Heap Sort

강지훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터공학과

“How Fast Can We Sort ?”

HOW FAST CAN WE SORT?

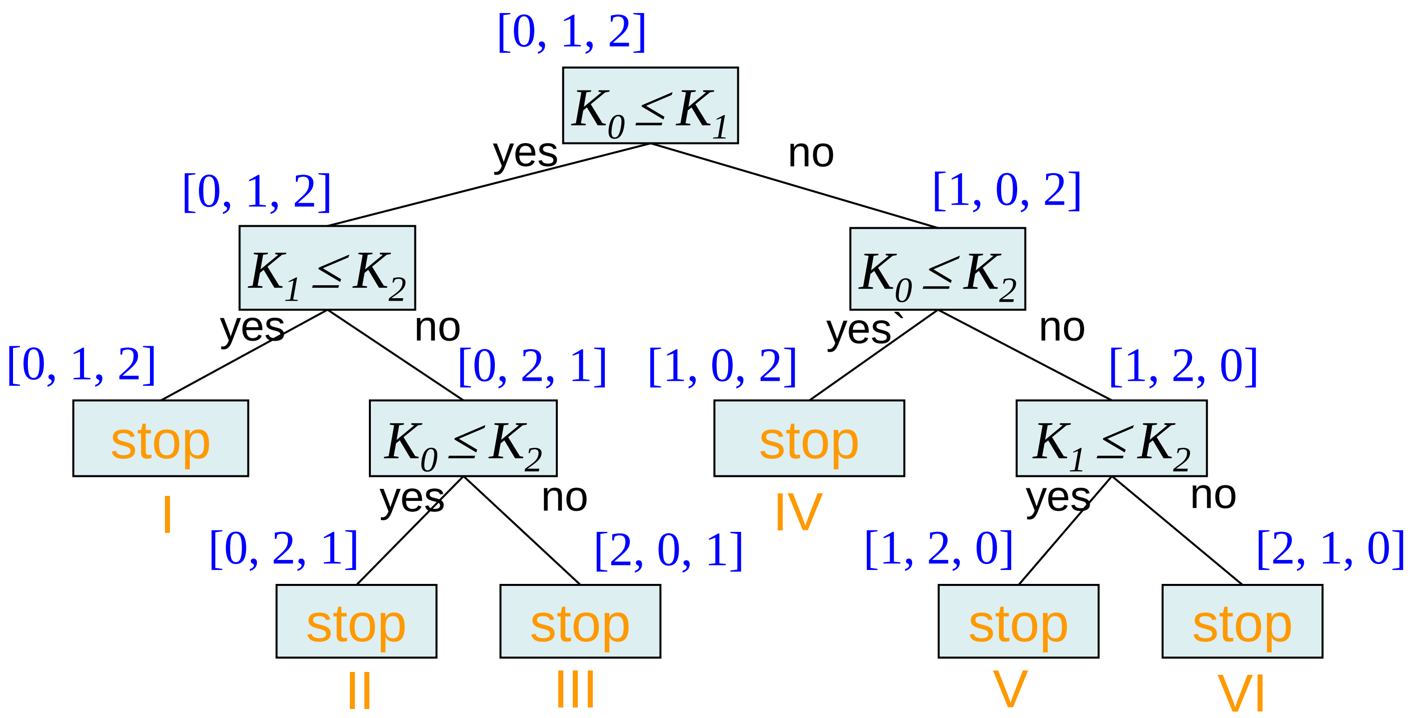
- *"What is the best computing time for sorting that we can hope for ?"*
  - We assume that the only operations permitted are comparison and interchange.
  - The conclusion is: $\Omega(n \log n)$.

Decision Tree

- Vertex and Branch
  - Vertex : key comparison
  - Branch : the result of Key comparison

- Each path in the tree represents a possible sequence of computations that an algorithm could produce.

- Decision Tree for Insertion Sort with (R_0, R_1, R_2) .



- Example: all the permutations of 7, 9, 10.

Leaf	Permutations	Sample key values that give the permutation.
I	0 1 2	(7, 9, 10)
II	0 2 1	(7, 10, 9)
III	2 0 1	(10, 7, 9)
IV	1 0 2	(9, 7, 10)
V	1 2 0	(9, 10, 7)
VI	2 1 0	(10, 9, 7)

- Theorem 7.1
  Any decision tree that sorts n distinct elements has a height of at least

  $$log(n!) + 1$$

(Proof)

There are $n!$ leaves in the decision tree.

The decision tree is a binary tree.

A binary tree can have at most 2^{k-1} leaves.

Height $k \Rightarrow$ at most 2^{k-1} leaves.

2^{k-1} leaves $\Rightarrow$ height of at least k.

n! leaves $\Rightarrow$ height of at least $\log_2(n!) + 1$.

Therefore , the height of the decision tree is at least $\log(n!) + 1$.

