Data Structures

Lecture 8: Priority Queue

Dongbo Min

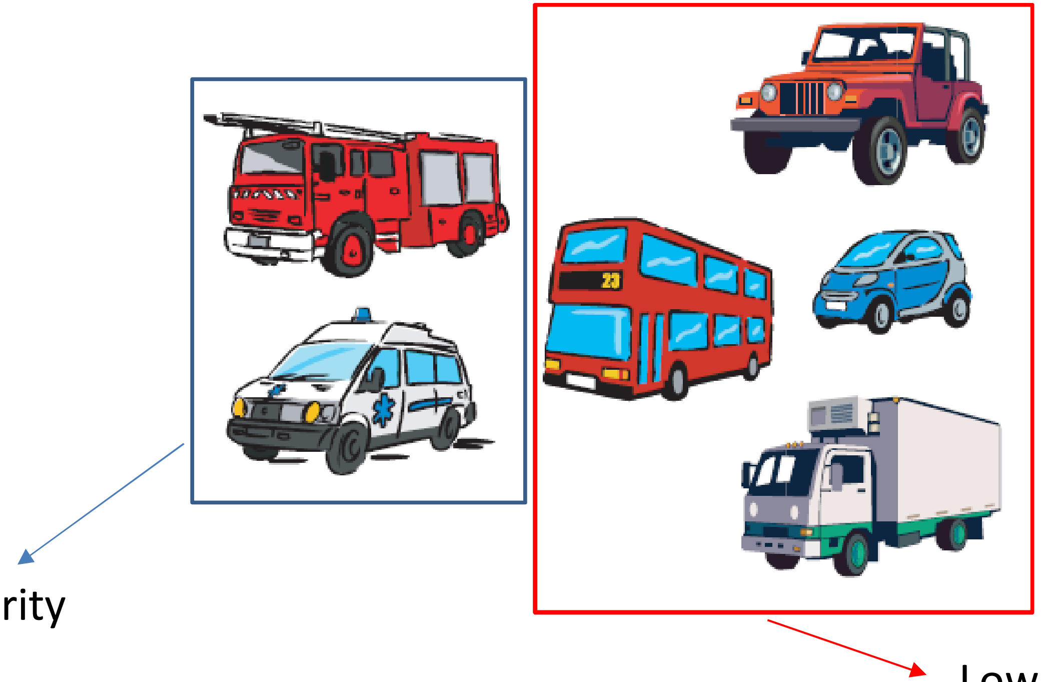
Department of Computer Science and Engineering

Ewha Womans University, Korea

E-mail: dbmin@ewha.ac.kr

Priority Queue

- Priority queue
  - A queue that stores items with priority
  - The data with the higher priority is output first, not the FIFO order.



High priority

Low priority

이화여자대학교
EWHA WOMANS UNIVERSITY

Priority Queue

- Priority queue: most common data structure
  - You can implement stack or FIFO queues as priority queues.

Data structure	Elements to be removed
Stack	Most recent data
Queue	First incoming data
Priority queue	Highest priority data

- Applications
  - Simulation system (priority: the event time)
  - Network traffic control
  - Scheduling Jobs in the Operating System

이화여자대학교
EWHA WOMANS UNIVERSITY

ADT of Priority Queue

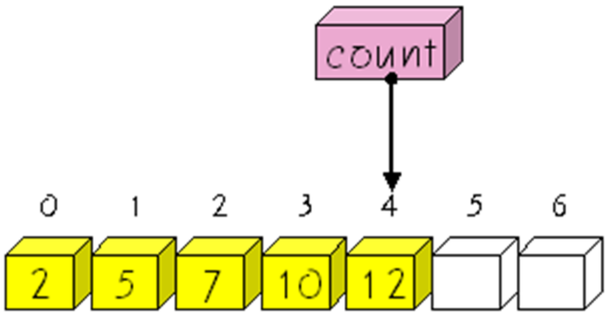
- Object: A collection of elements with a priority of n element types
- Operation:
  - Create () :: = Creates a priority queue.
  - Init (q) :: = Initializes the priority queue q.
  - is_empty (q) :: = Checks if the priority queue q is empty.
  - is_full (q) :: = Checks if the priority queue q is full.
  - insert (q, x) :: = Add an element x to the priority queue q.
  - delete (q) :: = Removes the highest priority element from the priority queue and returns this element.
  - find (q) :: = Returns the highest priority element without deleting it.

Priority Queue Types

- Priority queues are divided into two categories
  - Minimum priority queue
  - Maximum priority queue

Priority Queue Implementation

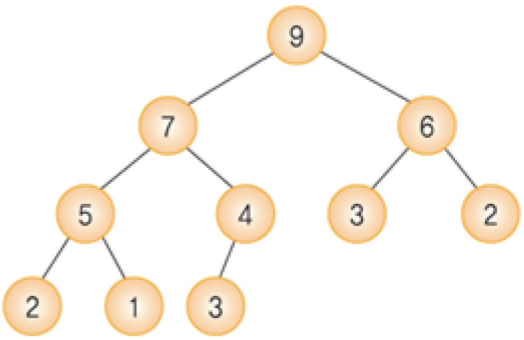
- Priority queue with arrays



- Priority queue with linked list

Head pointer

- Priority queue with heap



이화여자대학교 EWHA WOMANS UNIVERSITY

Priority Queue Implementation

Data structure	*Insertion operation*	*Deletion operation*
Unordered array	*O(1)*	*O(n)*
Unordered linked list	*O(1)*	*O(n)*
Ordered array	*O(n)*	*O(1)*
Ordered linked list	*O(n)*	*O(1)*
Heap	*O(logn)*	*O(logn)*

이화여자대학교 EWHA WOMANS UNIVERSITY

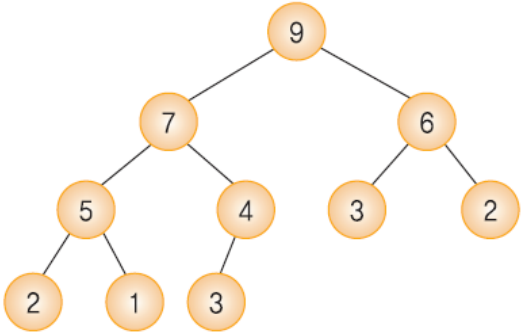
Heap

- Heap
  - Complete binary tree satisfying the following conditions
- Max heap

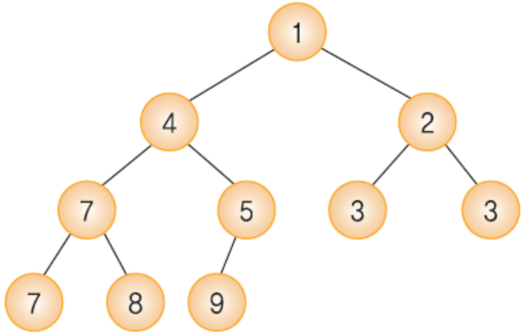
key (parent node) ≥key (child node)

- Min heap

key (parent node) ≤key (child node)



Map heap

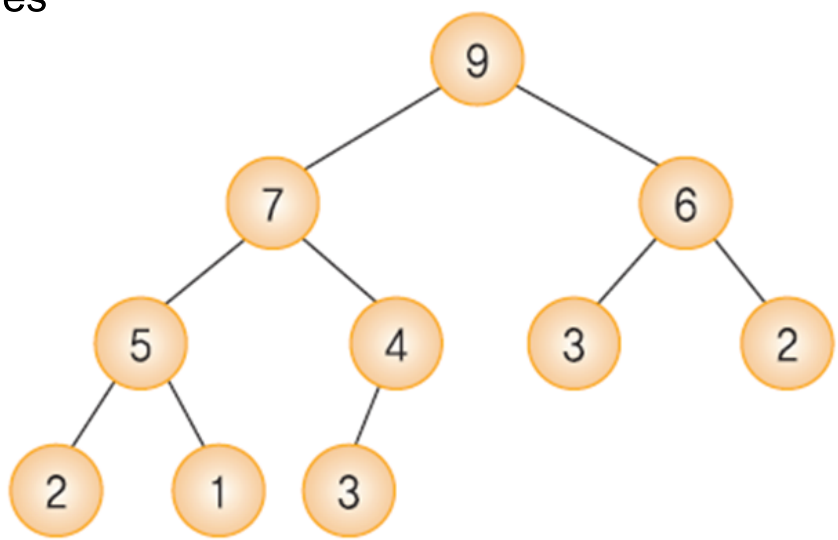


Min heap

Heap Height

- The height of the heap with n nodes is $O(log_2 n)$
  - Note) the heap is a complete binary tree
  - Except for the last level, there are 2^{i-1} nodes at each level i

