Data Structures

# Sorting Technique – Heap Sort

Team Emertxe

# Heap Sort

# Heap Sort

- arr[SIZE]

SIZE = 10

# Heap Sort

- arr[SIZE]

SIZE = 10

| 1 | 14 | 10 | 8 | 7 | 9 | 3 | 2 | 4 | 6 |
|---|---|---|---|---|---|---|---|---|---|

arr[0] arr[1] arr[2] arr[3] arr[4] arr[5] arr[6] arr[7] arr[8] arr[9]

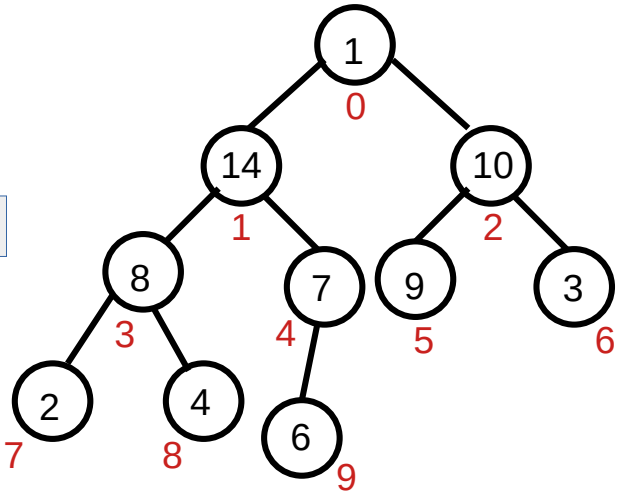
# Heap Sort

- arr[SIZE]

SIZE = 10

| 1 | 14 | 10 | 8 | 7 | 9 | 3 | 2 | 4 | 6 |
|---|---|---|---|---|---|---|---|---|---|

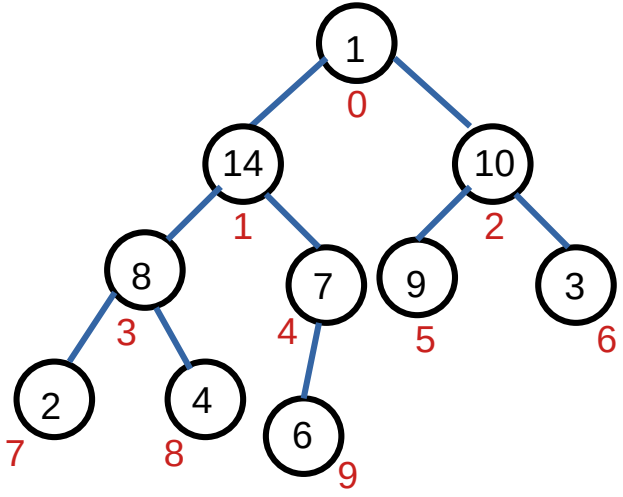
arr[0] arr[1] arr[2] arr[3] arr[4] arr[5] arr[6] arr[7] arr[8] arr[9]



# Heap Sort

- arr[SIZE]

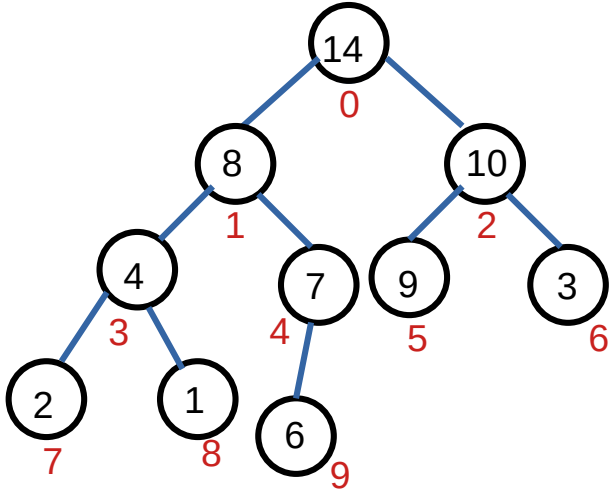
SIZE = 10



# Heap Sort

- arr[SIZE]

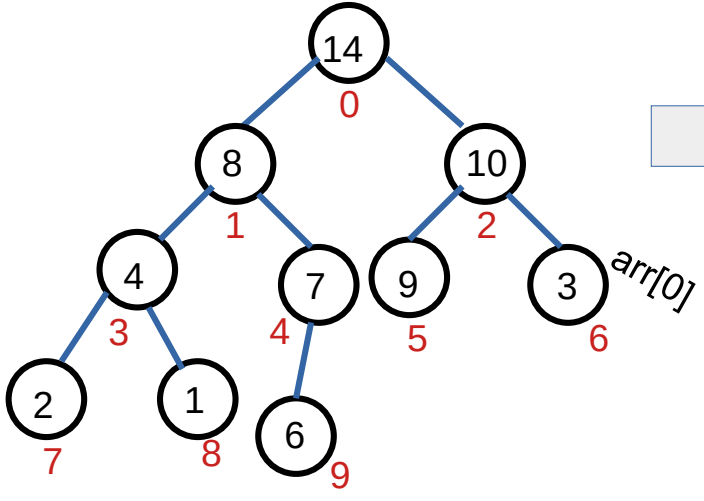
SIZE = 10



# Heap Sort

- arr[SIZE]

SIZE = 10

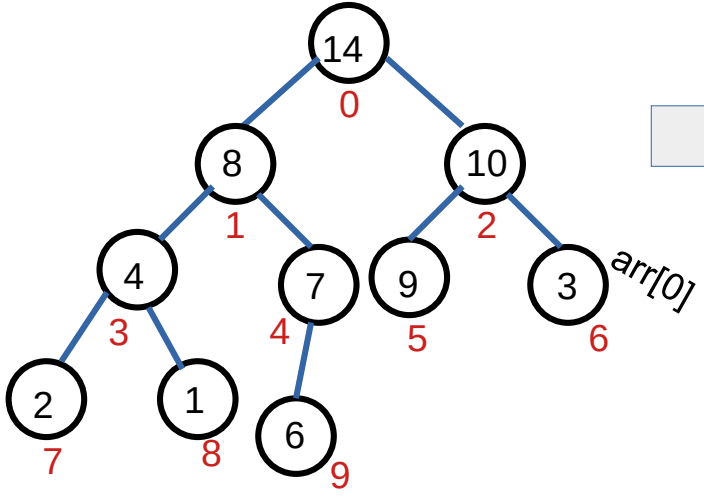


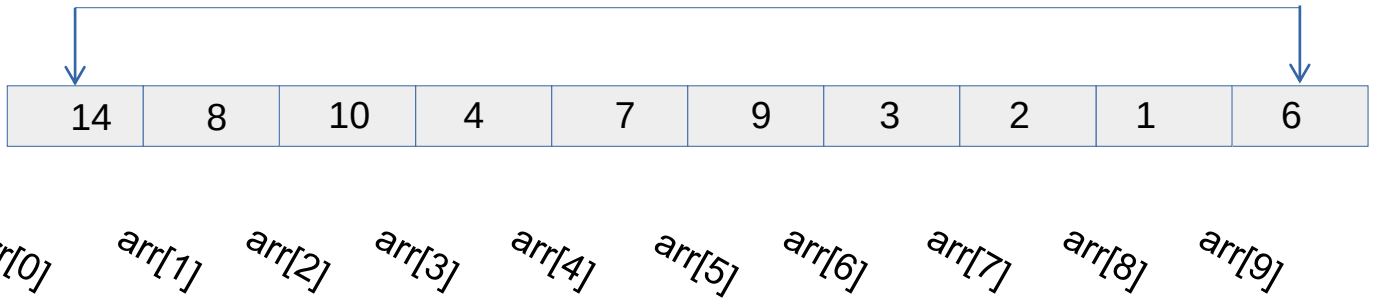

| 14 | 8 | 10 | 4 | 7 | 9 | 3 | 2 | 1 | 6 |
|---|---|---|---|---|---|---|---|---|---|
| arr[0] | arr[1] | arr[2] | arr[3] | arr[4] | arr[5] | arr[6] | arr[7] | arr[8] | arr[9] |

# Heap Sort

- arr[SIZE]

