[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.09.22 화요일)

서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

- 본 강의에서 제작하여 제공하는 PDF 파일, 동영상, 그리고 예제 코드 등의 강의 자료의 저작권은 특별히 명기되어 있지 않은 한 서강대학교에 있습니다.
- 본인의 학습 목적 외에 공개된 장소에 올리거나 타인에게 배포하는 등의 행위를 금합니다. 협조 부탁합니다.

[주제 2]
Heap-based Priority Queues and Heap Sort
(Review)

Priority Queue 1: Max(Min) Heap

[Horowitz 5.6.2]
[Neapolitan 7.6]

- **Problem**
  - The following operations must be performed as mixed in data processing:
    - Store a record with a key in an arbitrary order.
    - Fetch the record with the current largest key.

- A solution: **Design a data structure** that offers an efficient implementation of the following operations:

  - **Insert an element with an arbitrary key.**

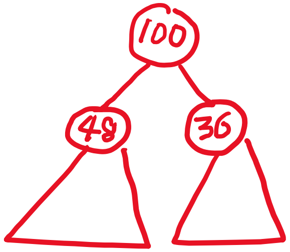
  - **Delete an element with the largest key.**

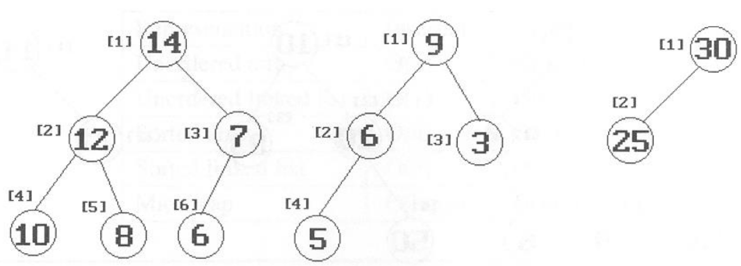
Max(Min) Heap: Definitions

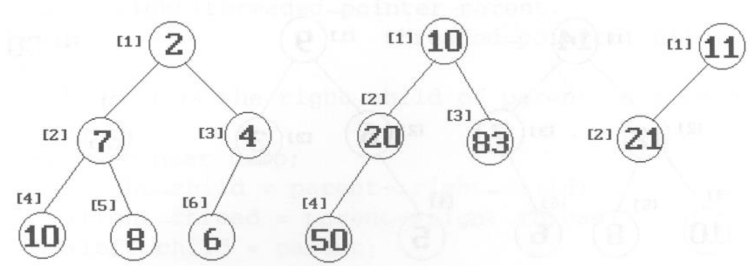
[Horowitz 5.6.2]
[Neapolitan 7.6]

- **Definition 1**
  - A max(min) heap is a complete binary tree where the key value in each internal node is no smaller(larger) than the key values in its children.

- **Definition 2**
  - A binary tree has the **max(min) heap property** if and only if
    - ① The number of nodes of the tree is either 0 or 1, *or*
    - ② For the tree that has at least two nodes, the key in the root is no smaller(larger) than that in each child and the subtree rooted at the child has the **max(min) heap property**.
  - A max(min) heap is a complete binary tree that has the max(min) heap property.

