COMP2521 23T3
Priority Queues and Heaps

Kevin Luxa

cs2521@cse.unsw.edu.au

priority queues
binary heaps

Motivation

We have learned about types of collections where items are inserted and then deleted based on insertion order

stack

last in, first out

queue

first in, first out

Motivation

There are applications where
we want to process items
based on priority

Examples:
Huffman coding
Dijkstra's algorithm
Prim's algorithm

Priority Queues

A priority queue is an abstract data type
where each item has an associated priority.

It supports the following operations:

insert
insert an item with an associated priority

delete
delete (and return) the item with the highest priority

peek
get the item with the highest priority, without deleting it

is empty
check if the priority queue is empty

Priority is often given by an integer value.

Depending on the application,
either a large priority value or small priority value
could be taken to mean “high priority”.

Here we’ll take a larger priority value to mean higher priority.

Priority Queues

Interface

```
typedef struct pq *Pq;

/** Creates a new, empty pq */
Pq PqNew(void);

/** Frees memory allocated to a pq */
void PqFree(Pq pq);

/** Adds an item with priority to a pq */
void PqInsert(Pq pq, Item item, int priority);

/** Deletes and returns the item with the highest priority */
Item PqDelete(Pq pq);

/** Returns the item with the highest priority */
Item PqPeek(Pq pq);

/** Returns true if the pq is empty, false otherwise */
bool PqIsEmpty(Pq pq);
```

Priority Queues

Example Usage

```
Pq pq = PqNew();

PqInsert(pq, "alice", 4);
PqInsert(pq, "bob", 3);
PqInsert(pq, "andrew", 30);
PqInsert(pq, "jas", 35);

printf("%s\n", PqDelete(pq)); // jas
printf("%s\n", PqDelete(pq)); // andrew

PqInsert(pq, "jake", 23);
PqInsert(pq, "sasha", 25);

printf("%s\n", PqPeek(pq));   // sasha
printf("%s\n", PqDelete(pq)); // sasha
printf("%s\n", PqDelete(pq)); // jake
printf("%s\n", PqDelete(pq)); // alice
printf("%s\n", PqDelete(pq)); // bob

if (PqIsEmpty(pq)) {
    printf("the queue is empty\n");
}

PqFree(pq);
```

How to implement a priority queue?

unordered array

ordered array

linked list (unordered/ordered)

Priority Queue

Unordered array implementation

unordered array

[0]	[1]	[2]	[3]	[4]	[5]
alice 4	bob 3	andrew 30	jas 35	jake 23	sasha 25

Performance?

Insert: $O(1)$

Delete: $O(n)$

Peek: $O(n)$

Is empty: $O(1)$

Priority Queue

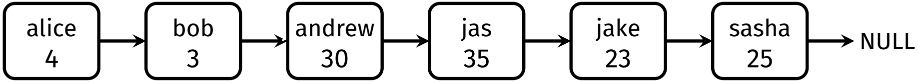
Ordered array implementation

ordered array

[0]	[1]	[2]	[3]	[4]	[5]
bob 3	alice 4	jake 23	sasha 25	andrew 30	jas 35

Performance?
Insert: $O(n)$
Delete: $O(1)$
Peek: $O(1)$
Is empty: $O(1)$

unordered linked list

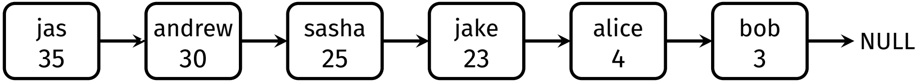


Performance?
Insert: $O(1)$
Delete: $O(n)$
Peek: $O(n)$
Is empty: $O(1)$

Priority Queue

Ordered linked list implementation

ordered linked list



Performance?

Insert: $O(n)$

Delete: $O(1)$

Peek: $O(1)$

Is empty: $O(1)$

Summary (so far)

Data Structure	Insert	Delete	Peek	Is Empty
Unordered array	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Ordered array	$O(n)$	$O(1)$	$O(1)$	$O(1)$
Unordered linked list	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Ordered linked list	$O(n)$	$O(1)$	$O(1)$	$O(1)$

Heaps

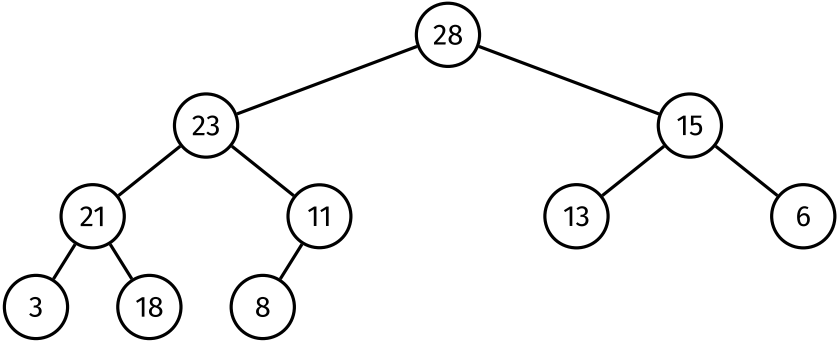
A heap is a tree-based data structure which satisfies the heap property.

The heap property specifies how values in the heap should be ordered, and depends on the kind of heap:

In a max heap, the value in each node must be greater than or equal to the values in its children.

In a min heap, the value in each node must be less than or equal to the values in its children.

Example max heap:



In this lecture we will focus on *max heaps*
(min heaps can be implemented very similarly)

There are many variants of heaps,
for example:

binary heap, binomial heap, Fibonacci heap,
leftist heap, pairing heap, soft heap,
...

We will consider just the binary heap.

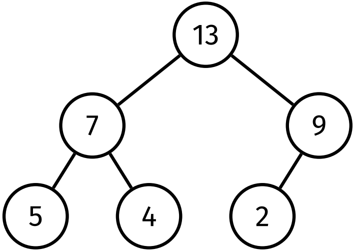
Binary Heaps

A binary heap is a heap that
takes the form of a binary tree,
and satisfies the following properties:

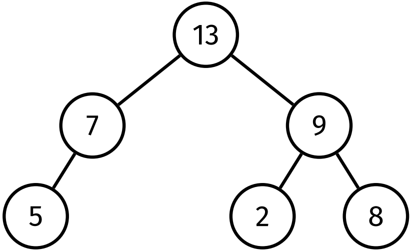
heap property
as defined above

completeness property
all levels of the tree (except possibly the last) must be fully filled
and the last level must be filled from left to right

Binary Heaps



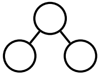
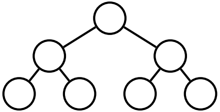
satisfies heap property
satisfies completeness
⇒ is a binary heap



satisfies heap property
does *not* satisfy completeness
⇒ is *not* a binary heap

A result of the completeness property
is that binary heaps always contain $\lfloor \log_2 n \rfloor + 1$ levels
where n is the number of nodes.