SIZE = 10

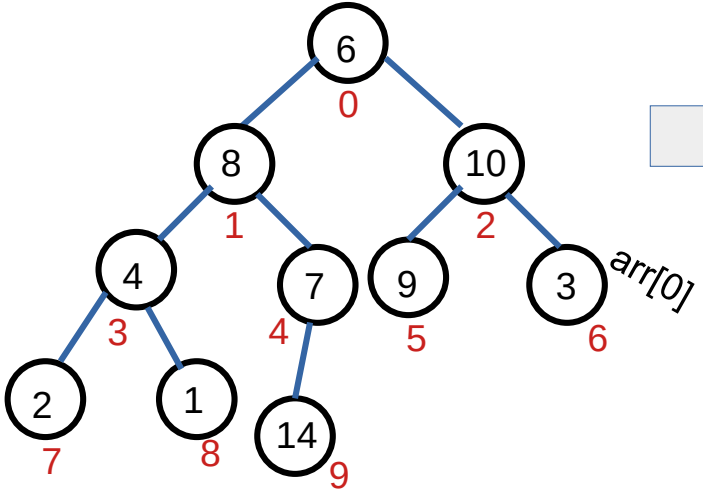


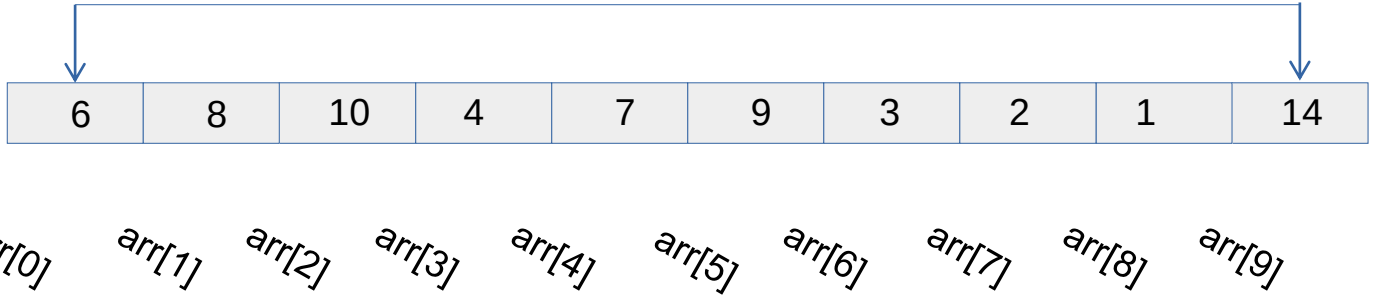



# Heap Sort

- arr[SIZE]

SIZE = 10

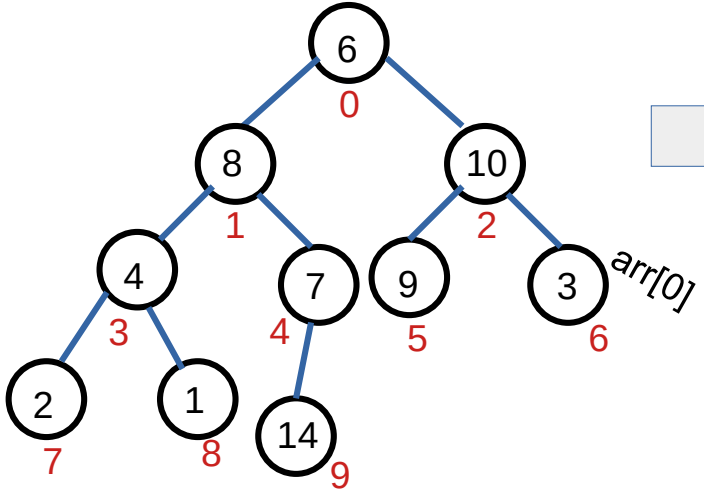


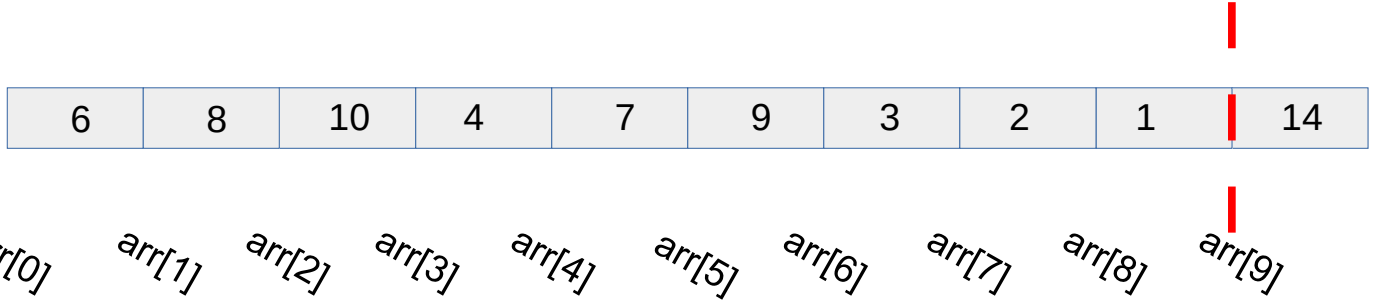



# Heap Sort

- arr[SIZE]

SIZE = 10

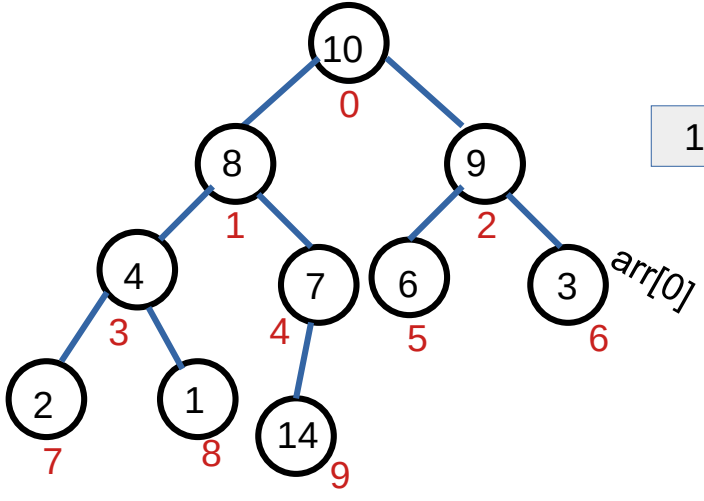


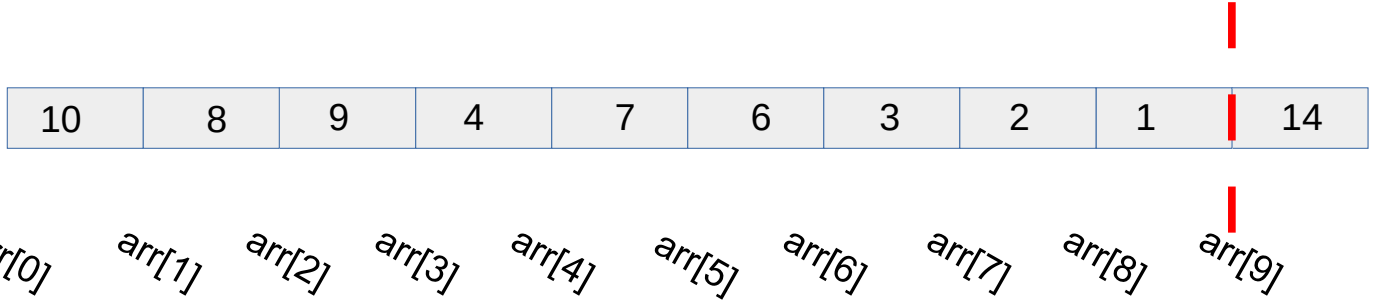



# Heap Sort

- arr[SIZE]