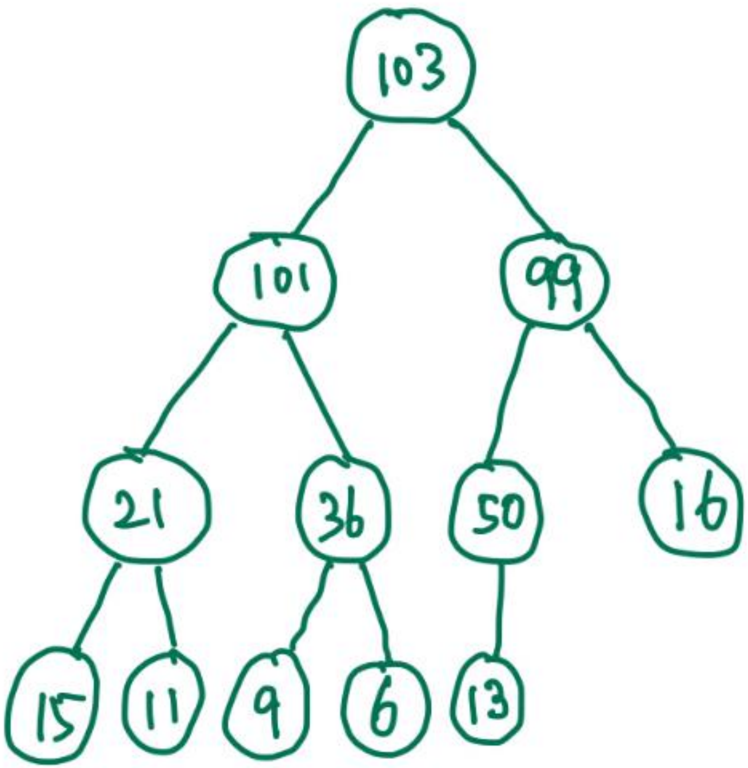




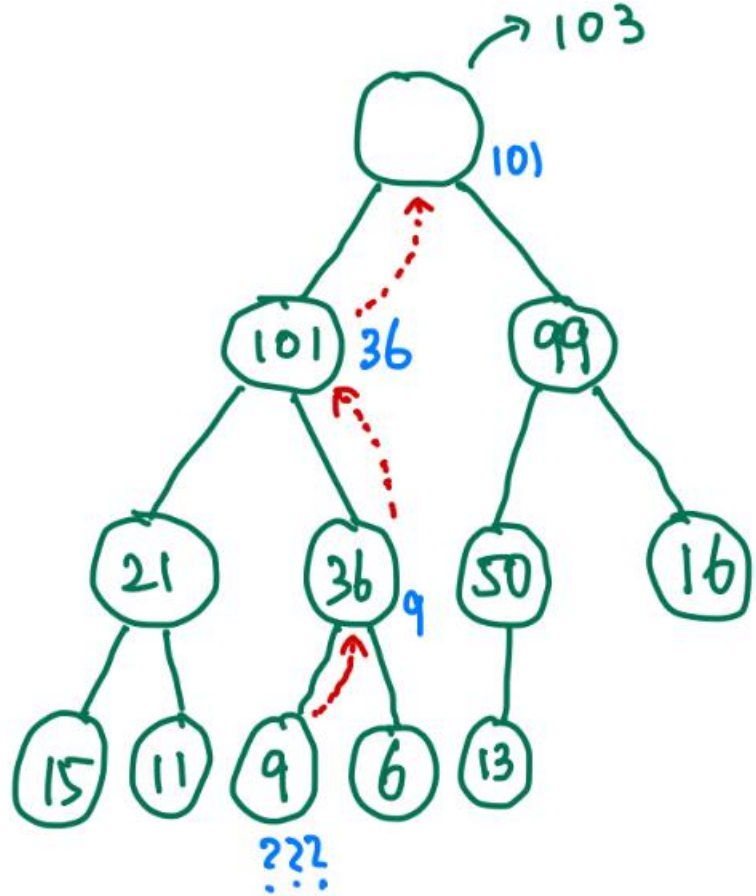


Brainstorming on Max Heap Operations

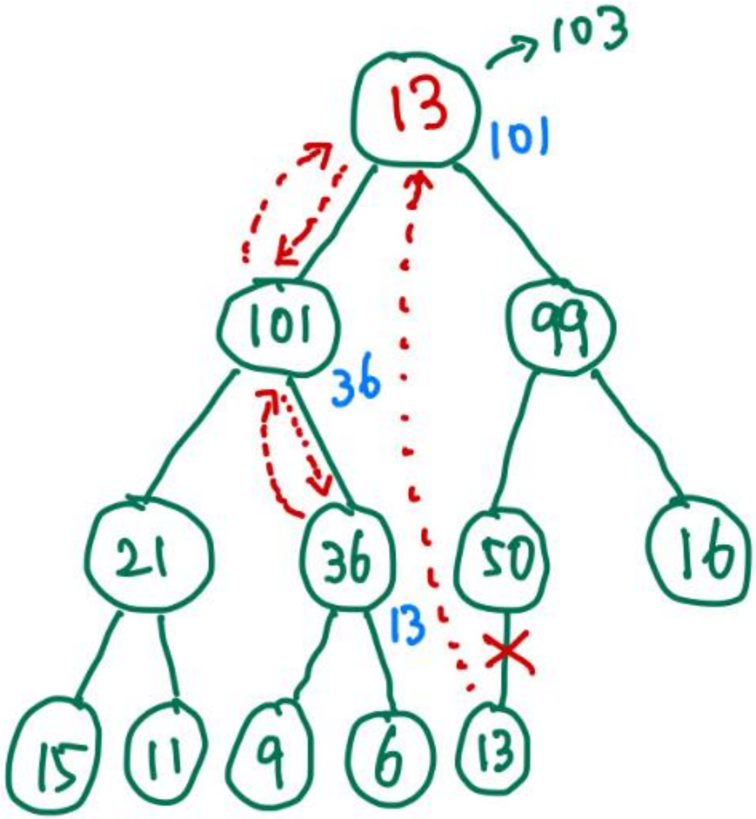
Max Heap Example



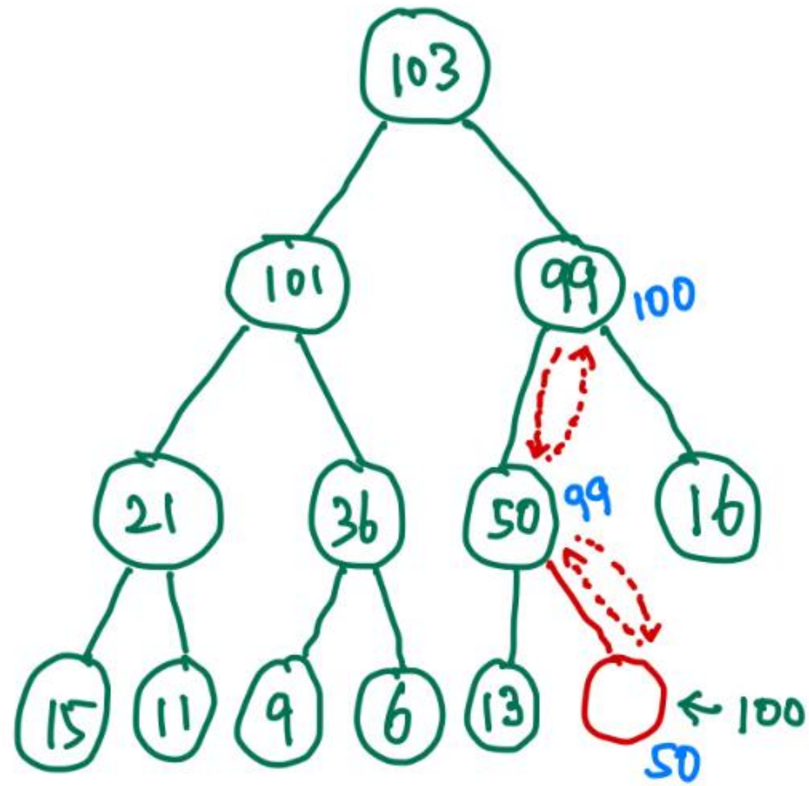
Deletion Example 1



Deletion Example 2

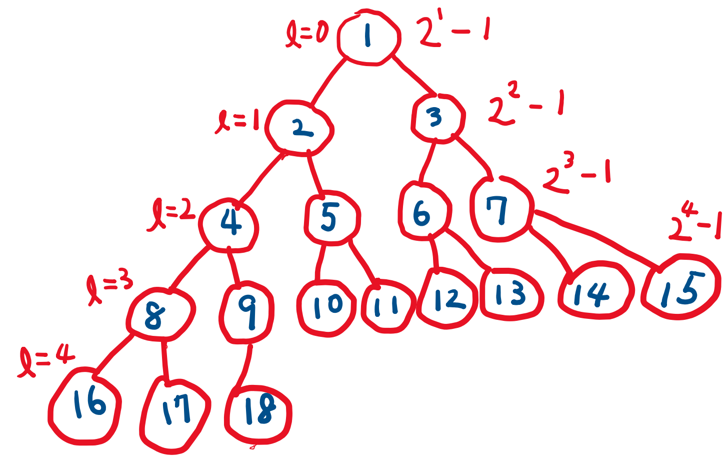


Insertion Example