This will be relevant for analysis.

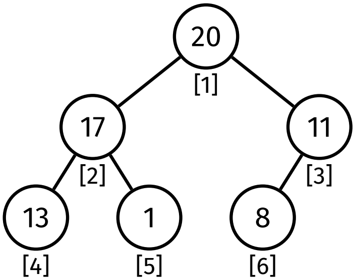
n	number of levels	heap
1	1	
2-3	2	
4-7	3	
...	...	...

Binary Heaps

...as arrays

Heaps are usually implemented with an array.

For a binary heap,
index 1 of the array contains the root item,
the next two indices contain the root's children,
the next four indices contain the children of the root's children,
and so on.



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	20	17	11	13	1	8

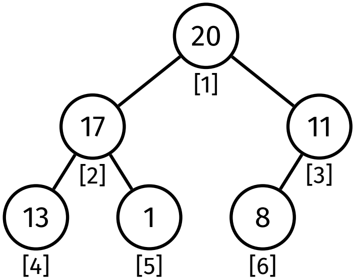
Binary Heaps

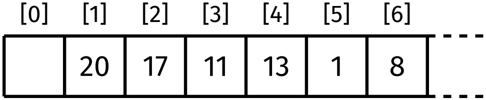
...as arrays

This arrangement gives rise to a useful property:

- For an item at index i:
  - Its left child is located at index $2i$
  - Its right child is located at index $2i + 1$
  - Its parent is located at index $\lfloor i/2 \rfloor$

This makes it efficient to move "up" and "down" the tree.

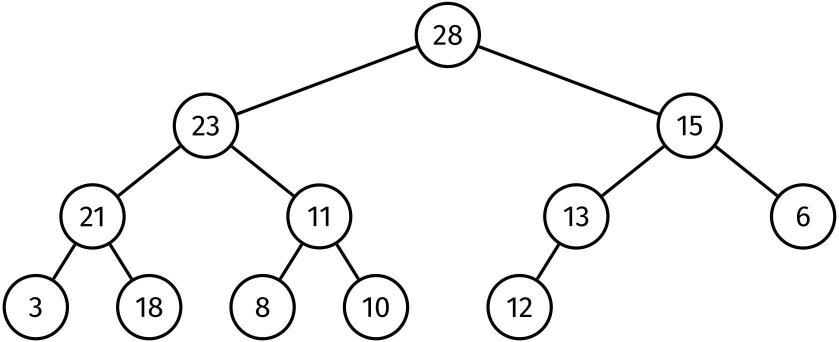




Binary Heaps

...as arrays

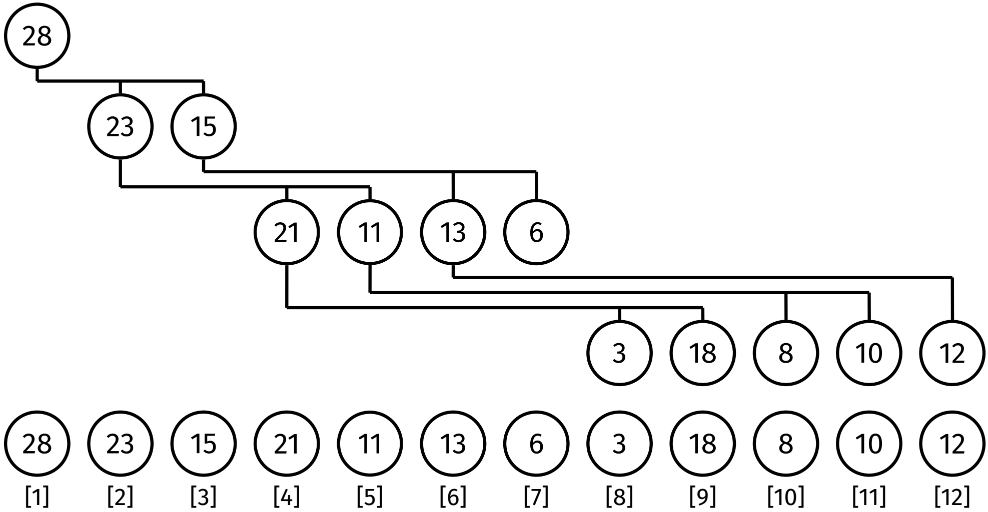
Consider this max heap:



Binary Heaps

...as arrays

The heap as an array:



Assuming integer items:

```
struct heap {
    int *items;
    int  numItems;
    int  capacity;
};
```

Binary Heaps

Constructor

```
struct heap *heapNew(void) {
    struct heap *heap = malloc(sizeof(struct heap));

    heap->numItems = 0;
    heap->capacity = INITIAL_CAPACITY;
    heap->items = malloc((heap->capacity + 1) * sizeof(int));

    return heap;
}
```

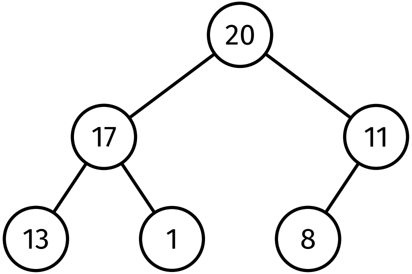
Binary Heap Insertion

Insertion is a two-step process:

1. Add new item at next available position on bottom level
   i.e., after the last item
   - New item may violate the heap property
2. **Fix up**: While new item is greater than its parent (and not at the root), swap with its parent
   - This re-organises items along the path to the root and restores the heap property

Binary Heap Insertion

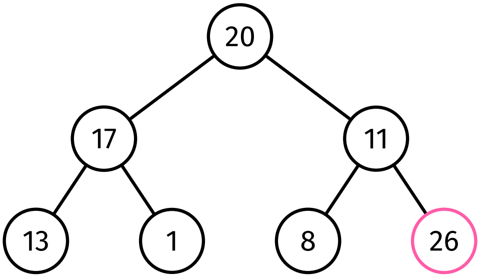
Example: Insert 26



Binary Heap Insertion

Example: Insert 26

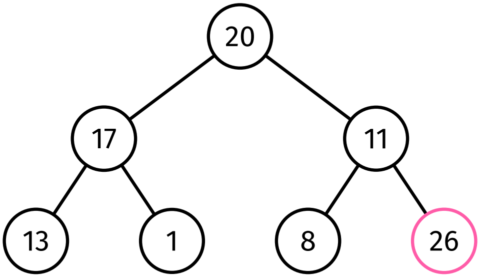
Insert 26 after the last item (8)