SIZE = 10

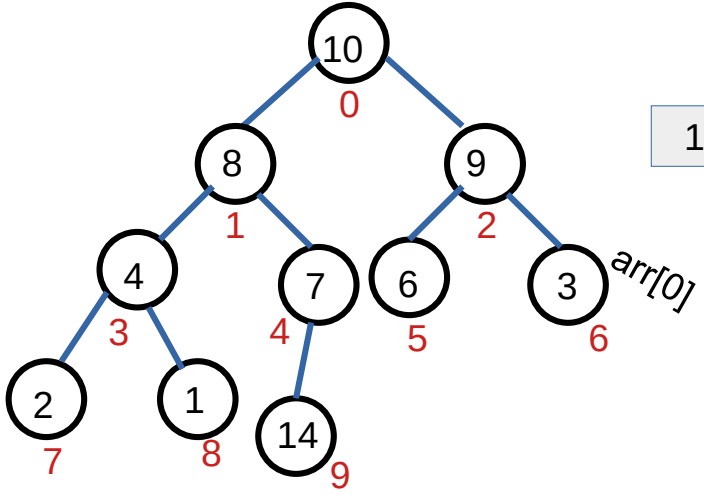


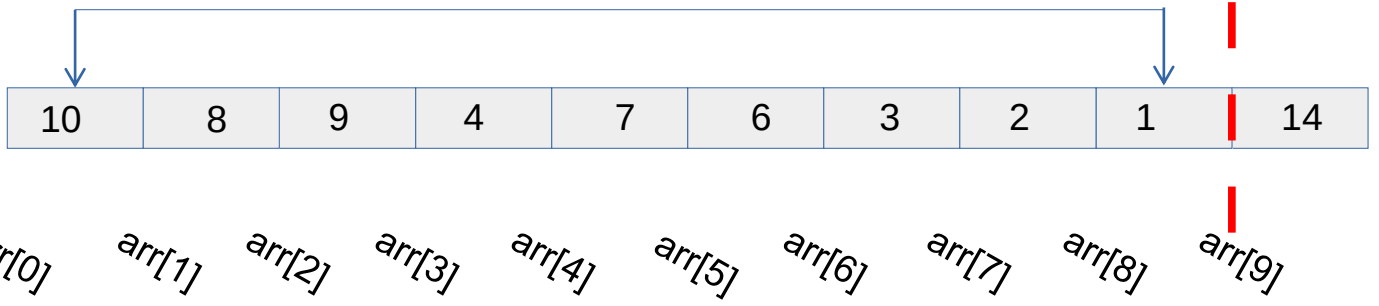



# Heap Sort

- arr[SIZE]

SIZE = 10

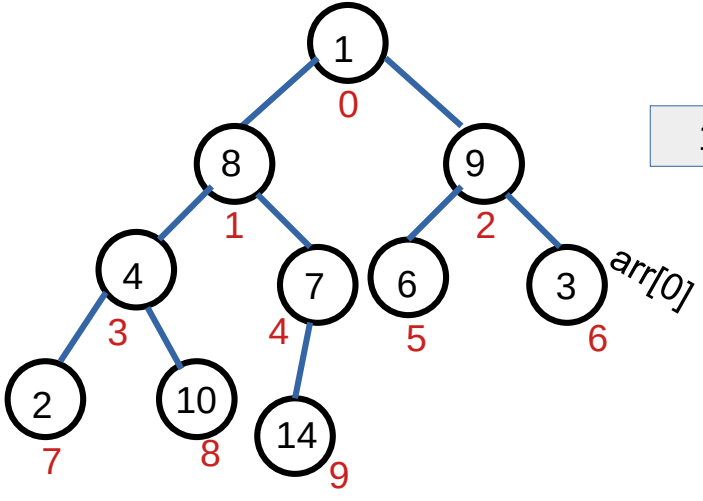


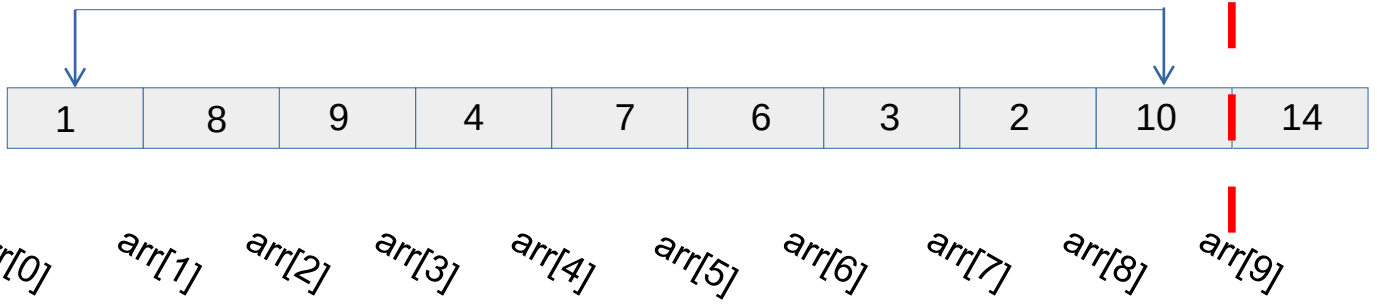



# Heap Sort

- arr[SIZE]

SIZE = 10

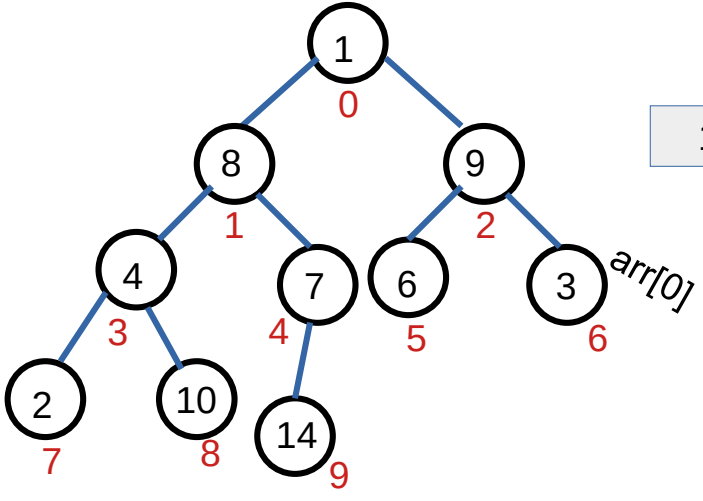


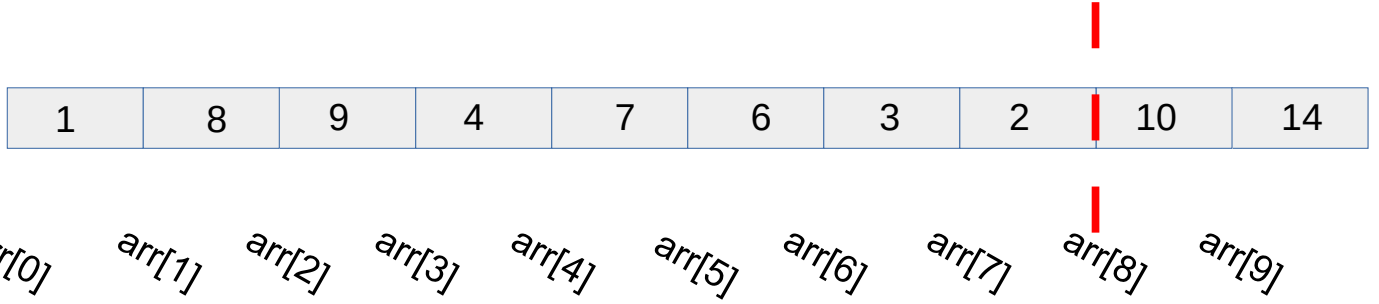



# Heap Sort

- arr[SIZE]

SIZE = 10

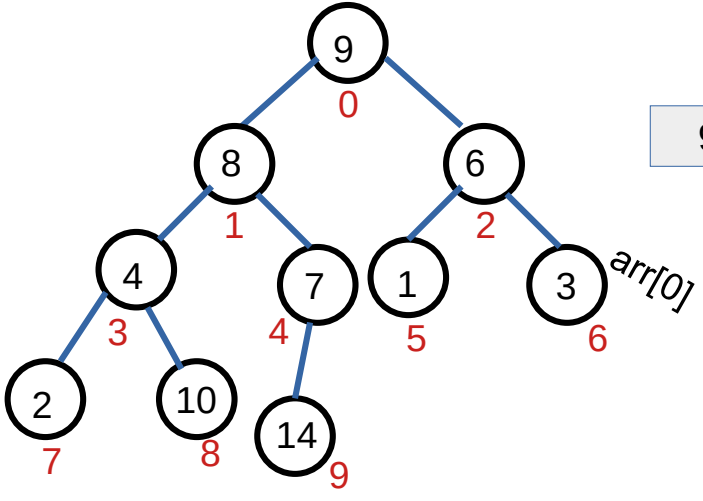


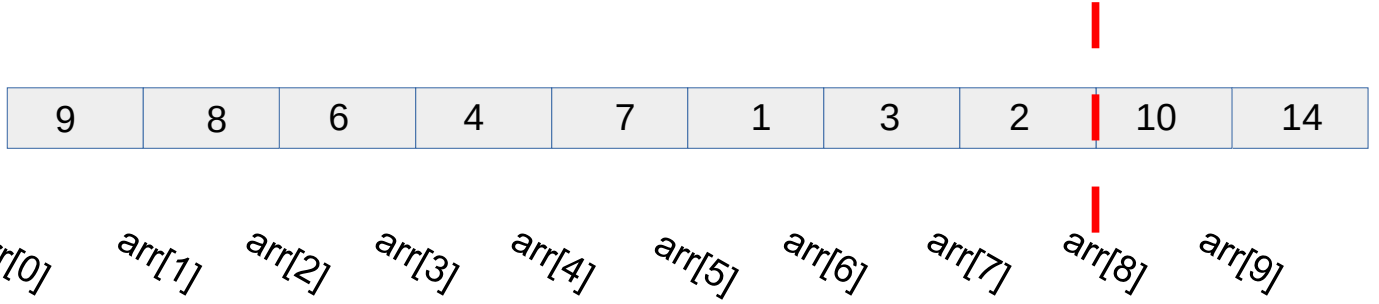



# Heap Sort

- arr[SIZE]