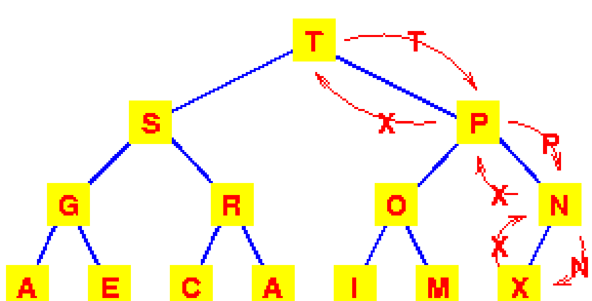


[Horowitz 5.6.2]

```
#define MAX_ELEMENTS 200
#define HEAP_FULL(n) (n == MAX_ELEMENTS-1)
#define HEAP_EMPTY(n) (!n)
typedef struct {
  int key;
  /* other fields */
} element;
element heap[MAX_ELEMENTS];
int n = 0;
```



$2^c \le n < 2^{c+1} \Rightarrow c \le \log_2 n < c+1$

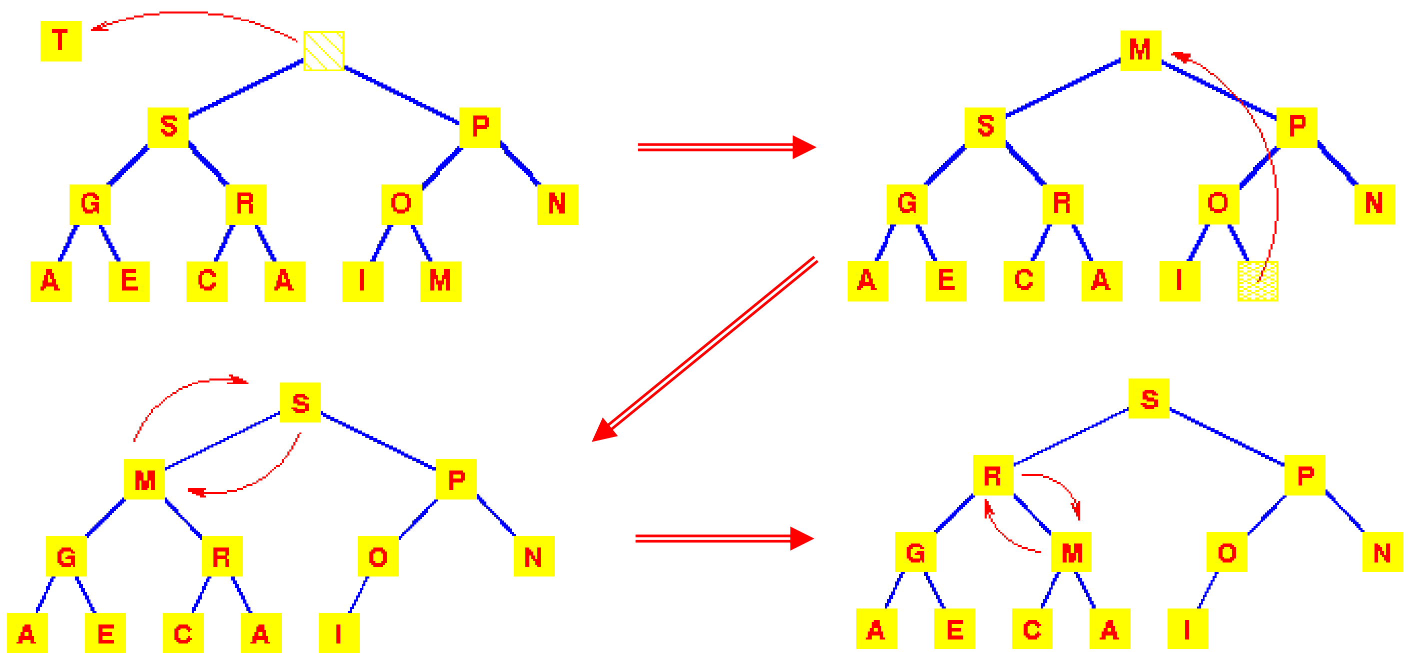



```
void insert_max_heap(element item, int *n)
{
/*insert item into a max heap of current size *n */
  int i;
  if (HEAP_FULL(*n)){
    fprintf(stderr, "The heap is full. \n");
    exit(1);
  }
  i = ++(*n);
  while ((i != 1) && (item.key > heap[i/2].key)) {
    heap[i] = heap[i/2];
    i /= 2;
  }
  heap[i] = item;
}
```