Level	# of nodes
1	$1=2^0$
2	$2=2^1$
3	$4=2^2$
4	3

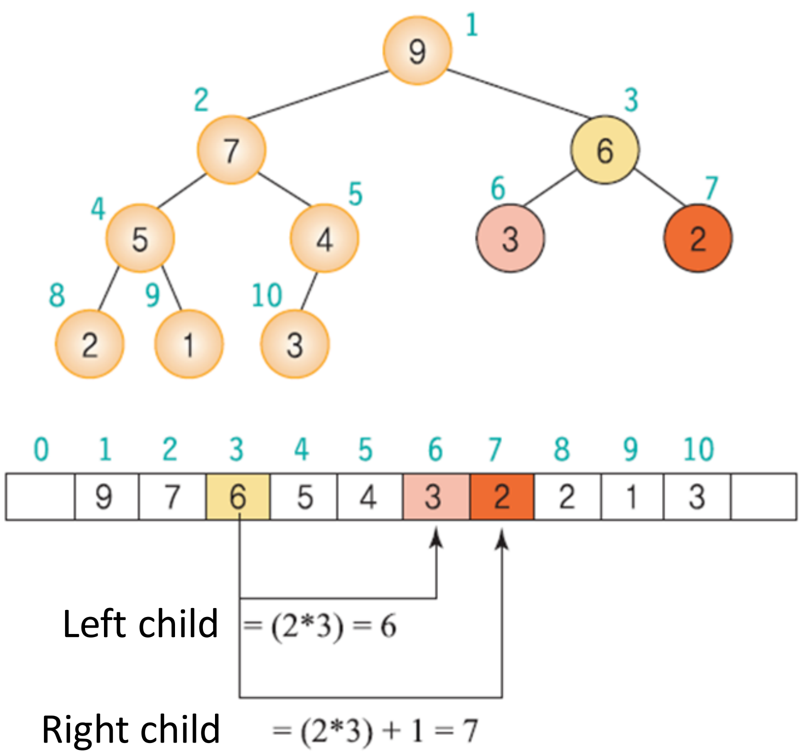


이화여자대학교
EWHA WOMANS UNIVERSITY

Heap Implementation

- Heap is implemented using arrays
  - Since it is a fully binary tree, each node can be numbered
  - Think of this number as the index of the array

- Index of parent and child
  - Parent node of node i: $i/2$
  - Left child node of node i: $2i$
  - Right child node of node i: $2i + 1$



이화여자대학교 EWHA WOMANS UNIVERSITY

Insertion in Map Heap

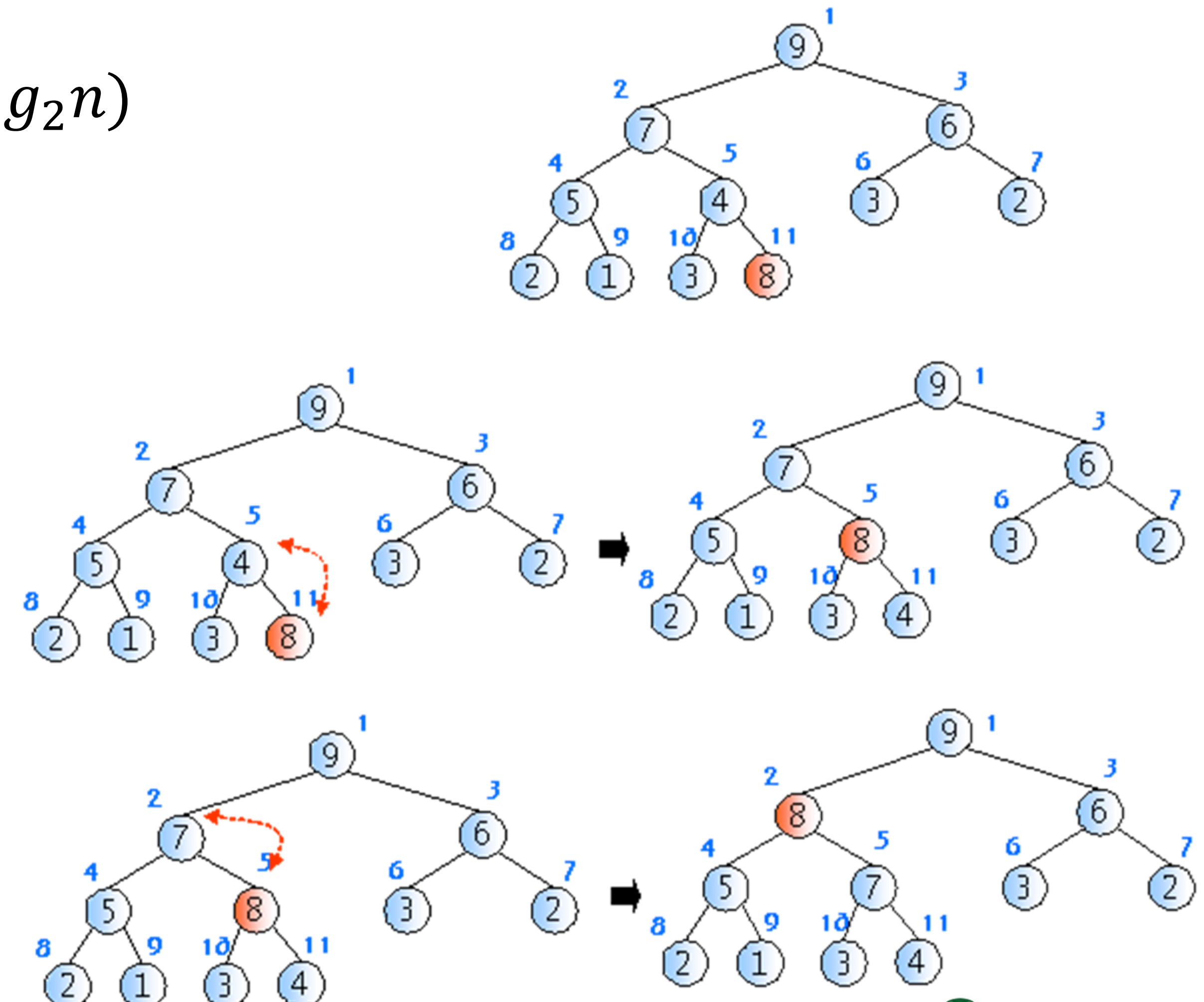
- Procedure
  1. The new node is inserted next to the last node in the heap
  2. The nodes in the path from the inserted node to the root are compared and exchanged to satisfy the heap property.

Note) Insertion in min heap is also similar

이화여자대학교 EWHA WOMANS UNIVERSITY

Insertion in Map Heap

- If the key k is less than or equal to the parent node, the comparison terminates.
- Time complexity: $O(log_2 n)$



이화여자대학교
EWHA WOMANS UNIVERSITY

Insertion in Map Heap

Pseudo code

```
insert_max_heap(A, key)

heap_size ← heap_size + 1;
i ← heap_size;
A[i] ← key;
while i ≠ 1 and A[i] > A[PARENT(i)] do
                  A[i] ↔ A[PARENT];
                  i ← PARENT(i);
```

```
// Insert the item at heap h, (# of elements: heap_size)
void insert_max_heap(HeapType *h, element item)
{
         int i;
         i = ++(h->heap_size);

         // The process of comparing with the parent node as it traverses the tree
         while ( (i!=1) && (item.key > h->heap[i/2].key) ) {
                  h->heap[i] = h->heap[i/2];
                  i /= 2;
         }
         h->heap[i] = item; // Insert new node
}
```

Note) Exchange operation is not used here for more efficient implementation.