SIZE = 10

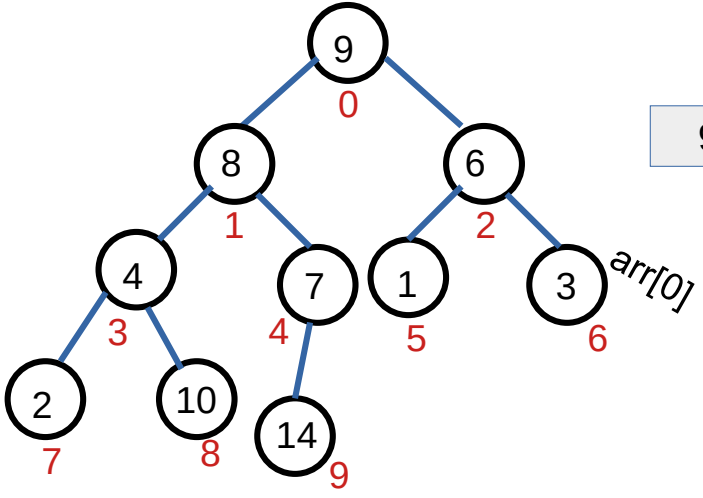


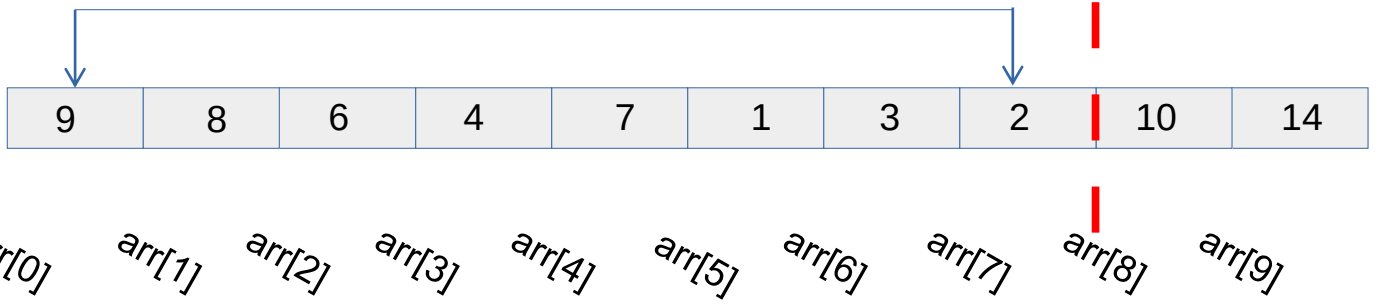



# Heap Sort

- arr[SIZE]

SIZE = 10

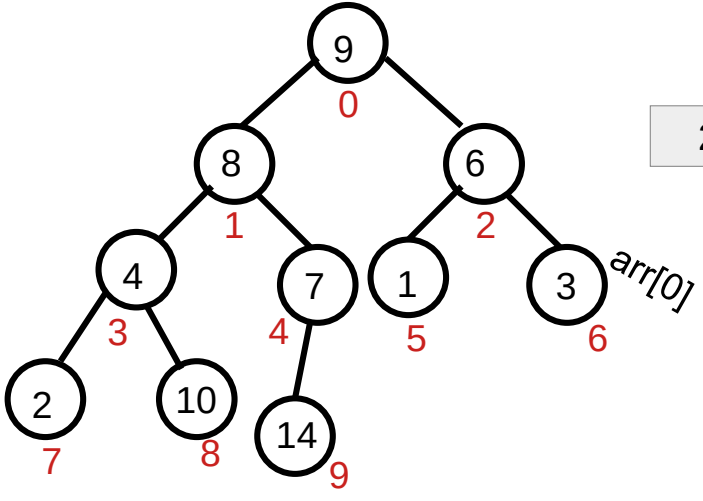


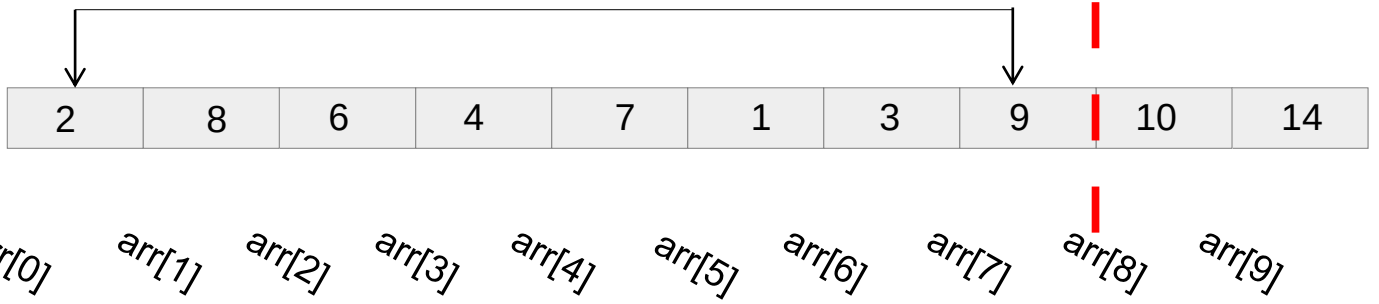



# Heap Sort

- arr[SIZE]

SIZE = 10

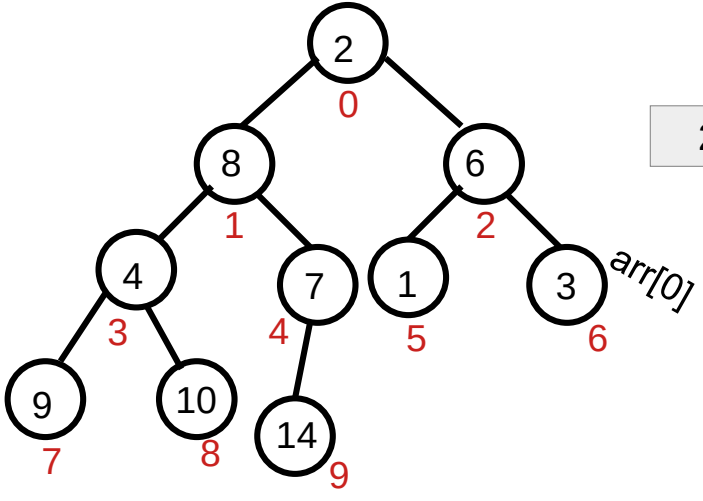


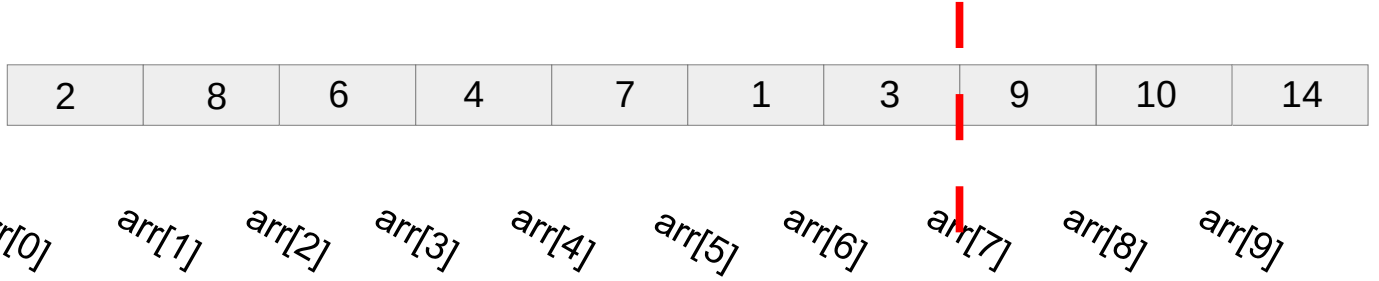



# Heap Sort

- arr[SIZE]