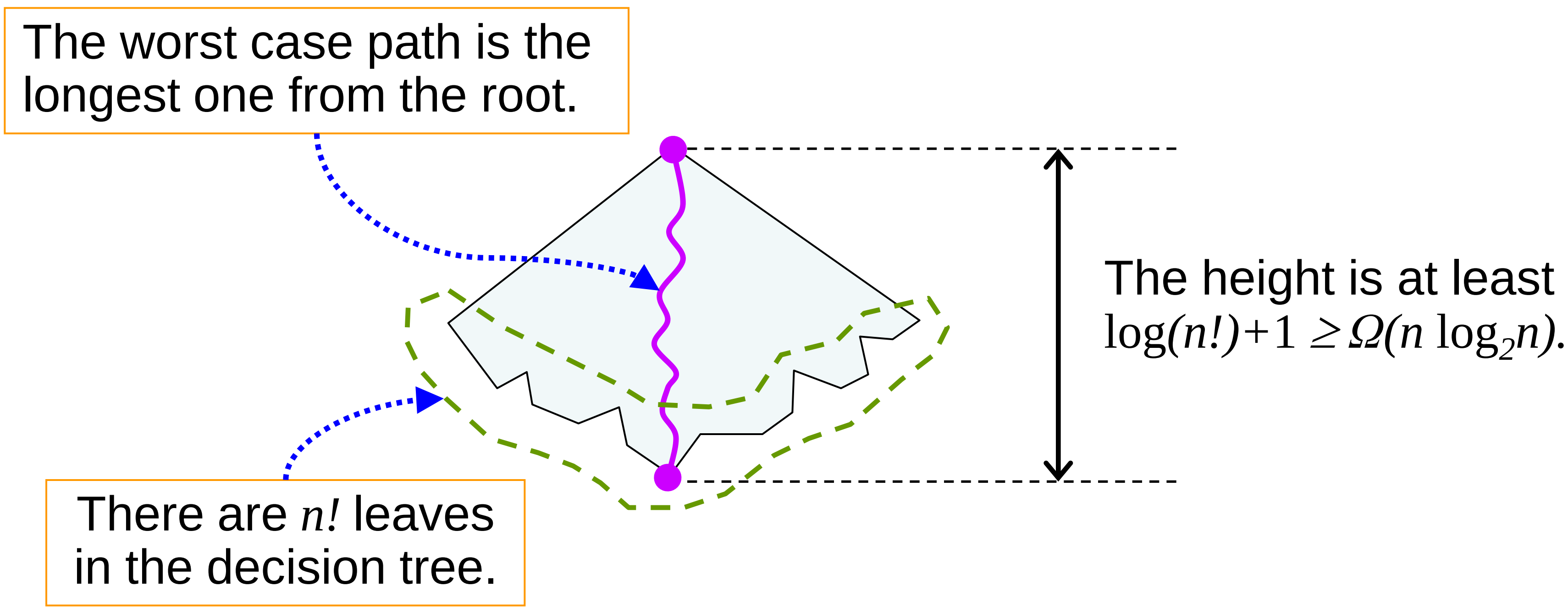
Corollary

Any algorithm which sorts by comparisons only must have a worst case computing time of $\Omega(n \log_2 n)$.

(Proof)

$n! = n(n-1)(n-2) \cdots 3 \cdot 2 \cdot 1 \geq (n/2)^{n/2}$

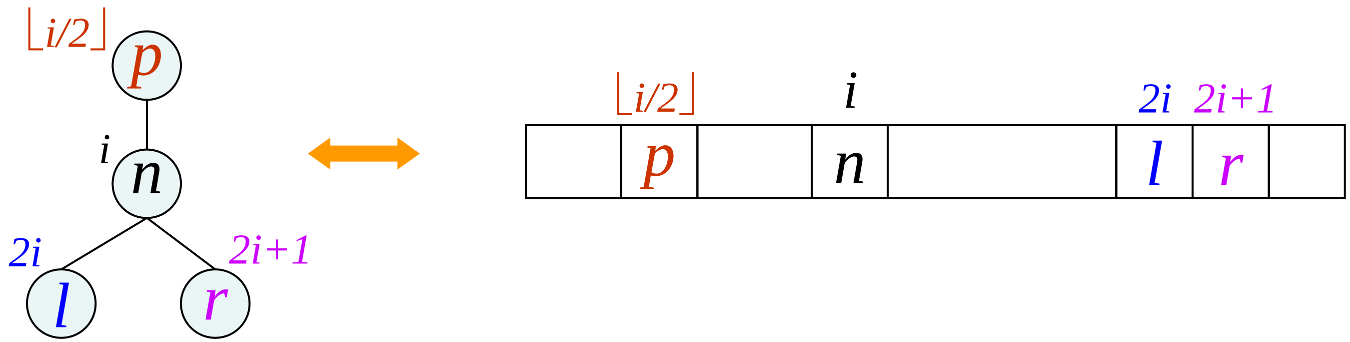
So, $\log_2(n!) \geq (n/2)\log_2(n/2) = \Omega(n \log_2 n)$.

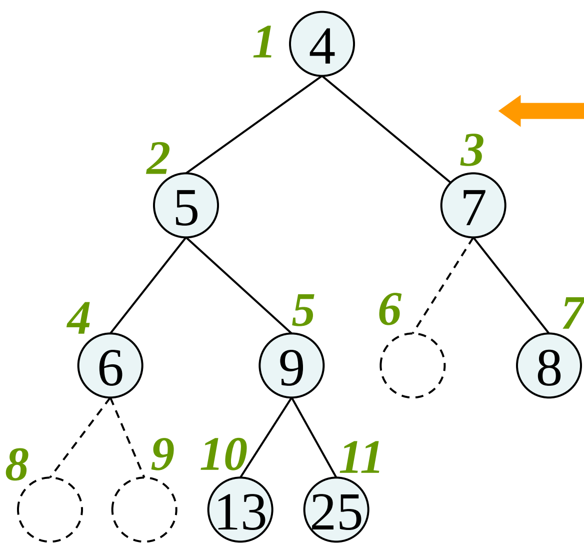


Heap Sort

Heap Sort

- Sequential Representation of Binary trees.
  - Especially, suitable for complete binary trees.





[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
-	4	5	7	6	9	-	8	-	-	13	25

Current Node: $i = 5$ $[i] = 9$

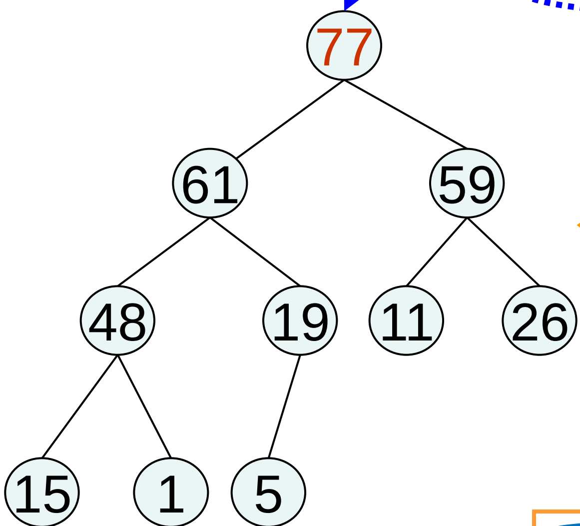
Left child : $2i = 10$ $[2i] = 13$

Right child: $(2i+1) = 11$ $[2i+1] = 25$

Parent: $\lfloor i/2 \rfloor = 2$ $[\lfloor i/2 \rfloor] = 5$

- **Heap**: A complete binary tree such that the value of each node is at least as large as the value of its children nodes.

The root of a heap has the largest value in the tree.



77	61	59	48	19	11	26	15	1	5
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]

The nodes in any path from the root to a leaf are in non-increasing order.