이화여자대학교 EWHA WOMANS UNIVERSITY

Deletion in Map Heap

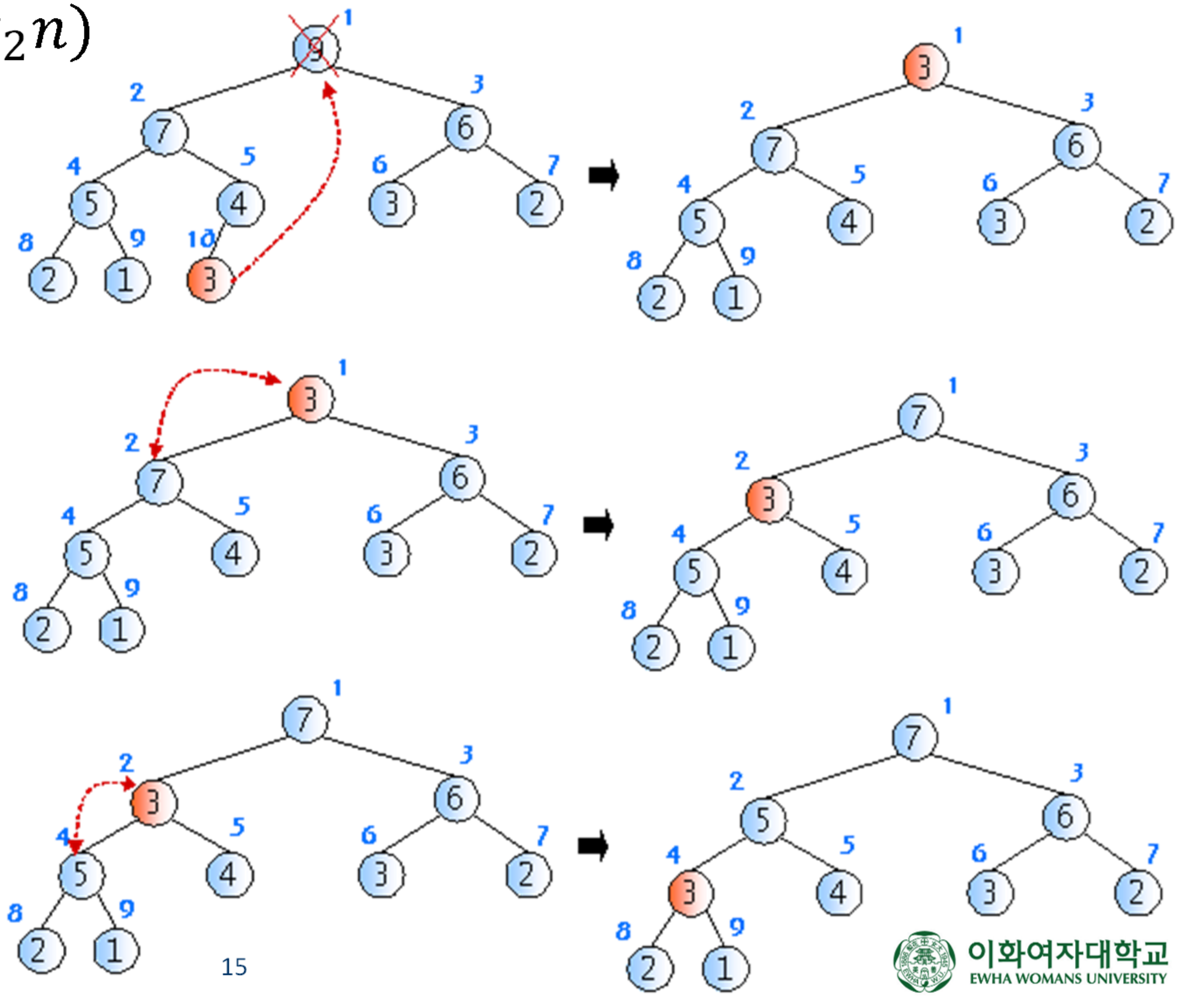
- Deletion in max heap
  – Deleting the node with the largest key value

- Procedure
  1. Delete the root node
  2. Move the last node to the root node.
  3. The nodes in the path from the root to the leaf nodes are compared and exchanged to satisfy the heap property.

이화여자대학교
EWHA WOMANS UNIVERSITY

Deletion in Max Heap

- If the key k is larger than or equal to the child nodes, the comparison terminates.
- Time complexity: $O(log_2 n)$



이화여자대학교 EWHA WOMANS UNIVERSITY

Deletion in Max Heap

Pseudo code

```
delete_max_heap(A)

item ← A[1];
A[1] ← A[heap_size];
heap_size ← heap_size-1;
i ← 2;
while i ≤ heap_size do
         if i < heap_size and A[i+1] > A[i]
                  then largest ← i+1;
         else largest ← i;

         if A[PARENT(largest)] > A[largest]
                  then break;
         A[PARENT(largest)] ↔ A[largest];
         i ← CHILD(largest);

return item;
```


이화여자대학교
EWHA WOMANS UNIVERSITY

Deletion in Max Heap

```
// Delete the root at heap h, (# of elements: heap_size)
element delete_max_heap(HeapType *h)
{
	int parent, child;
	element item, temp;

	item = h->heap[1];
	temp = h->heap[(h->heap_size)--];
	parent = 1;
	child = 2;
	while (child <= h->heap_size) {
		// Find a smaller child node
		if ((child < h->heap_size) &&
			(h->heap[child].key) < h->heap[child + 1].key)
			child++;
		if (temp.key >= h->heap[child].key) break;
		// Move down one level
		h->heap[parent] = h->heap[child];
		parent = child;
		child *= 2;
	}
	h->heap[parent] = temp;
	return item;
}
```

```c
#include <stdio.h>
#define MAX_ELEMENT 200
typedef struct {
        int key;
} element;
typedef struct {
        element heap[MAX_ELEMENT];
        int heap_size;
} HeapType;
```

```c
// Initialization
init(HeapType *h) {
        h->heap_size = 0;
}

void main()
{
        element e1 = { 10 }, e2 = { 5 }, e3 = { 30 };
        element e4, e5, e6;
        HeapType heap;
        init(&heap);

        // Insertion
        insert_max_heap(&heap, e1);
        insert_max_heap(&heap, e2);
        insert_max_heap(&heap, e3);

        // Deletion
        e4 = delete_max_heap(&heap);
        printf("< %d > ", e4.key);
        e5 = delete_max_heap(&heap);
        printf("< %d > ", e5.key);
        e6 = delete_max_heap(&heap);
        printf("< %d > ", e6.key);
}
```

Building the max heap by iteratively inserting elements

Time complexity $O(nlog_2n)$

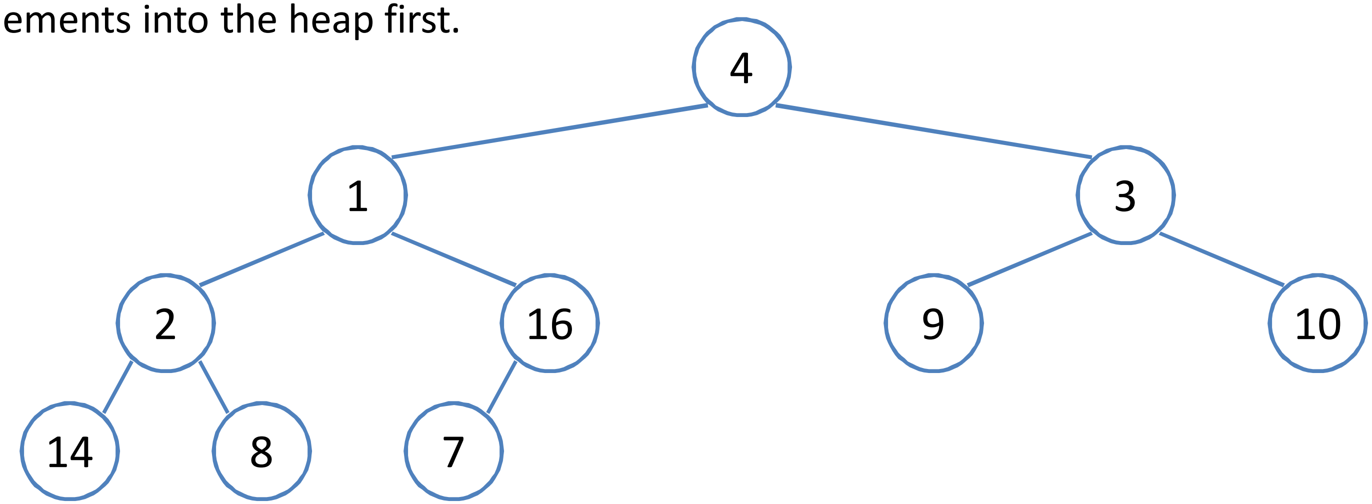
Building Max Heap

Extra

- Building max heap can be performed in $O(n)$

A = {4, 1, 3, 2, 16, 9, 10, 14, 8, 7}

When an array is given,
put all elements into the heap first.