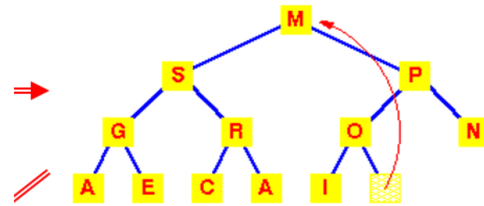
$c \approx \log_2 n$

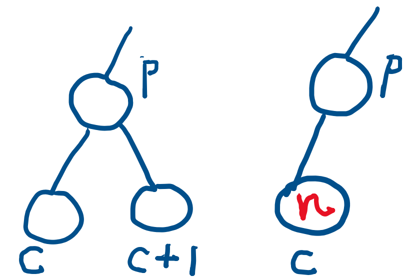
$O(\log n)$

Deletion from a Max Heap



```
element delete_max_heap(int *n)
{
/* delete element with the highest key from the heap */
  int parent, child;
  element item, temp;
  if (HEAP_EMPTY(*n)) {
    fprintf(stderr, "The heap is empty\n");
    exit(1);
  }
  /* save value of the element with the highest key */
  item = heap[1];
  /* use last element in heap to adjust heap */
  temp = heap[(*n)--];
  parent = 1;
  child = 2;
  while (child <= *n) {
    /* find the larger child of the current parent */
    if   (child  <   *n)   &&   (heap[child].key   <
   heap[child+1].key)
      child++;
    if (temp.key >= heap[child].key) break;
    /* move to the next lower level */
    heap[parent] = heap[child];
    parent = child;
    child *= 2;
  }
  heap[parent] = temp;
  return item;
}
```





$O(\log n)$

Another Heap Implementation (Min Heap)

[Sedgewick 9.3]

```
void PQinit(int);
int PQempty();
void PQinsert(int);
int PQdelmin();

static int *pq;
static int N;

void PQinit(int maxN) {
  pq = malloc(maxN*sizeof(int));
  N = 0;
}

int PQempty() { return N == 0; }

void PQinsert(int v) {
  pq[++N] = v;
  fixUp(pq, N);
}

Item PQdelmin() {
  exch(pq[1], pq[N]);
  fixDown(pq, 1, N-1);
  return pq[N--];
}
```

```
fixUp(int a[], int k) {
  while (k > 1 && a[k/2] > a[k]) {
    exch(a[k], a[k/2]);
    k = k/2;
  }
}

fixDown(int a[], int k, int N) {
  int j;

  while (2*k <= N) {
    j = 2*k;
    if (j < N && a[j] > a[j+1])
      j++;
    if (a[k] <= a[j]) break;
    exch(a[k], a[j]);
    k = j;
  }
}
```

- What will be the worst-case time complexity of each operation?

Comparisons of Priority Queue Implementations

Representation	Insertion	Deletion
Unordered array	$O(1)$	$O(n)$
Unordered linked list	$O(1)$	$O(n)$
Sorted array	$O(n)$	$O(1)$
Sorted linked list	$O(n)$	$O(1)$
Max heap	$O(\log n)$	$O(\log n)$

Heap Sort

[Horowitz 7.7]
[Neapolitan 7.6]

1	2	3	4	5	6	7	8	9	10	
26	5	77	1	61	11	59	15	48	19	← unordered
77	61	59	48	19	11	26	15	1	5	← max heap
1	5	11	15	19	26	48	59	61	77	← ordered

① ②

- **Method**
  - ① Convert an input array of *n* unordered items into a max heap.
  - ② Extract the items from the heap one at a time to build an ordered array.

