Data Structures and Algorithms

Mohammad GANJTABESH

Department of Computer Science,
School of Mathematics, Statistics and Computer Science,
University of Tehran,
Tehran, Iran.

mgtabesh@ut.ac.ir

Dept. of Computer Science
University of Tehran

Priority Queue (Heap)

min - heap

max - heap

Data Structures and Algorithms
Undergraduate course

Mohammad GANJTABESH
mgtabesh@ut.ac.ir

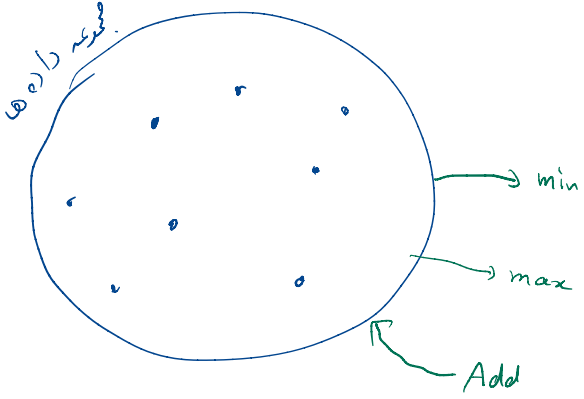
Dept. of Computer Science
University of Tehran

Motivations

مهمه (اولویت بالاتر)

P_1 P_2 ... P_n

10 2 100



CPU

Operations and Naive Implementations

	Add	Remove	Maintain
Array	$O(1)$	$O(n)$	$O(n)$
SortedArray	$O(n)$	$O(1)$	$O(n)$
Linked List	$O(1)$	$O(n)$	~~$O(1)$~~
Sorted L.L.	$O(n)$	$O(1)$	~~$O(1)$~~

Tree Implementation: Rules

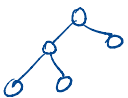
Heap is a complete Binary tree in which

- min-heap : parent < child (recursively)
- max-heap : parent > child (")

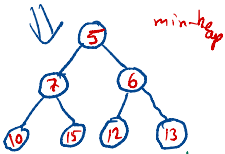
Complete :
- Every non-leaf node has two children except the last level.
- The last level is filled left-to-right.

Full :
- Every non-leaf node has two children
- All the leaves are at the same level.

Full X
Complete X

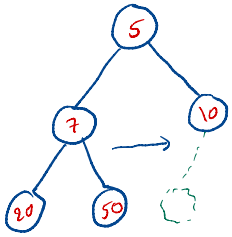


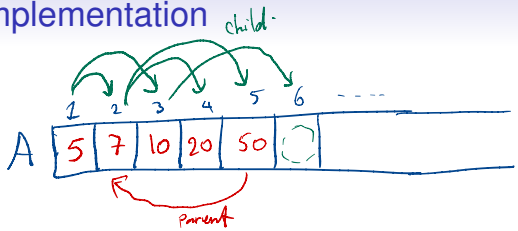
Full X
complete ✓



Full & complete

Tree Implementation





$$Parent(i) = \lfloor i/2 \rfloor$$

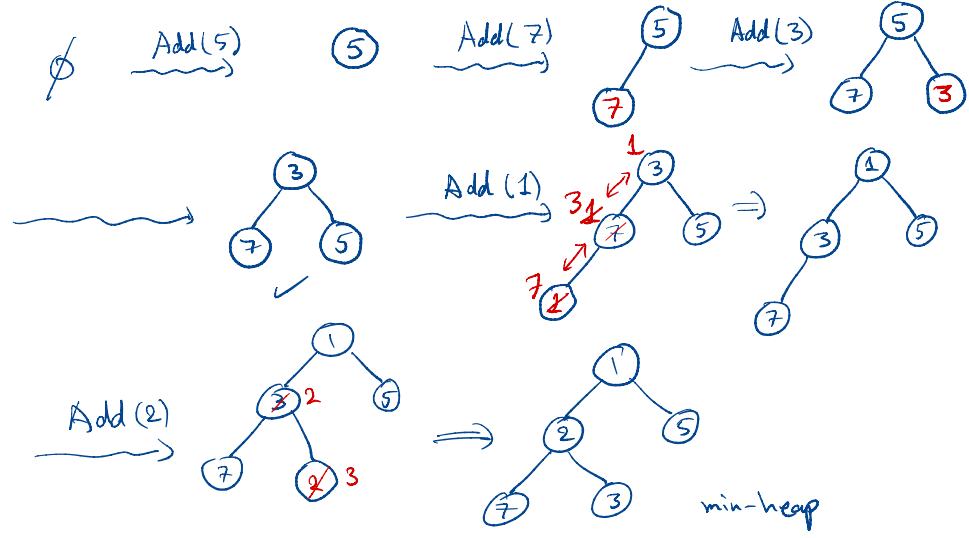
$$left\text{-}child(i) = 2i$$

$$right\text{-}child(i) = 2i + 1$$

Tree Implementation

Add to
Ex. min-heap.