SIZE = 10

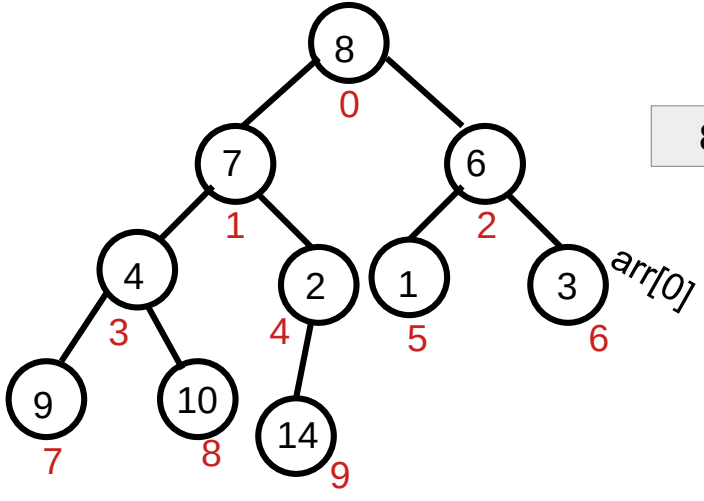


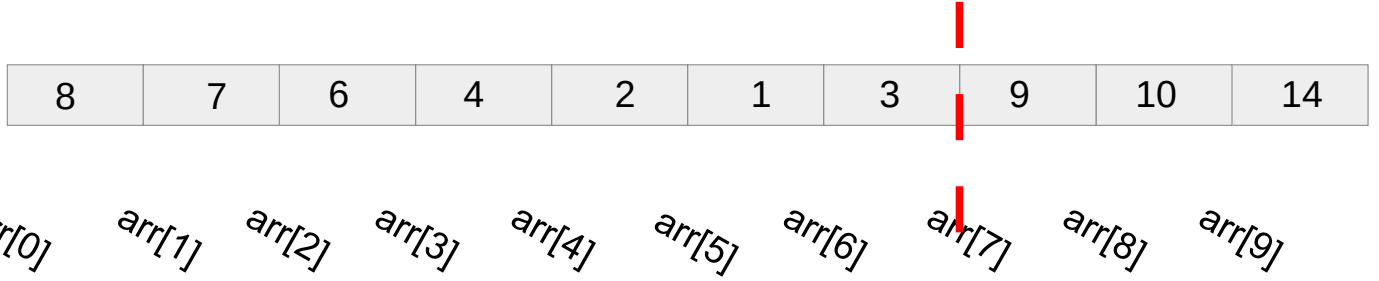



# Heap Sort

- arr[SIZE]

SIZE = 10

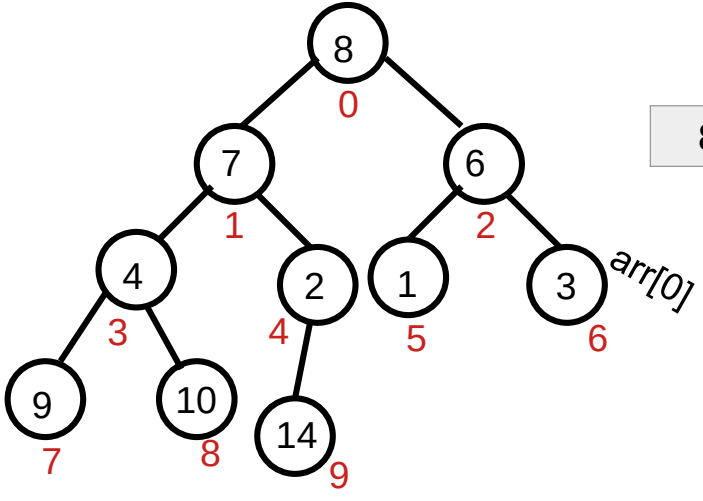


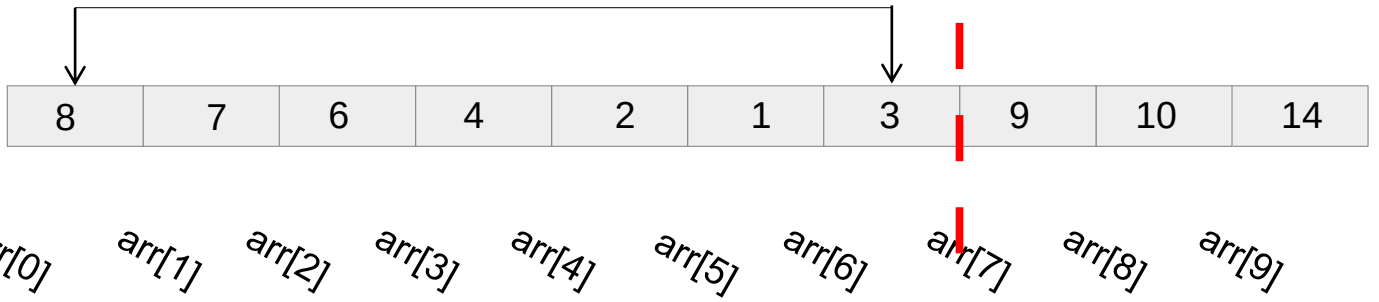



# Heap Sort

- arr[SIZE]

SIZE = 10

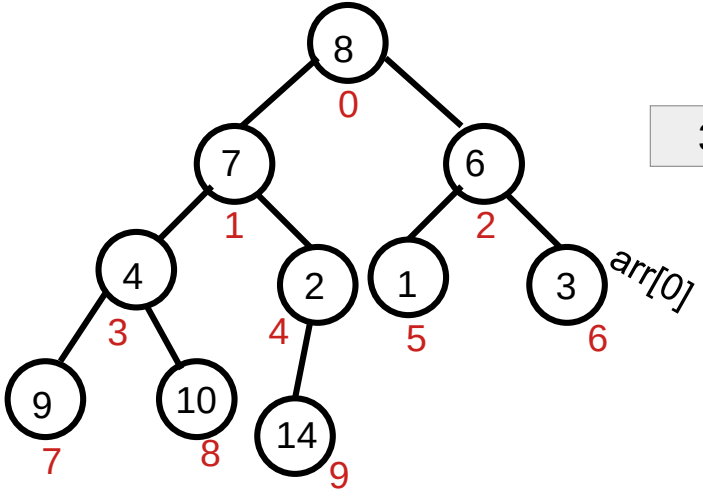


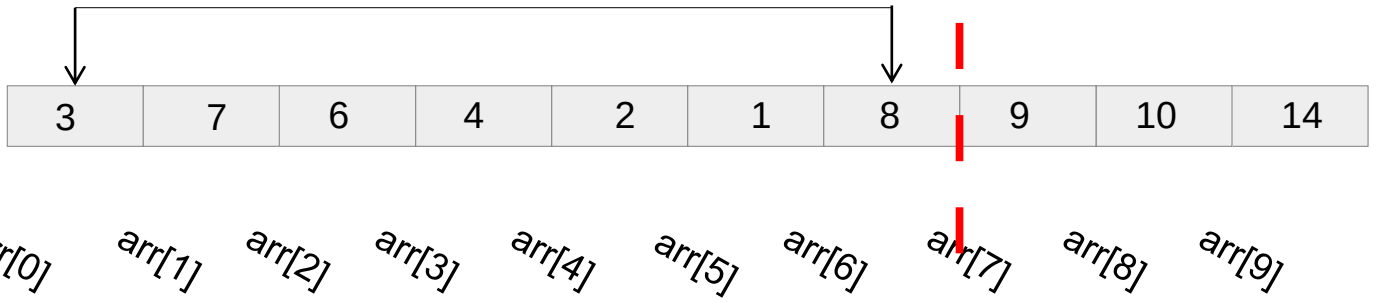



# Heap Sort

- arr[SIZE]

