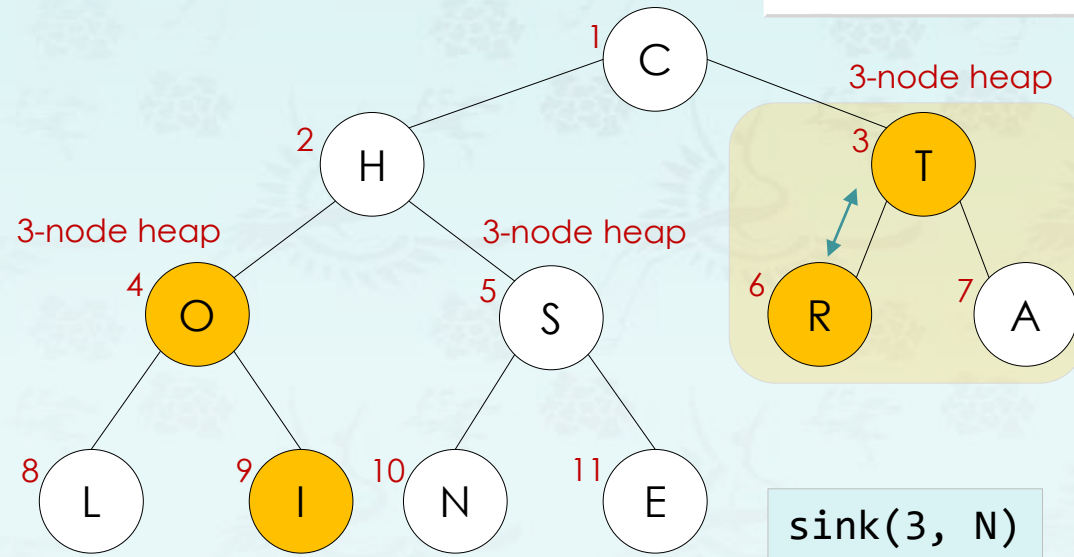- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;                              k=2, j=4
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;    k=2, j=5
    swap(h, k, j);
    k = j;                          k=5
  }                                 exit while()
}
```

heap-ordered?

3-node heap

3-node heap

3-node heap

3-node heap

sink(2, N)

	C	H	T	O	S	R	A	L	I	N	E
0	1	2	3	4	5	6	7	8	9	10	11

N

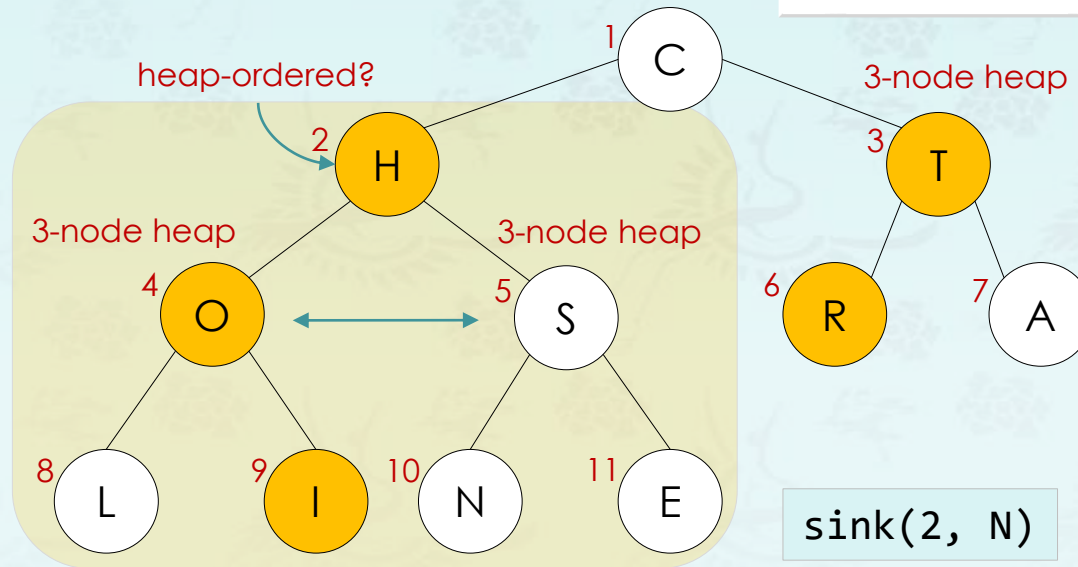Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

28

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;                          k=2, j=4
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;   k=2, j=5
    swap(h, k, j);
    k = j;                          k=5
  }                                 exit while()
}
```


heap-ordered?

3-node heap

3-node heap   3-node heap

sink(2, N)

	C	H	T	O	S	R	A	L	I	N	E
0	1	2	3	4	5	6	7	8	9	10	11
											N

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

29

Heapsort

1ˢᵗ Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;                              k=2, j=4
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;                  k=2, j=5
    swap(h, k, j);                              swap()
    k = j;
  }
}
```
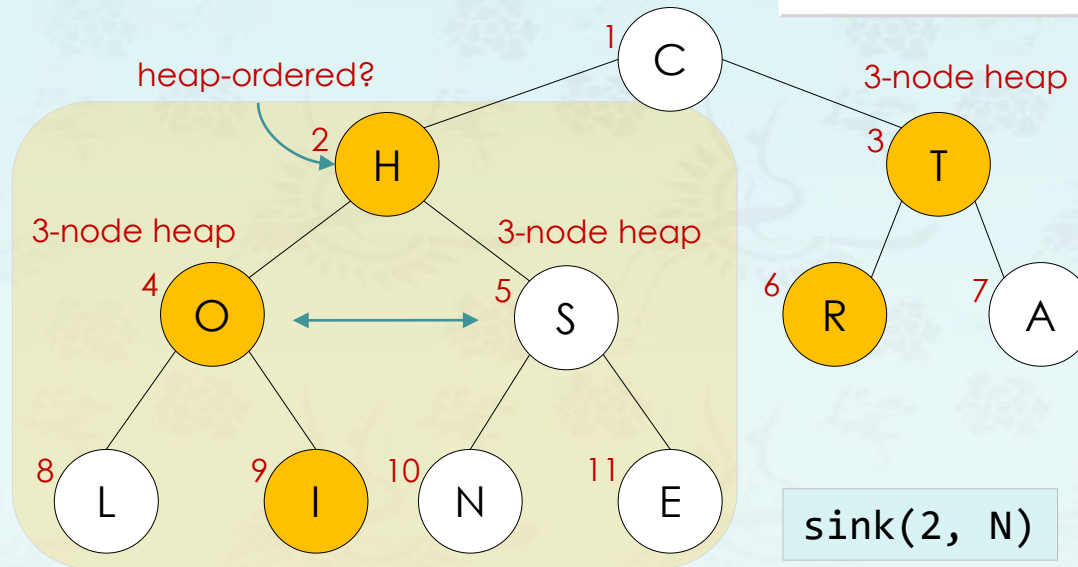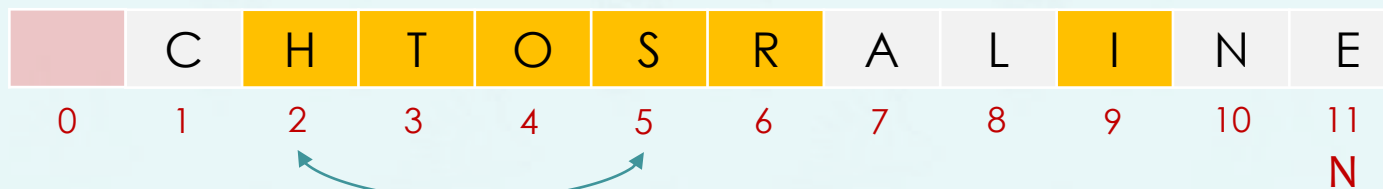

sink(2, N)

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

30

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;                                  k=2, j=4
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;                      k=2, j=5
    swap(h, k, j);                                  swap()
    k = j;                                          k=5
  }
                                                    while()
}
                                                    What's next?
```
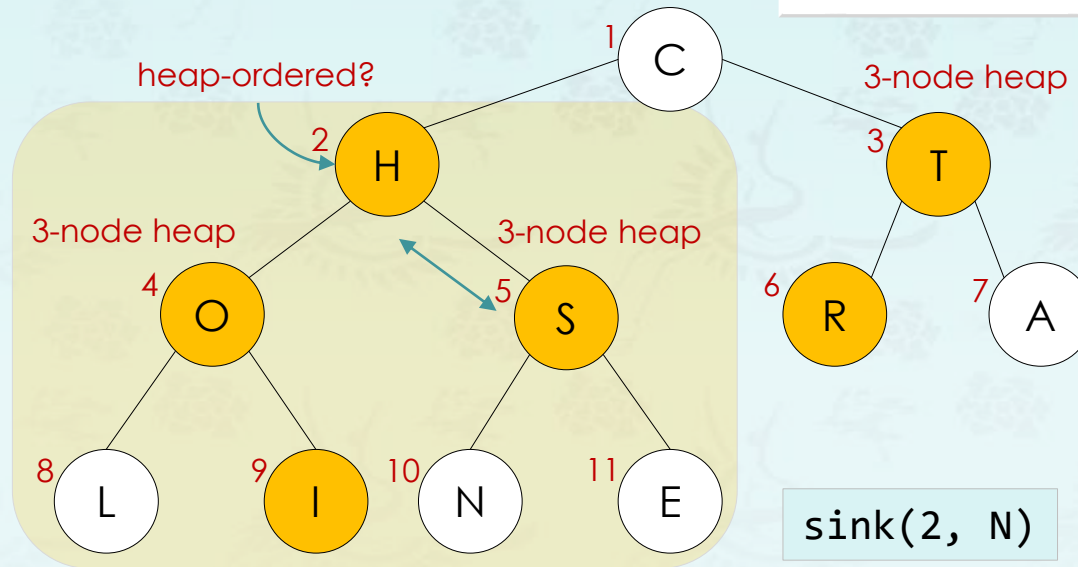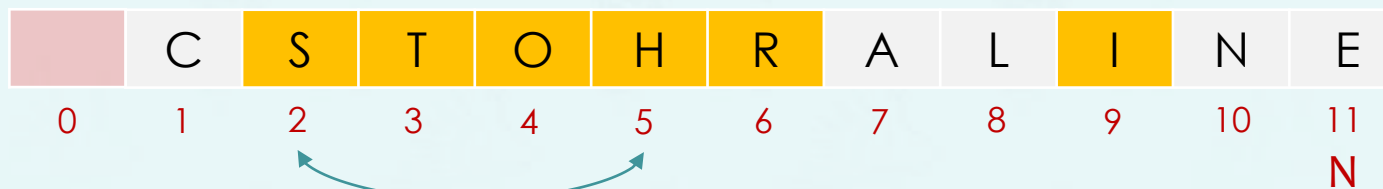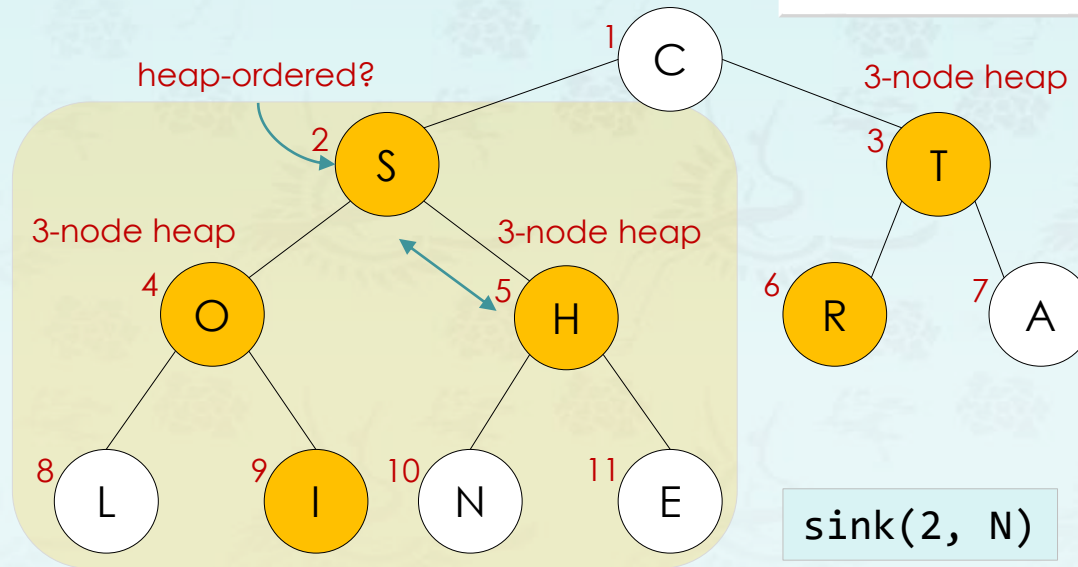

heap-ordered?

3-node heap

3-node heap   3-node heap

sink(2, N)

	C	S	T	O	H	R	A	L	I	N	E
0	1	2	3	4	5	6	7	8	9	10	11
											N

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;
    swap(h, k, j);
    k = j;
  }
}
```

k=2, j=4

k=2, j=5

swap()

k=5

while()

What's next?

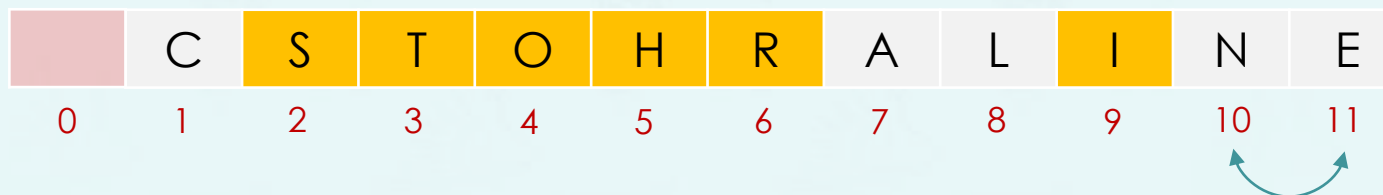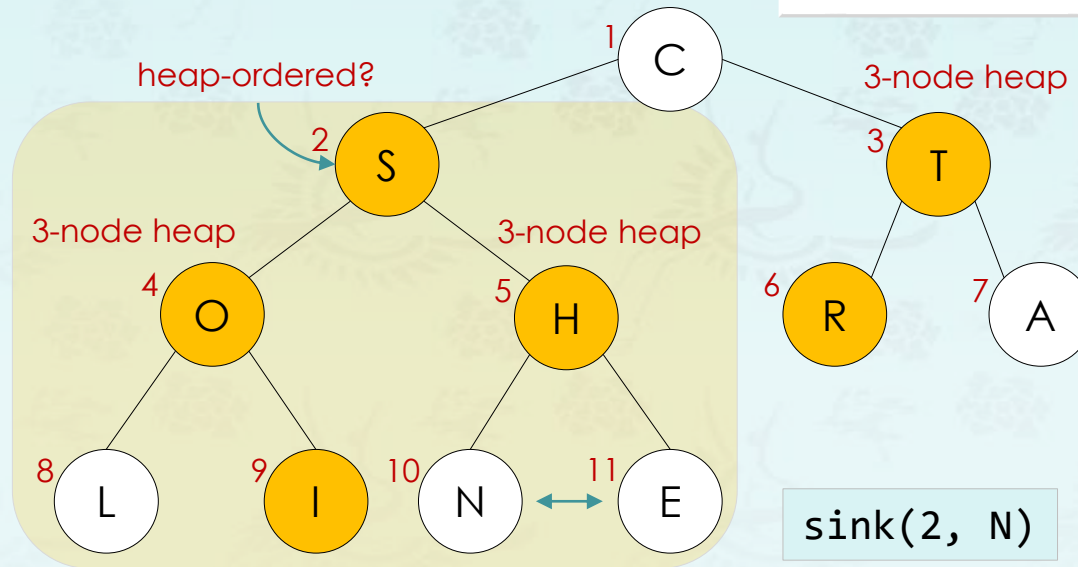k=5, j=10
k=5, j=10

sink(2, N) ➡ sink(5, N)

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;
    swap(h, k, j);
    k = j;
  }
}
```

k=2, j=4
k=2, j=5
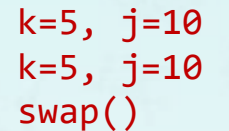swap()
k=5
while()
What's next?

k=5, j=10
k=5, j=10
swap()

heap-ordered?