Binary Heap Insertion

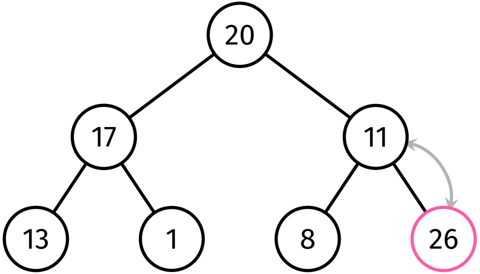
Example: Insert 26

Fix up



Example: Insert 26

Fix up
26 is greater than its parent (11) $\Rightarrow$ swap

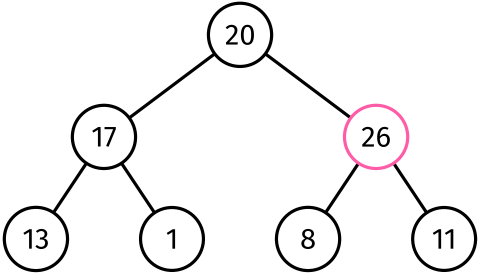


Binary Heap Insertion

Example: Insert 26

Fix up

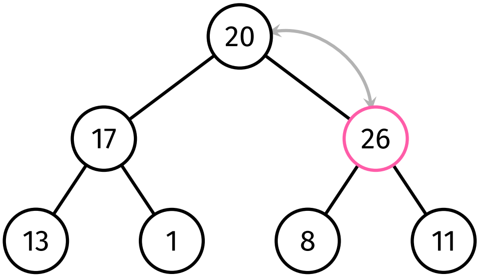
26 is greater than its parent (11) $\Rightarrow$ swap



Binary Heap Insertion

Example: Insert 26

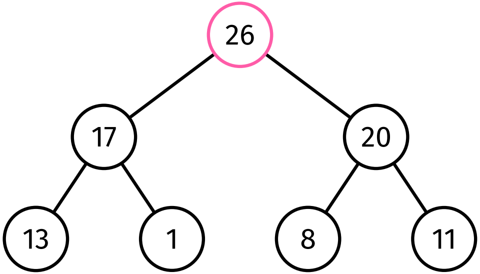
Fix up
26 is greater than its parent (20) $\Rightarrow$ swap



Binary Heap Insertion

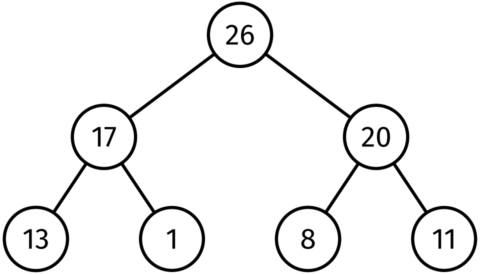
Example: Insert 26

Fix up
26 is greater than its parent (20) $\Rightarrow$ swap



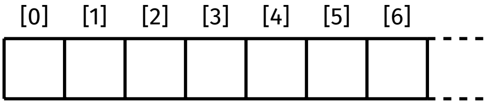
Example: Insert 26

Done



Insert the following items into an initially empty max heap:

17 25 8 6 30 13

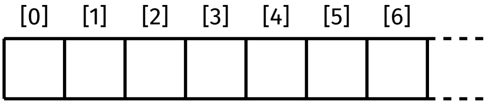


Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

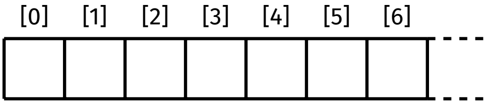


Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

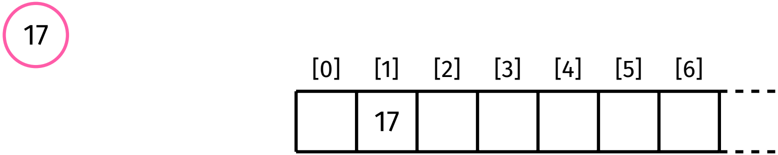
17 25 8 6 30 13



Insert the following items into an initially empty max heap:

17 25 8 6 30 13

Add 17 to the heap



Insert the following items into an initially empty max heap:

17 25 8 6 30 13

17 is at the root - done

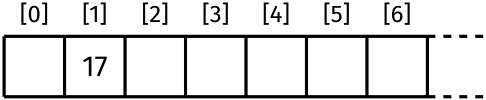
[0]	[1]	[2]	[3]	[4]	[5]	[6]
	17					

Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13



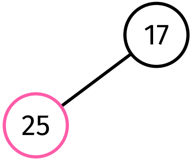
Binary Heap Insertion

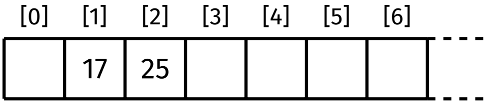
Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

Add 25 after the last item





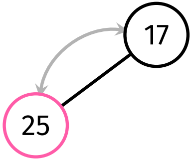
Binary Heap Insertion

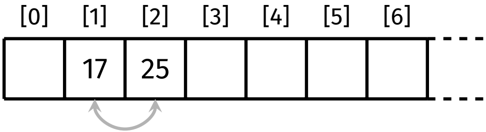
Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

25 is greater than its parent (17) - swap





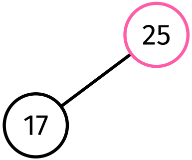
Binary Heap Insertion

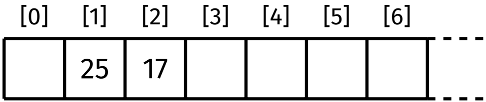
Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

25 is greater than its parent (17) - swap



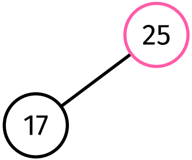


Binary Heap Insertion
Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

25 is at the root - done



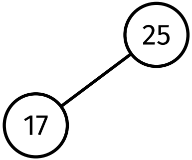
[0]	[1]	[2]	[3]	[4]	[5]	[6]
	25	17				

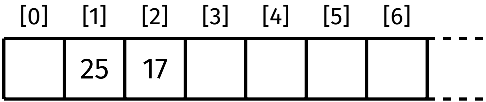
Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13