Key observations

1. Nodes with 14, 8, 7, 9, and 10 already meet the heap property.
2. Order of processing (i.e., moving elements) does matter.

이화여자대학교 EWHA WOMANS UNIVERSITY

Building Max Heap

Extra

- Building max heap can be performed in $O(n)$

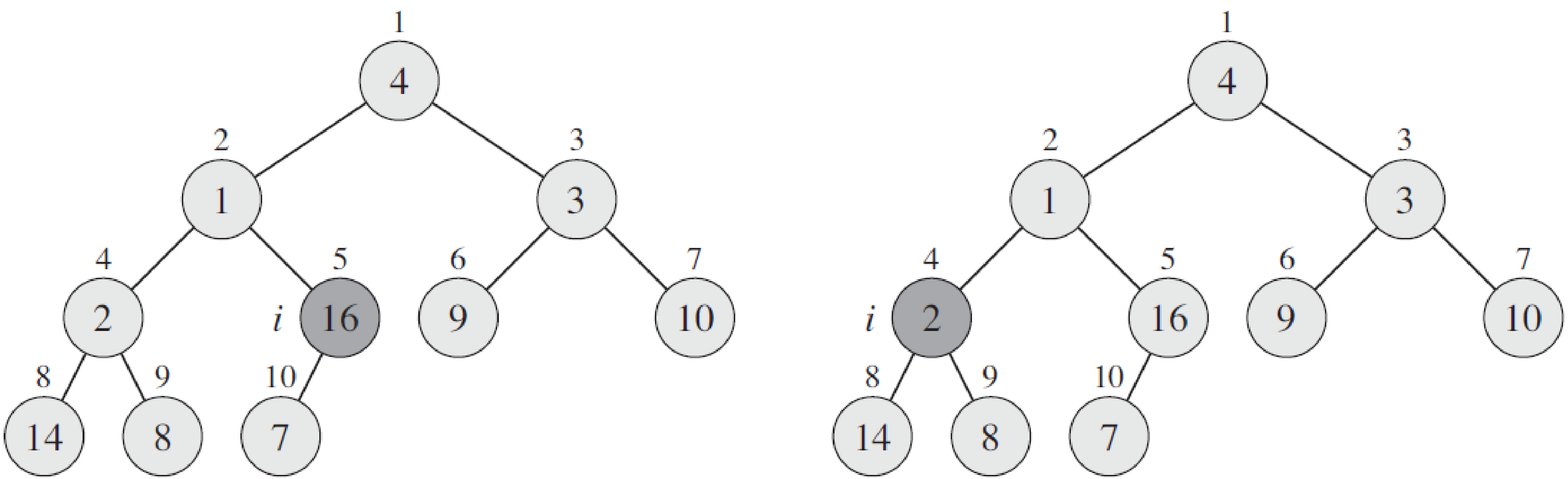
- `Build-Max-Heap()`
  - When an array is given, put all elements into the heap first.
  - Then, we can build a heap in a bottom-up manner by moving the element to meet the heap property.
  - For the array of length *n*, all elements in $\lfloor n/2 \rfloor + 1 \ldots n$ already meet the heap property!
  - Thus,
    - Walk backwards through the array from $\lfloor n/2 \rfloor$ to 1, moving the element on each node until it meets the heap property.
    - The order of processing guarantees that the children of node *i* are heaps when *i* is processed

이화여자대학교
EWHA WOMANS UNIVERSITY

Building Max Heap

Extra

A										
	4	1	3	2	16	9	10	14	8	7

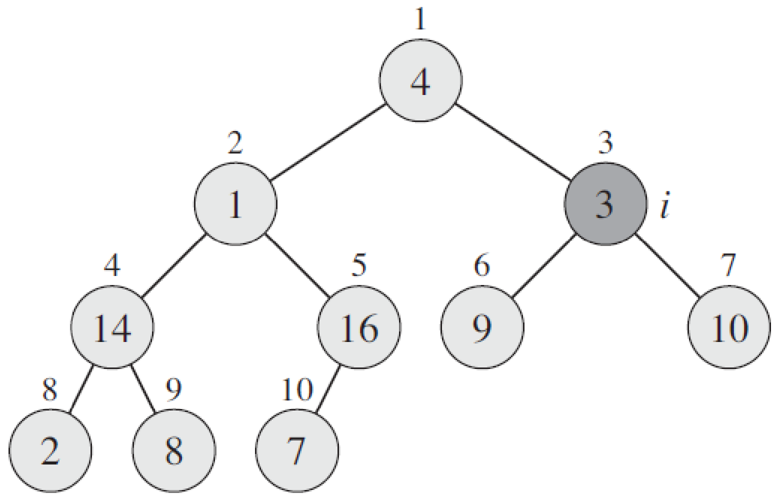


(a) (b)

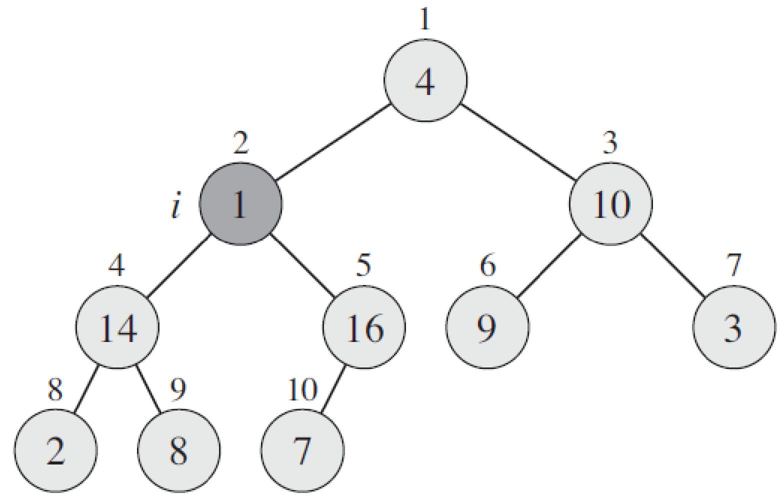
Moving the node in the heap can be implemented in a manner similar to the insertion operation.

Building Max Heap

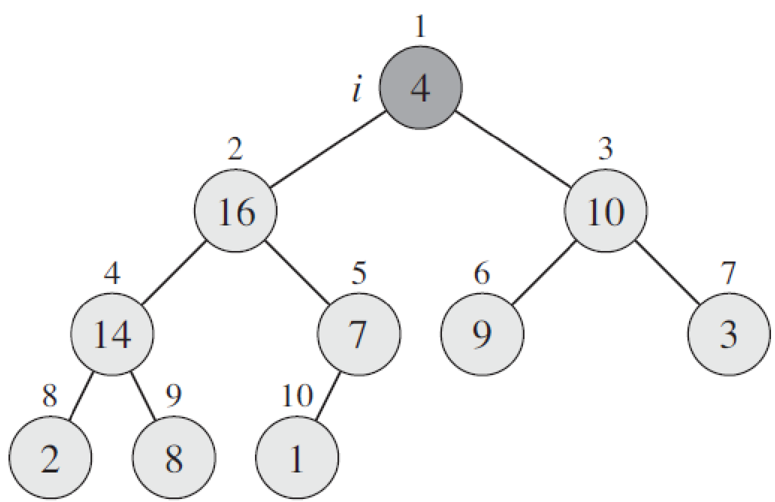
Extra



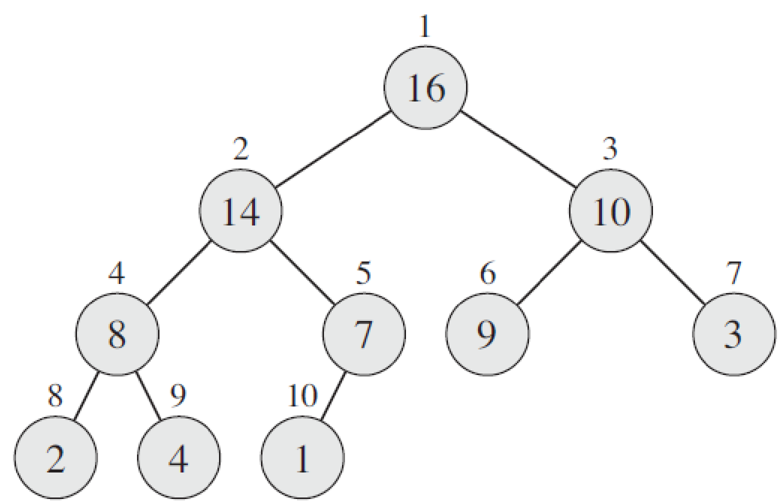
(c)