- $77 \rightarrow 61 \rightarrow 48 \rightarrow 15$
- $77 \rightarrow 61 \rightarrow 48 \rightarrow 1$
- $77 \rightarrow 61 \rightarrow 19 \rightarrow 5$
- $77 \rightarrow 59 \rightarrow 11$
- $77 \rightarrow 59 \rightarrow 26$

Basic idea of Heap Sort

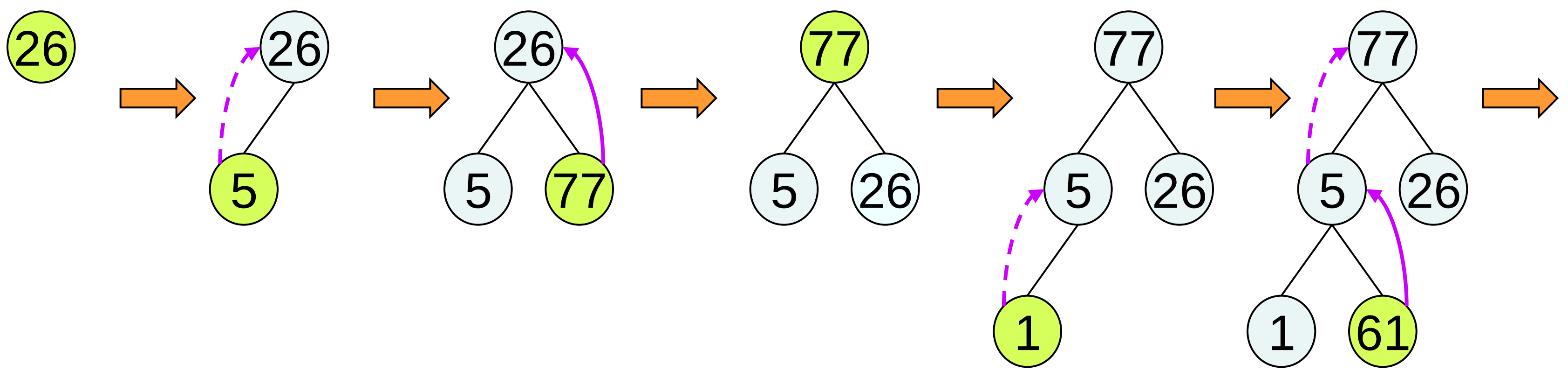
1. Make a heap from input.
2. Repeat the next step until the heap becomes empty.
   - Output and delete the root , and adjust the heap.

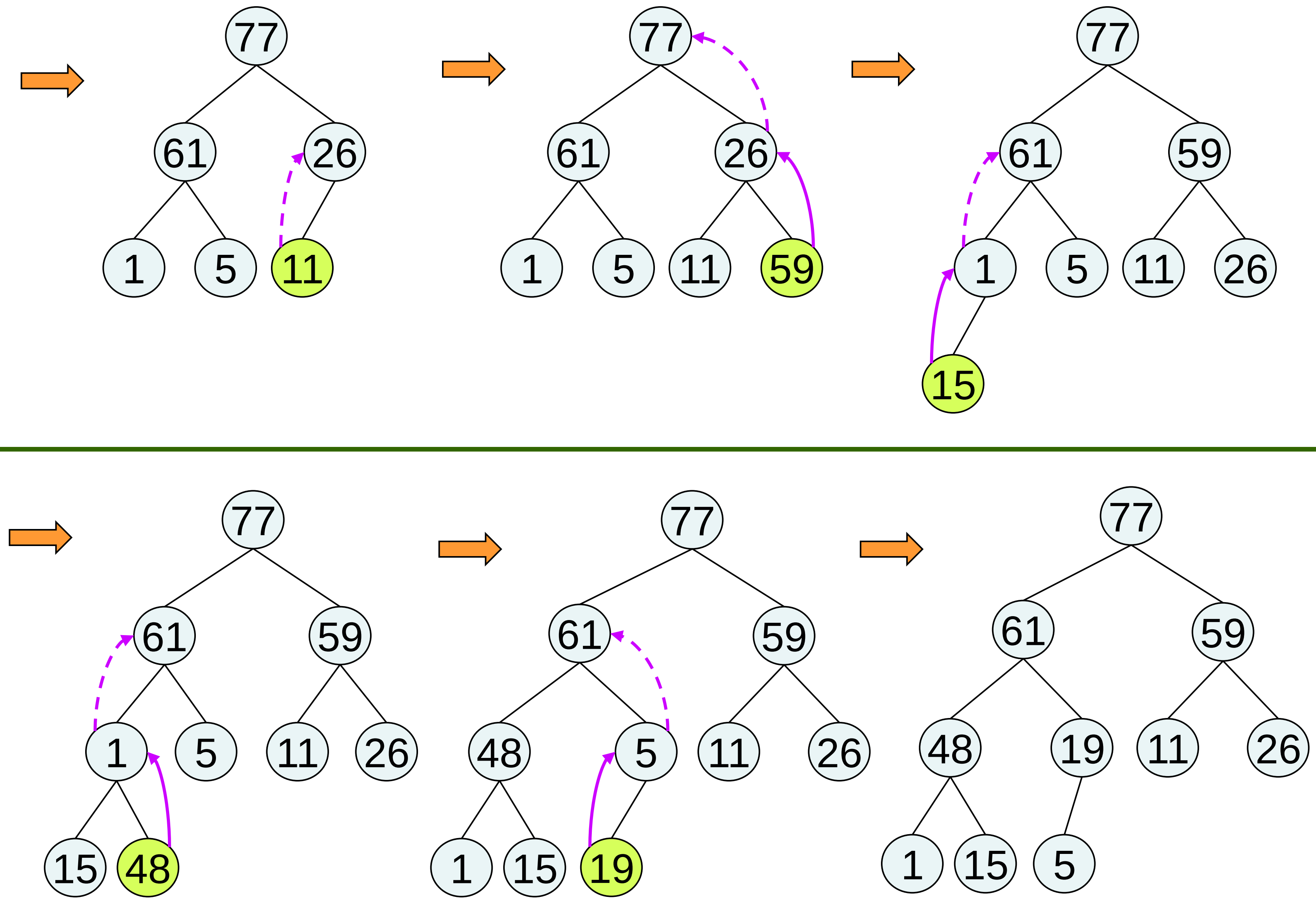
How to make a HEAP

- By “INSERT”
- By “ADJUST”

Making Initial Heaps by "INSERT"

- Example
  (26, 5, 77, 1, 61, 11, 59, 15, 48, 19)