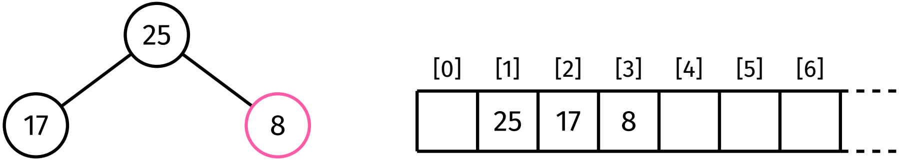
Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

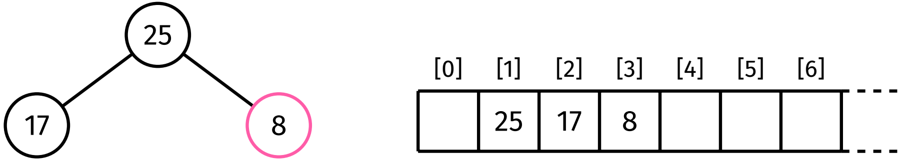
Add 8 after the last item



Insert the following items into an initially empty max heap:

17 25 8 6 30 13

8 is not greater than its parent (25) - done

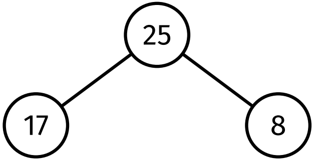


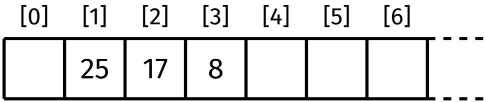
Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13





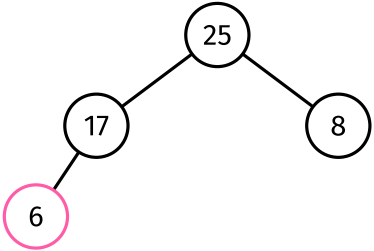
Binary Heap Insertion

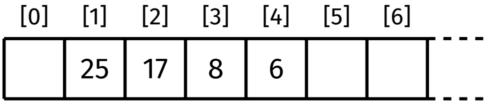
Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

Add 6 after the last item





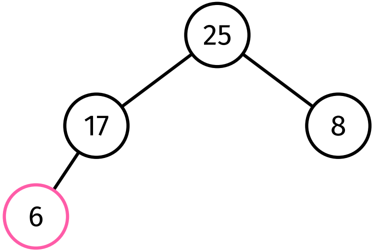
Binary Heap Insertion

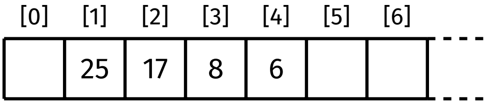
Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

6 is not greater than its parent (17) - done



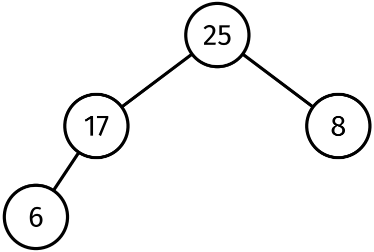


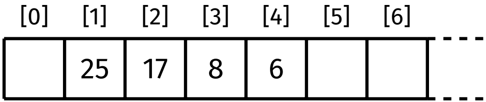
Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13





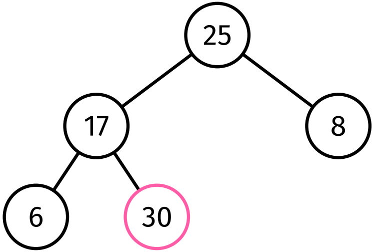
Binary Heap Insertion

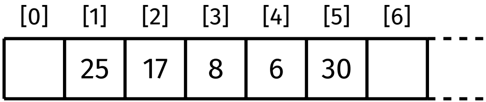
Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

Add 30 after the last item





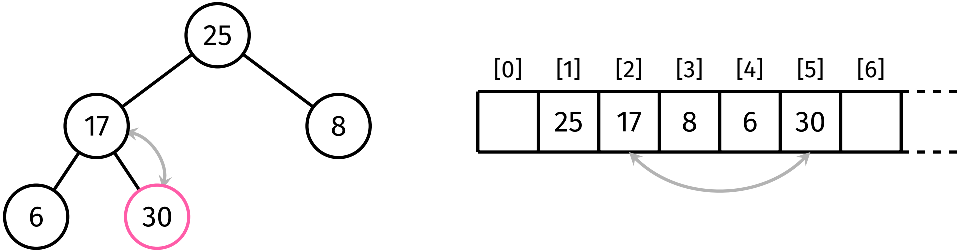
Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

30 is greater than its parent (17) - swap



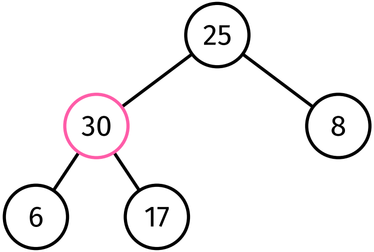
Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

30 is greater than its parent (17) - swap

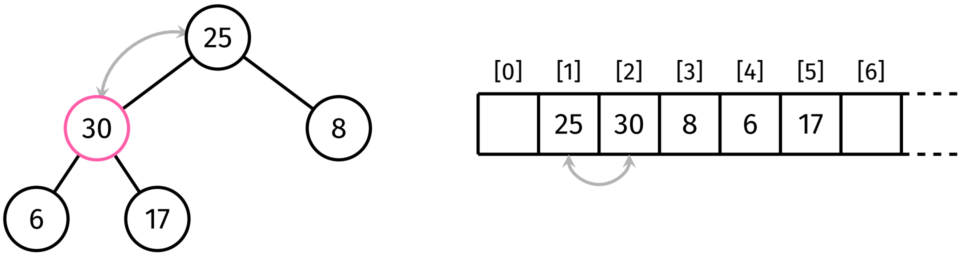


[0]	[1]	[2]	[3]	[4]	[5]	[6]
	25	30	8	6	17	

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

30 is greater than its parent (25) - swap



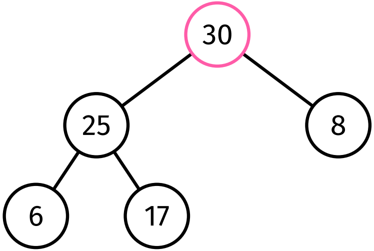
Binary Heap Insertion

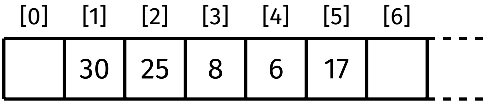
Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

30 is greater than its parent (25) - swap





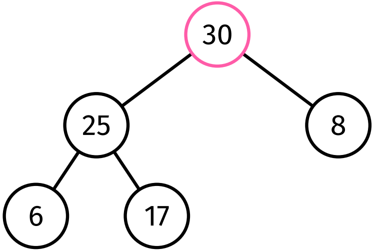
Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