1 C

3-node heap

2 S

3 T

3-node heap

3-node heap

4 O

5 H

6 R

7 A

8 L

9 I

10 N

11 E

sink(2, N) → sink(5, N)

	C	S	T	O	H	R	A	L	I	N	E
0	1	2	3	4	5	6	7	8	9	10	11

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University
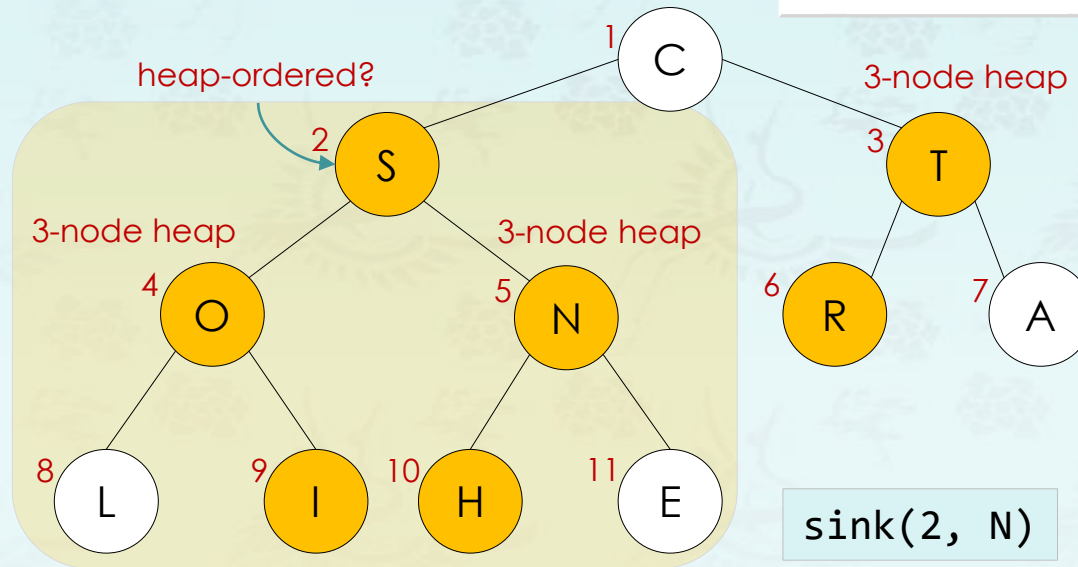
Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

```
void sink(heap h, int k) {
  while (2 * k <= h->N) {
    int j = 2 * k;                          k=2, j=4
    if (j < h->N && less(h, j, j + 1)) j++;
    if (!less(h, k, j)) break;              k=2, j=5
    swap(h, k, j);                          swap()
    k = j;                                  k=5
  }                                         while()
}                                           What's next?
```

k=5, j=10
k=5, j=10
swap()
k=10
while() exit

heap-ordered?

1 C

3-node heap

2 S   3 T

3-node heap   3-node heap

4 O   5 N   6 R   7 A

8 L   9 I   10 H   11 E

sink(2, N) ➡ sink(5, N)

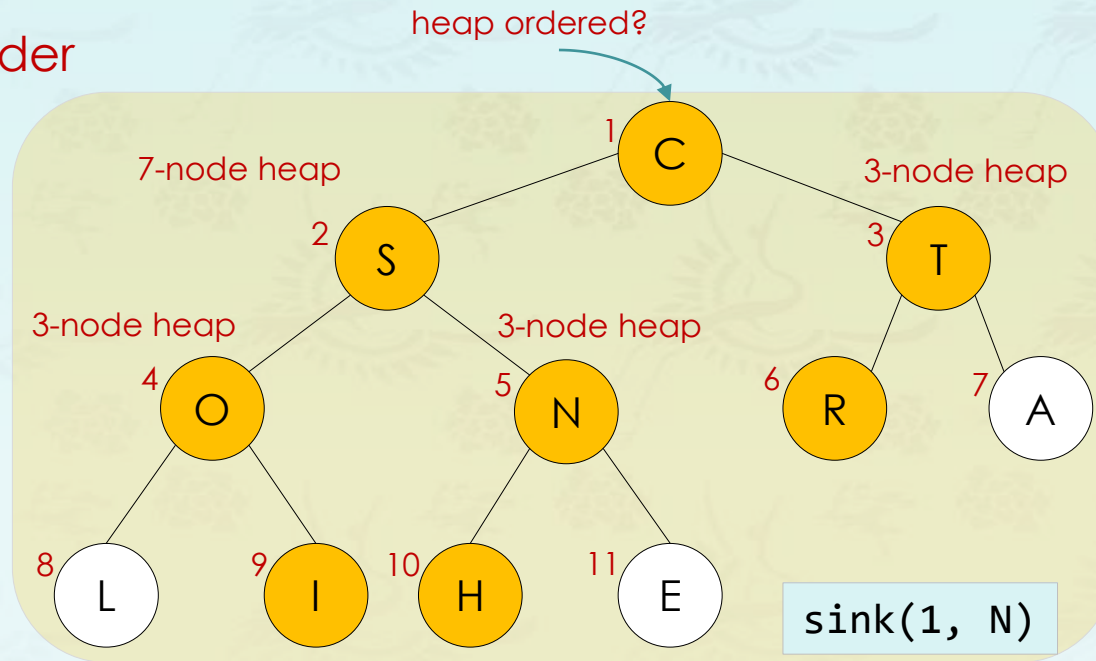|	C	S	T	O	N	R	A	L	I	H	E
0	1	2	3	4	5	6	7	8	9	10	11

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.
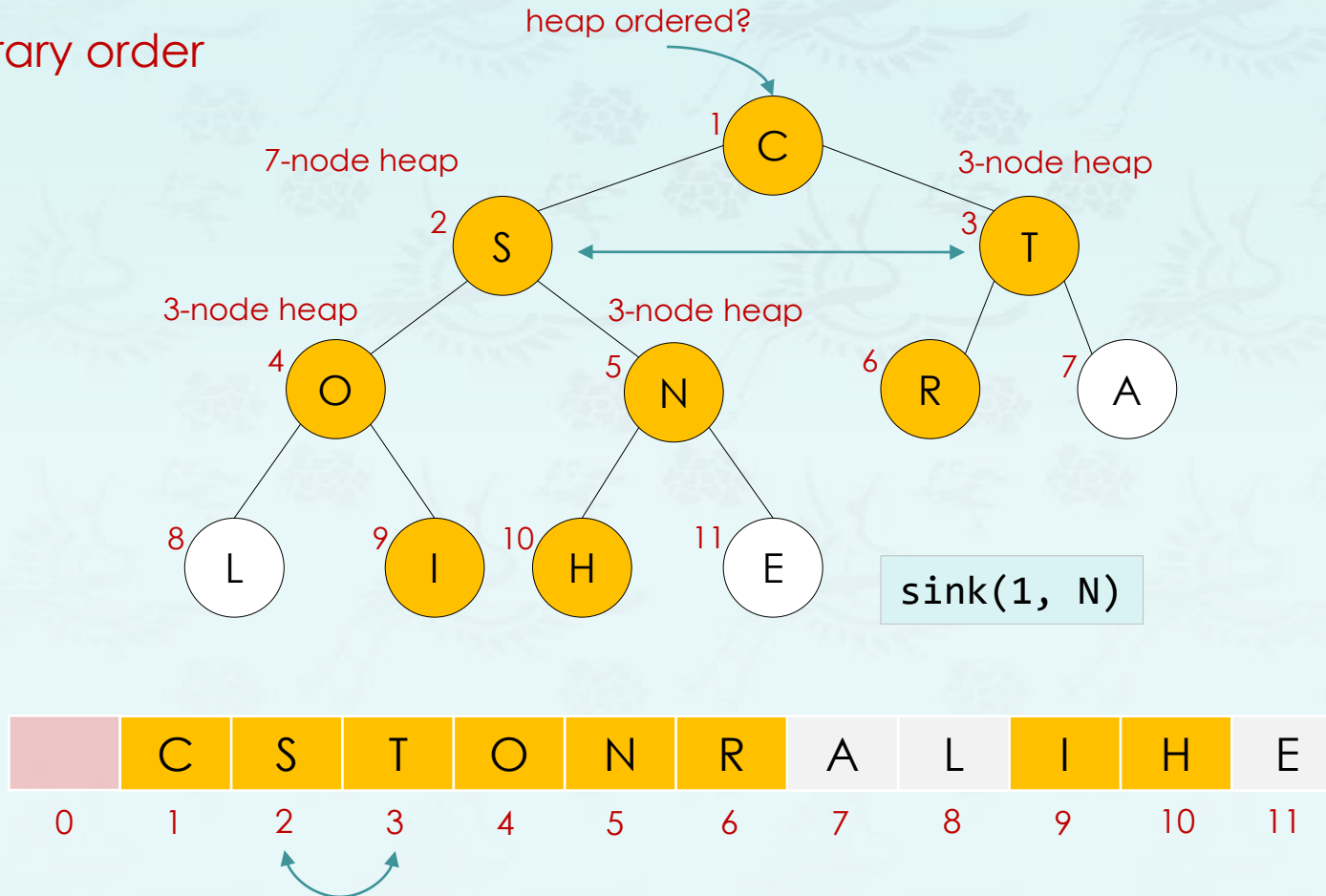
array in arbitrary order

heap ordered?

7-node heap

3-node heap

3-node heap

3-node heap

3-node heap

sink(1, N)

	C	S	T	O	N	R	A	L	I	H	E
0	1	2	3	4	5	6	7	8	9	10	11

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.
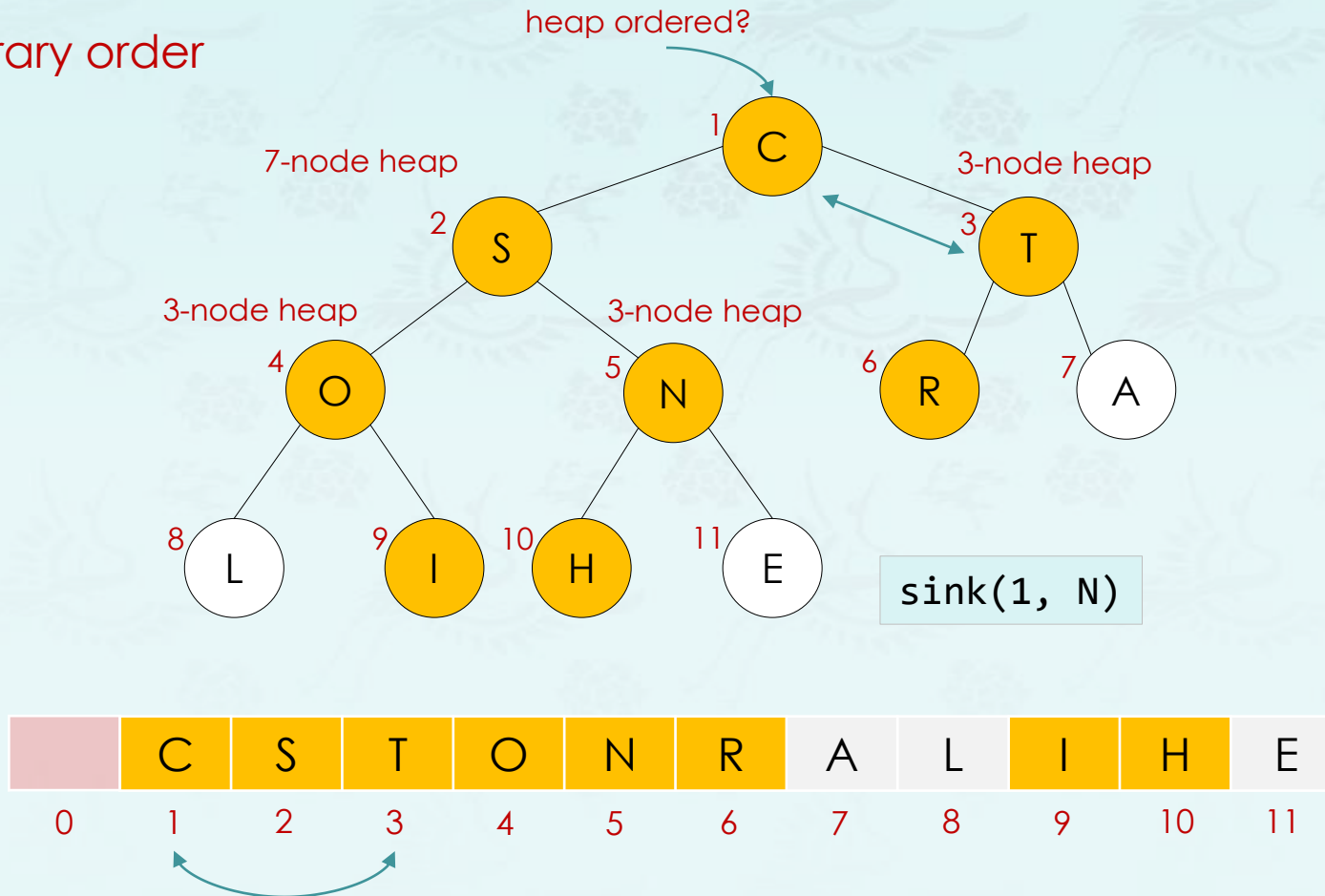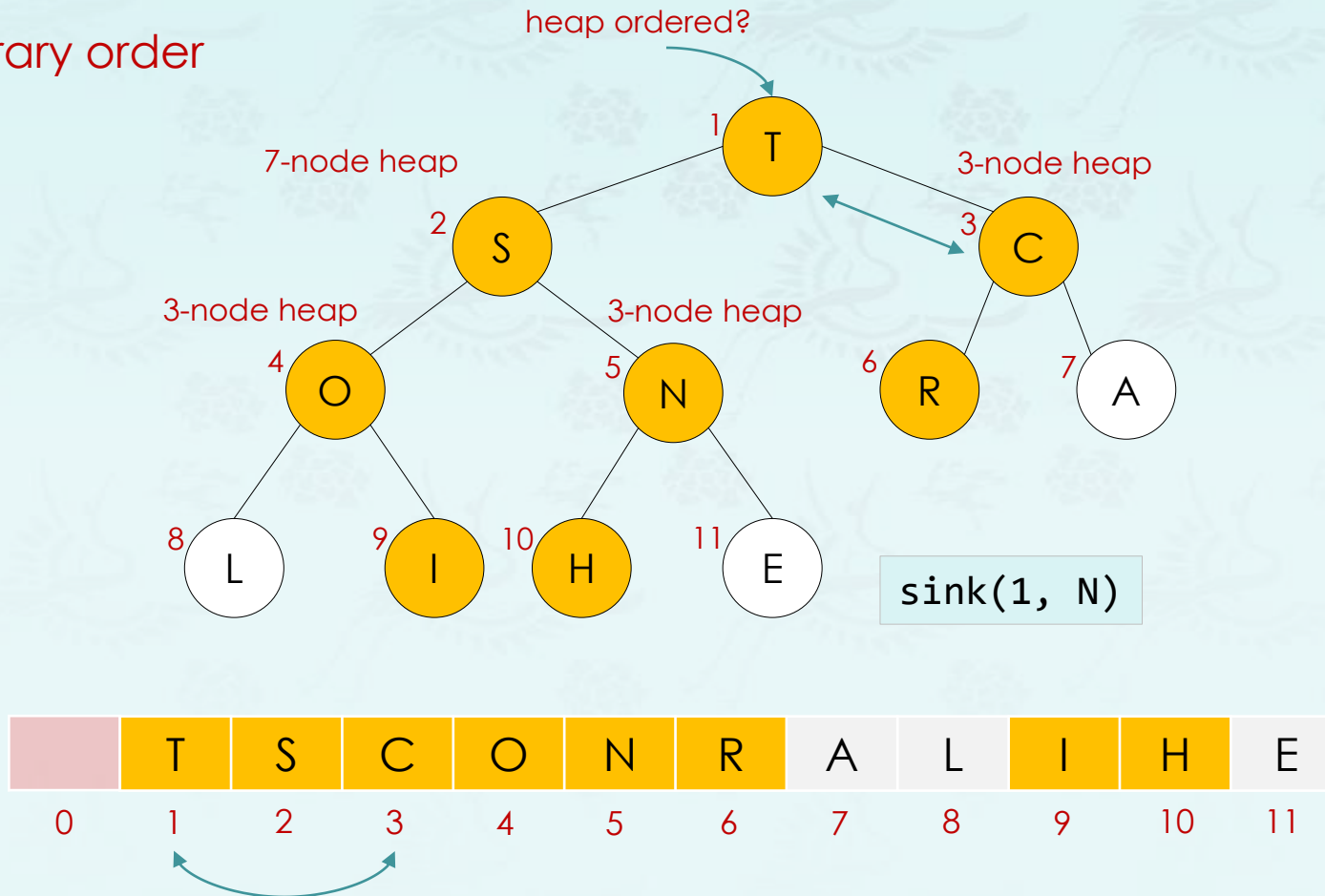
array in arbitrary order

heap ordered?