SIZE = 10

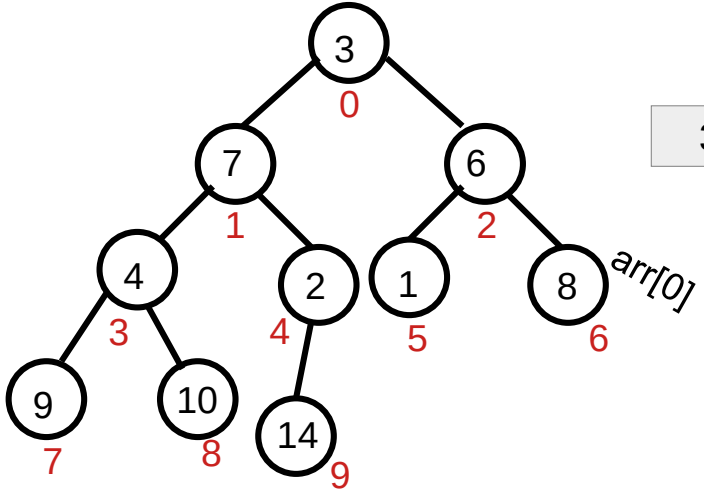


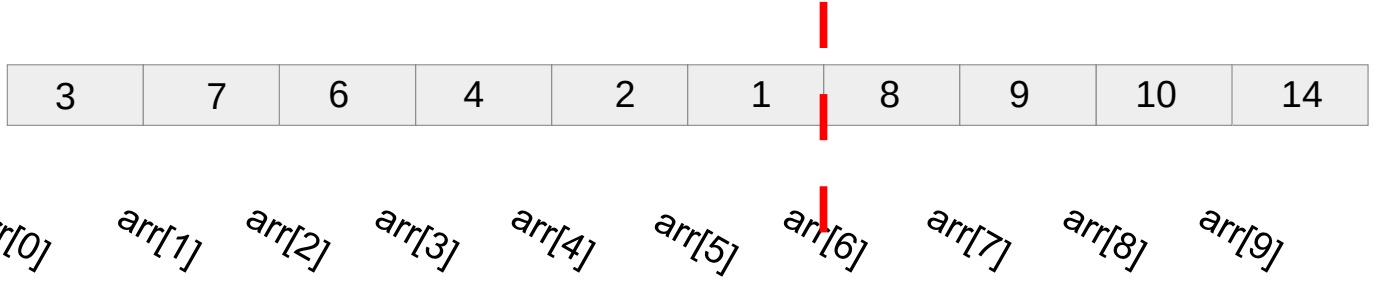



# Heap Sort

- arr[SIZE]

SIZE = 10

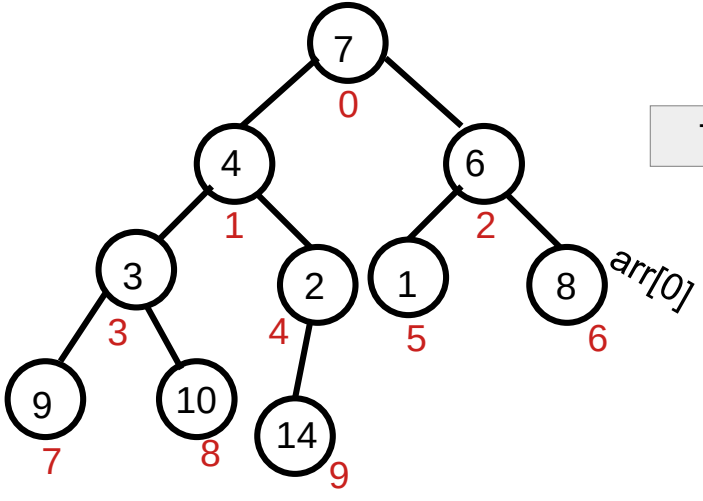


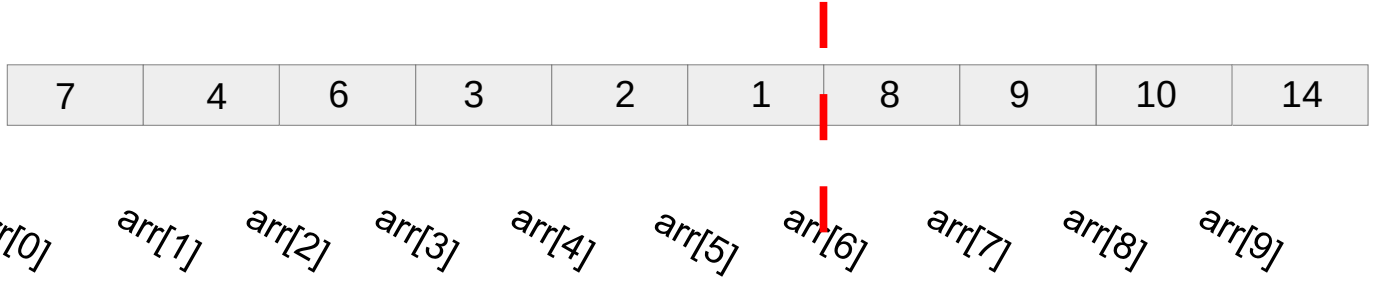



# Heap Sort

- arr[SIZE]

SIZE = 10

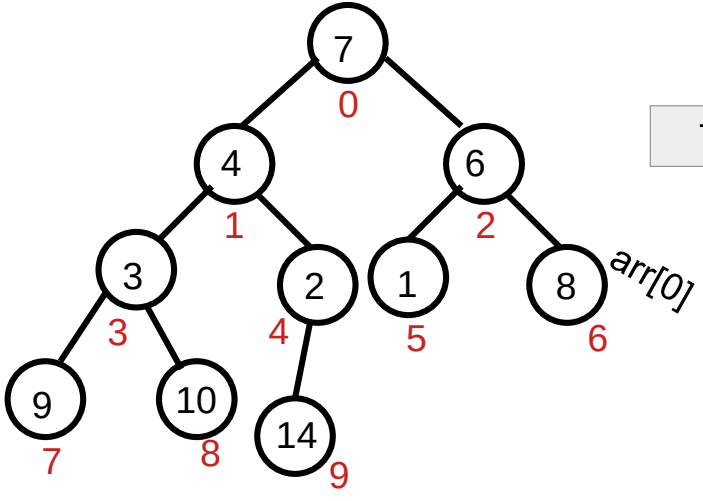


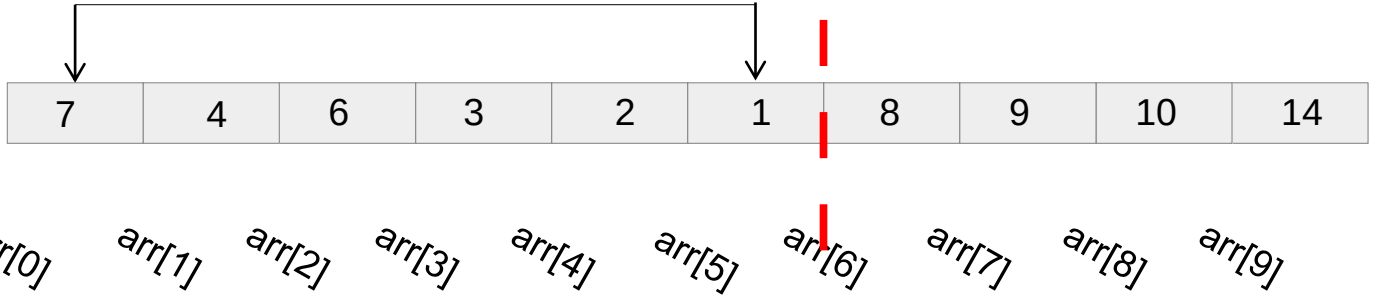



# Heap Sort

- arr[SIZE]