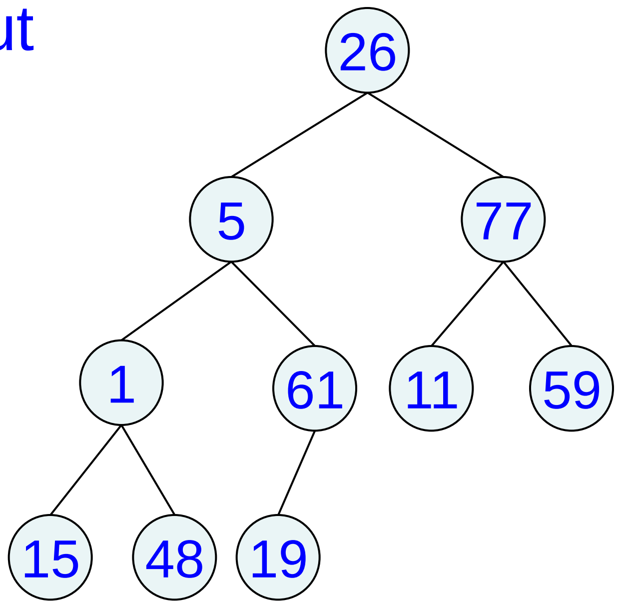


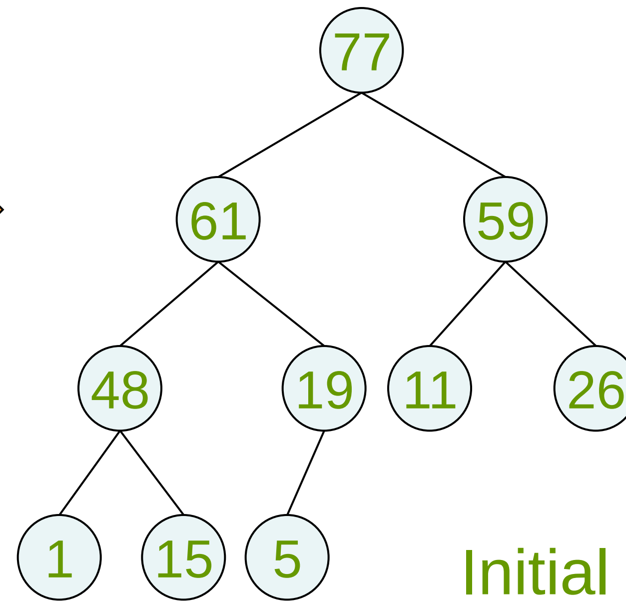

77 61 26 1 5 11
77 61 26 1 5 11 59
77 61 59 1 5 11 26 15
77 61 59 1 5 11 26 15 48
77 61 59 48 5 11 26 1 15 19
77 61 59 48 19 11 26 1 15 5

Representation of input and Initial Heap

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

Input





Initial Heap

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

Worst Case Analysis of INSERT

- It is when the elements are inserted in ascending order.
- Each new element will rise to become the new root.
- Complexity: $O(n \log n)$

Average case of INSERT: $O(n)$

Proof of Worst Case Complexity of INSERT

- The level i in a complete binary tree has at most 2^{i-1} nodes.
- A complete binary tree with n nodes has maximum level $\log_2(n+1)$.

$$\sum_{i=2}^{\lceil \log_2(n+1) \rceil} (i-1)\, 2^{i-1}, \ \text{(Let } k = \lceil \log_2(n+1) \rceil \text{)}.$$

$$= \sum_{i=2}^{k} (i-1)\, 2^{i-1} = \sum_{i=1}^{k-1} i \cdot 2^{i}$$

$$= (k-1)2^{k+1} - k\,2^{k} + 2$$

$$= 2k \cdot 2^{k} - 2 \cdot 2^{k} - k \cdot 2^{k} + 2$$

$$= k \cdot 2^{k} - 2(2^{k} - 1) \le k \cdot 2^{k}$$

$$= \lceil \log_2(n+1) \rceil 2^{\lceil \log_2(n+1) \rceil} = O(n \log n)$$

Lemma : $\sum_{i=1}^{\mathrm{k}} i \cdot x^{i} = \dfrac{k \cdot x^{x+2} - (k+1)x^{k+1} + x}{(x-1)^{2}}, x \neq 1$

(Proof)

Let $S = \sum_{i=1}^{k} i \cdot x^{i}$

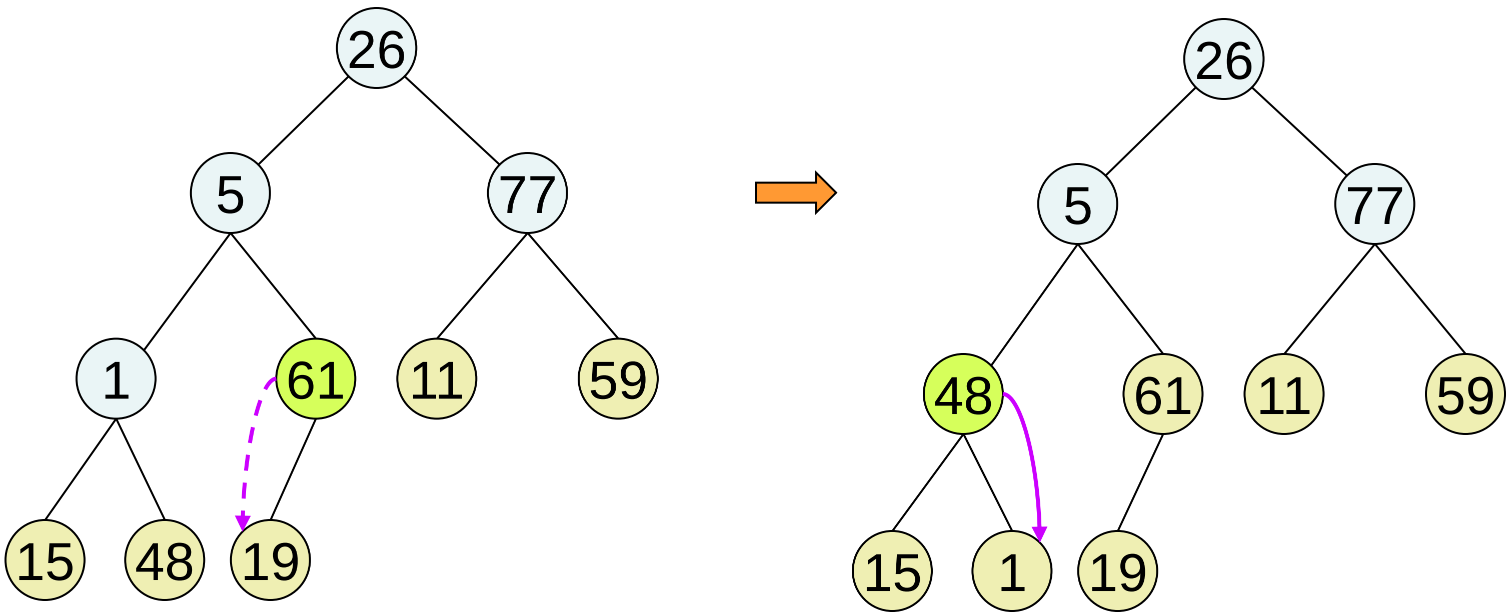
Then, $S - x \cdot S = \sum_{i=1}^{k} i \cdot x^{i} - x\sum_{i=1}^{k} i \cdot x^{i}$