7-node heap

3-node heap

3-node heap

3-node heap

3-node heap

sink(1, N)

	C	S	T	O	N	R	A	L	I	H	E
0	1	2	3	4	5	6	7	8	9	10	11

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

36

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.
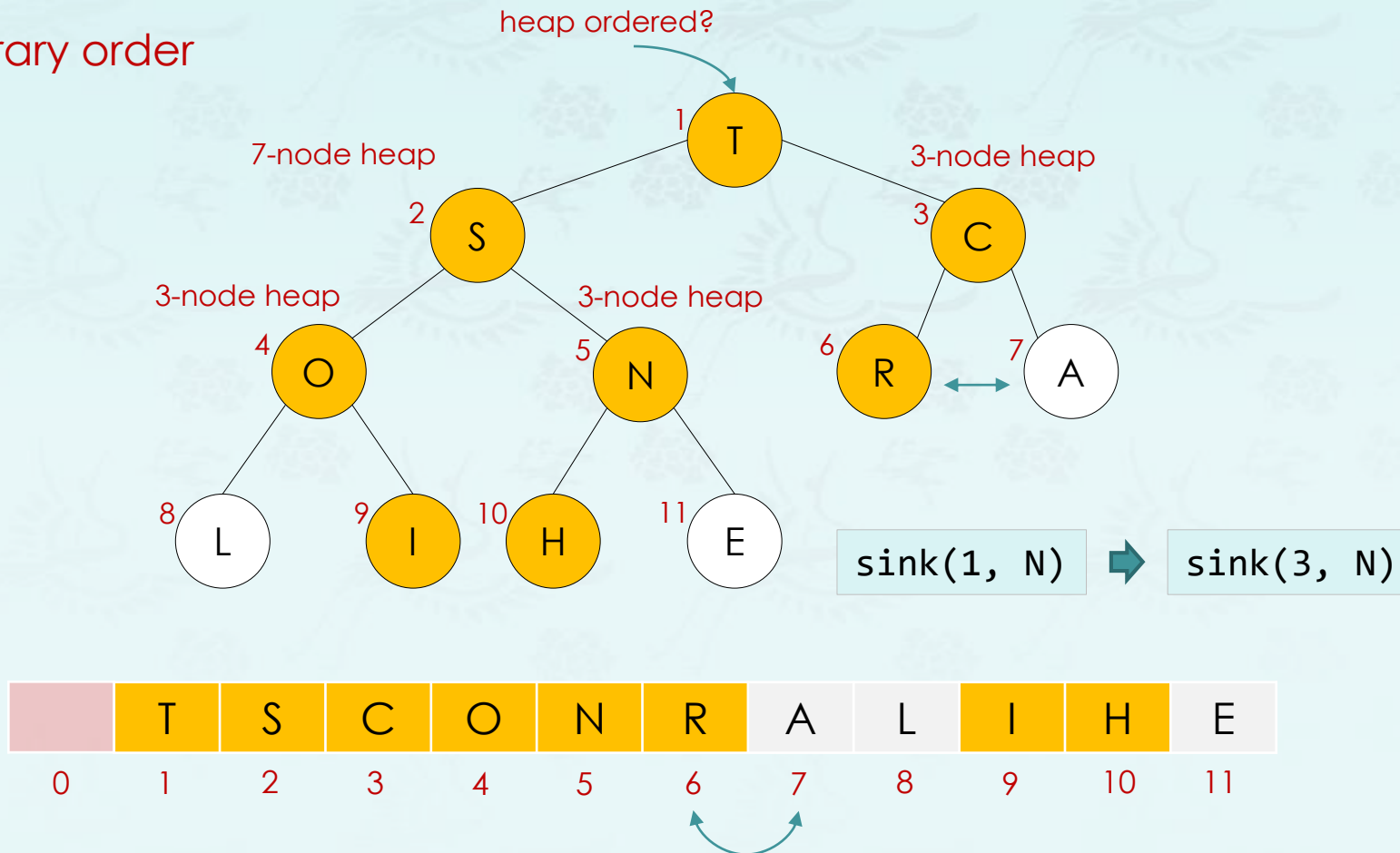
array in arbitrary order

heap ordered?

7-node heap

3-node heap

3-node heap

3-node heap

3-node heap

sink(1, N)

	C	S	T	O	N	R	A	L	I	H	E
0	1	2	3	4	5	6	7	8	9	10	11

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

37

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
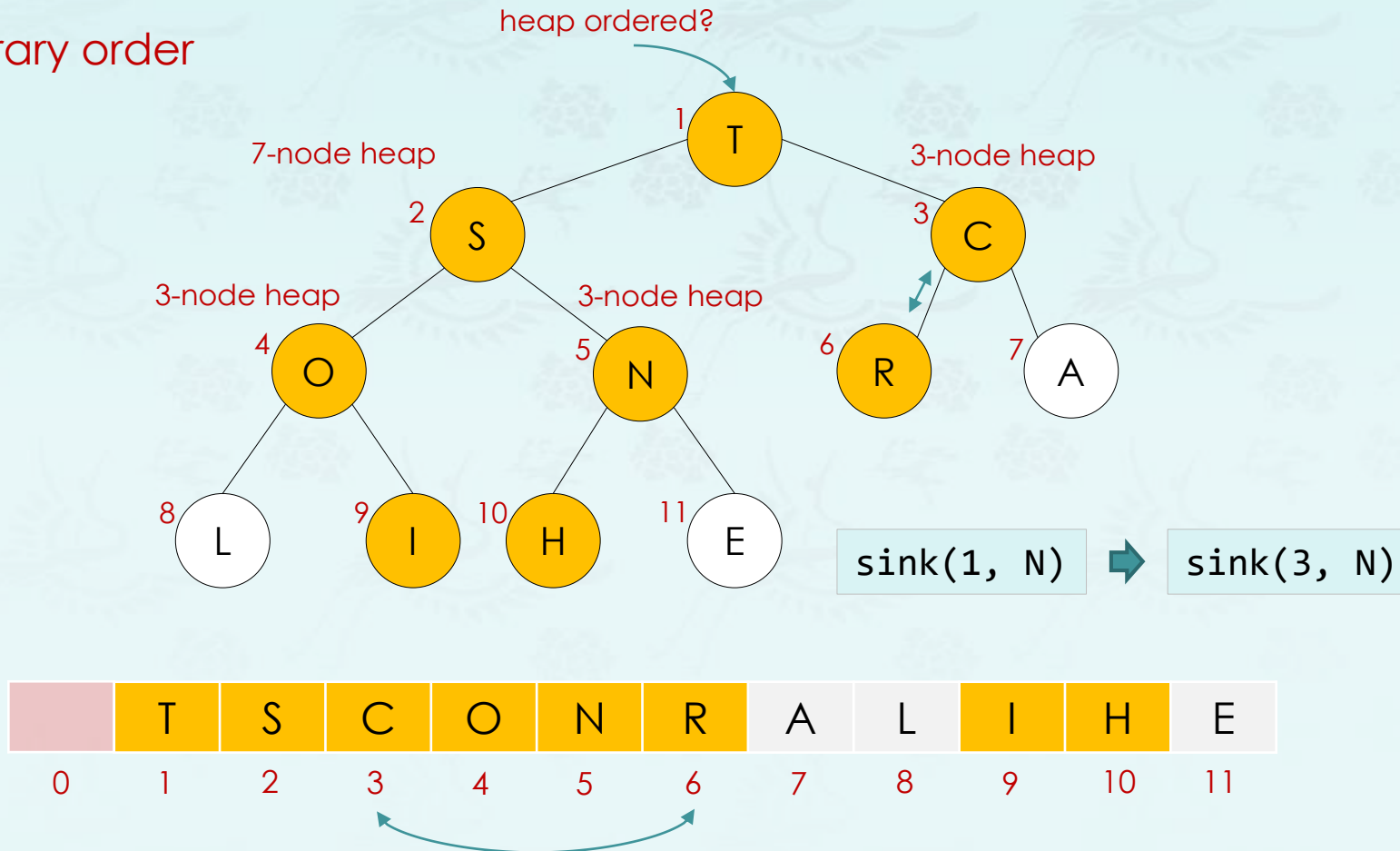- Assume array entries are indexed from 1 to N.

array in arbitrary order

heap ordered?

7-node heap

3-node heap

3-node heap

3-node heap

3-node heap

```
sink(1, N)
```

	T	S	C	O	N	R	A	L	I	H	E
0	1	2	3	4	5	6	7	8	9	10	11

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

38

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

heap ordered?

7-node heap

3-node heap

3-node heap

3-node heap

3-node heap

1 T
2 S
3 C
4 O
5 N
6 R
7 A
8 L
9 I
10 H
11 E

sink(1, N) ➡ sink(3, N)

	T	S	C	O	N	R	A	L	I	H	E
0	1	2	3	4	5	6	7	8	9	10	11

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

39

Heapsort

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

heap ordered?

7-node heap

3-node heap

3-node heap

3-node heap

3-node heap

sink(1, N) ➡ sink(3, N)

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in heap ordered
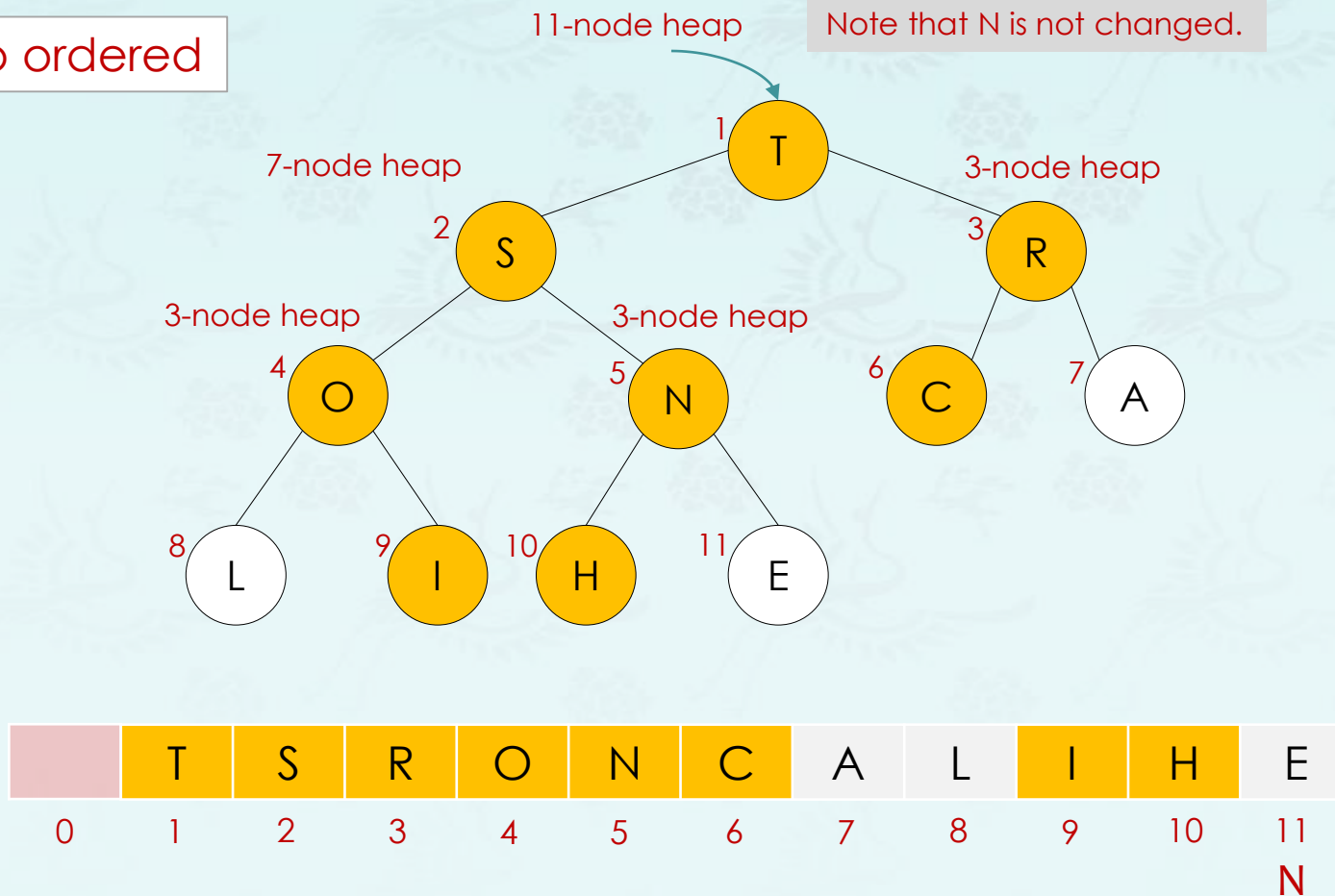
11-node heap

Note that N is not changed.

	T	S	R	O	N	C	A	L	I	H	E
0	1	2	3	4	5	6	7	8	9	10	11

N

Heapsort

Basic plan for in-place sort

- **1st Pass**: Build maxheap with all **N** keys.
- **2nd Pass**: Repeatedly remove the maximum key.

an array of **N** keys
in arbitrary order

1st Pass

build a maxheap
(in place)

2nd Pass

sorted in place

unsorted	C	H	R	I	S	T	A	L	O	N	E
heap-ordered	T	S	R	O	N	C	A	L	I	H	E
sorted	A	C	E	H	I	L	N	O	R	S	T
	1	2	3	4	5	6	7	8	9	10	11

2ⁿᵈ Pass: Repeatedly remove the maximum key.

(1) Swap root (the max) and the last node.
(2) Reduce N by one, but the value in memory.
(3) Sink the new root to make it heap-ordered.

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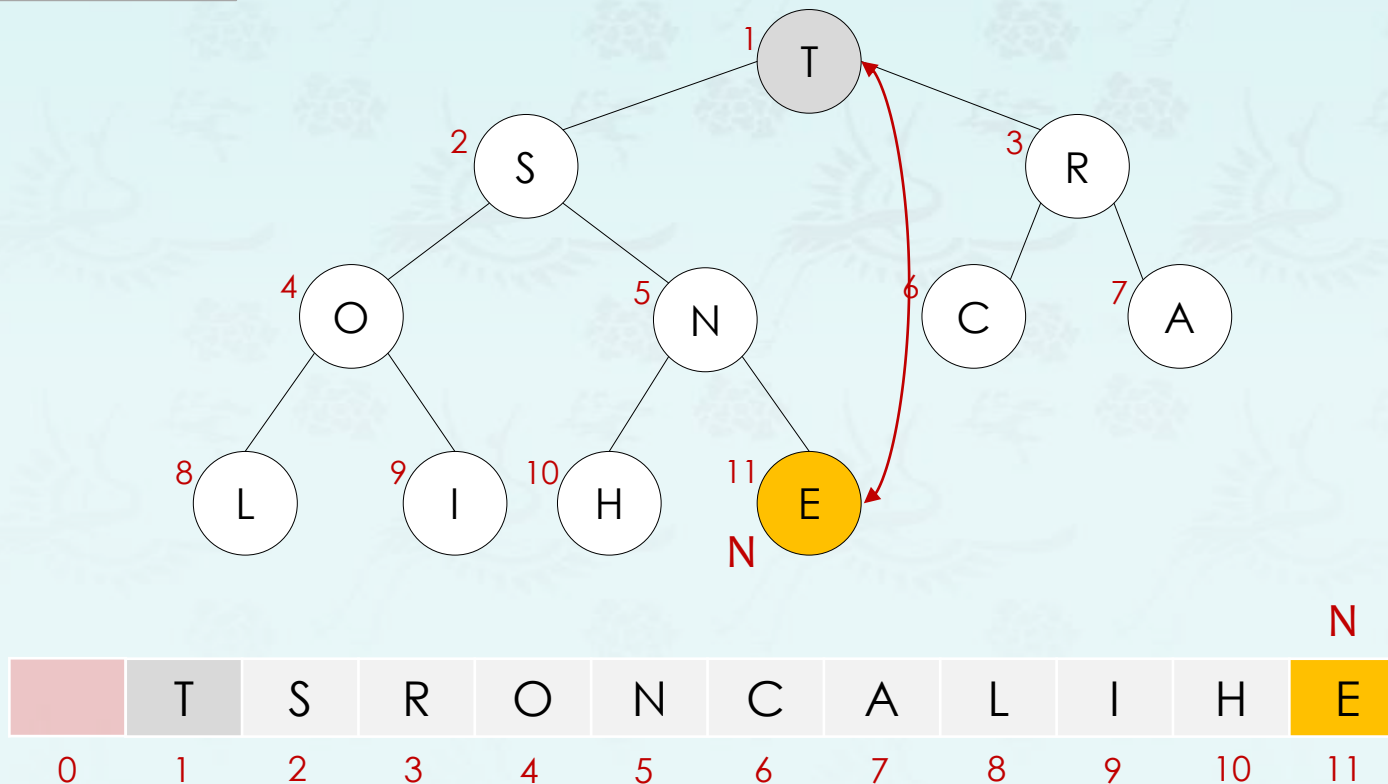- Leave them in array, instead of nulling out

array in heap ordered

	T	S	R	O	N	C	A	L	I	H	E
0	1	2	3	4	5	6	7	8	9	10	11

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

43

Heapsort

2nd Pass: Repeatedly remove the maximum key.

(1) Swap root (the max) and the last node.
(2) Reduce N by one, but the value in memory.
(3) Sink the new root to make it heap-ordered.

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

array in heap ordered

`swap(1, N);`   N=11
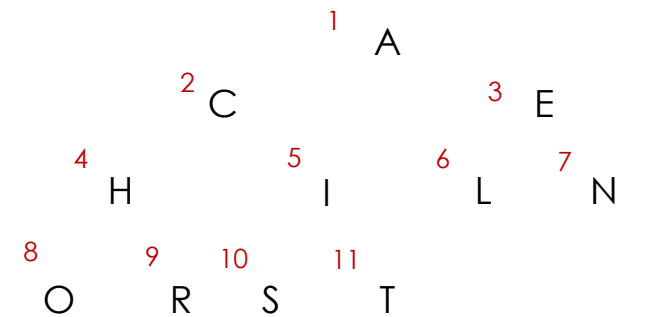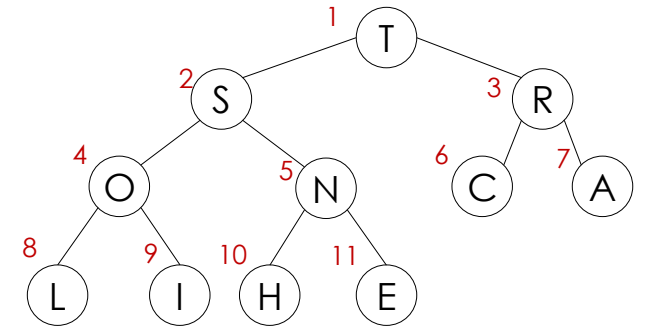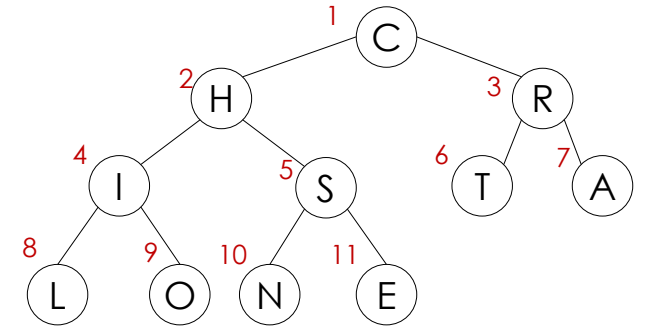
Heapsort

2nd Pass: Repeatedly remove the maximum key.

