힙 정렬-C

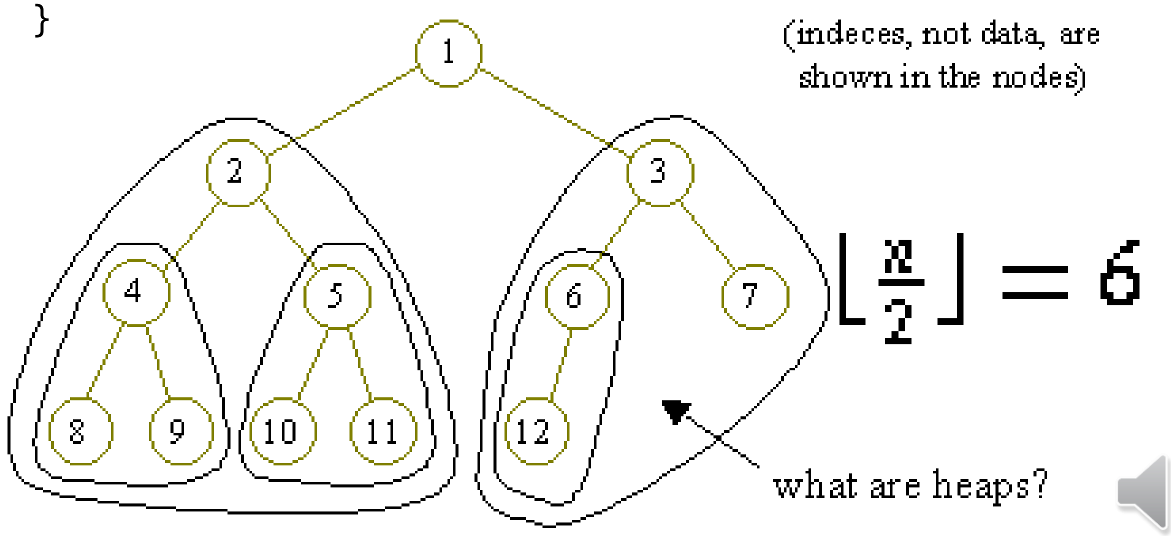
```
// To heapify a subtree rooted with node i which is
// an index in arr[]. N is size of heap
void heapify(int arr[], int n, int i)
{
    int smallest = i; // Initialize smallest as root
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2

    // If left child is smaller than root
    if (l < n && arr[l] < arr[smallest])
         smallest = l;
    // If right child is smaller than largest so far
    if (r < n && arr[r] < arr[smallest])
         smallest = r;

    // If smallest is not root
    if (smallest != i) {
        swap(arr[i], arr[smallest]);
        // Recursively heapify the affected sub-tree
        heapify(arr, n, smallest);
    }
}
```

```
// build a Min-Heap from the given array
void buildHeap(int arr[], int n)
{
    // Index of last non-leaf node
    int startIdx = (n / 2) - 1;

    // Perform reverse level order traversal
    // from last non-leaf node and heapify
    // each node
    for (int i = startIdx; i >= 0; i--) {
        heapify(arr, n, i);
    }
}
```


(indeces, not data, are shown in the nodes)
1
2
3
4
5
6
7
8
9
10
11
12
$\lfloor \frac{n}{2} \rfloor = 6$
what are heaps?