(d)



(e)



(f)

이화여자대학교
EWHA WOMANS UNIVERSITY

Analyzing Build-Max-Heap()

Extra

- Moving an element in the heap takes $O(log_2n)$ time

- There are $O(n)$ moving operations (specifically, $\lfloor n/2 \rfloor$)

- Thus, the time complexity may be seen as $O(nlog_2n)$
  - Is this a correct asymptotic upper bound?
  - Is this an asymptotically tight bound?

- A tighter bound is $O(n)$.

이화여자대학교
EWHA WOMANS UNIVERSITY

Analyzing Build-Max-Heap()

Extra

- For a subtree, the moving operation takes $O(h)$ time
  - h: the height of the subtree
  - $h = O(log_2 m)$, m = # nodes in subtree
  - The height of most subtrees is small.

- An n-element heap has at most $\lceil n/2^{h+1} \rceil$ nodes of height h

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right)$$

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2} = 2 .$$

$$\Rightarrow \quad O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) = O(n) .$$

이화여자대학교
EWHA WOMANS UNIVERSITY

Heap Application: Heap Sort

- Heap sort: the sorting algorithm using the heap
- Procedure
  1. Insert n elements to be sorted in the max heap
  2. Delete a root from the max heap and save it in an array n times
- Time complexity: $O(nlog_2n)$
  - Step 1: $O(nlog_2n)$ + Step 2: $O(nlog_2n)$

n: the number of elements

> Note) When using 'build-max-heap()' in step 1, it takes $O(n)$,
> but the time complexity of the heap sort is still $O(nlog_2n)$.

- Heap sort is useful, when you need a few of the largest values, not sorting the entire data.
  - Note: it is recommended to implement step 1) using 'build-max-heap' to make this true.

이화여자대학교 EWHA WOMANS UNIVERSITY

Heap Application: Heap Sort

```
void heap_sort(element a[], int n)
{
	int i;
	HeapType h;

	init(&h);
	for (i = 0; i<n; i++) {
		insert_max_heap(&h, a[i]);
	}
	for (i = (n - 1); i >= 0; i--) {
		a[i] = delete_max_heap(&h);
	}
}
```

To store the data in an ascending order

이화여자대학교 EWHA WOMANS UNIVERSITY

Heap Application: Discrete Event Simulation

- Computer simulation
  - Designs a model of physical system, runs the model using computers, and then analyzes the execution results
  - Type of computer simulation
    - Continuous time simulation
    - Discrete time simulation:
      is performed by generating an event as time passes
    - Discrete event simulation:
      is performed by the occurrence of an event

```
while (clock < duration) {
        clock++;
        .....
}
```

Discrete time simulation

```
while (t < test_time) {
        t += random(max_arr_interval+1);
        //Generate an event below.
}
```

Discrete event simulation

Note) Banking simulation in the lecture note ‘queue’ is a discrete time simulation

이화여자대학교 EWHA WOMANS UNIVERSITY

Heap Application: Discrete Event Simulation

- Discrete event simulation
  - All time progress is made by the occurrence of an event.
  - Events are stored using priority queues, and they are processed based on the event time.

Example) Ice cream shop simulation

Guests visit the ice cream shop. If there is no seat available, then they will just leave. Our goal is to predict how many chairs can be placed to maximize profits.

Definition of event

```
typedef struct {
        int id;// Guest group ID
        int type;// Event type
        int key;// Time when the event occurred
        int number;// Number of guests for the event
} element;
```


이화여자대학교
EWHA WOMANS UNIVERSITY

Ice Cream Shop Simulation

- Event types
  - ARRIVAL: the guest arrives the ice cream shop
  - ORDER: the guest orders the ice cream
  - LEAVE: the guest leaves the ice cream shop
- ARRIVAL event
  - If # of guests for this event < # of remaining chairs,
    then receive the guests and # of remaining chairs is reduced by # of guests.
  - Otherwise, guests will leave without ordering.
- ORDER event
  - This event receives the order according to # of guests, and then they will leave after a while.
- LEAVE event
  - This event increases # of remaining chairs by # of guests leaving.

이화여자대학교
EWHA WOMANS UNIVERSITY

Ice Cream Shop Simulation

- Random variables
  - ARRIVAL event
    - Arrival time of guests
    - The number of guests of a single event
  - ORDER event
    - Time to order after arrival
    - The number of scoops that each guest orders
  - LEAVE event
    - Time to stay in the shop before leaving

- Note that min heap is used in this simulation, since the event that occurs first must be processed.

이화여자대학교
EWHA WOMANS UNIVERSITY

Simple example

Heap

```
event.id = 0;
event.type = ARRIVAL;
event.key = 1;
event.number = 5;
```

```
event.id = 1;
event.type = ARRIVAL;
event.key = 3;
event.number = 2;
```

```
typedef struct {
     int id;// Guest group ID
     int type;// Event type
     int key;// Time when the event occurred
     int number;// Number of guests for the event
} element;
```

Initially, free_seats = 10

이화여자대학교
EWHA WOMANS UNIVERSITY

Simple example

```c
typedef struct {
	int id;// Guest group ID
	int type;// Event type
	int key;// Time when the event occurred
	int number;// Number of guests for the event
} element;
```

Heap

```c
event.id = 1;
event.type = ARRIVAL;
event.key = 3;
event.number = 2;
```

free_seats = 10

1. Based on arrival time (key), extract this event from heap

```c
event.id = 0;
event.type = ARRIVAL;
event.key = 1;
event.number = 5;
```

Simple example

Heap

```
event.id = 1;
event.type = ARRIVAL;
event.key = 3;
event.number = 2;
```

```
typedef struct {
    int id;// Guest group ID
    int type;// Event type
    int key;// Time when the event occurred
    int number;// Number of guests for the event
} element;
```

free_seats = 5

1. Based on arrival time (key), extract this event from heap

```
event.id = 0;
event.type = ARRIVAL;
event.key = 1;
event.number = 5;
```

2. Since free_seats(=10) > event.number(=5), this event stays in the shop (ORDER).
So, this will be added back to heap.
event.key = 6 by adding the time to order (5)
free_seats = 5 (=10-5)

```
event.id = 0;
event.type = ORDER;
event.key = 6; (=1+5)
event.number = 5;
```


이화여자대학교
EWHA WOMANS UNIVERSITY

Simple example

```c
typedef struct {
	int id;// Guest group ID
	int type;// Event type
	int key;// Time when the event occurred
	int number;// Number of guests for the event
} element;
```

Heap

```
event.id = 0;
event.type = ORDER;
event.key = 6;
event.number = 5;
```

```
event.id = 1;
event.type = ARRIVAL;
event.key = 3;
event.number = 2;
```

free_seats = 5

1. Based on arrival time (key), extract this event from heap

```
event.id = 0;
event.type = ARRIVAL;
event.key = 1;
event.number = 5;
```

2. Since free_seats(=10) > event.number(=5), this event stays in the shop (ORDER).
So, this will be added back to heap.
event.key = 6 by adding the time to order (5)
free_seats = 5 (=10-5)