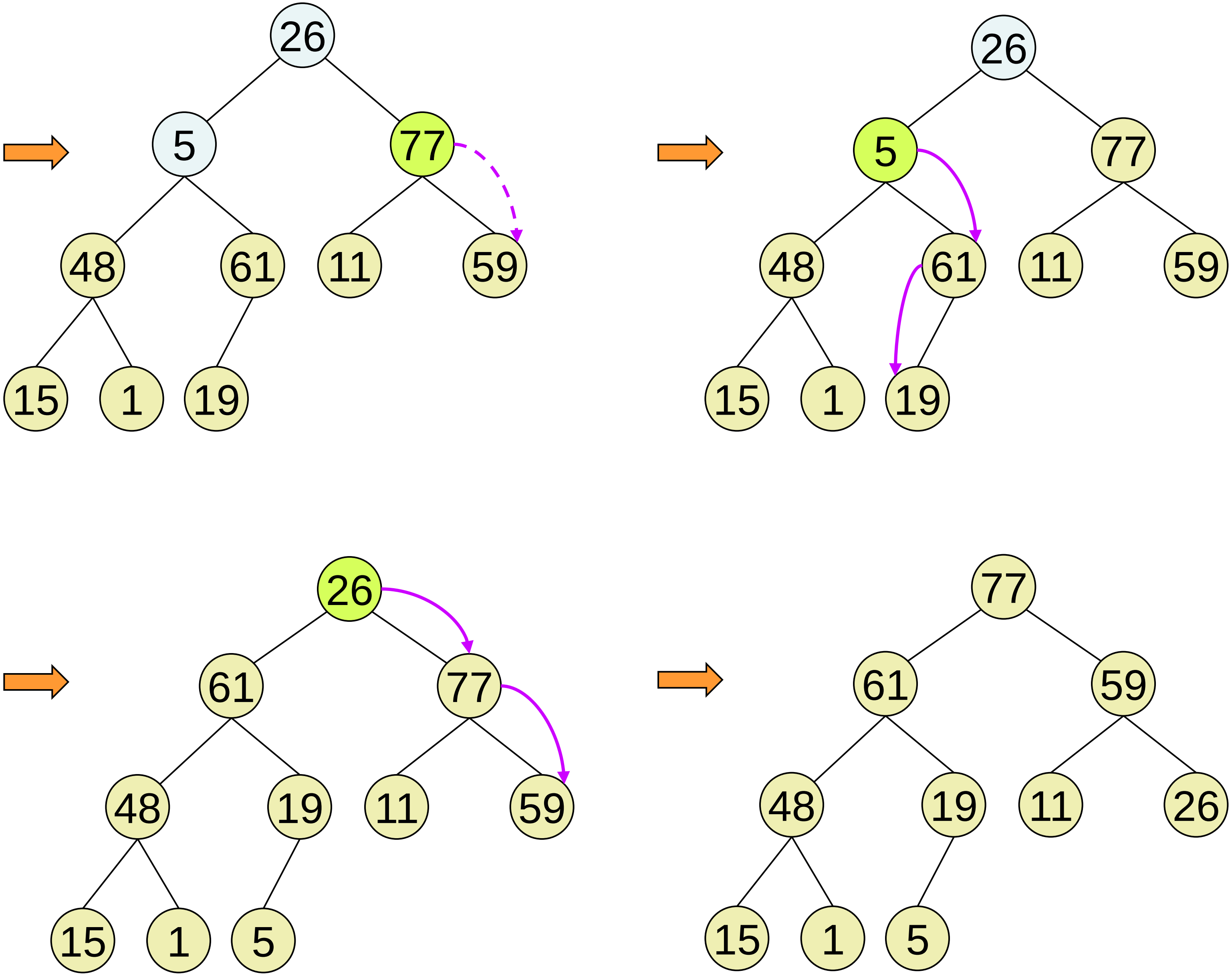
$$
\begin{aligned}
(1-x)S &= \sum_{i=1}^{k} i \cdot x^{i} - \sum_{i=1}^{k} i \cdot x^{i+1} \\
&= \sum_{i=1}^{k} i \cdot x^{i} - \sum_{i=1}^{k} (i+1-1) \cdot x^{i+1} \\
&= \sum_{i=1}^{k} i \cdot x^{i} - \left(\sum_{i=1}^{k} (i+1) \cdot x^{i+1} - \sum_{i=1}^{k} x^{i+1}\right) \\
&= \left(\sum_{i=1}^{k} i \cdot x^{i} - \sum_{i=2}^{k+1} i \cdot x^{i}\right) + \sum_{i=1}^{k} x^{i+1} \\
&= \left(x - (k+1)x^{k+1}\right) + \frac{x^{2}(1-x^{k})}{(1-x)} \\
&= \frac{x(1-x) - (k+1)x^{k+1}(1-x) + x^{2}(1-x^{k})}{(1-x)} \\
&= \frac{x - x^{2} - (k+1)x^{k+1} + (k+1)x^{k+2} + x^{2} - x^{k+2}}{(1-x)} \\
&= \frac{kx^{k+2} - (k+1)x^{k+1} + x}{(1-x)}
\end{aligned}
$$

$$\therefore S = \sum_{i=1}^{k} i \cdot x^{i} = \frac{kx^{k+2} - (k+1)x^{k+1} + x}{(x-1)^{2}}$$

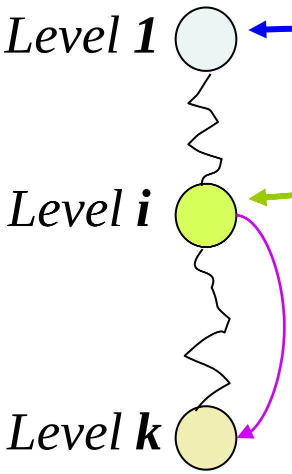
Making Initial Heaps by "ADJUST"

- Initially, each leaf is a heap.