30 is at the root - done



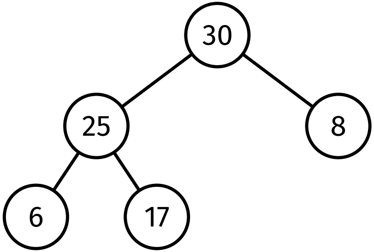
[0]	[1]	[2]	[3]	[4]	[5]	[6]
	30	25	8	6	17	

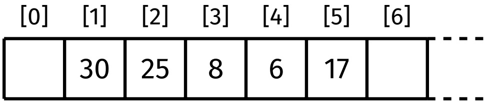
Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13





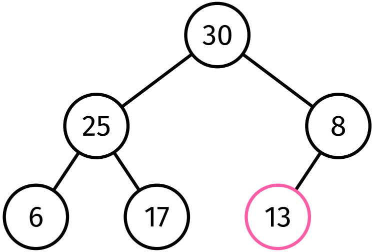
Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

Add 13 after the last item



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	30	25	8	6	17	13

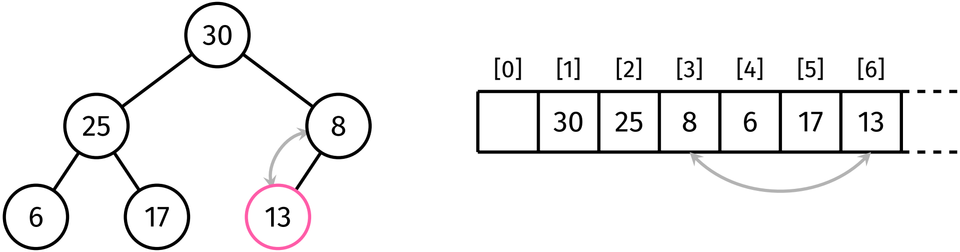
Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

13 is greater than its parent (8) - swap



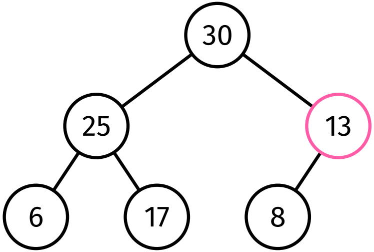
Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

13 is greater than its parent (8) - swap



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	30	25	13	6	17	8

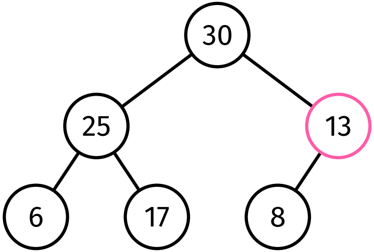
Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13

13 is not greater than its parent (30) - done



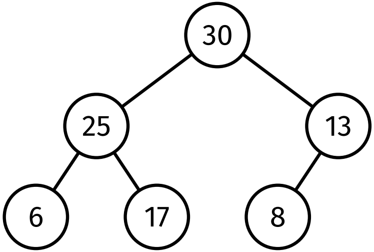
[0]	[1]	[2]	[3]	[4]	[5]	[6]
	30	25	13	6	17	8

Binary Heap Insertion

Example

Insert the following items into an initially empty max heap:

17 25 8 6 30 13



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	30	25	13	6	17	8

Binary Heap Insertion

Implementation


```c
void heapInsert(struct heap *heap, Item it) {
    if (heap->numItems == heap->capacity) {
        // resize
    }
    heap->numItems++;
    heap->items[heap->numItems] = it;
    fixUp(heap->items, heap->numItems);
}

void fixUp(Item items[], int i) {
    // while index i is not the root and
    // item at index i is greater than its parent
    while (i > 1 && items[i] > items[i / 2]) {
        swap(items, i, i / 2);
        i = i / 2;
    }
}
```

Binary Heap Insertion

Analysis

Cost of insertion:

- Add new item after last item $\Rightarrow O(1)$
- Fix up considers one item on each level in the worst case
- Heap is a complete tree $\Rightarrow O(\log n)$ levels
- Therefore, worst-case time complexity is $O(\log n)$

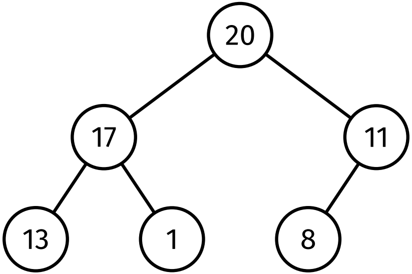
Binary Heap Deletion

Deletion is a three-step process:

1. Replace root item with last item
   - Last item = bottom-most, rightmost item
   - Let this item be i
2. Remove last item
3. Fix down: While i is less than its greater child, swap it with its greater child
   - This restores the heap property

Binary Heap Deletion

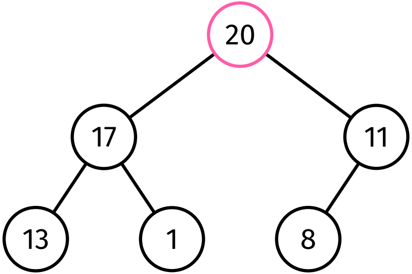
Example: Delete from this max heap



Binary Heap Deletion

Example: Delete from this max heap

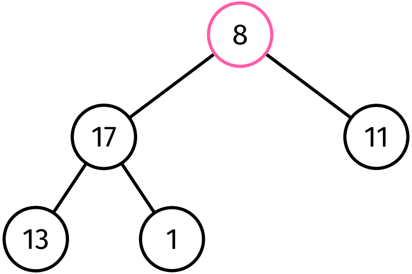
Delete 20, replace with 8



Binary Heap Deletion

Example: Delete from this max heap

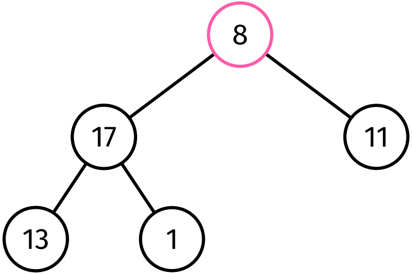
Delete 20, replace with 8



Binary Heap Deletion

Example: Delete from this max heap

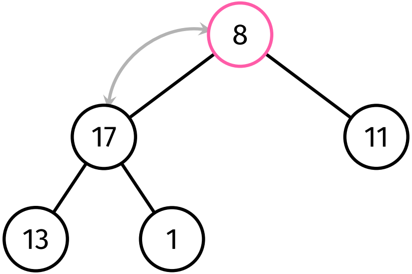
Fix down



Binary Heap Deletion

Example: Delete from this max heap

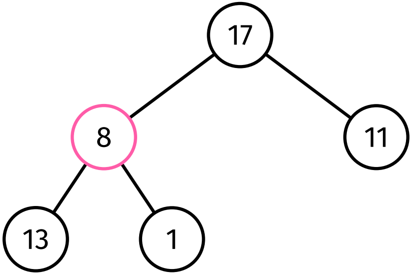
Fix down
8 is less than its greater child (17) $\Rightarrow$ swap