힙 정렬-C

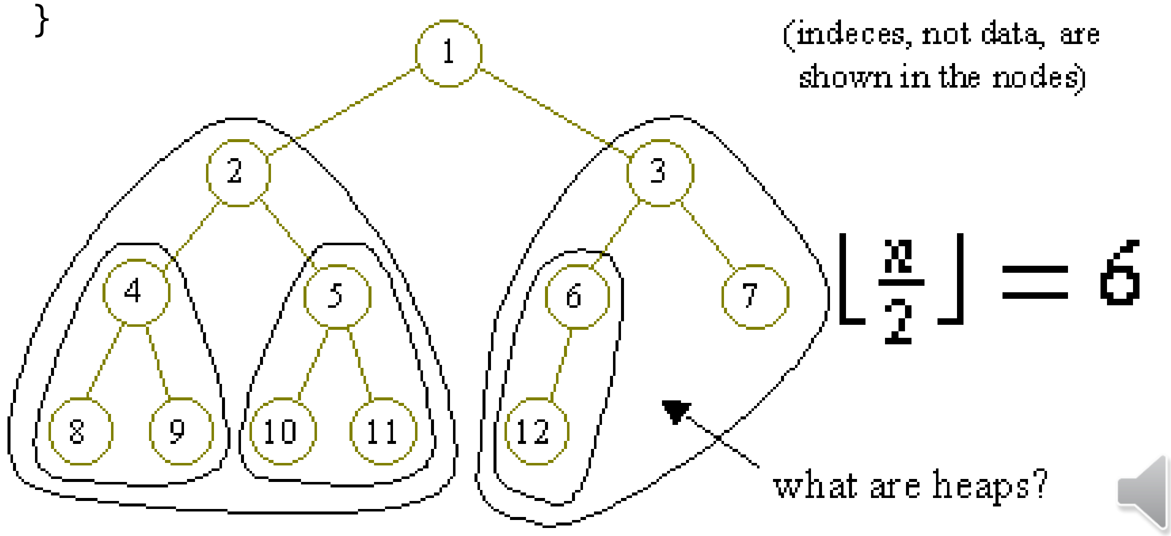
```
// To heapify a subtree rooted with node i which is
// an index in arr[]. N is size of heap
void heapify(int arr[], int n, int i)
{
    int smallest = i; // Initialize smallest as root
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2

    // If left child is smaller than root
    if (l < n && arr[l] < arr[smallest])
         smallest = l;
    // If right child is smaller than largest so far
    if (r < n && arr[r] < arr[smallest])
         smallest = r;

    // If smallest is not root
    if (smallest != i) {
        swap(arr[i], arr[smallest]);
        // Recursively heapify the affected sub-tree
        heapify(arr, n, smallest);
    }
}
```

```
// build a Min-Heap from the given array
void buildHeap(int arr[], int n)
{
    // Index of last non-leaf node
    int startIdx = (n / 2) - 1;

    // Perform reverse level order traversal
    // from last non-leaf node and heapify
    // each node
    for (int i = startIdx; i >= 0; i--) {
        heapify(arr, n, i);
    }
}
```


(indeces, not data, are shown in the nodes)
1
2
3
4
5
6
7
8
9
10
11
12
$\lfloor \frac{n}{2} \rfloor = 6$
what are heaps?

힙 정렬-C

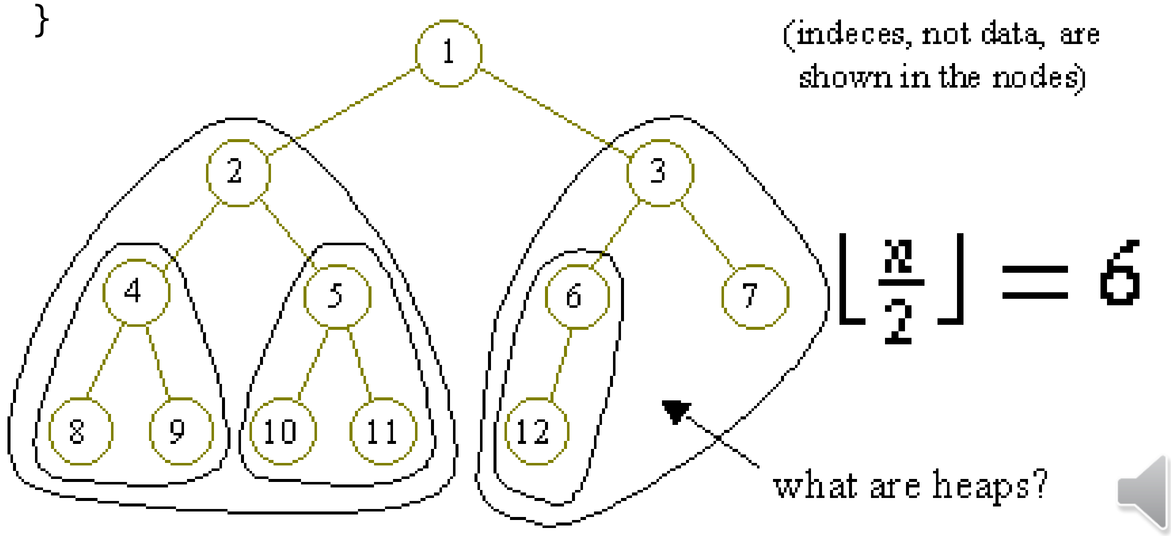
```
// To heapify a subtree rooted with node i which is
// an index in arr[]. N is size of heap
void heapify(int arr[], int n, int i)
{
    int smallest = i; // Initialize smallest as root
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2

    // If left child is smaller than root
    if (l < n && arr[l] < arr[smallest])
         smallest = l;
    // If right child is smaller than largest so far
    if (r < n && arr[r] < arr[smallest])
         smallest = r;

    // If smallest is not root
    if (smallest != i) {
        swap(arr[i], arr[smallest]);
        // Recursively heapify the affected sub-tree
        heapify(arr, n, smallest);
    }
}
```

```
// build a Min-Heap from the given array
void buildHeap(int arr[], int n)
{
    // Index of last non-leaf node
    int startIdx = (n / 2) - 1;

    // Perform reverse level order traversal
    // from last non-leaf node and heapify
    // each node
    for (int i = startIdx; i >= 0; i--) {
        heapify(arr, n, i);
    }
}
```