- Remove the maximum, one at a time.
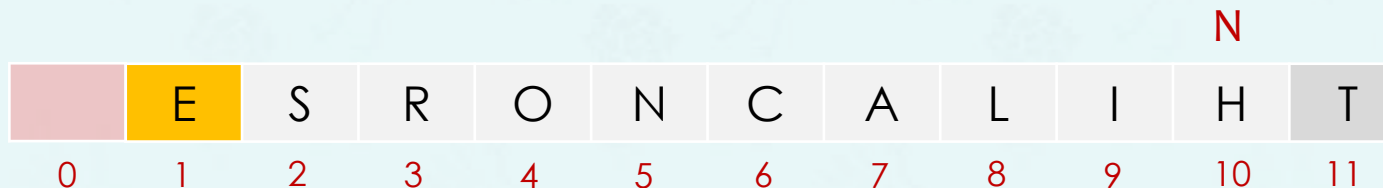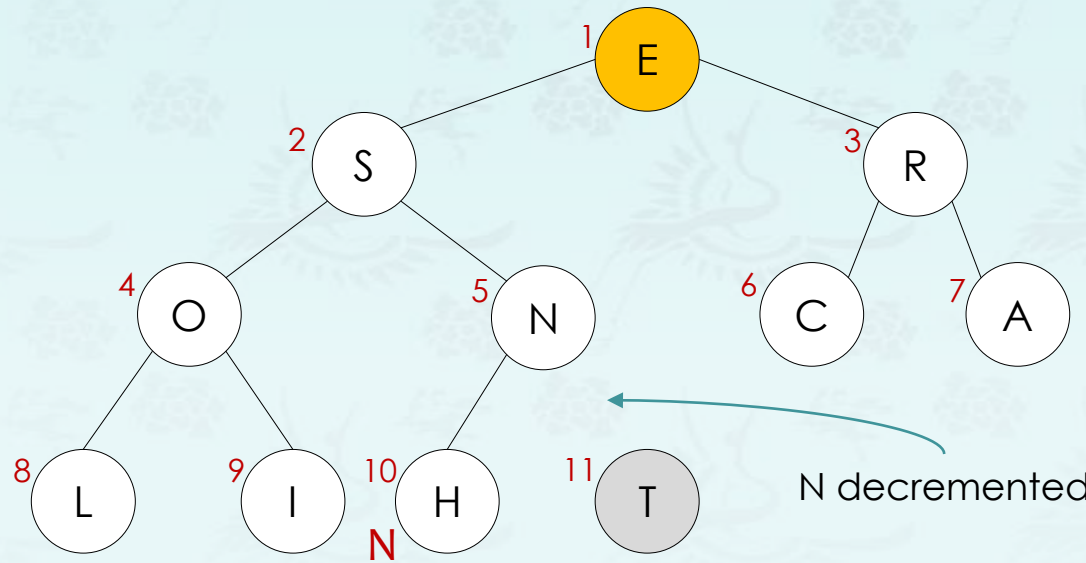- Leave them in array, instead of nulling out
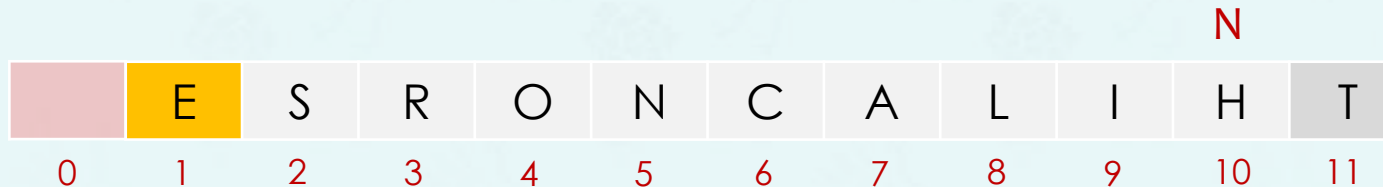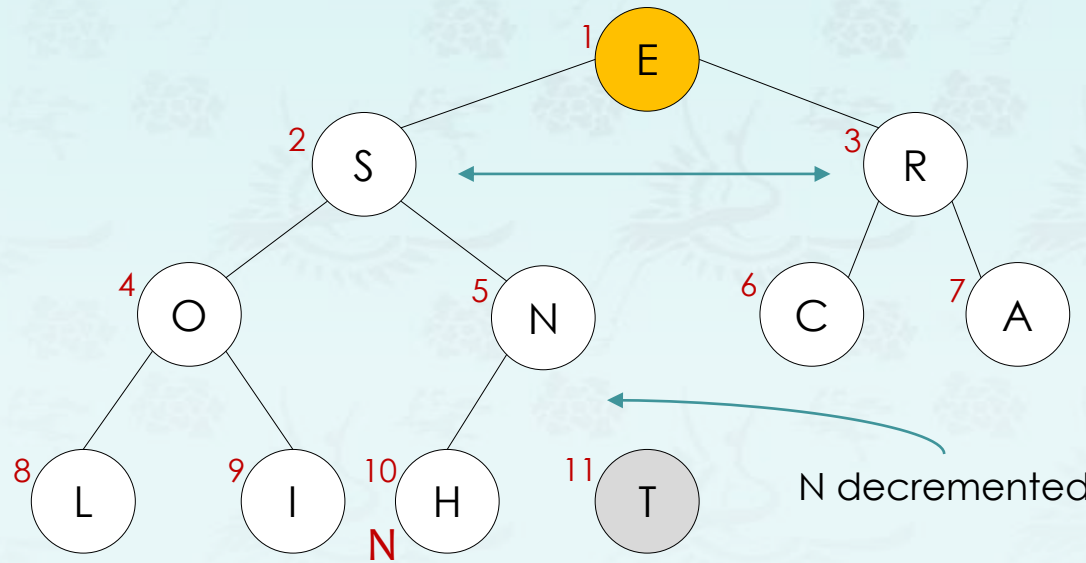
(1) Swap root (the max) and the last node.
(2) Reduce N by one, but the value in memory.
(3) Sink the new root to make it heap-ordered.

array in heap ordered

```
swap(1, N);        N=11
sink(1, --N);      N=10
```


N decremented

	E	S	R	O	N	C	A	L	I	H	T
0	1	2	3	4	5	6	7	8	9	10	11

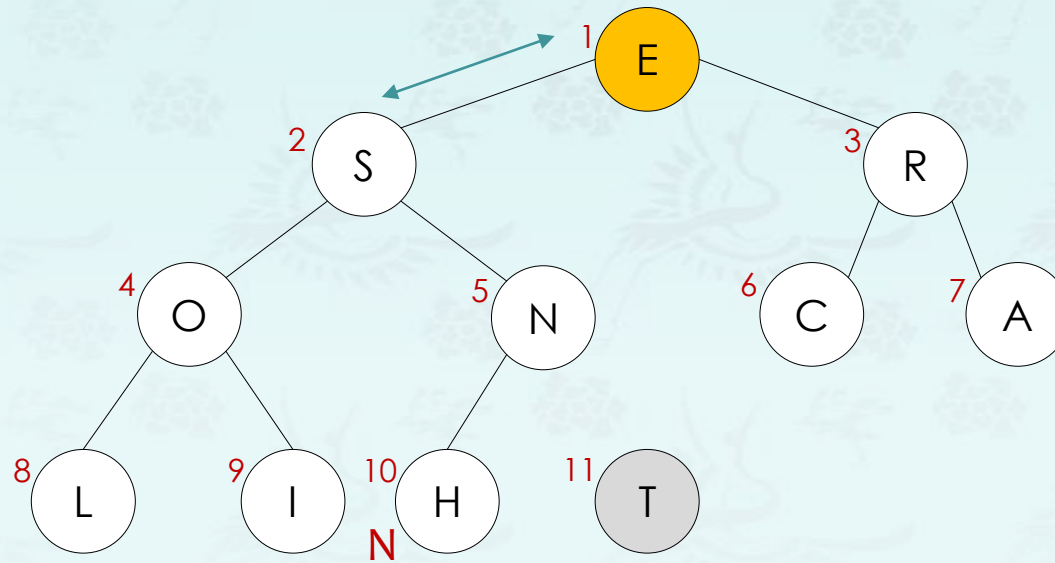Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered

```
swap(1, N);        N=11
sink(1, --N);      N=10
```


N decremented

	E	S	R	O	N	C	A	L	I	H	T
0	1	2	3	4	5	6	7	8	9	10	11

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

46

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered

```
swap(1, N);        N=11
sink(1, --N);      N=10
```
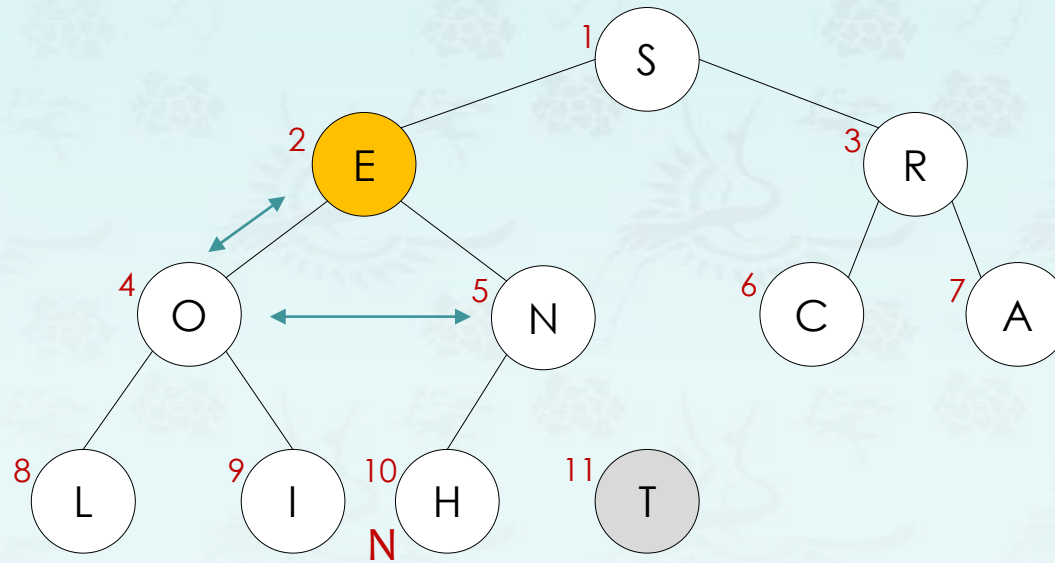

N decremented

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University
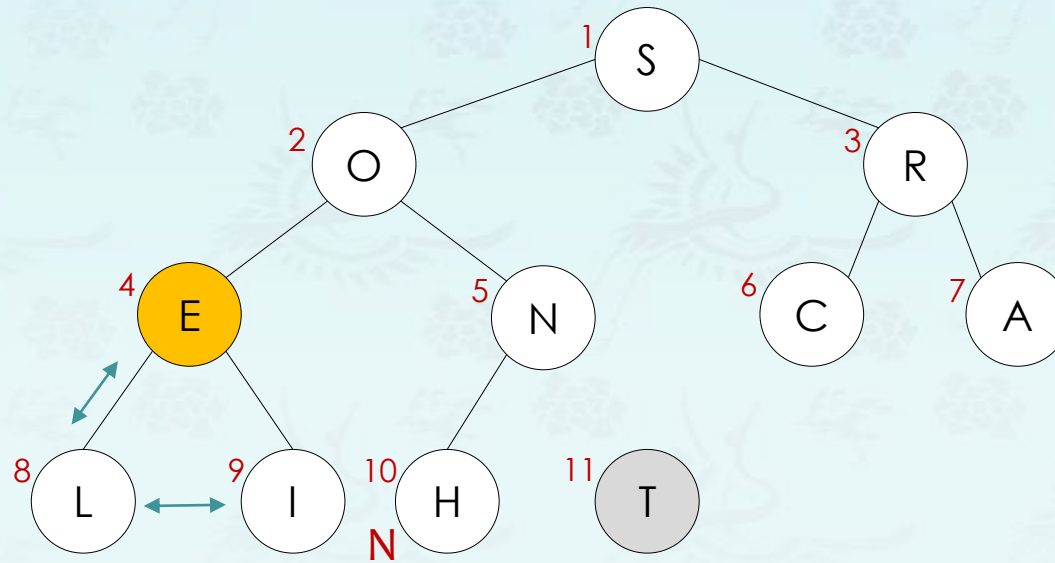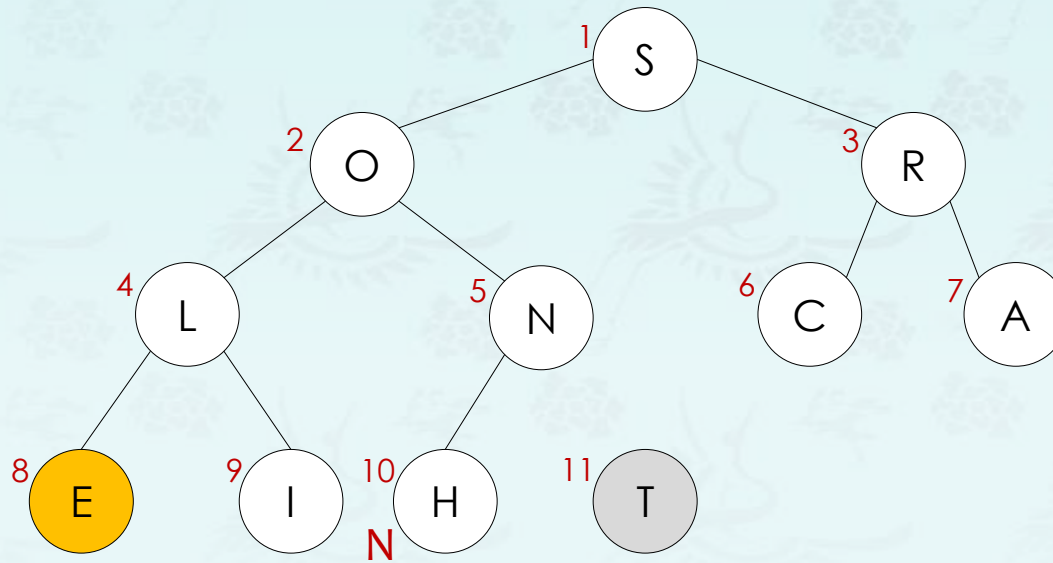
47

Heapsort

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered


```
swap(1, N);       N=11
sink(1, --N);     N=10
```

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered


```
swap(1, N);        N=11
sink(1, --N);      N=10
```

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

Heapsort

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered

```
swap(1, N);        N=11
sink(1, --N);      N=10
```

Heapsort

2nd Pass: Repeatedly remove the maximum key.