Binary Heap Deletion

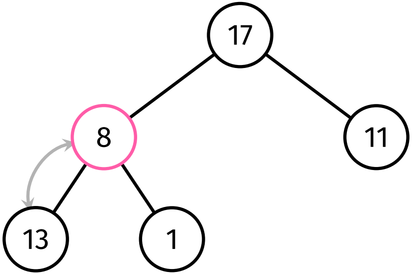
Example: Delete from this max heap

Fix down
8 is less than its greater child (17) $\Rightarrow$ swap



Example: Delete from this max heap

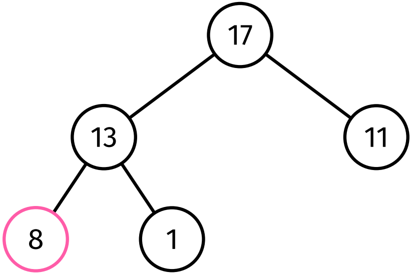
Fix down
8 is less than its greater child (13) $\Rightarrow$ swap



Binary Heap Deletion

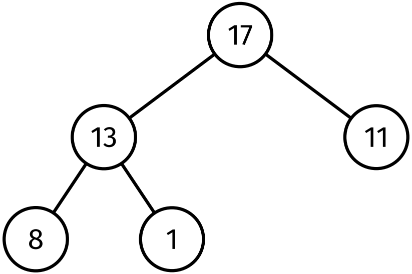
Example: Delete from this max heap

Fix down
8 is less than its greater child (13) $\Rightarrow$ swap



Example: Delete from this max heap

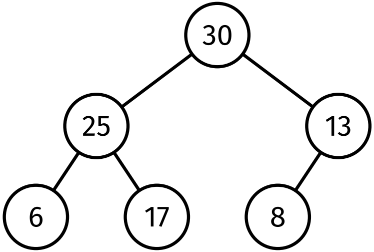
Done



Binary Heap Deletion

Example

Delete from the following max heap until it is empty:



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	30	25	13	6	17	8

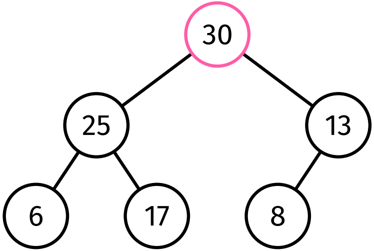
Binary Heap Deletion

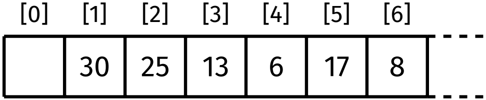
Example

Delete from the following max heap until it is empty:

30

Deleting 30





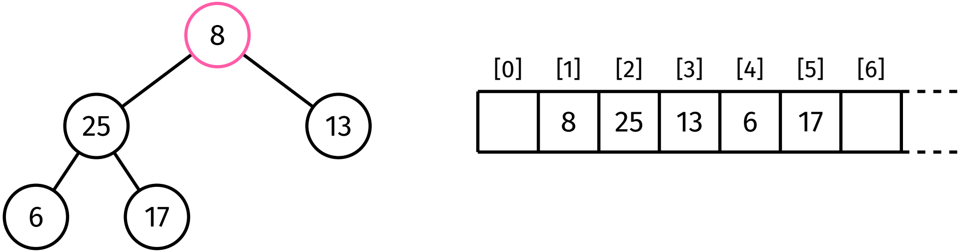
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30

Replace 30 with last item (8)



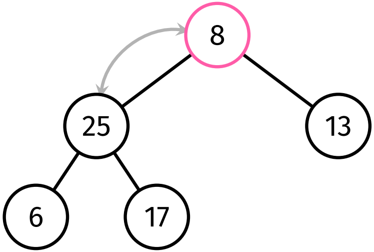
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30

8 is less than its greater child (25) - swap



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	8	25	13	6	17	

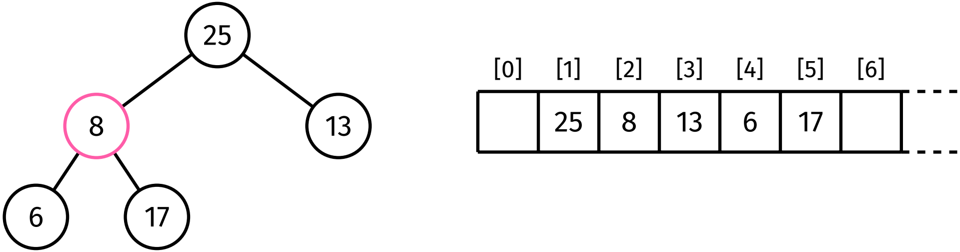
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30

8 is less than its greater child (25) - swap



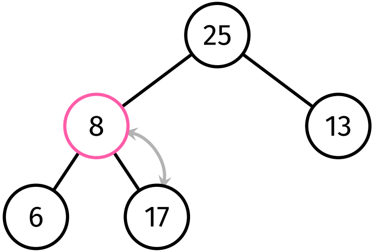
Binary Heap Deletion

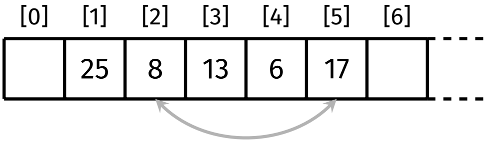
Example

Delete from the following max heap until it is empty:

30

8 is less than its greater child (17) - swap





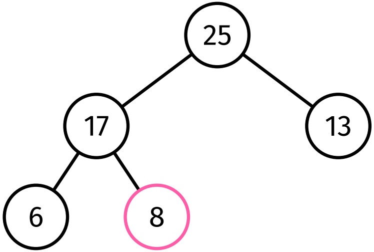
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30

8 is less than its greater child (17) - swap



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	25	17	13	6	8	

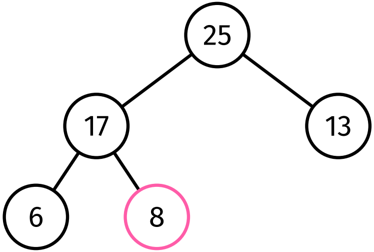
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30