이화여자대학교 EWHA WOMANS UNIVERSITY

Simple example

Heap

```
event.id = 0;
event.type = ORDER;
event.key = 6;
event.number = 5;
```

```
typedef struct {
     int id;// Guest group ID
     int type;// Event type
     int key;// Time when the event occurred
     int number;// Number of guests for the event
} element;
```

free_seats = 5

3. Extract the following event from heap.

```
event.id = 1;
event.type = ARRIVAL;
event.key = 3;
event.number = 2;
```

Simple example

Heap

```
event.id = 0;
event.type = ORDER;
event.key = 6;
event.number = 5;
```

```
typedef struct {
      int id;// Guest group ID
      int type;// Event type
      int key;// Time when the event occurred
      int number;// Number of guests for the event
} element;
```

free_seats = 3

3. Extract the following event from heap.

```
event.id = 1;
event.type = ARRIVAL;
event.key = 3;
event.number = 2;
```

4. Since free_seats(=5) > event.number(=3), this event stays in the shop (ORDER).
So, this will be added back to heap.
event.key becomes 5 by adding the time to order (2)
free_seats becomes 3 (=5-2)

```
event.id = 1;
event.type = ORDER;
event.key = 5; (=3+2)
event.number = 2;
```


이화여자대학교
EWHA WOMANS UNIVERSITY

Simple example

```c
typedef struct {
     int id;// Guest group ID
     int type;// Event type
     int key;// Time when the event occurred
     int number;// Number of guests for the event
} element;
```

Heap

```c
event.id = 0;
event.type = ORDER;
event.key = 6;
event.number = 5;
```

```c
event.id = 1;
event.type = ORDER;
event.key = 5;
event.number = 2;
```

free_seats = 3

3. Extract the following event from heap.

```c
event.id = 1;
event.type = ARRIVAL;
event.key = 3;
event.number = 2;
```

4. Since free_seats(=5) > event.number(=3), this event stays in the shop (ORDER).
So, this will be added back to heap.
event.key becomes 5 by adding the time to order (2)
free_seats becomes 3 (=5-2)

이화여자대학교 EWHA WOMANS UNIVERSITY

Simple example

Heap

```
event.id = 0;
event.type = ORDER;
event.key = 6;
event.number = 5;
```

```
event.id = 1;
event.type = ORDER;
event.key = 5;
event.number = 2;
```

```
typedef struct {
    int id;// Guest group ID
    int type;// Event type
    int key;// Time when the event occurred
    int number;// Number of guests for the event
} element;
```

free_seats = 3

Simple example

```c
typedef struct {
     int id;// Guest group ID
     int type;// Event type
     int key;// Time when the event occurred
     int number;// Number of guests for the event
} element;
```

Heap

```c
event.id = 0;
event.type = ORDER;
event.key = 6;
event.number = 5;
```

free_seats = 3

5. Based on key, extract this event from heap

```c
event.id = 1;
event.type = ORDER;
event.key = 5;
event.number = 2;
```

Simple example

Heap

```
event.id = 0;
event.type = ORDER;
event.key = 6;
event.number = 5;
```

```
typedef struct {
     int id;// Guest group ID
     int type;// Event type
     int key;// Time when the event occurred
     int number;// Number of guests for the event
} element;
```

free_seats = 3

5. Based on key, extract this event from heap

```
event.id = 1;
event.type = ORDER;
event.key = 5;
event.number = 2;
```

6. For 2 guests, start the order.
After the order, convert it into ‘LEAVE’.
So, this will be added back to heap.
event.key becomes 12 by adding the time to stay (7)

```
event.id = 1;
event.type = LEAVE;
event.key = 12; (=5+7)
event.number = 2;
```

Simple example

Heap

```
event.id = 0;
event.type = ORDER;
event.key = 6;
event.number = 5;
```

```
event.id = 1;
event.type = LEAVE;
event.key = 12;
event.number = 2;
```

```c
typedef struct {
    int id;// Guest group ID
    int type;// Event type
    int key;// Time when the event occurred
    int number;// Number of guests for the event
} element;
```

free_seats = 3

5. Based on key, extract this event from heap

```
event.id = 1;
event.type = ORDER;
event.key = 5;
event.number = 2;
```

6. For 2 guests, start the order.
After the order, convert it into ‘LEAVE’.
So, this will be added back to heap.
event.key becomes 12 by adding the time to stay (7)

이화여자대학교 EWHA WOMANS UNIVERSITY

Simple example

```c
typedef struct {
     int id;// Guest group ID
     int type;// Event type
     int key;// Time when the event occurred
     int number;// Number of guests for the event
} element;
```

Heap

```c
event.id = 0;
event.type = ORDER;
event.key = 6;
event.number = 5;
```

```c
event.id = 1;
event.type = LEAVE;
event.key = 12;
event.number = 2;
```

free_seats = 3

이화여자대학교 EWHA WOMANS UNIVERSITY

Simple example

```c
typedef struct {
	int id;// Guest group ID
	int type;// Event type
	int key;// Time when the event occurred
	int number;// Number of guests for the event
} element;
```

Heap

```c
event.id = 1;
event.type = LEAVE;
event.key = 12;
event.number = 2;
```

free_seats = 3

7. Based on key, extract this event from heap

```c
event.id = 0;
event.type = ORDER;
event.key = 6;
event.number = 5;
```

Simple example

```c
typedef struct {
    int id;// Guest group ID
    int type;// Event type
    int key;// Time when the event occurred
    int number;// Number of guests for the event
} element;
```

Heap

```c
event.id = 1;
event.type = LEAVE;
event.key = 12;
event.number = 2;
```

free_seats = 3

7. Based on key, extract this event from heap

```c
event.id = 0;
event.type = ORDER;
event.key = 6;
event.number = 5;
```

8. For 2 guests, start the order.
After the order, convert it into 'LEAVE'.
So, this will be added back to heap.
event.key becomes 14 by adding the time to stay (8)

```c
event.id = 0;
event.type = LEAVE;
event.key = 14; (=6+8)
event.number = 5;
```


이화여자대학교 EWHA WOMANS UNIVERSITY

Simple example

```c
typedef struct {
      int id;// Guest group ID
      int type;// Event type
      int key;// Time when the event occurred
      int number;// Number of guests for the event
} element;
```

Heap

```c
event.id = 0;
event.type = LEAVE;
event.key = 14;
event.number = 5;
```

```c
event.id = 1;
event.type = LEAVE;
event.key = 12;
event.number = 2;
```

free_seats = 3

7. Based on key, extract this event from heap

```c
event.id = 0;
event.type = ORDER;
event.key = 6;
event.number = 5;
```

8. For 2 guests, start the order.
After the order, convert it into ‘LEAVE’.
So, this will be added back to heap.
event.key becomes 14 by adding the time to stay (8)

Simple example

Heap

```
event.id = 0;
event.type = LEAVE;
event.key = 14;
event.number = 5;
```

```
event.id = 1;
event.type = LEAVE;
event.key = 12;
event.number = 2;
```

```
typedef struct {
    int id;// Guest group ID
    int type;// Event type
    int key;// Time when the event occurred
    int number;// Number of guests for the event
} element;
```

free_seats = 3

이화여자대학교 EWHA WOMANS UNIVERSITY

Simple example

Heap

```
event.id = 0;
event.type = LEAVE;
event.key = 14;
event.number = 5;
```

```
typedef struct {
    int id;// Guest group ID
    int type;// Event type
    int key;// Time when the event occurred
    int number;// Number of guests for the event
} element;
```

free_seats = 5

9. Based on key, extract this event from heap

```
event.id = 1;
event.type = LEAVE;
event.key = 12;
event.number = 2;
```