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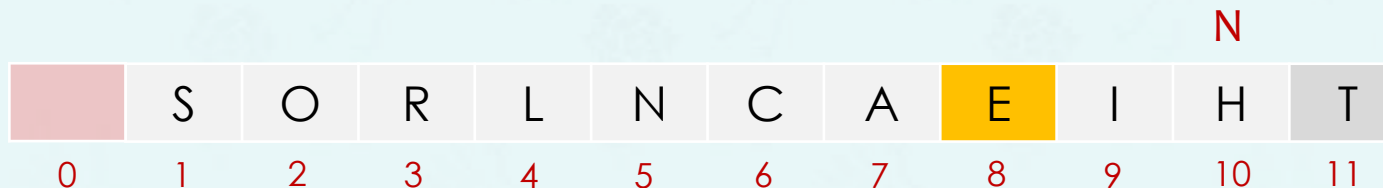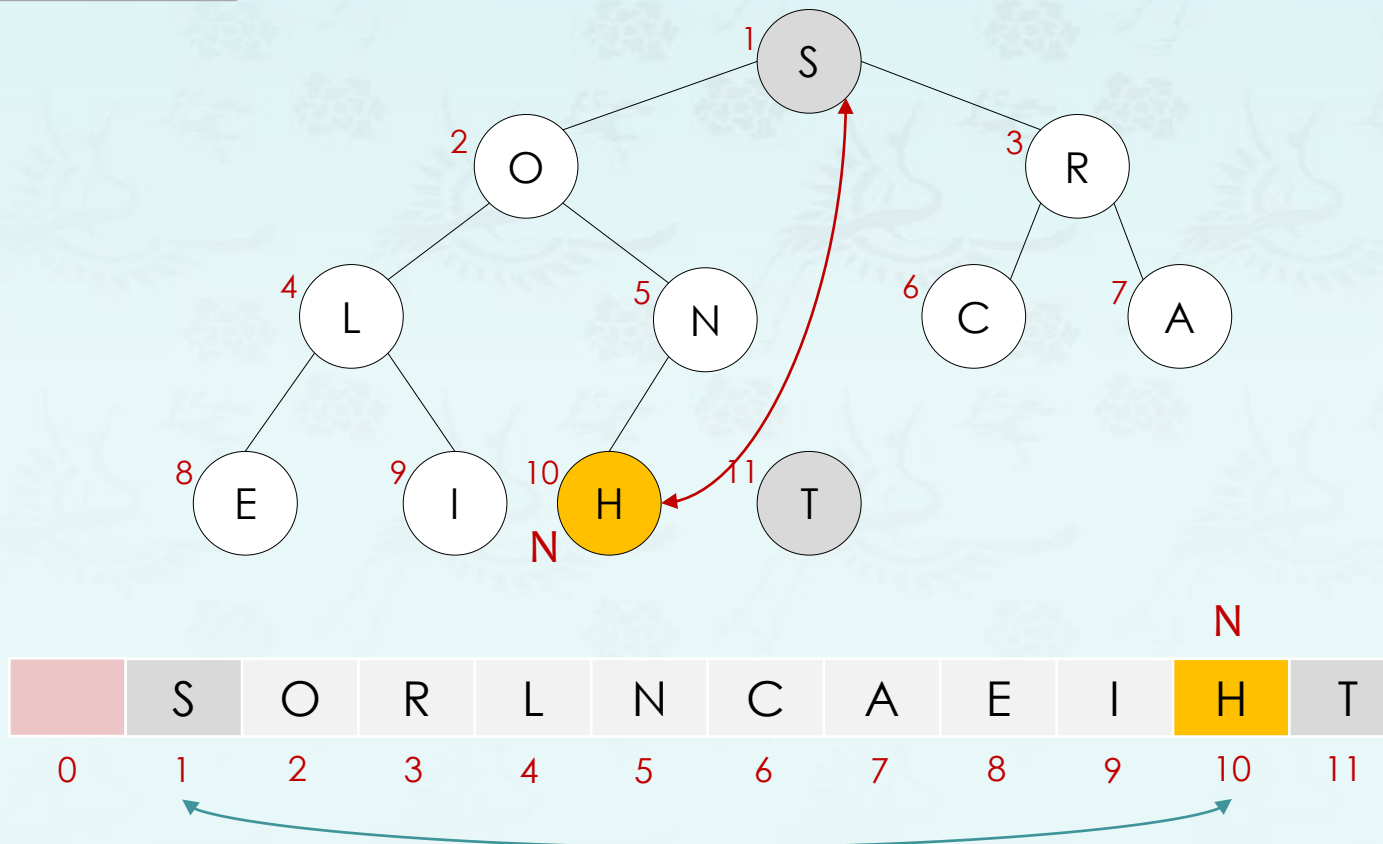- Leave them in array, instead of nulling out

array in heap ordered


```
swap(1, N);       N=11
sink(1, --N);     N=10
```

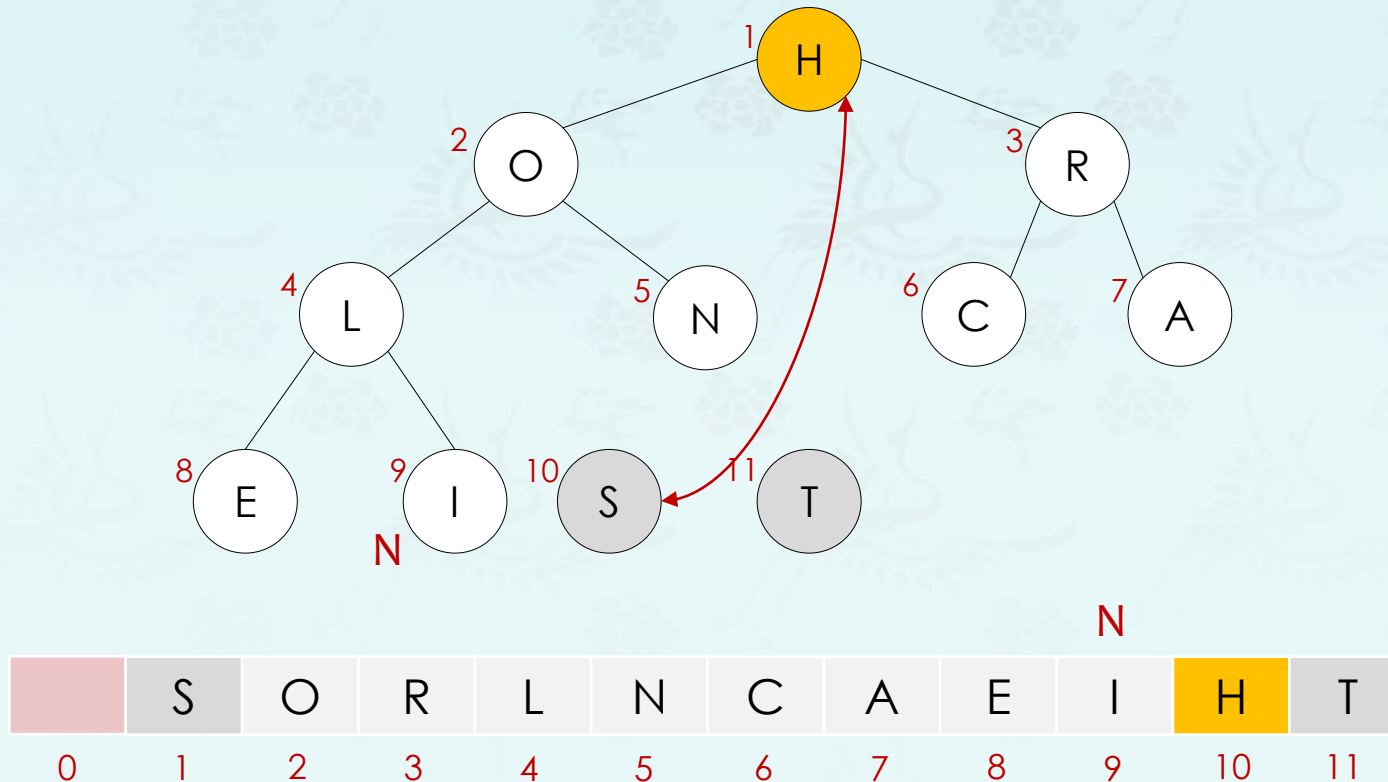What's next?

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered

```
swap(1, N);        N=11
sink(1, --N);      N=10
```

```
swap(1, N);        N=10
sink(1, --N);      N=9
```

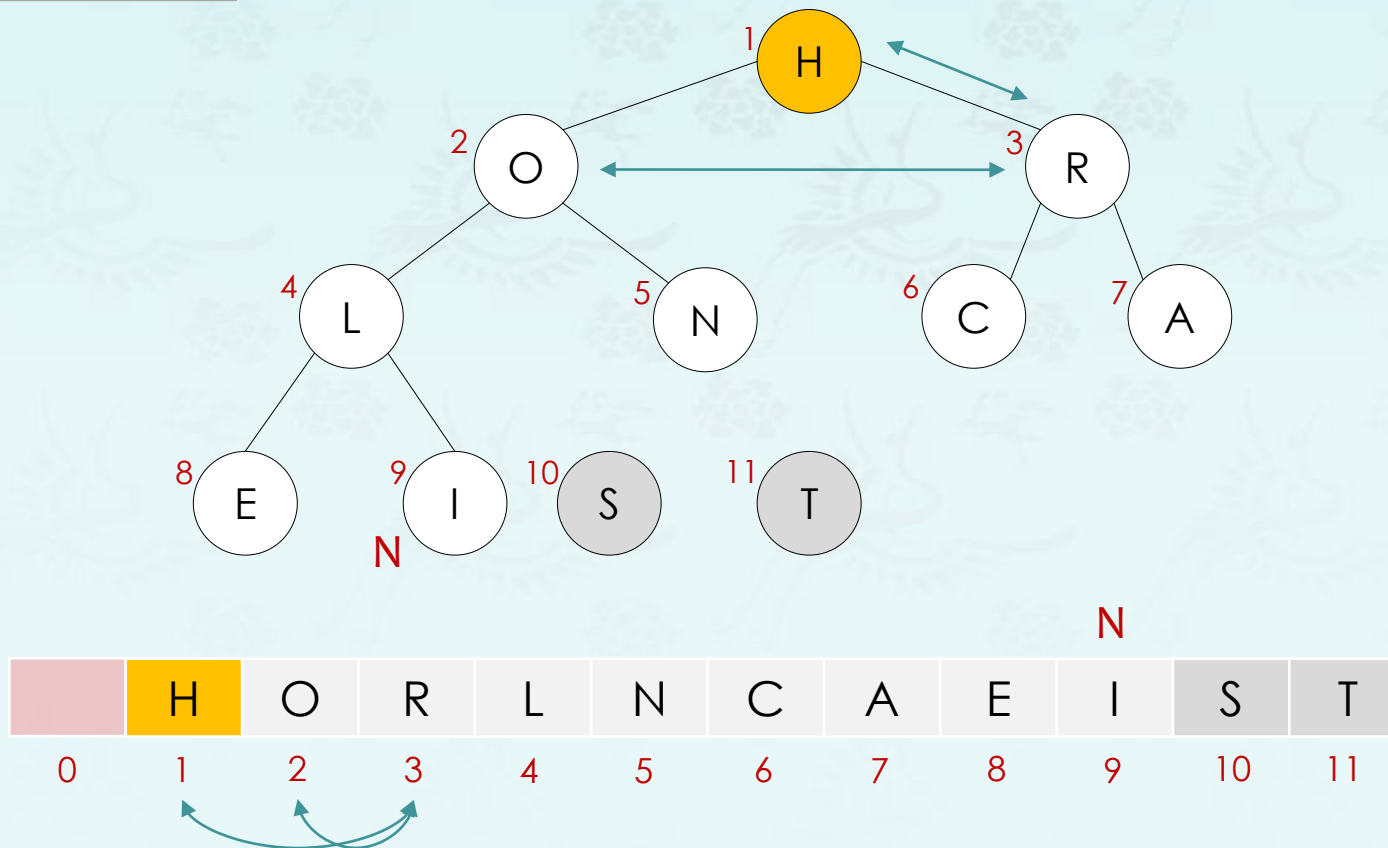Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

52

Heapsort

2ⁿᵈ Pass: Repeatedly remove the maximum key.

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

array in heap ordered

```
swap(1, N);      N=11
sink(1, --N);    N=10

swap(1, N);      N=10
sink(1, --N);    N=9
```

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered


```
swap(1, N);        N=11
sink(1, --N);      N=10
```
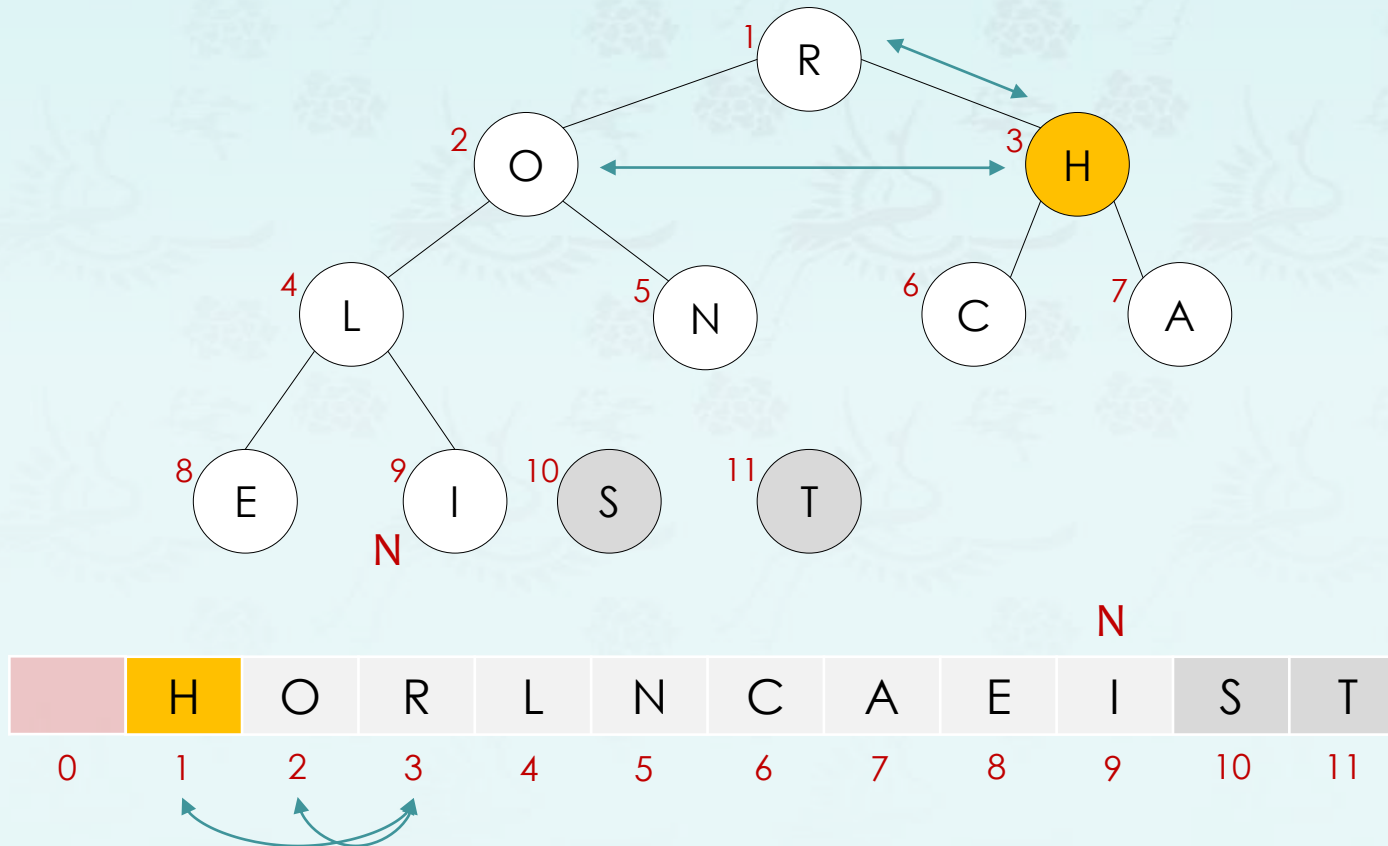
```
swap(1, N);        N=10
sink(1, --N);      N=9
```

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered

```
swap(1, N);      N=11
sink(1, --N);    N=10
```

```
swap(1, N);      N=10
sink(1, --N);    N=9
```

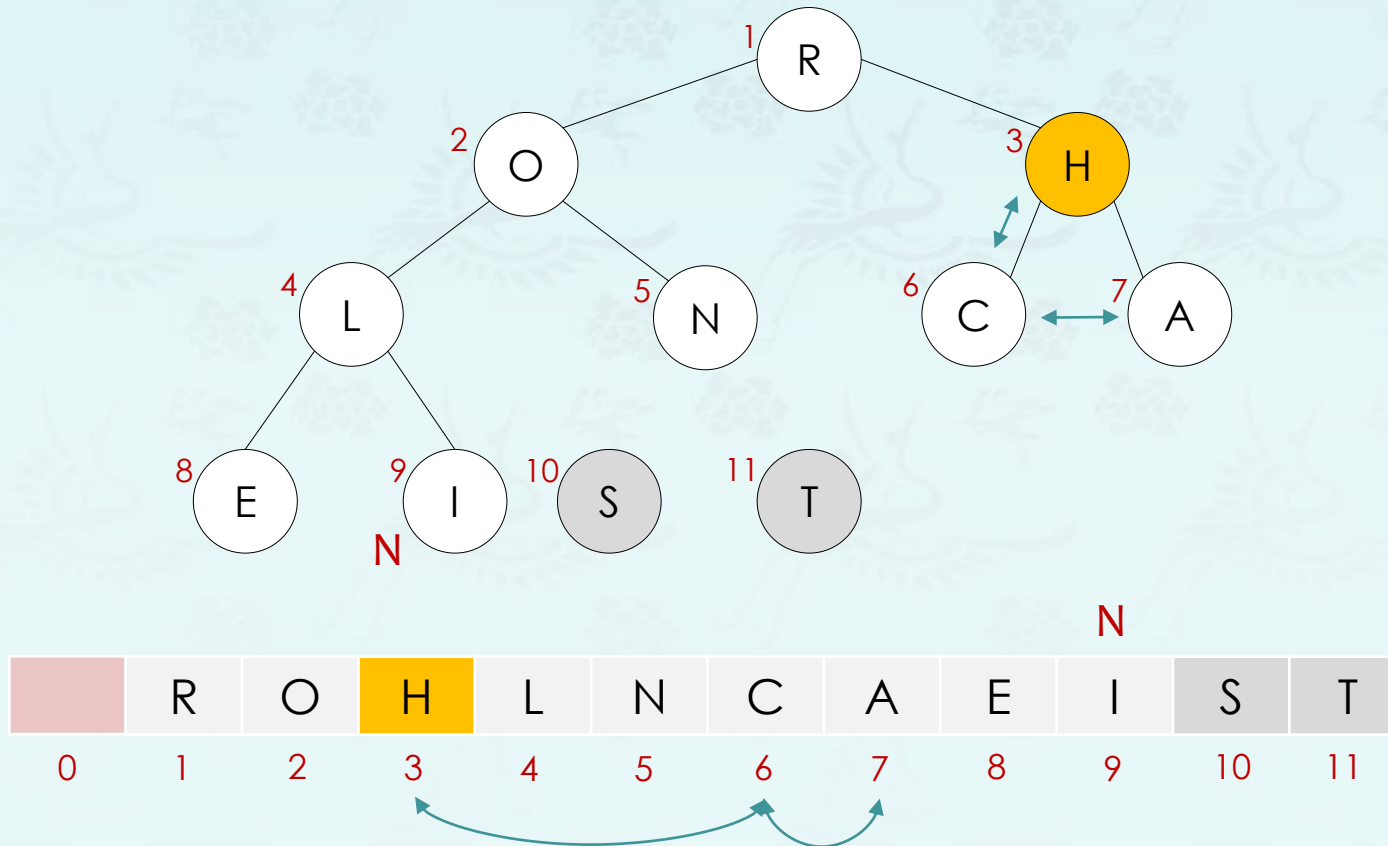*Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University*