8 is at a leaf - done



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	25	17	13	6	8	

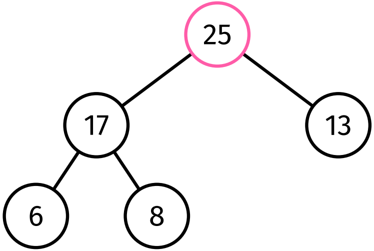
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25

Deleting 25



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	25	17	13	6	8	

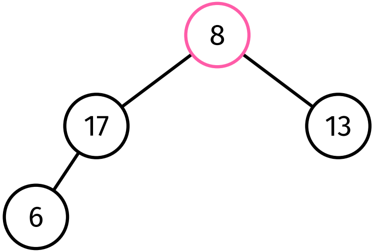
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25

Replace 25 with last item (8)

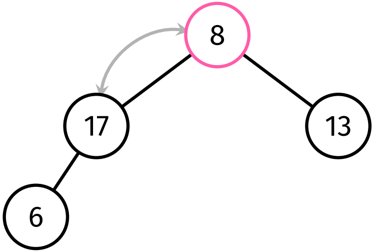


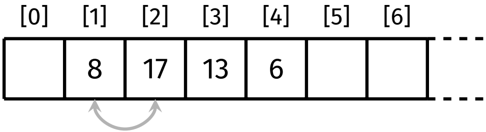
[0]	[1]	[2]	[3]	[4]	[5]	[6]
	8	17	13	6		

Delete from the following max heap until it is empty:

30 25

8 is less than its greater child (17) - swap

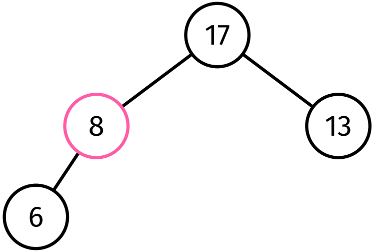


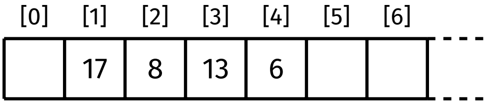


Delete from the following max heap until it is empty:

30 25

8 is less than its greater child (17) - swap





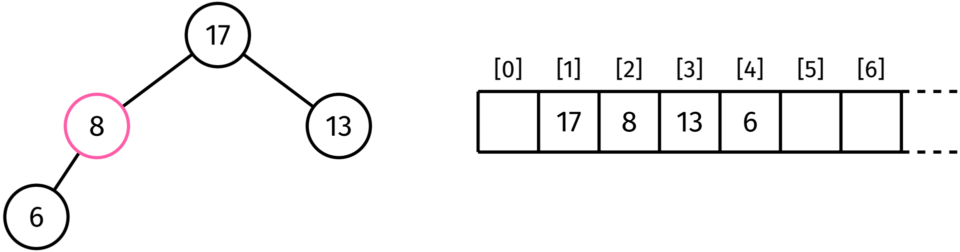
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25

8 is not less than its greater child (6) - done



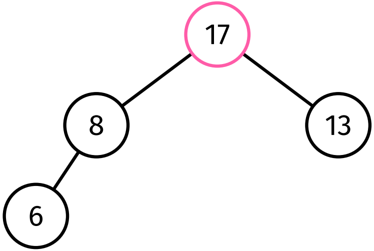
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25 17

Deleting 17



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	17	8	13	6		

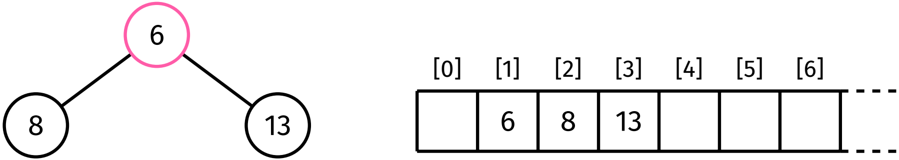
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25 17

Replace 17 with last item (6)



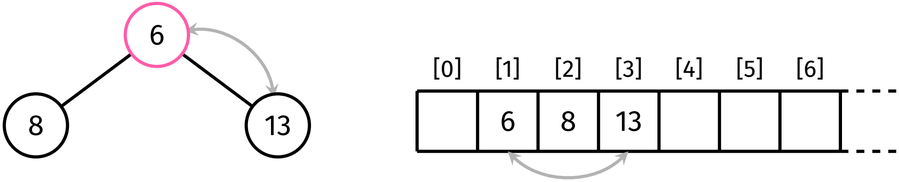
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25 17

6 is less than its greater child (13) - swap



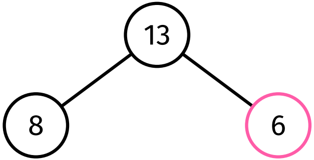
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25 17

6 is less than its greater child (13) - swap



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	13	8	6			

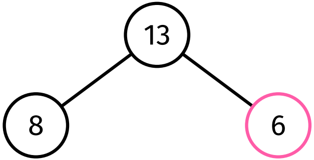
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25 17

6 is at a leaf - done



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	13	8	6			

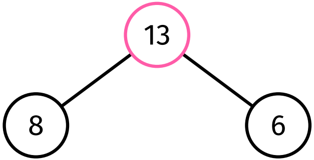
Binary Heap Deletion

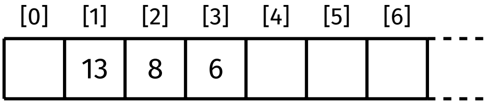
Example

Delete from the following max heap until it is empty:

30 25 17 13

Deleting 13





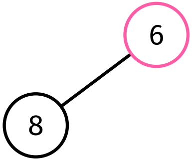
Binary Heap Deletion

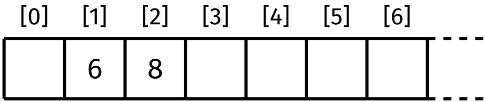
Example

Delete from the following max heap until it is empty:

30 25 17 13

Replace 13 with last item (6)

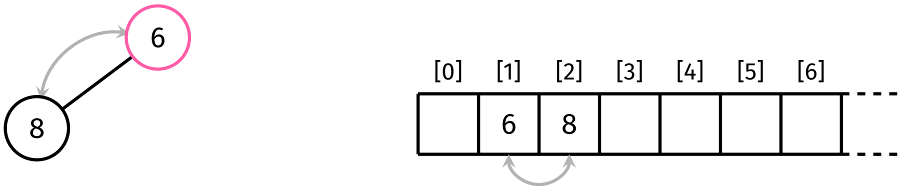




Delete from the following max heap until it is empty:

30 25 17 13

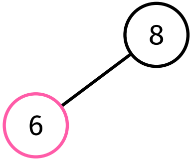
6 is less than its greater child (8) - swap



Delete from the following max heap until it is empty:

30 25 17 13

6 is less than its greater child (8) - swap



[0]	[1]	[2]	[3]	[4]	[5]	[6]
	8	6				

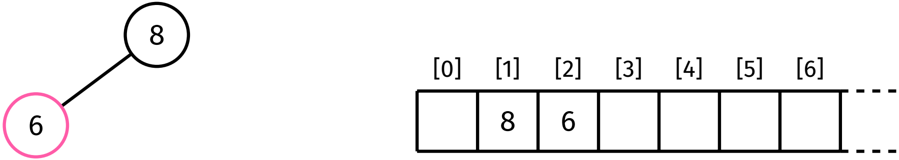
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25 17 13

6 is at a leaf - done



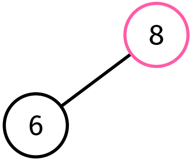
Binary Heap Deletion

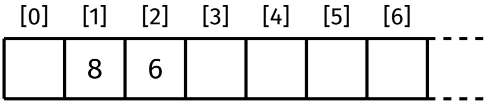
Example

Delete from the following max heap until it is empty:

30 25 17 13 8

Deleting 8





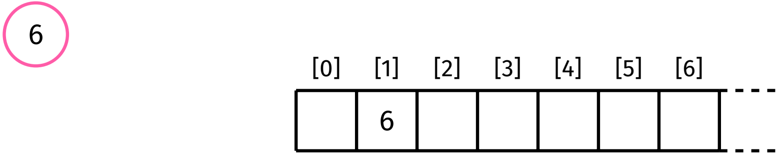
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25 17 13 8

Replace 8 with last item (6)



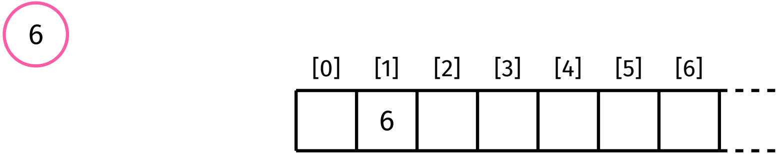
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25 17 13 8

6 is at a leaf - done



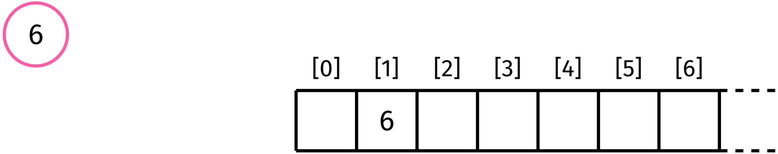
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25 17 13 8 6

Deleting 6



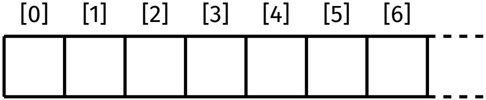
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25 17 13 8 6

Delete 6



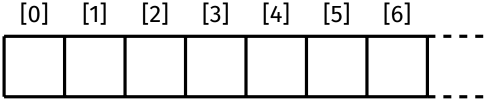
Binary Heap Deletion

Example

Delete from the following max heap until it is empty:

30 25 17 13 8 6

Heap is now empty



Binary Heap Deletion

Implementation (I)

```
Item heapDelete(struct heap *heap) {
    Item item = heap->items[1];
    heap->items[1] = heap->items[heap->numItems];
    heap->numItems--;
    fixDown(heap->items, 1, heap->numItems);
    return item;
}
```

Binary Heap Deletion

Implementation (II)


```
void fixDown(Item items[], int i, int N) {
    // while index i has at least one child
    while (2 * i <= N) {
        // let j be the index of index i's left child
        int j = 2 * i;

        // if index i's right child is greater than its left child
        if (j < N && items[j] < items[j + 1]) j++;

        // if the item at index i is greater than or equal to both children
        if (items[i] >= items[j]) break;

        swap(items, i, j);

        // move one level down the heap
        i = j;
    }
}
```

Binary Heap Deletion

Analysis

Cost of deletion:

- Replace root by item at end of array $\Rightarrow O(1)$
- Fix down considers two items on each level in the worst case
- Heap is a complete tree $\Rightarrow O(\log n)$ levels
- Therefore, worst-case time complexity is $O(\log n)$

PQ Implementation

Concrete data structures


```
struct pq {
    struct pqItem *items; // array of items
    int numItems;         // number of items stored
    int capacity;         // max number of items
};

struct pqItem {
    Item item;
    int priority;
};
```

PQ Implementation

Constructor


```
Pq PqNew(void) {
    Pq pq = malloc(sizeof(struct pq));

    pq->numItems = 0;
    pq->capacity = INITIAL_CAPACITY;
    pq->items = malloc((pq->capacity + 1) * sizeof(struct pqItem));
    return pq;
}
```

PQ Implementation

Insertion

```
void PqInsert(Pq pq, Item it, int priority) {
    if (pq->numItems == pq->capacity) {
        // resize array
    }

    pq->numItems++;
    pq->items[pq->numItems] = (struct pqItem){it, priority};
    fixUp(pq->items, pq->numItems);
}

void fixUp(struct pqItem items[], int i) {
    while (i > 1 && items[i].priority > items[i / 2].priority) {
        swap(items, i, i / 2);
        i = i / 2;
    }
}
```

PQ Implementation

Deletion


```c
Item PqDelete(Pq pq) {
    Item item = pq->items[1].item;
    pq->items[1] = pq->items[pq->numItems];
    pq->numItems--;
    fixDown(pq->items, 1, pq->numItems);
    return item;
}

void fixDown(struct pqItem items[], int i, int N) {
    while (2 * i <= N) {
        int j = 2 * i;
        if (j < N && items[j].priority < items[j + 1].priority) j++;
        if (items[i].priority >= items[j].priority) break;
        swap(items, i, j);
        i = j;
    }
}
```

Priority Queue ADT Summary

Data Structure	Insert	Delete	Peek	Is Empty
Unordered array	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Ordered array	$O(n)$	$O(1)$	$O(1)$	$O(1)$
Unordered linked list	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Ordered linked list	$O(n)$	$O(1)$	$O(1)$	$O(1)$
Binary heap	$O(\log n)$	$O(\log n)$	$O(1)$	$O(1)$

Feedback

https://forms.office.com/r/aPF09YHZ3X