주어진 정수들을 비감소 순서(non-decreasing order)대로 정렬하라.

```
void heapsort(element list[], int n)
/* perform a heapsort on the array */
{
   int i,j;
   element temp;

   for (i = n/2; i > 0; i--)
      adjust(list,i,n);
   for (i = n-1; i > 0; i--) {
      SWAP(list[1],list[i+1],temp);
      adjust(list,1,i);
   }
}
```

① O(n) ② O(nlogn) ⇒ O(nlogn)

```
typedef struct{
  int key;
  /* other fields */
} element;
Element list[MAX_SIZE];
```

1. Make a (max) heap.

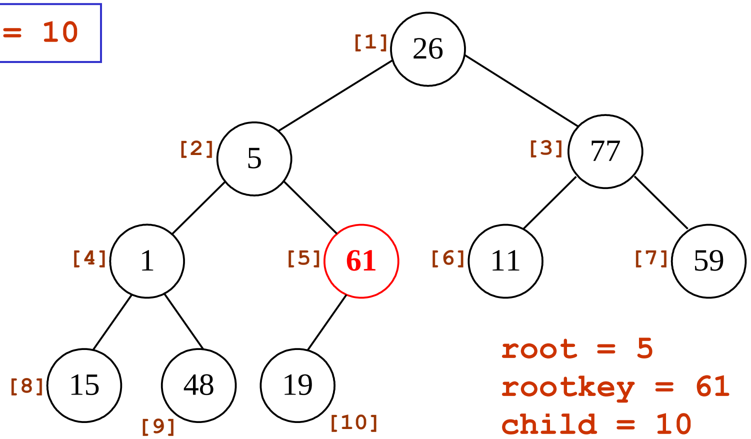
2. Extract items one by one.

1. Make a Max Heap.

1 2 3 4 5 6 7 8 9 10
① 26 5 77 1 61 11 59 15 48 19 ← unordered
77 61 59 48 19 11 26 15 1 5 ← max heap

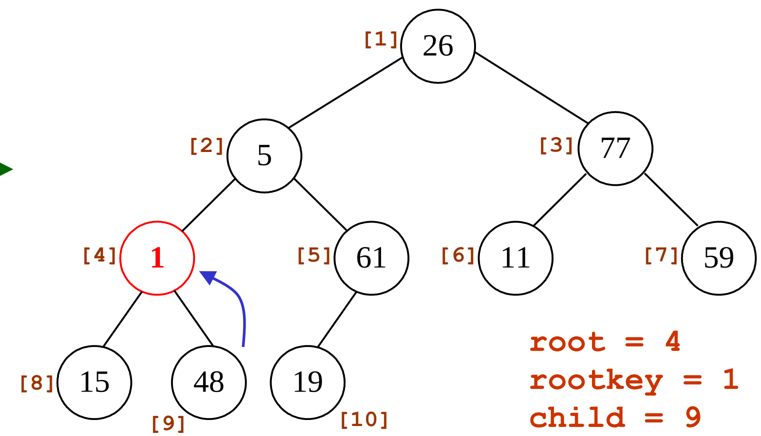
`adjust(list, 5, 10);`

n = 10



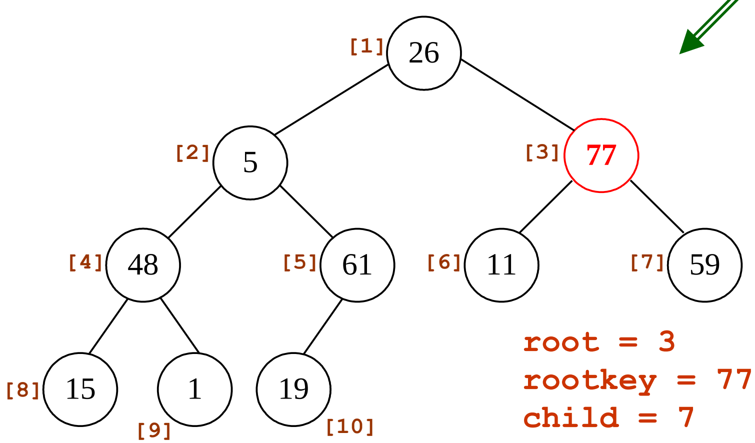

```
root = 5
rootkey = 61
child = 10
```