10. This event will leave.
free_seats = 5 (=3+2)

Simple example

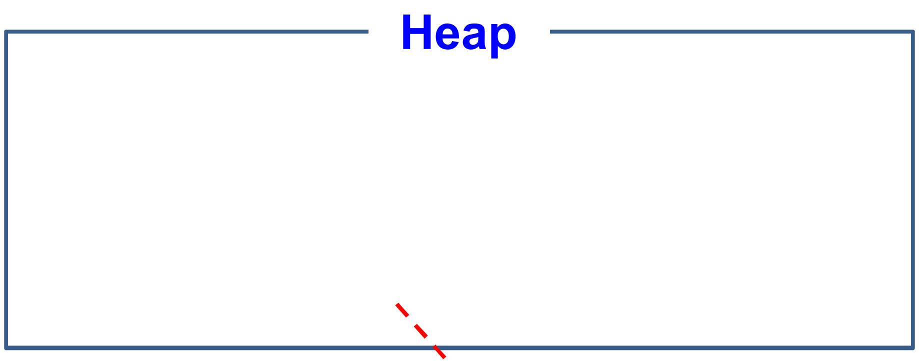
Heap

```
event.id = 0;
event.type = LEAVE;
event.key = 14;
event.number = 5;
```

```
typedef struct {
    int id;// Guest group ID
    int type;// Event type
    int key;// Time when the event occurred
    int number;// Number of guests for the event
} element;
```

free_seats = 5

Simple example




```
typedef struct {
     int id;// Guest group ID
     int type;// Event type
     int key;// Time when the event occurred
     int number;// Number of guests for the event
} element;
```

free_seats = 10

11. Based on key, extract this event from heap

```
event.id = 0;
event.type = LEAVE;
event.key = 14;
event.number = 5;
```

12. This event will leave.
free_seats = 10 (=5+5)

이화여자대학교
EWHA WOMANS UNIVERSITY

```c
typedef struct {
		int id;// Guest group ID
		int type;// Event type
		int key;// Time when the event occurred
		int number;// Number of guests for the event
} element;

typedef struct {
		element heap[MAX_ELEMENT];
		int heap_size;
} HeapType;
```

```c
#include "stdafx.h"
#include "stdlib.h"
#include "stdio.h"
#include "string.h"
#include "time.h"

#define ARRIVAL 1
#define ORDER 2
#define LEAVE 3
#define MAX_ELEMENT 1000


int free_seats = 10;//The number of chairs
int test_time = 10;// Simulation will be performed for 'test_time' (minute).
double profit_per_icecream = 0.35;//Profit per scoop

double profit = 0.0;
int groups = 0;// Total number of guest groups


// Random variables of ARRIVAL event
int max_arr_interval = 6;// Guests will arrive every [0 ~ 'max_interval_guest'] mins.
int max_num_of_guests = 4;//  1 <= The number of guests <= num_of_guests

// Random variables of ORDER event
int max_time_to_order = 4;// 1 <= Time to order after arrival <= 'max_time_to_order'
int max_num_icecream = 3;// 1 <= The number of ice creams that each guest orders <=
'max_num_icecream'

// Random variables of LEAVE event
int max_time_to_stay = 10;// 1 <= Time to stay before leaving <= 'max_time_to_stay'
```


이화여자대학교
EWHA WOMANS UNIVERSITY

Basic functions of heap

```
void init(HeapType *h) {
        h->heap_size = 0;
}
int is_empty(HeapType *h) {
        if (h->heap_size == 0)
                return true;
        else
                return false;
}
void insert_min_heap(HeapType *h, element item) {
        int i;
        i = ++(h->heap_size);
        // compare it with the parent node in an order from the leaf to the root
        while ((i != 1) && (item.key < h->heap[i/2].key)) {
                h->heap[i] = h->heap[i/2];
                i /= 2;
        }
        h->heap[i] = item; // Insert new node
}
element delete_min_heap(HeapType *h){
        int parent, child;
        element item, temp;
        item = h->heap[1];
        temp = h->heap[(h->heap_size)--];
        parent = 1;
        child = 2;
        while (child <= h->heap_size) {
                if ((child < h->heap_size) && (h->heap[child].key) > h->heap[child + 1].key)
                        child++;
                if (temp.key <= h->heap[child].key) break;
                h->heap[parent] = h->heap[child];
                parent = child;
                child *= 2;
        }
        h->heap[parent] = temp;
        return item;
}
```


이화여자대학교
EWHA WOMANS UNIVERSITY

```c
// Integer random number generation function between 0 and n-1
int random(int n)
{
		return rand() % n;
}

// If seats are available, reduce the number of remaining chairs by the number of guests.
int is_seat_available(element e)
{
		printf("Group %d of %d guests arrive.\n", e.id, e.number);
		if (free_seats >= e.number) {
				free_seats -= e.number;
				return true;
		}
		else {
				printf("Group %d of %d guests leave because there is no seat.\n", e.id, e.number);
				return false;
		}
}

// When you receive an order, increase the variable representing the net profit.
void order(element e, int scoops)
{
		printf("In group %d, %d ice creams ordered.\n", e.id, scoops);
		profit += profit_per_icecream * ((double)scoops);
}
// Increases the number of free seats when guests leave.
void leave(element e)
{
		printf("Group %d of %d guests leaves.\n", e.id, e.number);
		free_seats += e.number;
}
```

```c
void process_event(HeapType *heap, element e)
{
	int i = 0;
	element new_event;

	printf("\nCurrent time = %d\n", e.key);
	switch (e.type) {
	case ARRIVAL:
		// If seats are available, create an 'order' event.
		if (is_seat_available(e)) {
			new_event.id = e.id;
			new_event.type = ORDER;
			new_event.key = e.key + 1 + random(max_time_to_order);
			new_event.number = e.number;
			insert_min_heap(heap, new_event);
		}
		break;
	case ORDER:
		// receive orders according to the number of guests.
		for (i = 0; i < e.number; i++) {
			order(e, 1 + random(max_num_icecream));
		}
		// Create a 'leave' event.
		new_event.id = e.id;
		new_event.type = LEAVE;
		new_event.key = e.key + 1 + random(max_time_to_stay);
		new_event.number = e.number;
		insert_min_heap(heap, new_event);
		break;
	case LEAVE:
		// Increases the number of free seats when guests leave.
		leave(e);
		break;
	}
}
```


이화여자대학교
EWHA WOMANS UNIVERSITY

```c
int main()
{
		time_t t1;
		/* Intializes random number generator */
		srand((unsigned)time(&t1));

		element event;
		HeapType heap;
		int t = 0;
		init(&heap);
		// Create some events.
		while (t < test_time) {
			t += random(max_arr_interval+1);//random(n) returns an integer (0 ~ n-1)
			event.id = groups++;
			event.type = ARRIVAL;
			event.key = t;
			event.number = 1 + random(max_num_of_guests);
			insert_min_heap(&heap, event);
		}

		// Process events based on their priority
		while (!is_empty(&heap)) {
			event = delete_min_heap(&heap);
			process_event(&heap, event);
		}
		printf("\nTotal net profit = % f.\n\n", profit);
}
```

Generate groups for 'test_time'

Process groups based on arrival time

Question1: How do we modify this code to predict the number of chairs that maximize profits.
Question2: How do we modify this code when more than two different kinds of ice cream are on sale in the shop? Namely, each ice cream has different profit.

이화여자대학교 EWHA WOMANS UNIVERSITY

Huffman Code

- Huffman coding tree
  - A binary tree can be used to compress the data for which the frequency of each element is known.
  - This kind of binary tree is called the Huffman coding tree.

Frequency analysis

A	80
B	16
C	32
D	36
E	123
F	22
G	26
H	51
I	71
…	
Z	1

이화여자대학교 EWHA WOMANS UNIVERSITY

Huffman Code

- Huffman code is for data compression (e.g., JPEG)
  - Designing a *binary code*
  - Each character is represented by a unique binary string (codeword)
  - The length of codeword varies according to frequency

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Fixed-length codeword: $45 \cdot 3 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 3 + 5 \cdot 3 = 300$ bits

Variable-length codeword: $45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4 = 224$ bits

- Prefix codes
  - No codeword is also a prefix of some other codewords.
  - It achieves an optimal data compression among any character code.