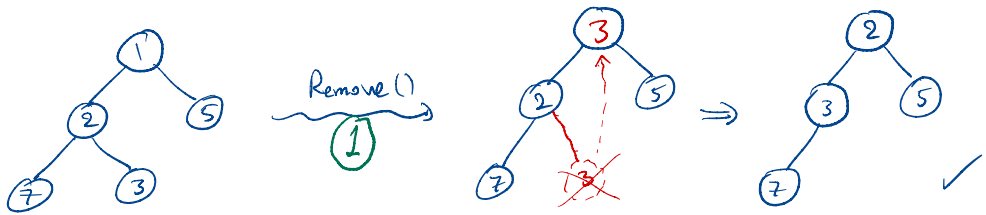
$O(\log n)$ ✓

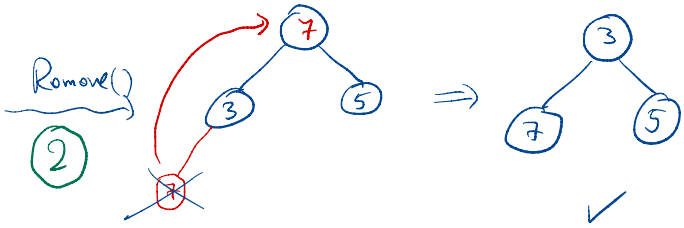


Tree Implementation

Ex. Remove from min-heap

$O(\log n)$





Tree Implementation

```
class Min-Heap {
    int *Heap;
    int capacity;
    int lastposition;
    Min-Heap (int n){
        Heap ← new int[n];
        capacity ← n;
        lastposition ← -1;
    }
    // methods
}
```

```
public Add (x) {
    lastposition ++;
    if (lastposition = capacity)
        // exception ("Array is full")
    Heap [lastposition] ← x;
    Fix Up (lastposition);
}

private Fix Up (position) {
    if (position = 0) return;
    parent ← ⌊(position -1)/2⌋;
    if (Heap [position] < Heap [parent])
        swap (Heap [position], Heap [Parent]);
    Fix Up (parent);
}
```

Tree Implementation

Public

```
Remove ( ) {
    if (lastposition < 0)
        // exception ("Array is empty!");
    min ← Heap [0];
    Heap [0] ← Heap [lastposition];
    lastposition --;
    FixDown (0);
    return (min);
}
```

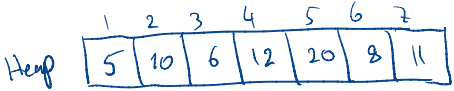
swap (Heap [0], Heap [lastposition]);

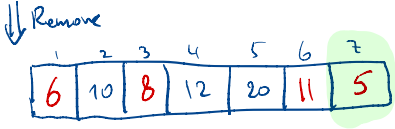
Leads us to have heap-sort.

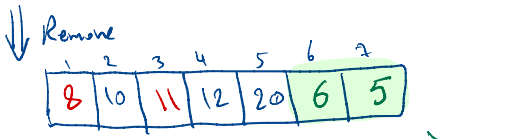
Private

```
FixDown ( position ) {
    lc ← 2*position + 1
    rc ← 2*position + 2
    if (lc = lastposition & Heap [lc] < Heap [position])
        swap ( Heap [lc], Heap [position] );
        return ;
    }
    if (rc = lastposition & Heap[rc] < Heap[position])
        swap (Heap [rc], Heap [position]);
        return ;
    }
    if( rc > lastposition & lc > lastposition)
        return ;
    if (Heap [lc] > Heap [rc] &
                    Heap (position) > Heap (rc))
        swap (Heap [rc], Heap [position]);
    } FixDown (rc);
    else if (Heap [lc] < Heap [position]) {
        swap ( _ _ );
    } } FixDown (lc);
```

Tree Implementation







Heap-Sort