26
5 77
48 61 11 59
15 1 19
26
5 77
48 61 11 59
15 1 19
26
61 77
48 19 11 59
15 1 5
77
61 59
48 19 11 26
15 1 5

Worst Case Analysis of ADJUST



- The root node
- The node to be adjusted.
  (There are maximum 2^{i-1} nodes in the level i.)
- The node at the level i can be moved down $(k - i)$ steps.
- $2^{k-1} \le n < 2^k$, where n is the number of all the nodes.

$$\sum_{i=1}^{k}(k-i)2^{i-1} = \sum_{j=0}^{k-1} j \cdot 2^{k-j-1} = 2^{k-1}\sum_{j=1}^{k-1} j(1/2)^j$$

$$(1)\quad 2^{k-1} \le n$$

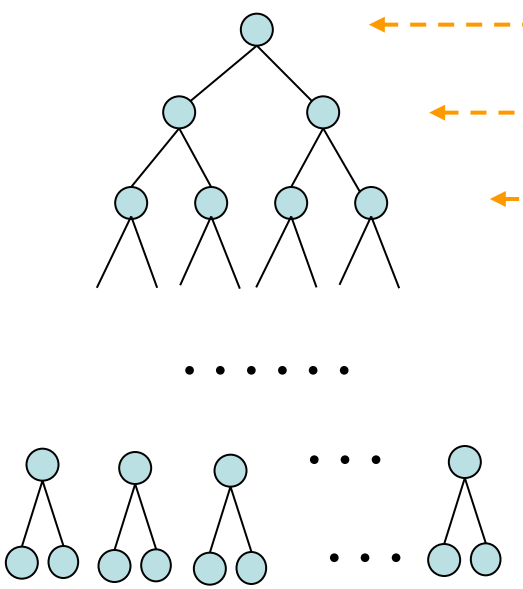
$$(2)\quad \sum_{j=1}^{k-1} j(1/2)^j = \frac{(k-1)(1/2)^{k+1} - k(1/2)^k + 1/2}{(1/2)^2}$$

$$= (k-1)(1/2)^{k-1} - k(1/2)^{k-2} + 2$$

$$= 2 - (k+1)\ (1/2)^{k-1} \le 2$$

$$\therefore \sum_{i=1}^{k}(k-i)2^{i-1} \le n \cdot 2 = O(n)$$

- Relationship between the depth and the number of nodes in the complete binary trees.
  - Let n be the number of nodes in a complete binary tree.
  - Let k be the depth of it (i.e., the maximum level).



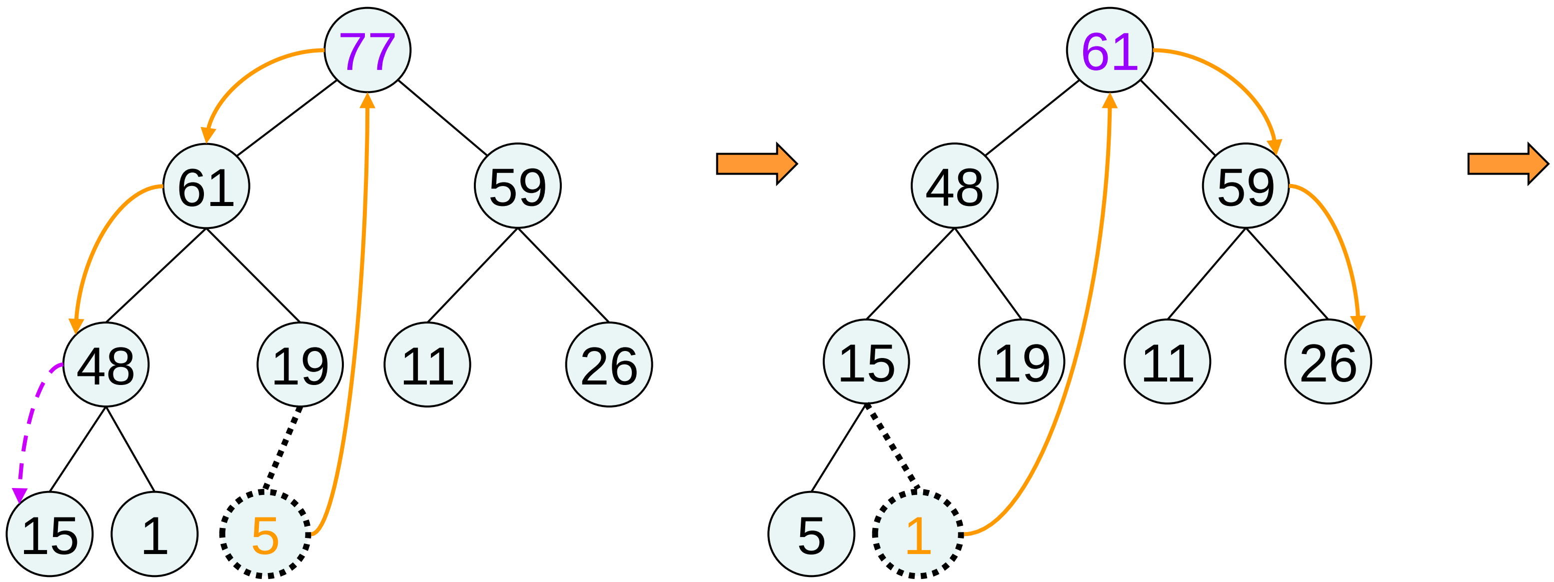
Level	Max Nodes	Total Nodes
1	2^0	2^1-1
2	2^1	2^2-1
3	2^2	2^3-1
⋮		
$k-1$	2^{k-2}	$2^{k-1}-1$
k	2^{k-1}	2^k-1

- Relationship between the depth and the number of nodes in the complete binary trees. (Cont'd)
  - From the complete binary tree,
    - $2^{k-1} \leq n < 2^k$
    - $2^{k-1} \leq n \leq 2^k - 1$
  - By taking log_2 at each side, we get:
    - $k - 1 \leq \log_2 n$, i.e. $k \leq \log_2 n + 1$
    - $n+1 \leq 2^k$, i.e. $\log_2(n+1) \leq k$
  - By combining the two inequalities,
    - $\log_2(n+1) \leq k \leq \log_2 n + 1 < \log_2(n+1)+1$,
      i.e., $\log_2(n + 1) \leq k < \log_2(n+1)+1$
  - Therefore, we finally get the equation:
    - $k = \lceil \log_2(n + 1) \rceil$

Heap sort
after making an initial heap

Heap sort after making an initial heap [1]