55

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered


```
swap(1, N);        N=11
sink(1, --N);      N=10
```

```
swap(1, N);        N=10
sink(1, --N);      N=9
```

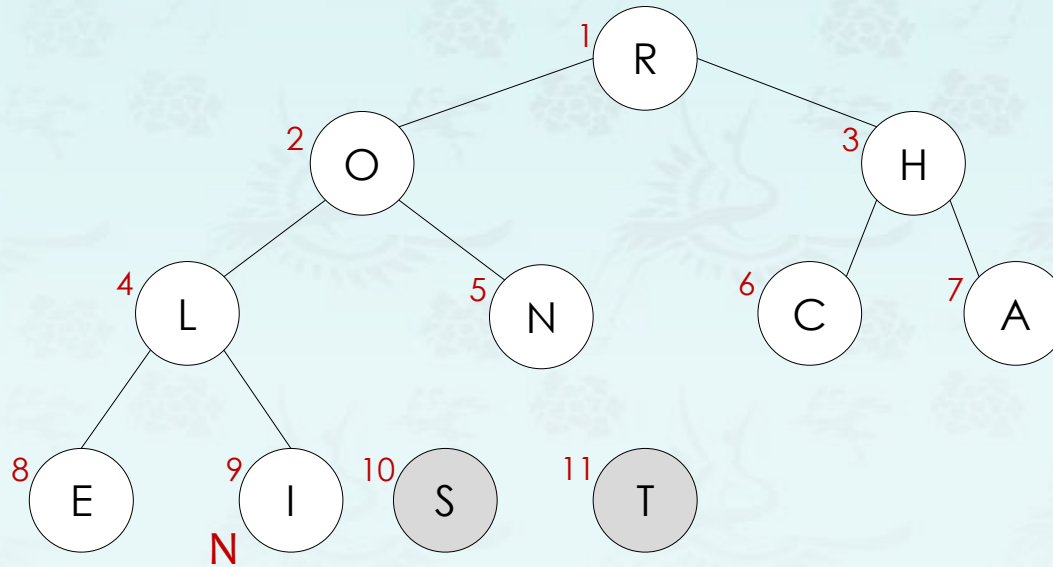|	R	O	H	L	N	C	A	E	I	S	T
0	1	2	3	4	5	6	7	8	9	10	11

2nd Pass: Repeatedly remove the maximum key.

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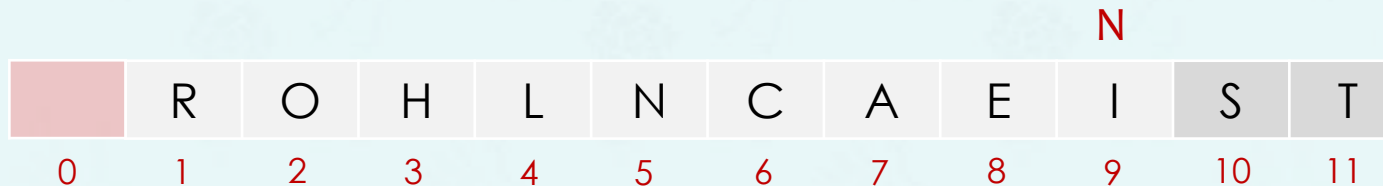- Leave them in array, instead of nulling out

array in heap ordered

```
swap(1, N);        N=11
sink(1, --N);      N=10
```

```
swap(1, N);        N=10
sink(1, --N);      N=9
```

What's next?

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

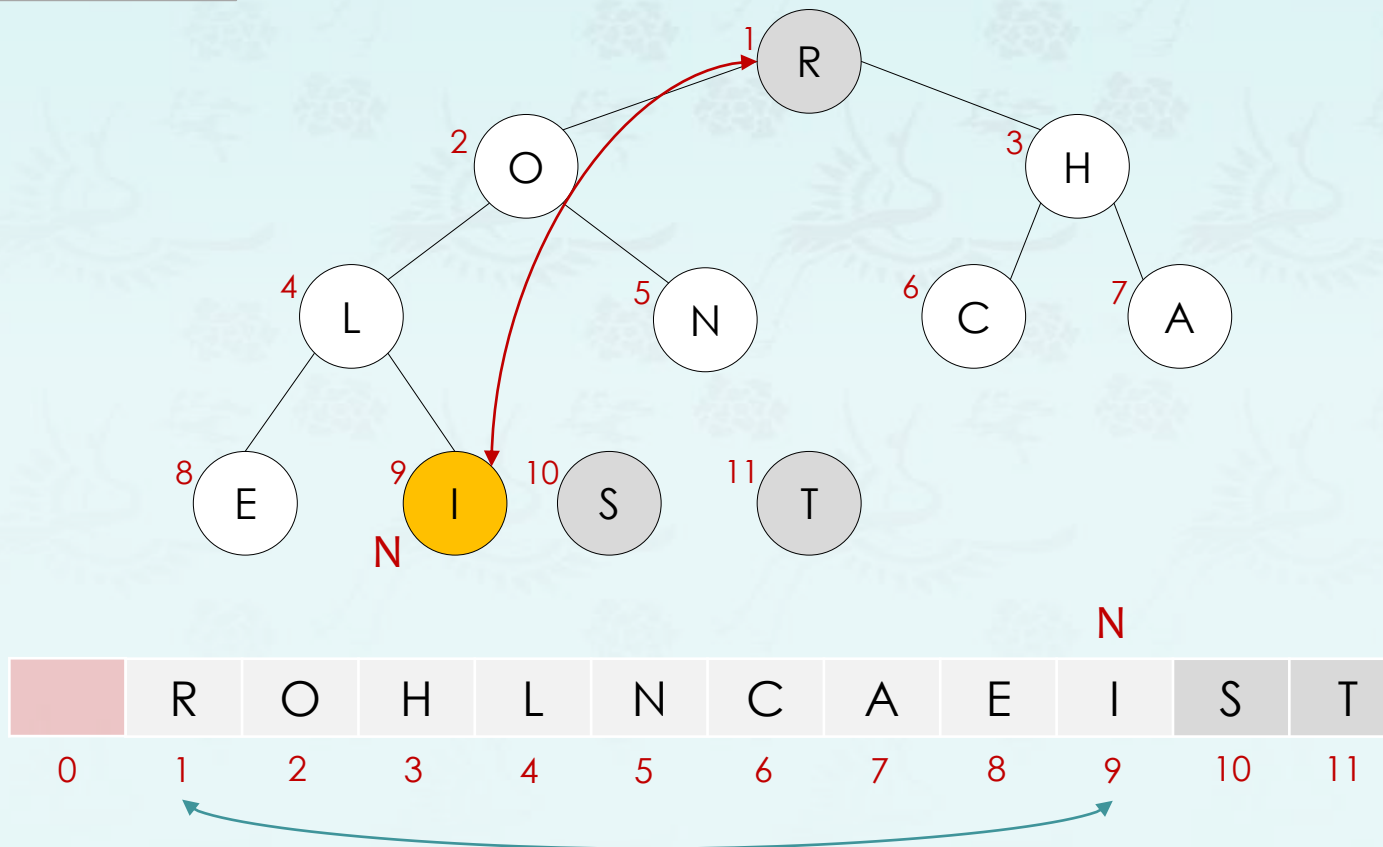**2nd Pass: Repeatedly remove the maximum key**.

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

array in heap ordered

`swap(1, N);`	N=11
`sink(1, --N);`	N=10

`swap(1, N);`	N=10
`sink(1, --N);`	N=9

`swap(1, N);`	N=9
`sink(1, --N);`	N=8

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

58
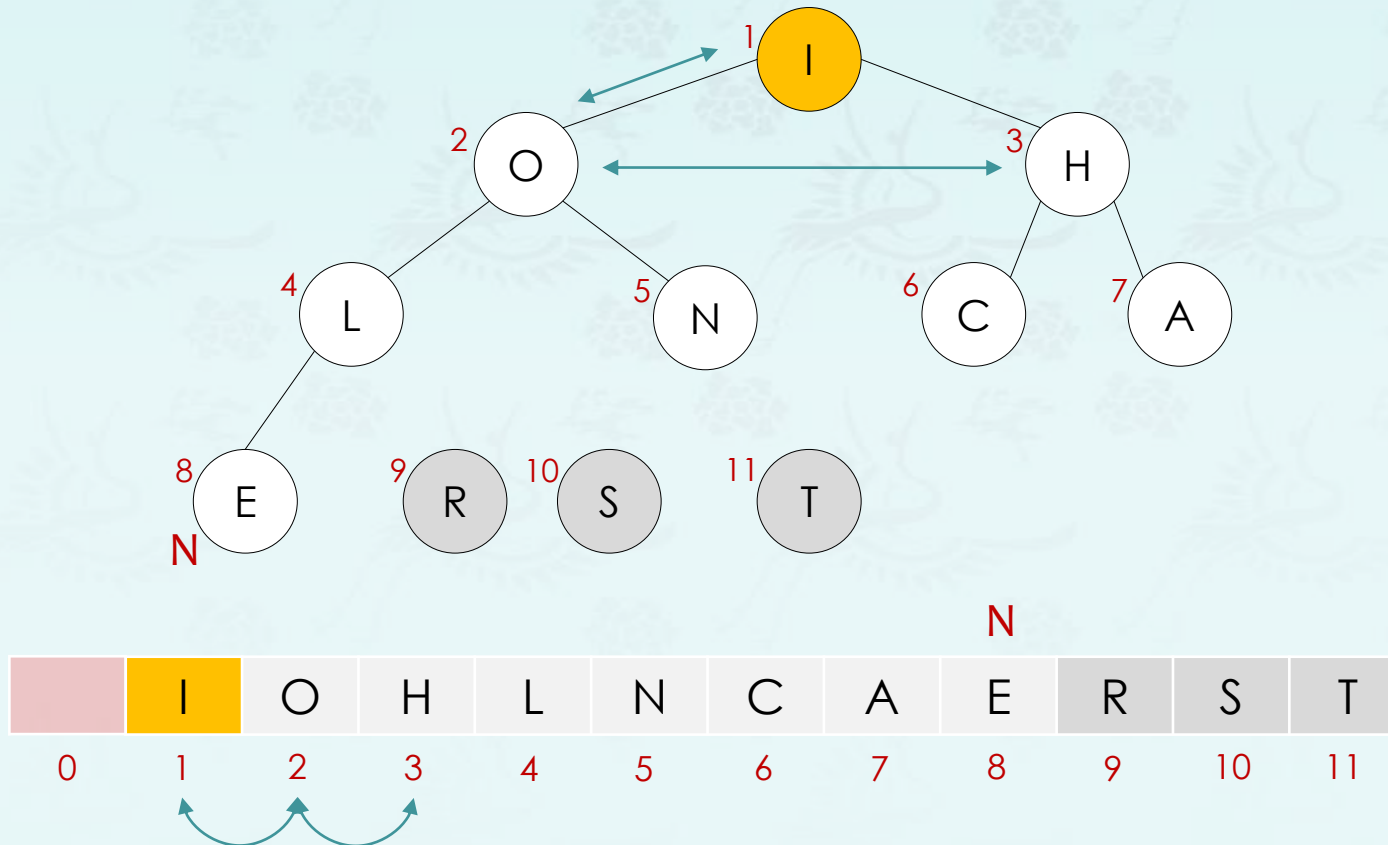
2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered

```
swap(1, N);        N=11
sink(1, --N);      N=10

swap(1, N);        N=10
sink(1, --N);      N=9

swap(1, N);        N=9
sink(1, --N);      N=8
```

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered


```
swap(1, N);        N=11
sink(1, --N);      N=10

swap(1, N);        N=10
sink(1, --N);      N=9

swap(1, N);        N=9
sink(1, --N);      N=8
```
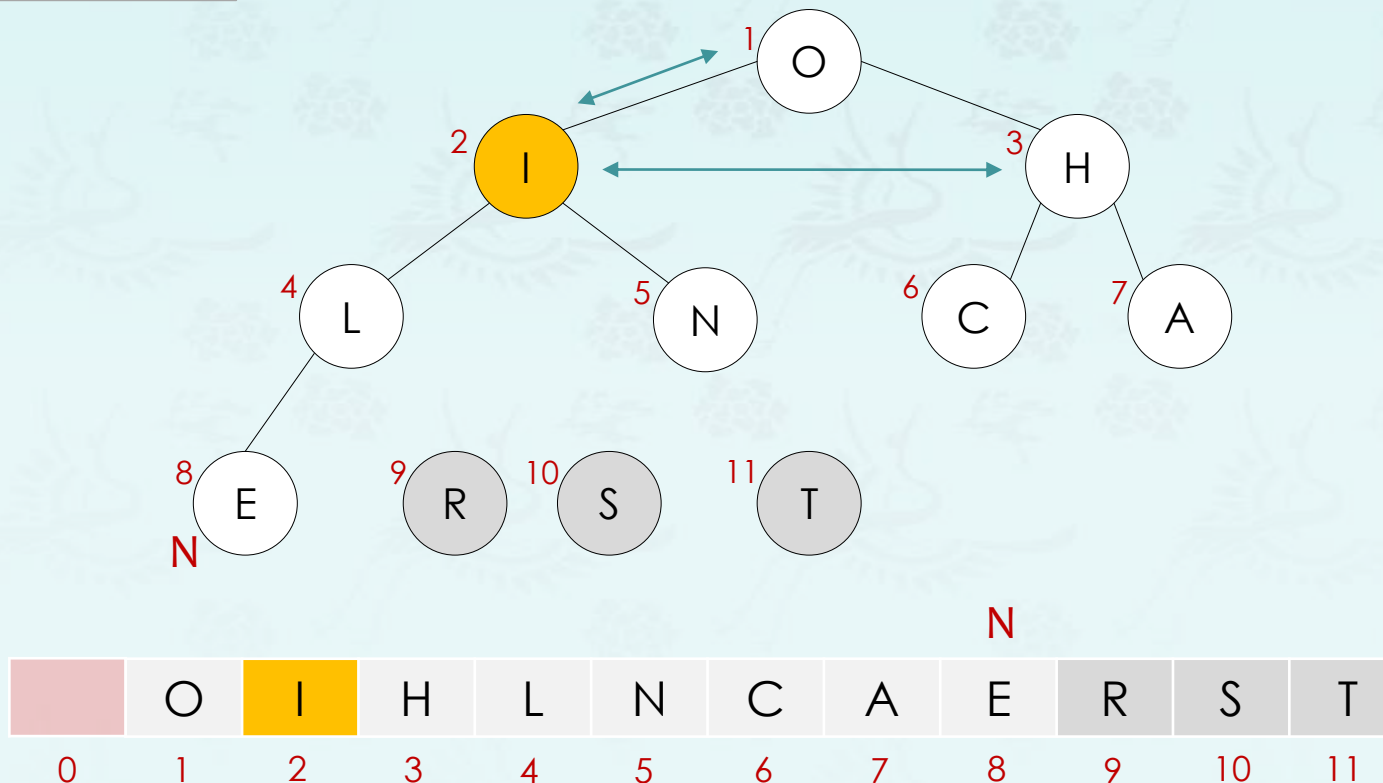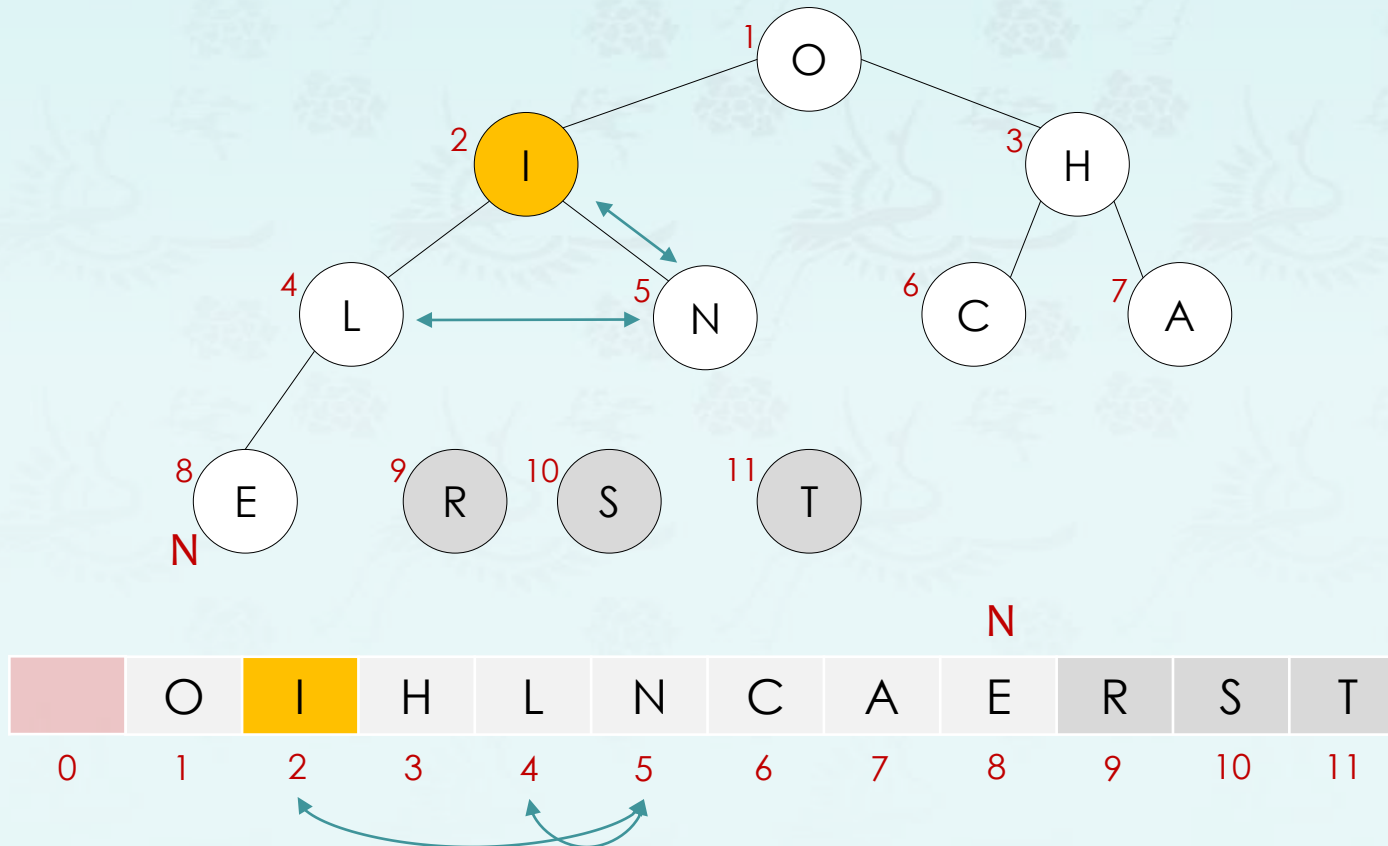
2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered

`swap(1, N);`	N=11
`sink(1, --N);`	N=10

`swap(1, N);`	N=10
`sink(1, --N);`	N=9

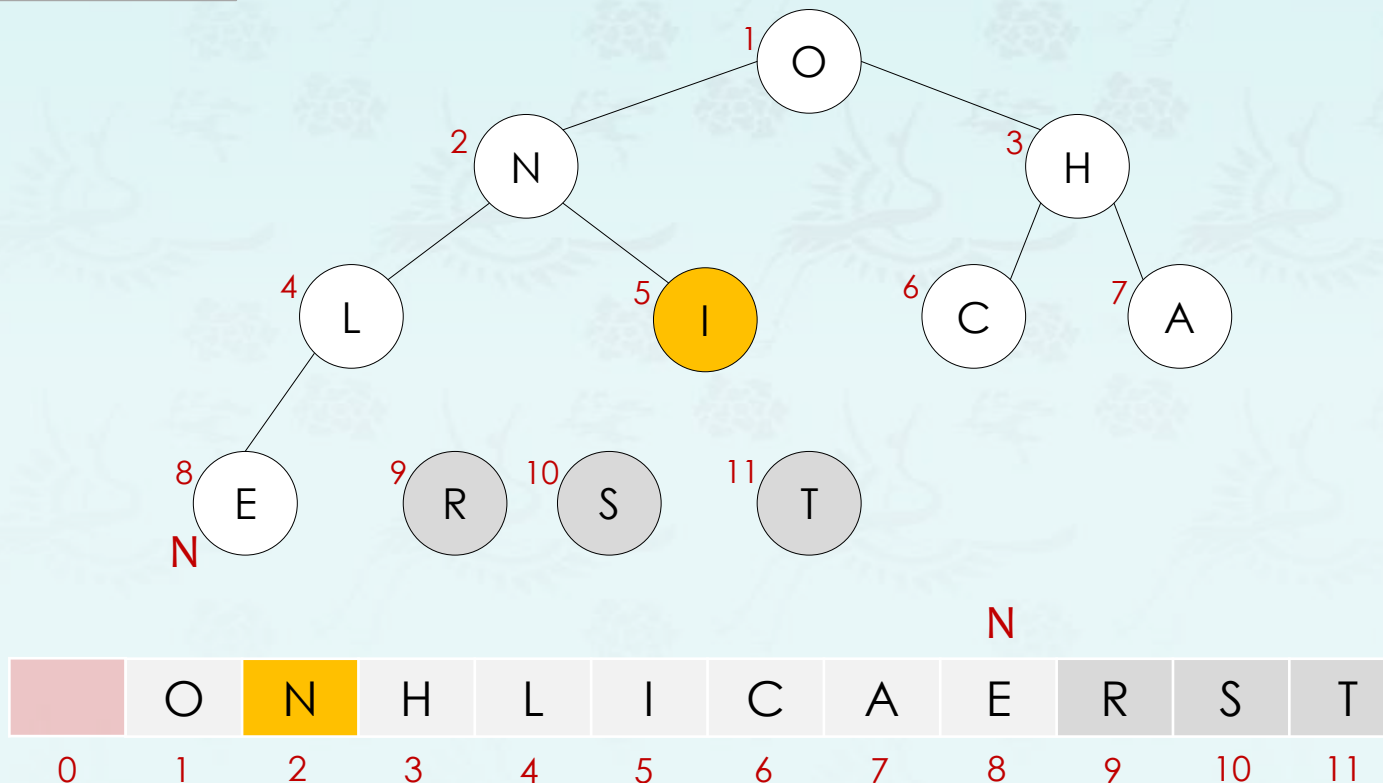|	
`swap(1, N);`	N=9
`sink(1, --N);`	N=8

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered

swap(1, N);	N=11
sink(1, --N);	N=10

swap(1, N);	N=10
sink(1, --N);	N=9

swap(1, N);	N=9
sink(1, --N);	N=8

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University
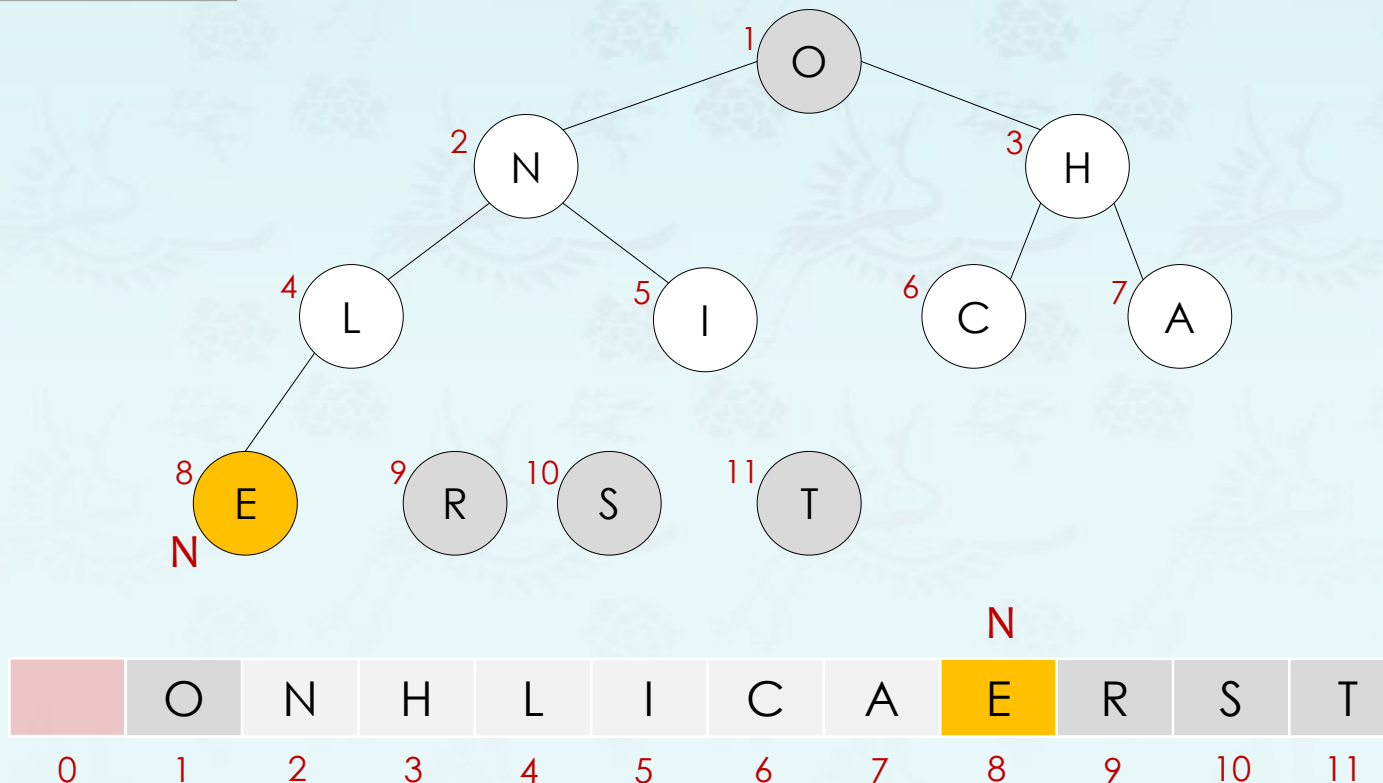
62

2nd **Pass: Repeatedly remove the maximum key**.

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

array in heap ordered

```
swap(1, N);         N=11
sink(1, --N);       N=10

swap(1, N);         N=10
sink(1, --N);       N=9

swap(1, N);         N=9
sink(1, --N);       N=8

swap(1, N);         N=8
sink(1, --N);       N=7
```

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

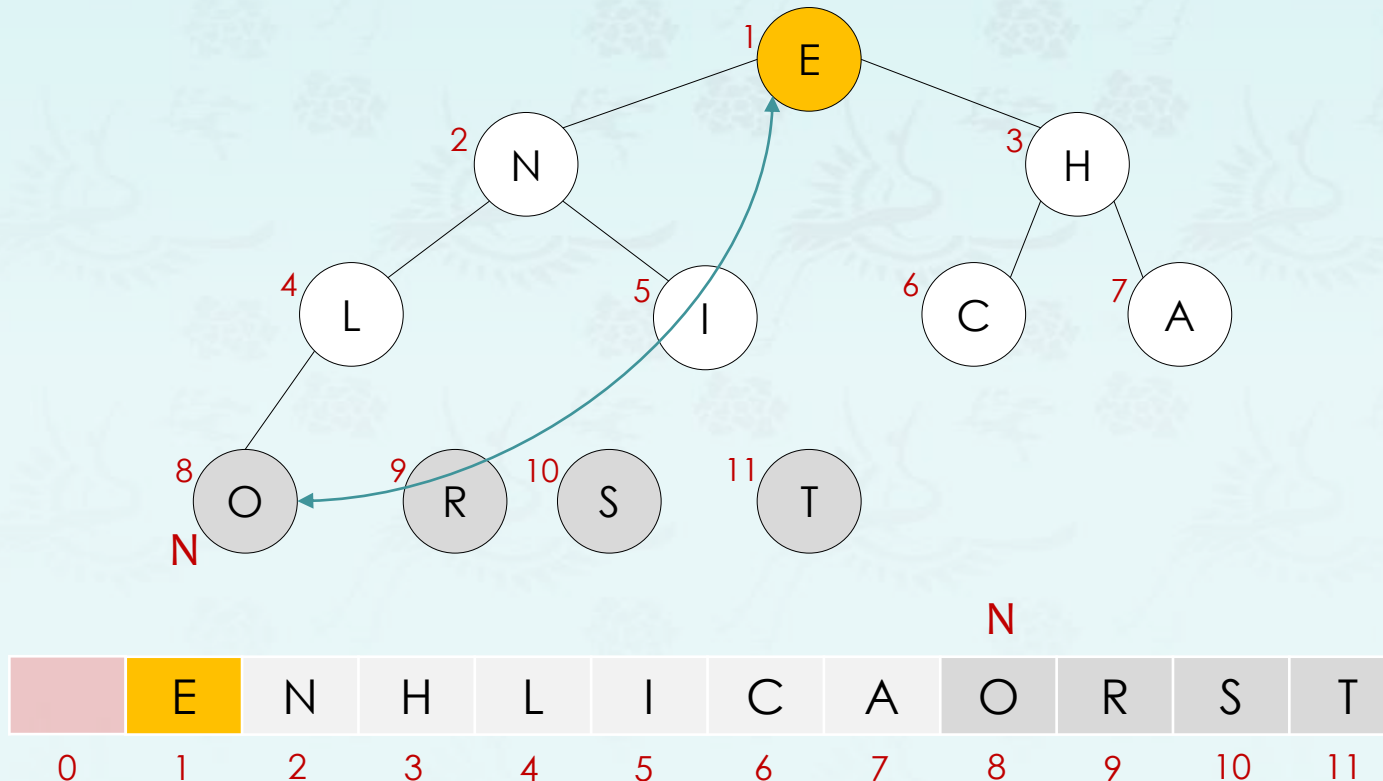**2ⁿᵈ Pass: Repeatedly remove the maximum key.**

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

array in heap ordered

```
swap(1, N);          N=11
sink(1, --N);        N=10

swap(1, N);          N=10
sink(1, --N);        N=9

swap(1, N);          N=9
sink(1, --N);        N=8

swap(1, N);          N=8
sink(1, --N);        N=7
```

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

64

2nd Pass: Repeatedly remove the maximum key.

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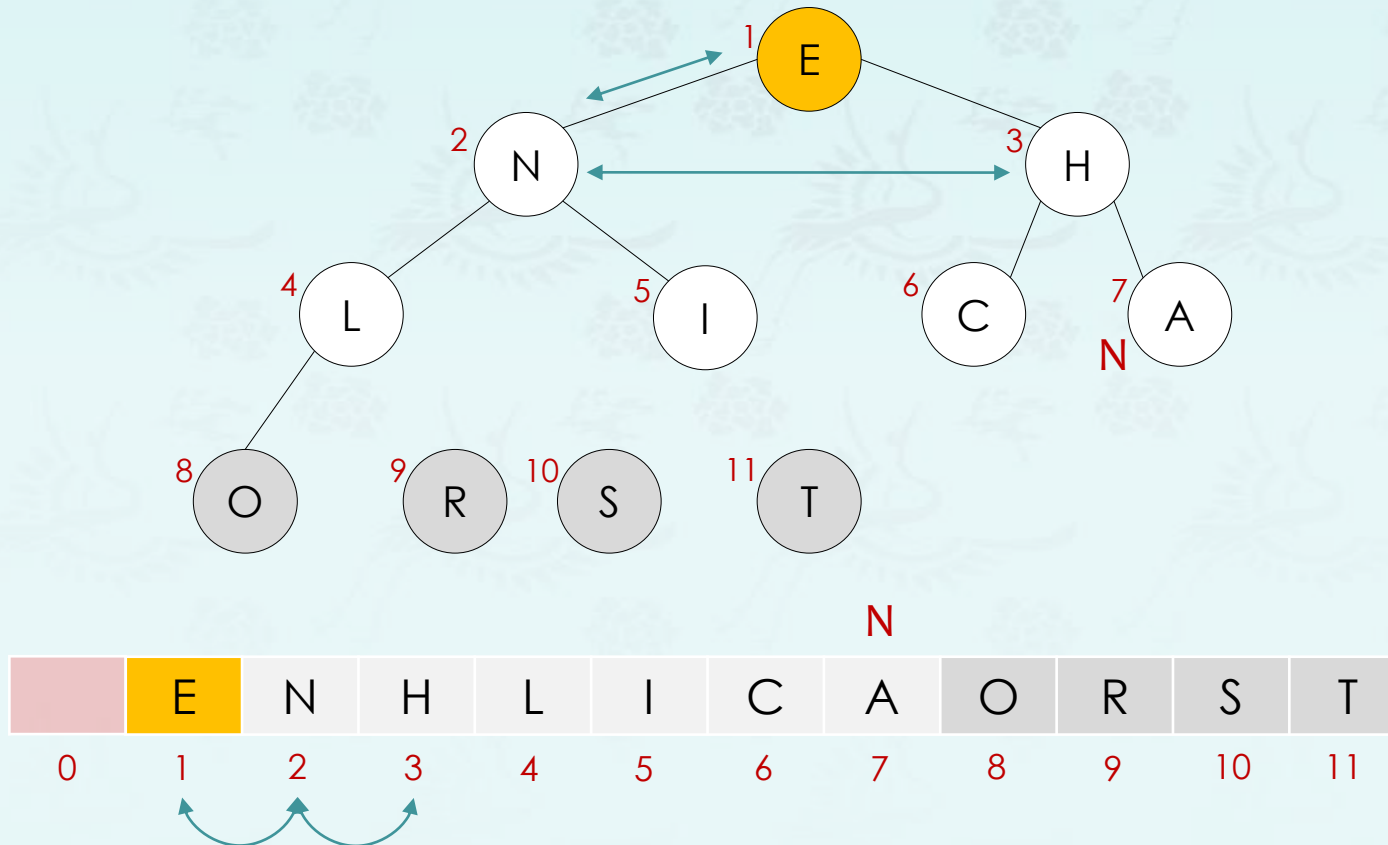

array in heap ordered

`swap(1, N);`	N=11
`sink(1, --N);`	N=10
`swap(1, N);`	N=10
`sink(1, --N);`	N=9
`swap(1, N);`	N=9
`sink(1, --N);`	N=8
`swap(1, N);`	N=8
`sink(1, --N);`	N=7