(indeces, not data, are shown in the nodes)
1
2
3
4
5
6
7
8
9
10
11
12
$\lfloor \frac{n}{2} \rfloor = 6$
what are heaps?

힙 정렬-파이썬

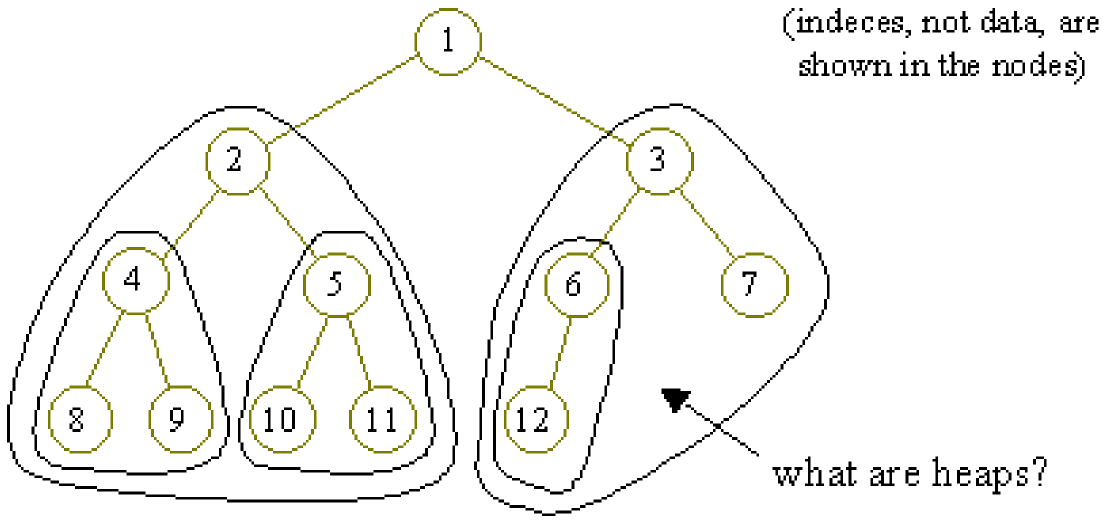
n =12 $\lfloor \frac{n}{2} \rfloor = 6$

```
def heapify(arr, n, i):
    smallest = i  # 가장작은 값을 root로 지정 (초기화)
    l = 2 * I + 1  # left = 2*I + 1  왼쪽 자식
    r = 2 * I + 2  # right = 2*I + 2 오른쪽 자식

    # 왼쪽과 오른쪽에서 작은값을  선택
    if l < n and arr[i] > arr[l]:
        smallest = l
    if r < n and arr[smallest] > arr[r]:
        smallest = r

   # 자식이 작은 값인 경우 부모와 교환
   if smallest != i:
        arr[i], arr[smallest] = arr[smallest], arr[i]  # swap

        #  재귀호출로 힙 구성 시작
        heapify(arr, n, smallest)
```



힙 정렬-파이썬

```
# Function to build a Max-Heap from the given array
def buildHeap(arr, n):
    # Index of last non-leaf node
    startIdx = int((n / 2)) - 1;

    # Perform reverse level order traversal
    # from last non-leaf node and heapify
    # each node
    for i in range(startIdx, -1, -1):
        heapify(arr, n, i);
```

```
# A utility function to print the array
# representation of Heap
def printHeap(arr, n):
      print("Array representation of Heap is:");
      for i in range(n):
          print(arr[i], end = " ");
       print();

# Driver Code
if __name__ == '__main__':
     arr = [ 1, 3, 5, 4, 6, 13,  10, 9, 8, 15, 17 ];
     n = len(arr);
     buildHeap(arr, n);
     printHeap(arr, n);
```