`adjust(list, 4, 10);`



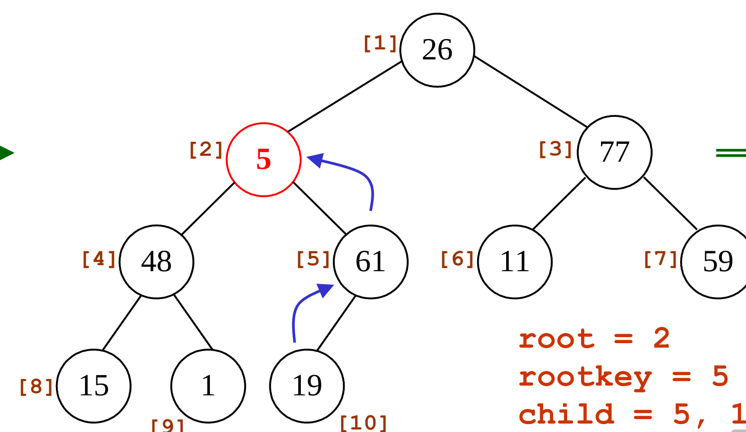

```
root = 4
rootkey = 1
child = 9
```

`adjust(list, 3, 10);`



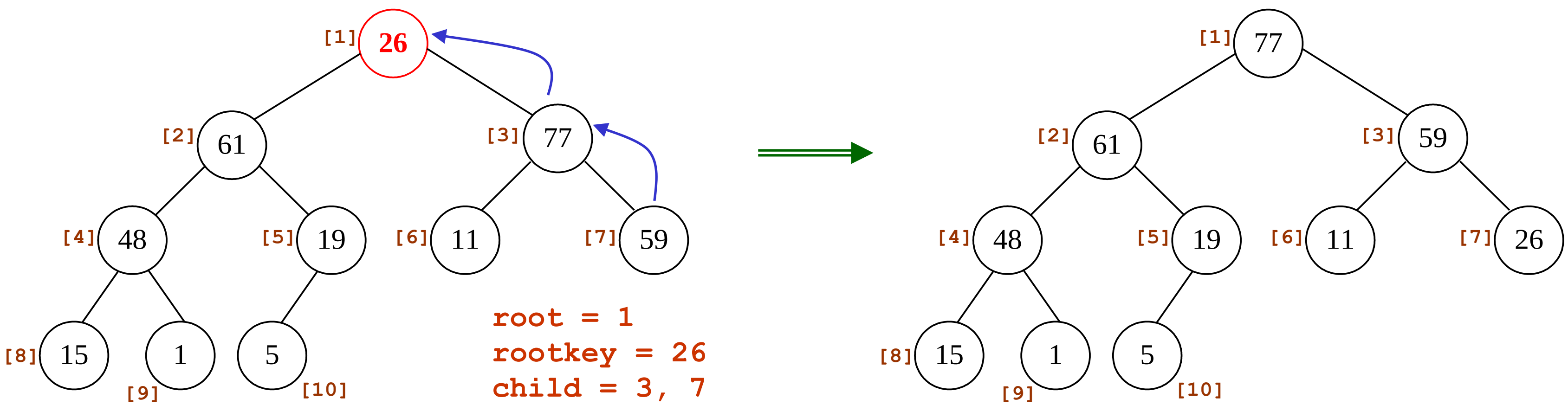

```
root = 3
rootkey = 77
child = 7
```

`adjust(list, 2, 10);`




```
root = 2
rootkey = 5
child = 5, 10
```

```
adjust(list, 1, 10);
```




```
root = 1
rootkey = 26
child = 3, 7
```

	1	2	3	4	5	6	7	8	9	10	
	26	5	77	1	61	11	59	15	48	19	← unordered
①	77	61	59	48	19	11	26	15	1	5	← max heap
②	1	5	11	15	19	26	48	59	61	77	← ordered