SIZE = 10

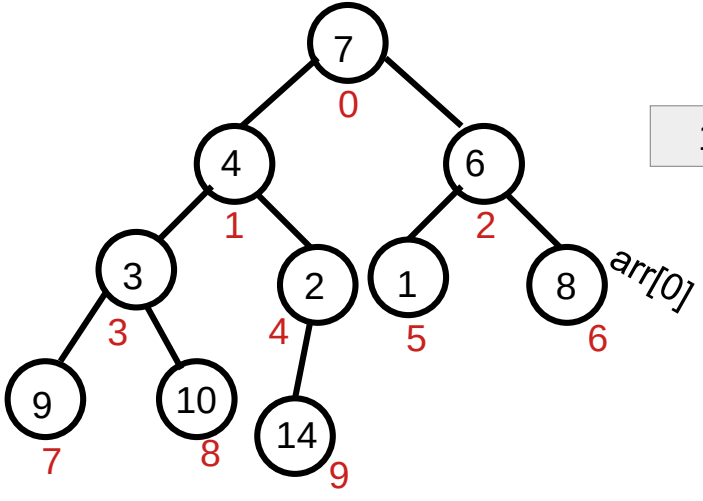


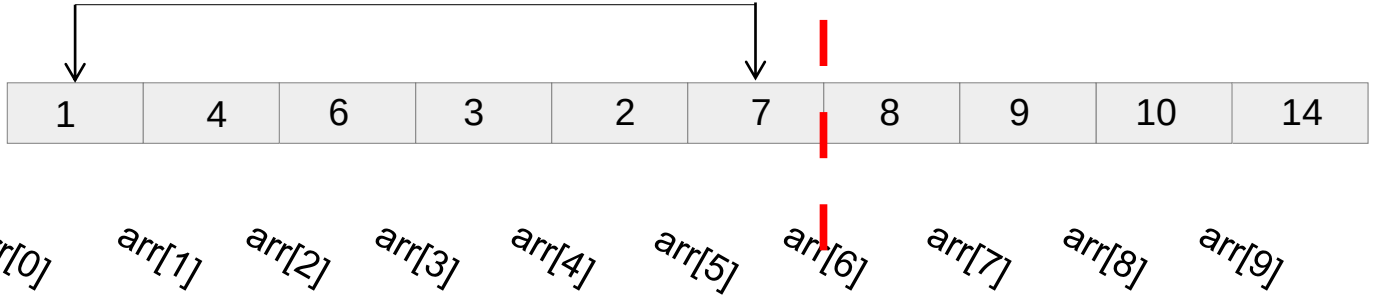



# Heap Sort

- arr[SIZE]

SIZE = 10

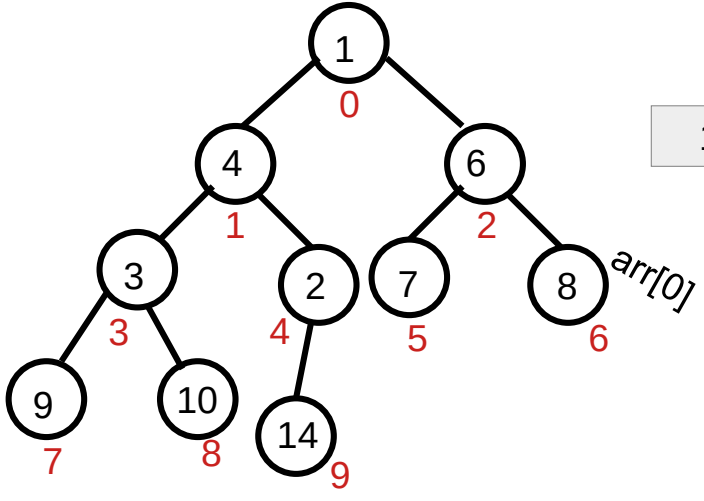


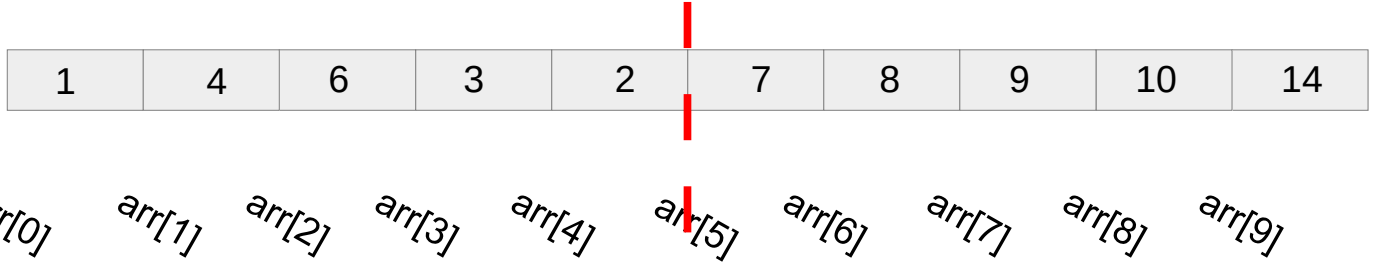



# Heap Sort

- arr[SIZE]

SIZE = 10

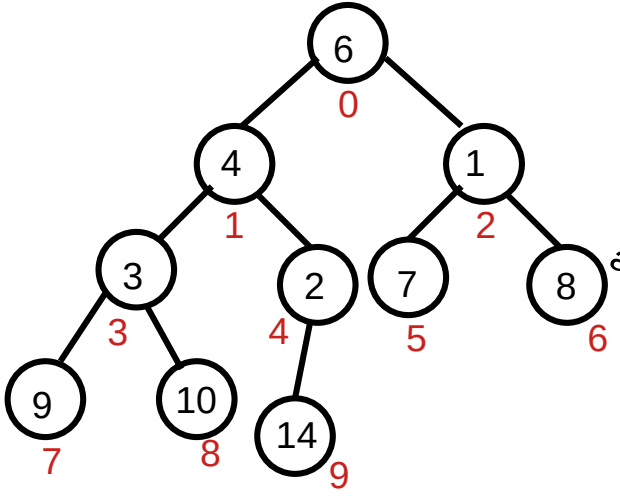


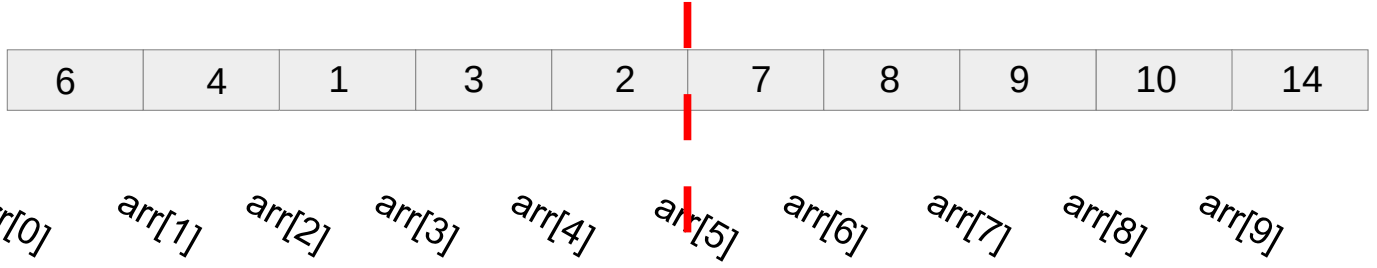



# Heap Sort

- arr[SIZE]

SIZE = 10

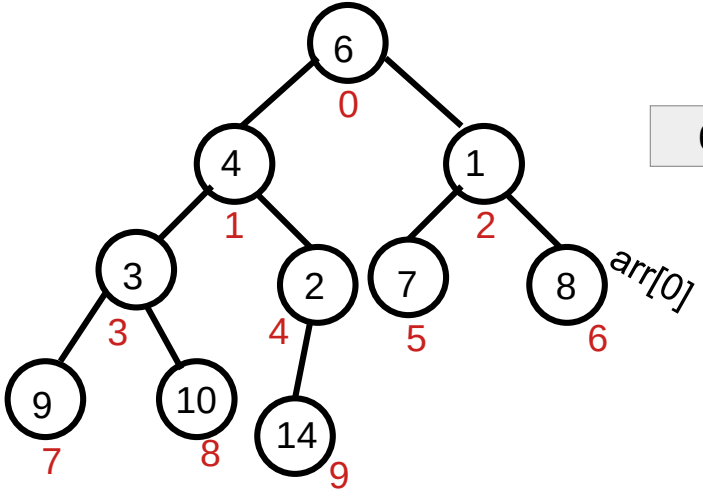


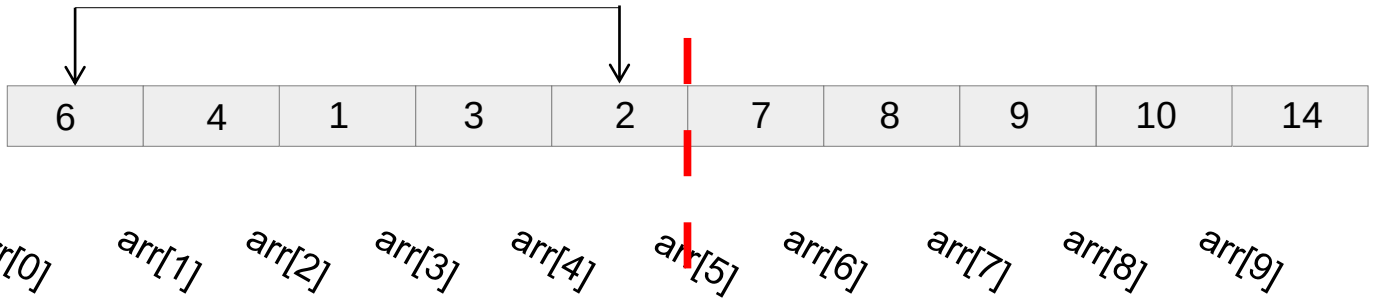



# Heap Sort

- arr[SIZE]