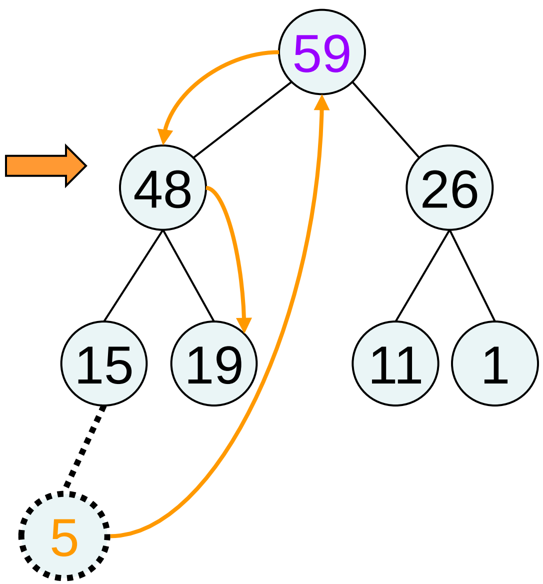
1	2	3	4	5	6	7	8	9	10
77	71	59	48	19	11	26	15	1	5



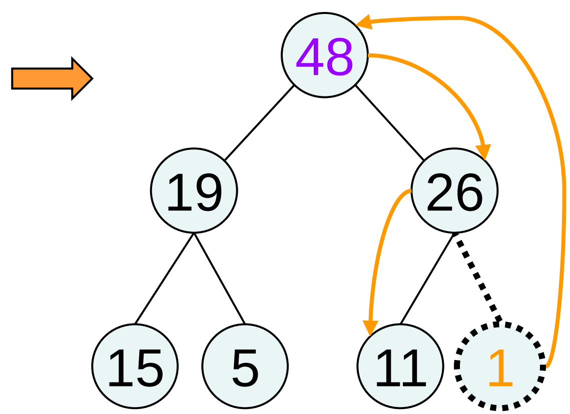
1	2	3	4	5	6	7	8	9	10
61	48	59	15	19	11	26	5	1	77

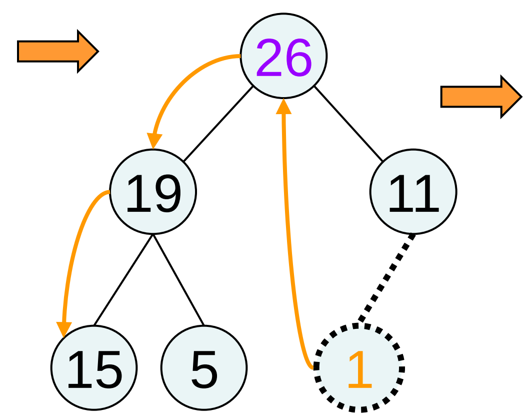
Heap sort after making an initial heap [2]

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



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

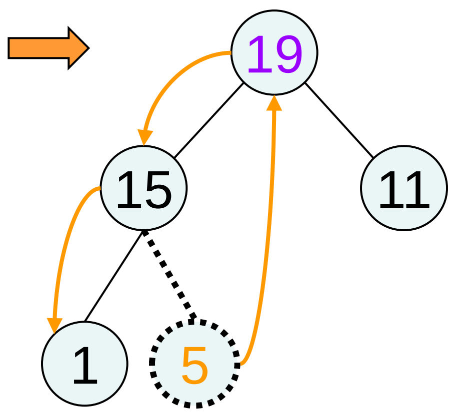




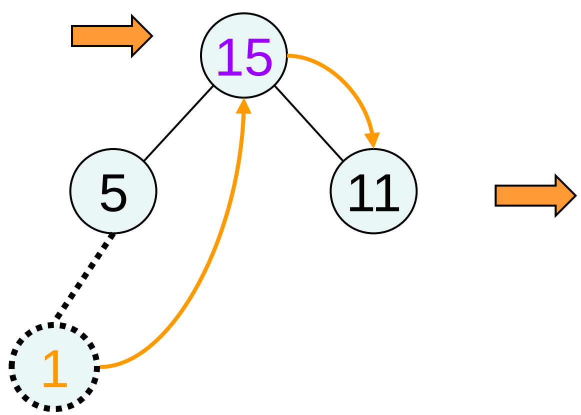
1	2	3	4	5	6	7	8	9	10
26	19	11	15	5	1	48	59	61	77

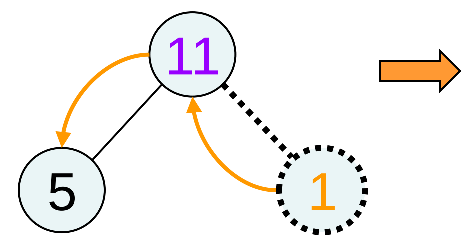
Heap sort after making an initial heap [3]

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



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

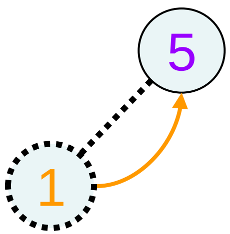




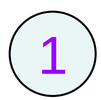
1	2	3	4	5	6	7	8	9	10
11	5	1	15	19	26	48	59	61	77

Heap sort after making an initial heap [4]

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



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



At this time, the tree has only one node. So, we stop the sorting. And the array has now been sorted.

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

Complexity of Heap Sort

- Overall Sorting time:
  - Total ADJUST time + Making an initial heap
- Total ADJUST time: $O(n \log n)$
  - Each ADJUST with n nodes. : $O(\log n)$
  - We need n times of ADJUST after deleting each root.
- Making an initial heap
  - $O(n \log n)$ by INSERT
  - $O(n)$ by ADJUST
- Therefore, totally $O(n \log n)$.

INSERT vs ADJUST

- ADJUST : $O(n)$
  - But it requires that all the elements should be available before the heap creation begins.
- INSERT : $O(n \log n)$
  - We can add a new element into the heap at any time.

Partial Sorting using Heap Sort

- We need only first k records in the sorted nodes from n records, especially when $k << n$.

  $$\Rightarrow O(k \log n)$$

Radix Sort
Bin Sort

RADIX SORT

- There are several keys.
  - Example: sorting a deck of cards.
    - [suit] ♣< ◊ < ♥ < ♠
    - [Face Value] 2 < 3 < . . . < 9 < 10 < J < Q < K < A
    - Two possible Sorted Decks:
      - 2♣ < 3♣ < . . . < A♣ < . . . . . . < 2♠ < 3♠ . . . < A♠
      - 2♣ < 2◊ < 2♥ < 2♠ < 3♣ < 3◊ < . . . < A♣ < A◊ < A♥ < A♠
  - Basic idea:
    - Sort by each key separately.