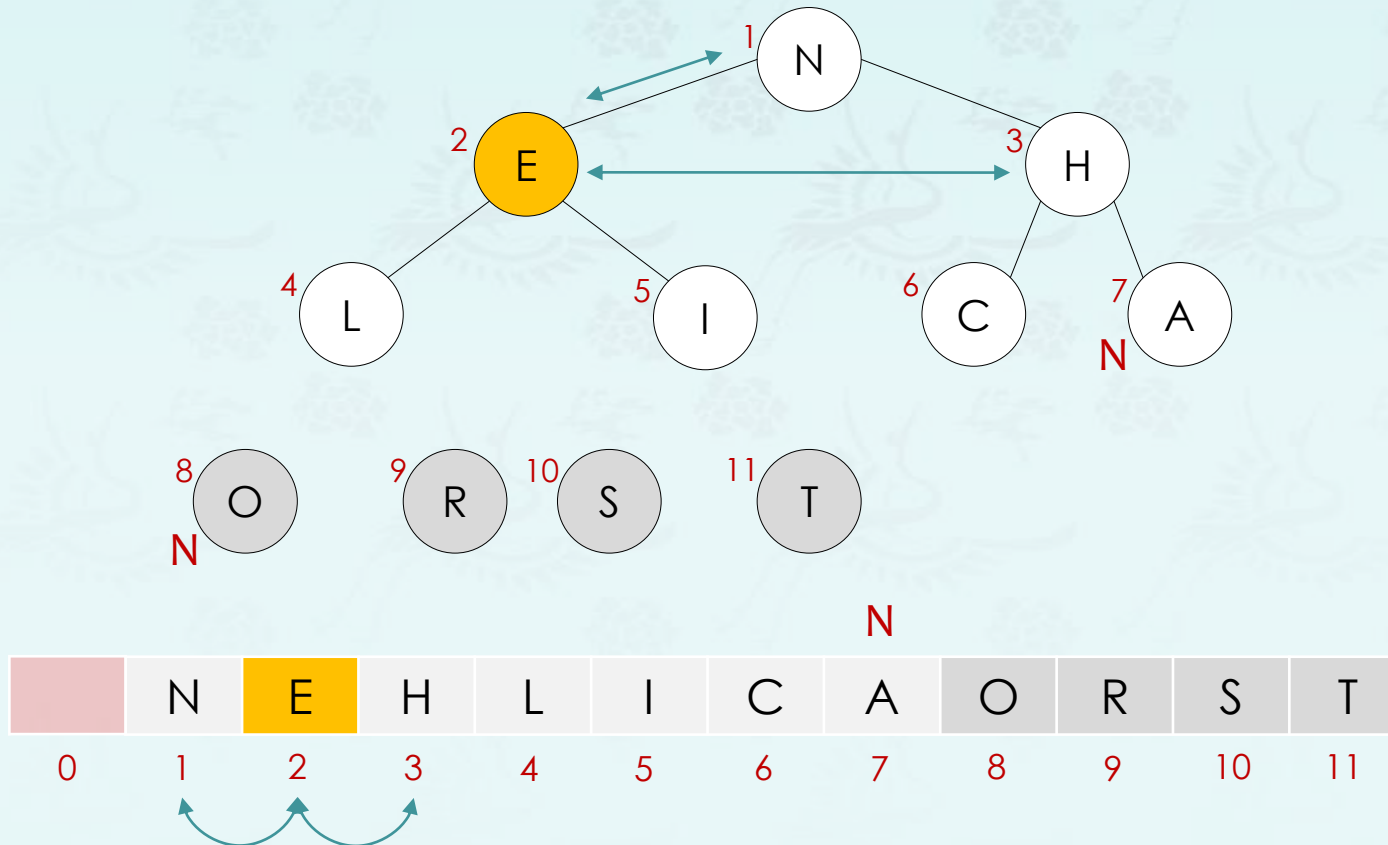*Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University*

Heapsort

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered


```
swap(1, N);        N=11
sink(1, --N);      N=10

swap(1, N);        N=10
sink(1, --N);      N=9

swap(1, N);        N=9
sink(1, --N);      N=8

swap(1, N);        N=8
sink(1, --N);      N=7
```
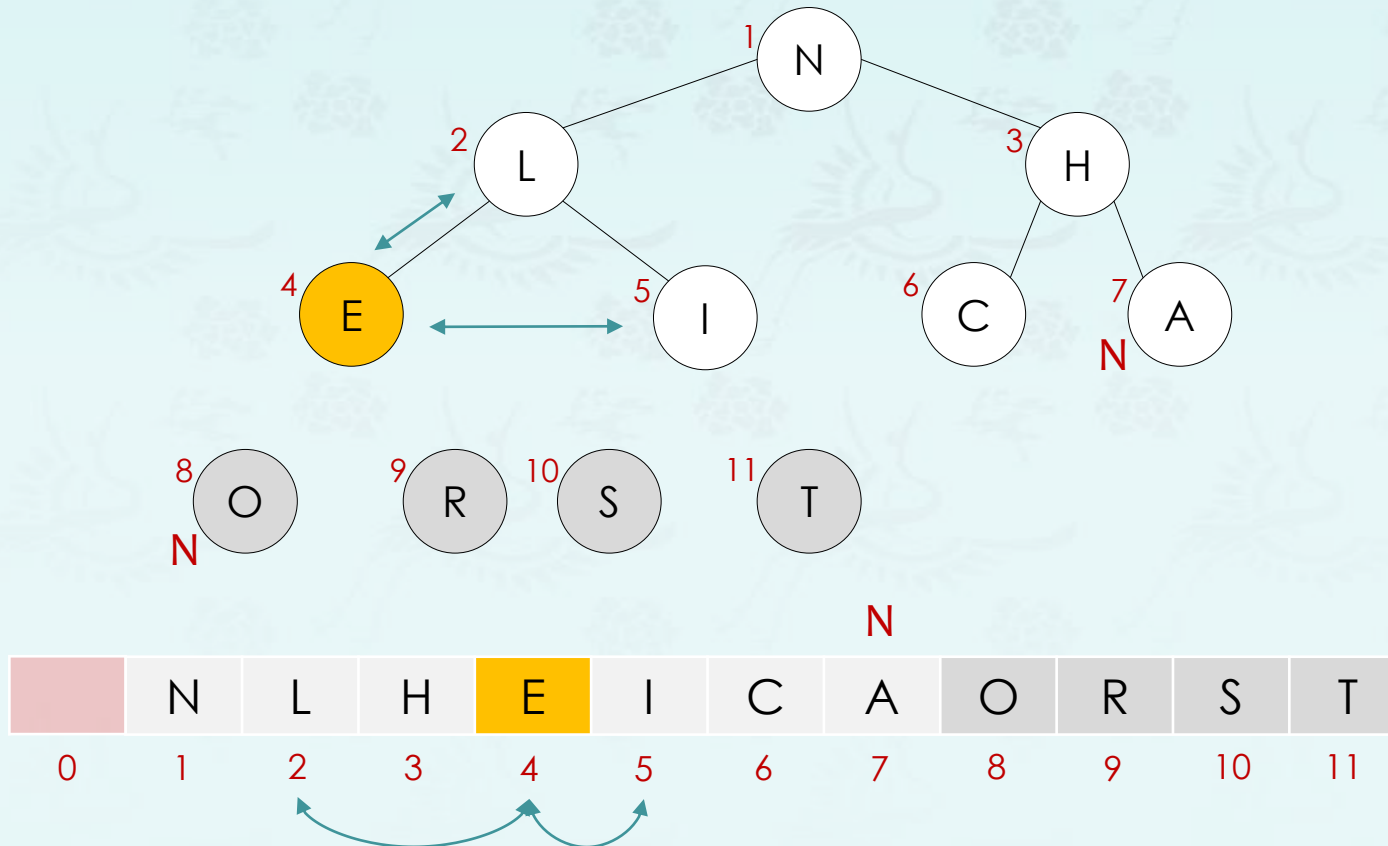
2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered

```
swap(1, N);        N=11
sink(1, --N);      N=10

swap(1, N);        N=10
sink(1, --N);      N=9

swap(1, N);        N=9
sink(1, --N);      N=8

swap(1, N);        N=8
sink(1, --N);      N=7
```

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

67

2nd Pass: Repeatedly remove the maximum key.

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

array in heap ordered

swap(1, N);	N=11
sink(1, --N);	N=10

swap(1, N);	N=10
sink(1, --N);	N=9

swap(1, N);	N=9
sink(1, --N);	N=8

swap(1, N);	N=8
sink(1, --N);	N=7

	N	L	H	E	I	C	A	O	R	S	T
0	1	2	3	4	5	6	7	8	9	10	11

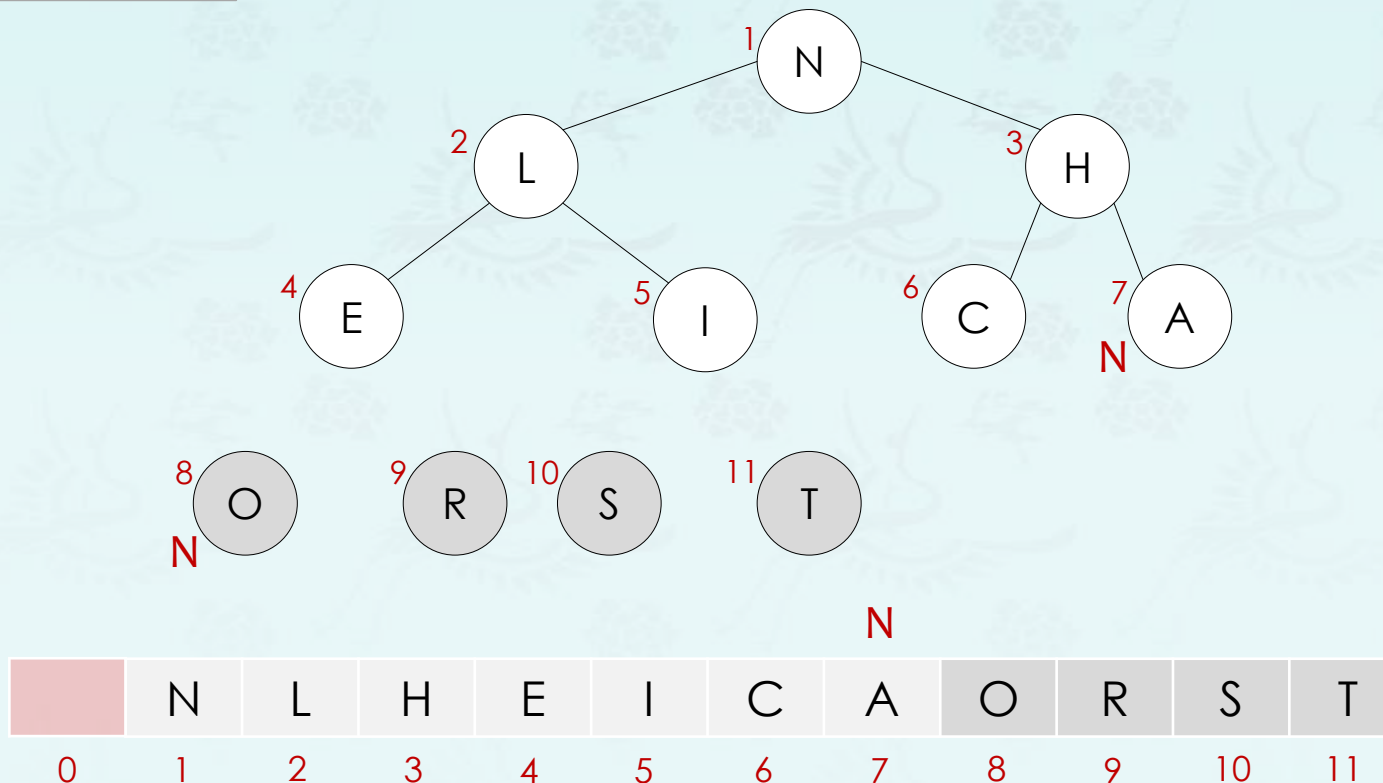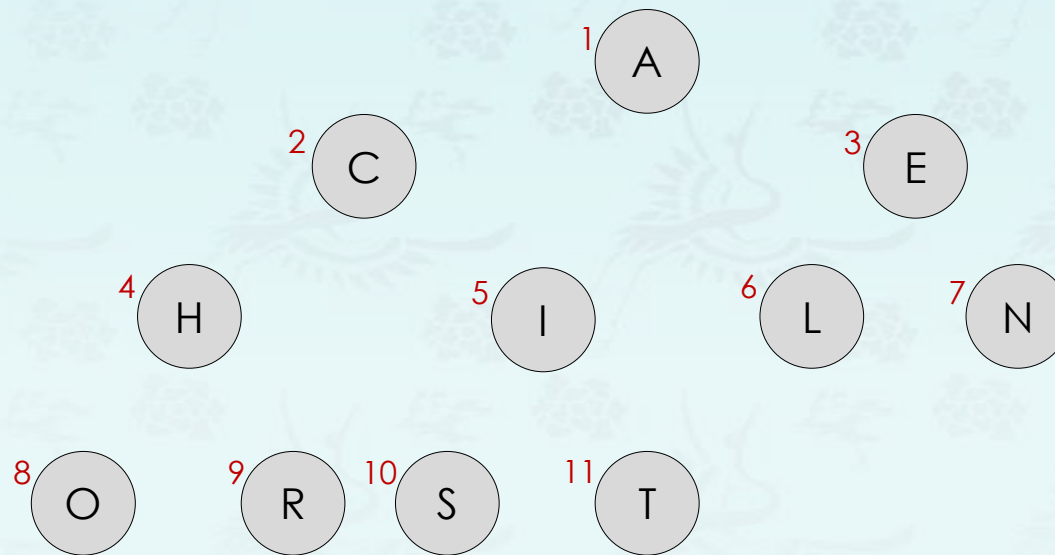*Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University*

Heapsort

2nd Pass: Repeatedly remove the maximum key.

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


```
swap(1, N);        N=11
sink(1, --N);      N=10
```

```
swap(1, N);        N=10
sink(1, --N);      N=9
```

```
swap(1, N);        N=9
sink(1, --N);      N=8
```

```
swap(1, N);        N=8
sink(1, --N);      N=7
```

```
swap(1, N);
s
```

```
swap(1, N);
s
```

```
swap(1, N);        N=2
sink(1, --N);      N=1
```

array sorted

	A	C	E	H	I	L	N	O	R	S	T
0	1	2	3	4	5	6	7	8	9	10	11

Heapsort

2nd Pass: Repeatedly remove the maximum key.

- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out
- After sorted, do not forget resetting N.

```
while (N > 1) {
    swap(a, 1, N);
    sink(a, 1, --N);
} // reset N = 11
```



```
swap(1, N);        N=11
sink(1, --N);      N=10
```

```
swap(1, N);        N=10
sink(1, --N);      N=9
```

```
swap(1, N);        N=9
sink(1, --N);      N=8
```

```
swap(1, N);        N=8
sink(1, --N);      N=7
```

```
swap(1, N);
```
```
swap(1, N);
```
```
swap(1, N);        N=2
sink(1, --N);      N=1
```

array sorted		A	C	E	H	I	L	N	O	R	S	T
	0	1	2	3	4	5	6	7	8	9	10	11

Heapsort tracing

```
Enter a word to sort: CHRISTALONE
Input String:[ CHRISTALONE ], N=11
Input  a[11]: C H R I S T A L O N E
ASCENDING:
1st pass(heapify - O(n)) begins:
    N=11 k=5 C H R I S T A L O N E
    N=11 k=4 C H R O S T A L I N E
    N=11 k=3 C H T O S R A L I N E
    N=11 k=2 C S T O N R A L I H E
    N=11 k=1 T S R O N C A L I H E
HeapOrdered: T S R O N C A L I H E
2nd pass(swap and sink - O(n log n) begins:
    N=10 k=1 S O R L N C A E I H
    N=9 k=1 R O H L N C A E I
    N=8 k=1 O N H L I C A E
    N=7 k=1 N L H E I C A
    N=6 k=1 L I H E A C
    N=5 k=1 I E H C A
    N=4 k=1 H E A C
    N=3 k=1 E C A
    N=2 k=1 C A
    N=1 k=1 A
a[11]: A C E H I L N O R S T
```

← printed in main()
← printed in main()

1st path
printed in sink()

← printed in heapSort()

2nd path
printed in sink()

← printed in main()

NOTE: This implementation does not sort the first element in the array.
NOTE: N=?? k=? .... lines are outputs at the end of each sink()

Binary heap operations time complexity with N items:

- Level of heap is $\lfloor \log_2 N \rfloor$
- insert: O(log N) for each insert
    - In practice, expect less
- delete: O(log N)   // deleting root node in min/max heap
- decreaseKey: O(log N)
- increaseKey: O(log N)
- remove: O(log N)   // removing a node in any location

- **Heapify():** O(N)
- **Heapsort():** O(n log n)
-         Because O(N) heapify + O(n log n) remove nodes = O(n log n)
- **Proof:**
    - https://stackoverflow.com/questions/9755721/how-can-building-a-heap-be-on-time-complexity
    - https://www.growingwiththeweb.com/data-structures/binary-heap/build-heap-proof/
    - https://www.quora.com/How-is-the-time-complexity-of-building-a-heap-is-o-n
    - http://www.cs.umd.edu/~meesh/351/mount/lectures/lect14-heapsort-analysis-part.pdf

Data Structures
Chapter 5: Heap and Priority Queue

1. Heap & Priority Queue
2. Heapsort
   - Heap Construction – Heapify
   - Heapsort
   - Time Complexity
3. **Heap & PQ Coding**