SIZE = 10

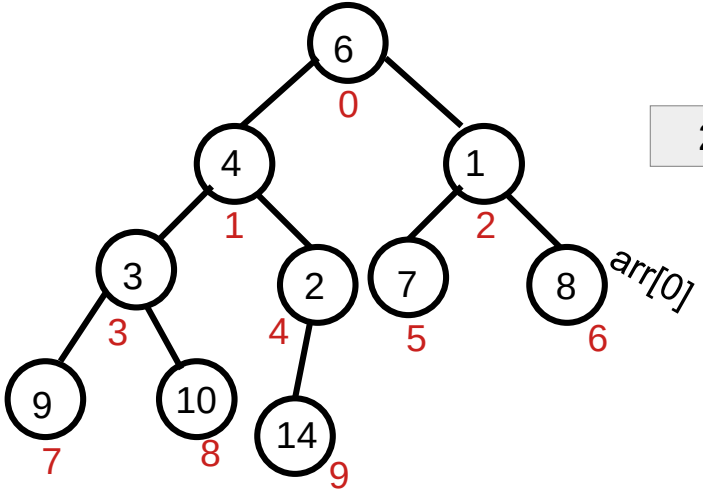


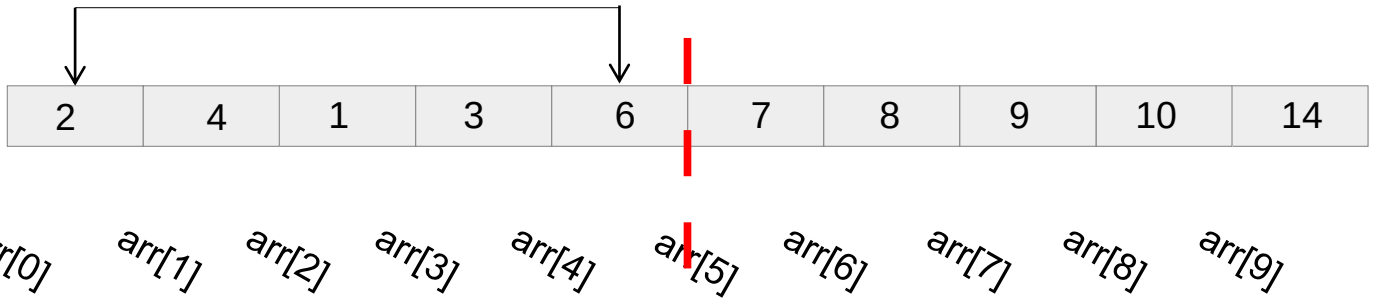



# Heap Sort

- arr[SIZE]

SIZE = 10

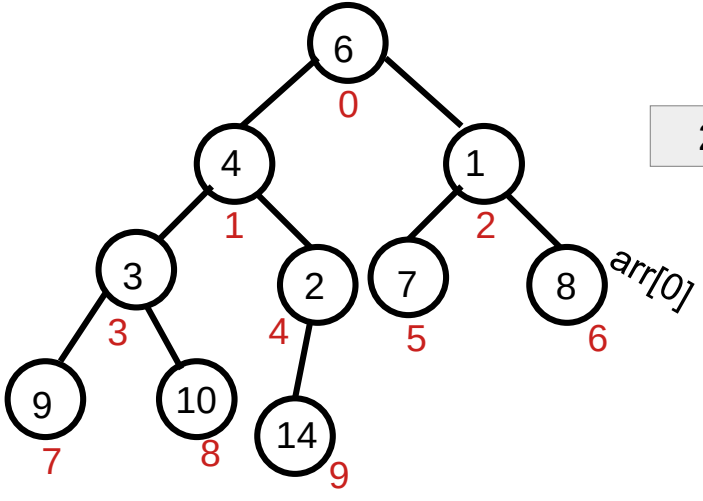


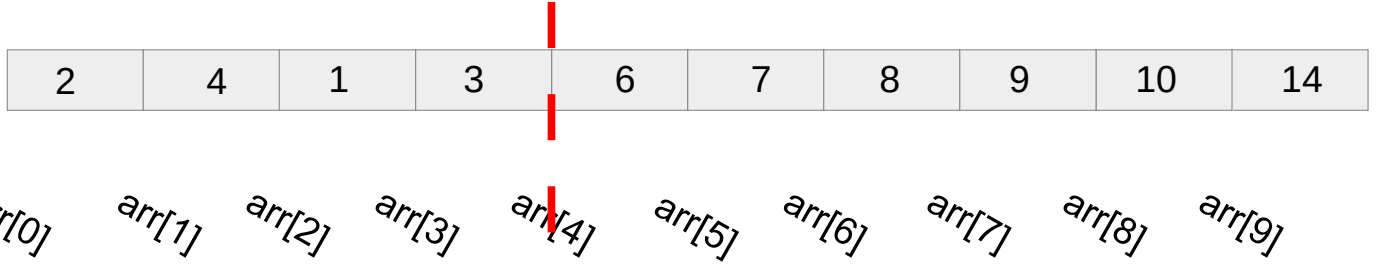



# Heap Sort

- arr[SIZE]

SIZE = 10

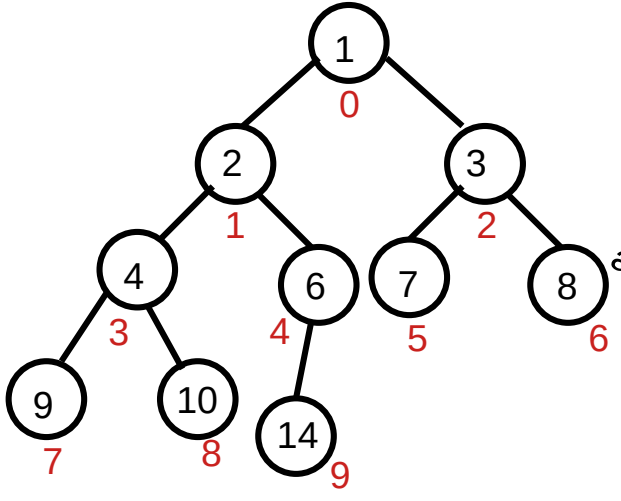


# Heap Sort

- arr[SIZE]

SIZE = 10




| 1 | 2 | 3 | 4 | 6 | 7 | 8 | 9 | 10 | 14 |
|---|---|---|---|---|---|---|---|---|---|
| arr[0] | arr[1] | arr[2] | arr[3] | arr[4] | arr[5] | arr[6] | arr[7] | arr[8] | arr[9] |

EMERTXE

# Algorithm

## heap_sort(arr,size):

```
build_maxheap(arr,size)
index = size – 1
while(index >= 0 )
        swap(arr[0],arr[index])
        maxheapify(arr,0,index)
        Decrement index
```

## build_maxheap(arr,size):

```
index = size/2 – 1
while(index >= 0 )
        maxheapify(arr,index,size)
        Decrement index
```

## maxheapify(arr,index,size)

```
L = 2*index +1
R = 2*index+2
if(arr[index] < arr[L] AND L < size)
  large = L
else
  large = index
if(arr[large]< arr[R] AND R < size)
  large = R
if(index != large)
  swap(arr[large] ,arr[index])
  maxheapify(arr,large,size)
```

# Heap Sort

## Advantages

- The Heap sort algorithm is widely used because of its efficiency.
- It is an in-place sorting technique
- The Heap sort algorithm exhibits consistent performance

# Heap Sort

## Advantages

- The Heap sort algorithm is widely used because of its efficiency.
- It is an in-place sorting technique
- The Heap sort algorithm exhibits consistent performance

## Disadvantages

- It is not a stable sort

## Time Complexity

- O(nlogn)

# Code -Heap Sort