- Notation:
  - Each Record has the keys $K^0, \ldots, K^{r-1}$
  - So, the record R_i has the keys $K_i^0, \ldots, K_i^{r-1}$
- Comparison of multiple keys:
  - $(x_0, x_1, \ldots, x_{r-1}) \leq (y_0, y_1, \ldots, y_{r-1})$

    iff either $x_i = y_i, 0 \leq i < j$ and $x_{j+1} < y_{j+1}$ for some $j < r - 1$

    or $x_i = y_i, 0 \leq i < r$.
- Definition of Sorted Order with several keys:
  - A list of records, $R_0, \ldots, R_{n-1}$, is lexically sorted with respect to the keys $K^0, K^1, \ldots, K^{r-1}$

    iff $(K_i^0, K_i^1, \ldots, K_i^{r-1}) \leq (K_{i+!}^0, K_{i+1}^1, \ldots, K_{i+1}^{r-1}), 0 \leq i < n-1$

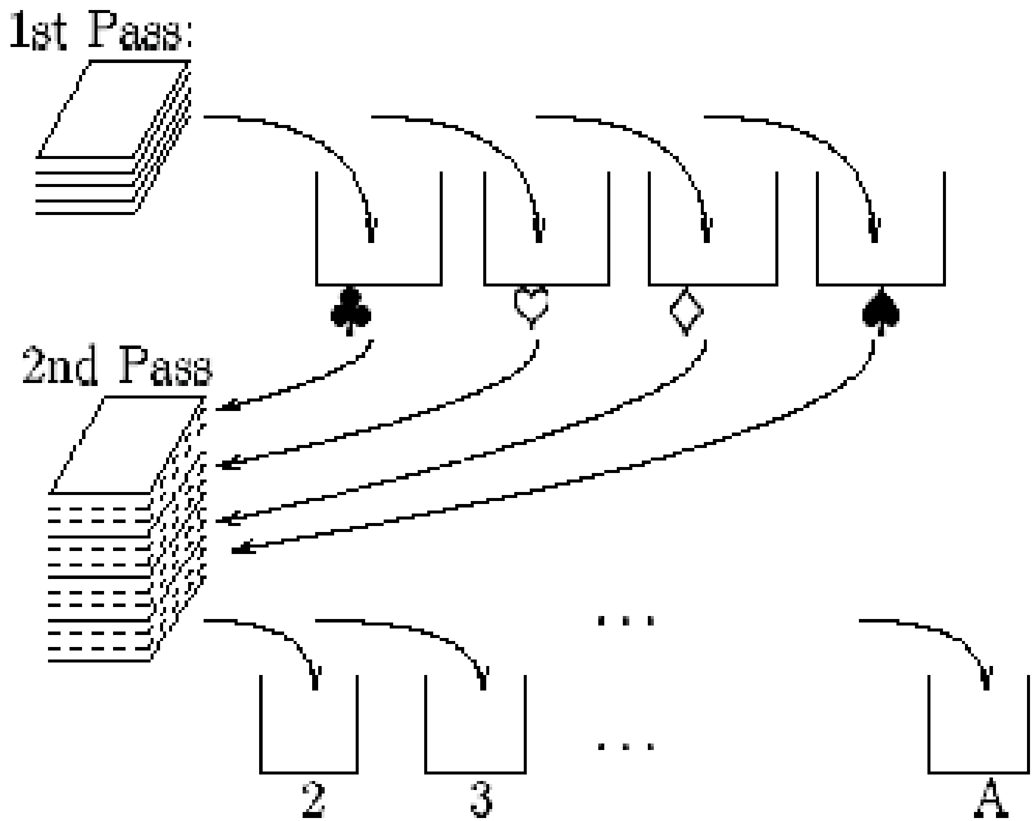
Two approaches for Radix Sort

- MSD (Most Significant Digit) Sort
  - K^0 first
  - Example for sorting a card deck: K^0[suit] and K^1[face value]
    1. Sort on suit, and have 4 piles of cards.
    2. Sort each suit file on face value separately.
    3. Stack the 4 files so that the space(♠) file is on the bottom and the club(♣) file is on the top.
- LSD (Least Significant Digit) Sort
  - K^{r-1} first
  - Example for sorting a card deck: K^0[suit] and K^1[face value]
    1. Sort on face value, and have 13 piles of cards.
    2. Stack the piles to obtain a single pile.
    3. Sort on suit, and have 4 piles.
    4. Stack the piles to obtain a sorted deck.
  - The sorting method in the second and the later passes must be stable.

- The LSD approach is simpler than the MSD approach.
  - In LSD approach, we do not have to sort subpiles independently.
- The term LSD or MSD indicates only the order in which the keys are sorted
  - They do not specify how each key is to be sorted.
  - We usually create bins.

Bin Sort

- It is an MSD sort.
  - Time Complexity: O(rn)=O(n)



Applying Radix Sort to sorting with only one logical key

- We interpret the key as a composite of several keys
  - $K = (K^0, K^1, \ldots, K^{r-1})$
  - Example: $0 \leq K \leq 999$
    - Let each K^i be as follows:
      - K^0 is the digit in the 100s place
      - K^1 is the digit in the 10s place
      - K^2 is the digit in the units place
    - Then, each K^i is $0 \leq K^i \leq 999$.
    - The Sort for each key requires only 10 bins.
  - We decompose the sort key into digits using a radix r.
    - When r=10, we get the decimal decomposition.
    - When r=2, we get the binary decomposition.
  - With a radix of r, r bins are needed to sort each digit.
- LSD radix r Sort
  - Assume that the records, $R_0, \ldots, R_{n-1}$, have the keys that are d-tuples $(x_0, x_1, \ldots, x_{d-1})$, and $0 \leq x_i < r$.
  - Implementation using linked lists for bins: See Program 7.15
  - Analysis: $O(d(r+n))$

Summary

Practical Considerations

- Sorting method vs the data size
  - $n \le 20{\sim}25$: Use Insertion sort ($O(n^2)$)
  - $n > 20{\sim}25$: Use Quick sort ($O(n \log n)$)
  - For quick sort:
    - If the subfile size ≤ 20 after partitioning, then sort this subfile using Insertion Sort instead of Quick Sort.
    - This situation is similar in other sorts.
- Long Records
  - Exchanging records requires much time.
  - Use linked lists
- Running Time Comparisons among Sorts

	n=256	n=512
Insertion	336	1444
Bubble	1026	4054
Heap	110	241
Quick	60	146
Merge	102	242

End
of
Heap & Radix Sort