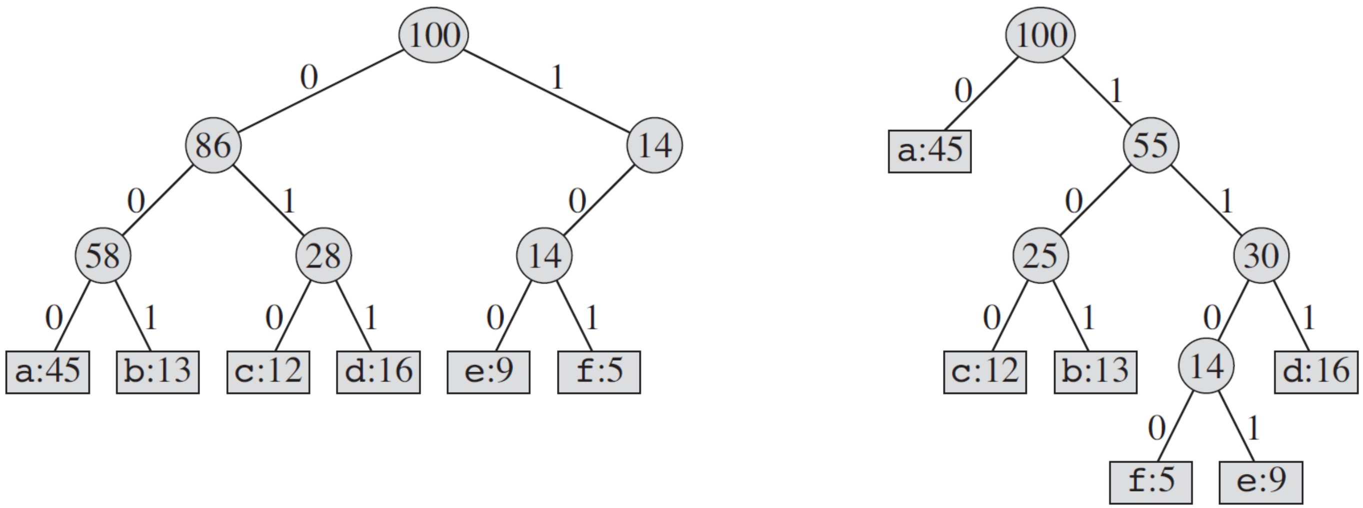
ex) 0101100 <-> abc (*unique* conversion)

이화여자대학교 EWHA WOMANS UNIVERSITY

Huffman Code

- From the tree's point of view

> Optimal codeword is represented by a ***full*** binary tree, in which every non-leaf node has two children.

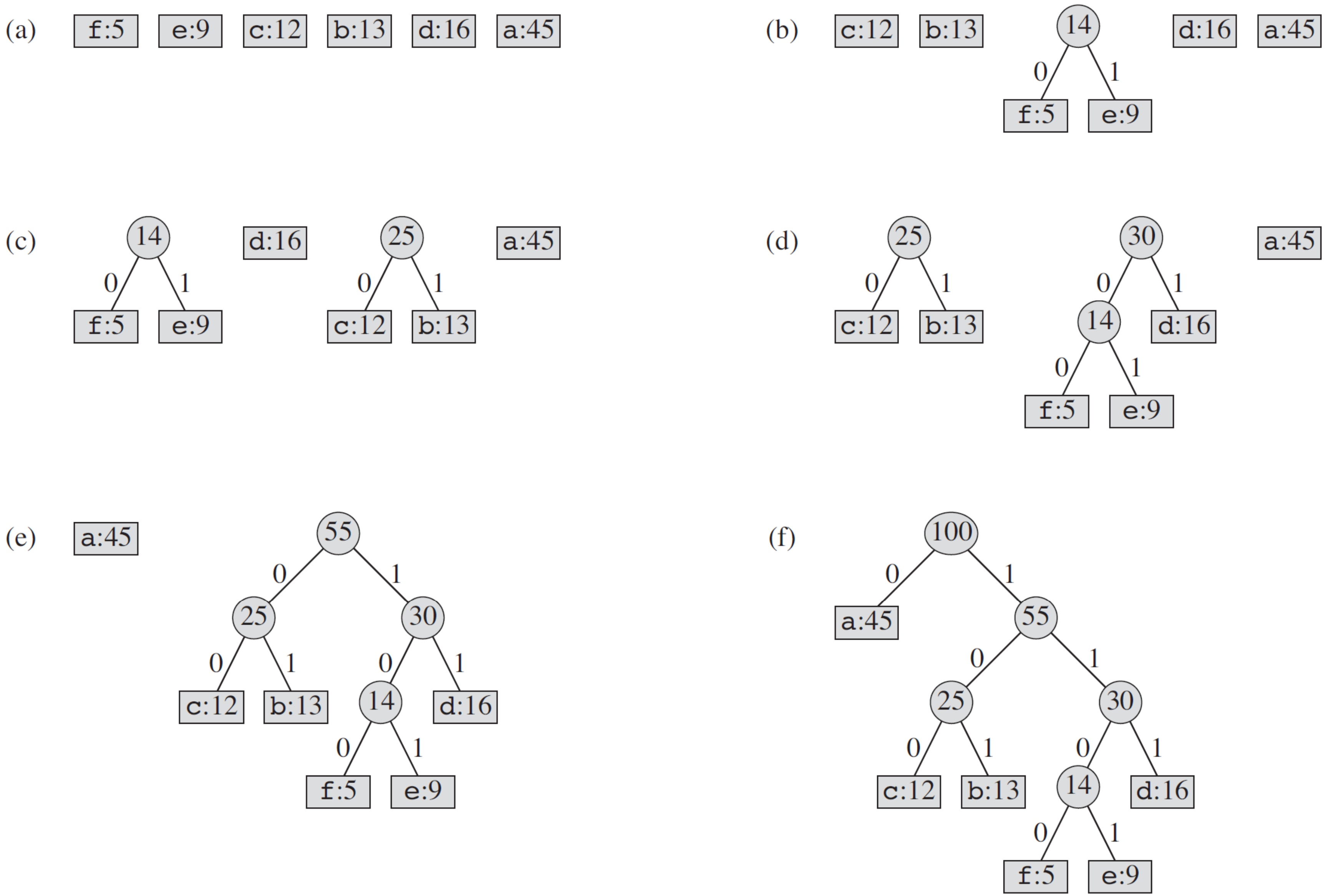


(a) Fixed-length codeword

(b) Variable-length codeword

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Generating Huffman Code



이화여자대학교
EWHA WOMANS UNIVERSITY

```c
#define MAX_ELEMENT 1000
typedef struct TreeNode {
		int weight;
		struct TreeNode *left_child;
		struct TreeNode *right_child;
} TreeNode;

typedef struct {
		TreeNode *ptree;
		int key;
} element;

typedef struct {
		element heap[MAX_ELEMENT];
		int heap_size;
} HeapType;

// Initialization
void init(HeapType *h) {
		h->heap_size = 0;
}
int is_empty(HeapType *h) {
		if (h->heap_size == 0)
				return true;
		else
				return false;
}

void insert_min_heap(HeapType *h, element item) {
		From previous slides
}

element delete_min_heap(HeapType *h) {
		From previous slides
}
```


이화여자대학교
EWHA WOMANS UNIVERSITY

```c
// Node generation in binary tree
TreeNode *make_tree(TreeNode *left, TreeNode *right)
{
	TreeNode *node = (TreeNode *)malloc(sizeof(TreeNode));
	if (node == NULL) {
		fprintf(stderr, "Memory allocation error\n");
		exit(1);
	}
	node->left_child = left;
	node->right_child = right;
	return node;
}

// Binary tree removal
void destroy_tree(TreeNode *root)
{
	if (root == NULL) return;
	destroy_tree(root->left_child);
	destroy_tree(root->right_child);
	free(root);
}
```

```c
// Huffman code generation
void huffman_tree(int freq[], int n)
{
		int i;
		TreeNode *node, *x;
		HeapType heap;
		element e, e1, e2;
		init(&heap);

		for (i = 0; i<n; i++) {
				node = make_tree(NULL, NULL);
				e.key = node->weight = freq[i];
				e.ptree = node;
				insert_min_heap(&heap, e);
		}

		for (i = 1; i<n; i++) {
				// Delete two nodes with minimum values
				e1 = delete_min_heap(&heap);
				e2 = delete_min_heap(&heap);

				// Merge two nodes
				x = make_tree(e1.ptree, e2.ptree);
				e.key = x->weight = e1.key + e2.key;
				e.ptree = x;
				insert_min_heap(&heap, e);
		}
		e = delete_min_heap(&heap); // Final Huffman binary tree
		destroy_tree(e.ptree);
}

void main()
{
		int freq[] = { 15, 12, 8, 6, 4 };
		huffman_tree(freq, 5);
}
```