



Department of Computer Science and Engineering

Data Structures and Object-Oriented Design

(CSE – 2050)

Hasan Baig

Office: UConn (Stamford), 305C
email: hasan.baig@uconn.edu

1

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Recalling Queues...

2

- Queues → First-In First-Out (FIFO)
 - The object which goes in first will be the first one to come out
- Example: Call center

Hasan Baig



2

Module 10

Priority Queues and Heaps

3

CSE-2050 – Data Structures and Object-Oriented Design


Priority Queues and Heaps

Priority Queues

4

- A priority queue stores a collection of items in a (key, value) pair
 - Key → defines the priority
 - Value → the actual data
- A priority queue may have multiple entries with same keys,
 - return an arbitrary choice of item having minimum key.
 - Values may be any type of object.

Hasan Baig



4

CSE-2050 – Data Structures and Object-Oriented Design


Priority Queues and Heaps

Priority Queues ADT

5

- `insert(key, value)` → insert an item (value) with key (priority)
- `findmin()` → Returns (but does not remove) the item with minimum priority
 - Same priority elements are chosen randomly
- `removemin()` → Remove and returns the item with minimum priority
 - Same priority elements are chosen randomly

Hasan Baig



5

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Priority Queues Implementation

6

Unsorted List


Sorted List

What will be the time complexity of the following methods on above data?

<code>insert(key, value)</code> → $O(1)$	<code>insert(key, value)</code> → $O(n)$
<code>findmin()</code> → $O(n)$	<code>findmin()</code> → $O(1)$
<code>removemin()</code> → $O(n)$	<code>removemin()</code> → $O(n)$

Can we improve this further?

Hasan Baig



6

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Priority Queues Implementation

7

Unsorted List

Sorted List

What will be the time complexity of the following methods on above data?

insert(key, value) → O(1)	insert(key, value) → O(n)
findmin() → O(n)	findmin() → O(1)
removemin() → O(n)	removemin() → O(1)

By storing the data in reverse order

Hasan Baig

7

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Priority Queues Implementation

8

Unsorted List

Sorted List

```
class Entry:
    def __init__(self, item, priority):
        self.item = item
        self.priority = priority

    def __lt__(self, other):
        return self.priority < other.priority
```

```
class UnsortedPriorityQueue:
    def __init__(self):
        self._entries = []

    def insert(self, item, priority):
        self._entries.append(Entry(item, priority))

    def findmin(self):
        return min(self._entries).item

    def removemin(self):
        entry = min(self._entries)
        self._entries.remove(entry)
        return entry.item
```

```
class SortedPriorityQueue:
    def __init__(self):
        self._entries = []

    def insert(self, item, priority):
        self._entries.append(Entry(item, priority))
        self._entries.sort(reverse = True)

    def findmin(self):
        return self._entries[-1].item

    def removemin(self):
        return self._entries.pop().item
```

Hasan Baig

8


CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Priority Queues ADT 9

- Tradeoff in implementations
 - Fast `insert()` BUT slow `removeMin()`
 - Slow `insert()` BUT fast `removeMin()`
- Come up with an efficient implementation of Priority Queues

Hasan Baig



9


CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps 10

- **Heaps** are data structures that are used to implement priority queues (ADT)
- Most common implementation of heap → binary tree
- Standard term used for *priority* is “key”
 - Unlike **maps** data structure, keys (priority) in **heaps** can be same
- Heaps are **complete** data structure
 - BST → each node has left and right child
 - Construct heap from left to right across each row
 - Last row may not be fully completed

Hasan Baig



10

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps


Heaps Types 11

1. MAX HEAP
In a **max heap** the keys of parent nodes are always greater than or equal to those of the children. The highest key (max value) is in the root node.

2. MIN HEAP
In a **min heap** the keys of parent nodes are less than or equal to those of the children and the lowest key (min item) is in the root node

Highest priority element (no matter if it is defined as min/max) should be always placed above its children. That is at the top!

Hasan Baig



11

CSE-2050 – Data Structures and Object-Oriented Design


Priority Queues and Heaps

Heaps Properties 12


1. Completeness

Heap is constructed from left to right across each row

- The last row may not be fully completed



Hasan Baig



12

CSE-2050 – Data Structures and Object-Oriented Design

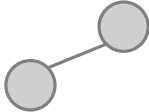
Priority Queues and Heaps

Heaps Properties

1. Completeness

Heap is constructed from left to right across each row

- The last row may not be fully completed



Hasan Baig

13

13

CSE-2050 – Data Structures and Object-Oriented Design

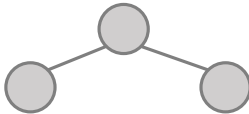
Priority Queues and Heaps

Heaps Properties

1. Completeness

Heap is constructed from left to right across each row

- The last row may not be fully completed



Hasan Baig

14

14

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

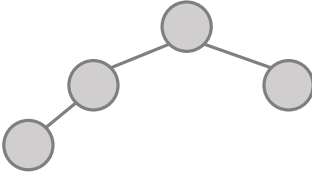
15

Heaps Properties

1. Completeness

Heap is constructed from left to right across each row

- The last row may not be fully completed



Hasan Baig

15

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

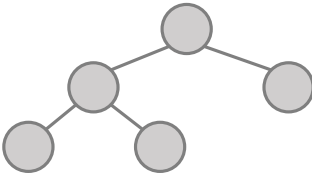
16

Heaps Properties

1. Completeness

Heap is constructed from left to right across each row

- The last row may not be fully completed



Hasan Baig

16

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

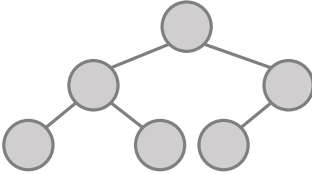
Heaps Properties

1. Completeness

17

Heap is constructed from left to right across each row

- The last row may not be fully completed



Hasan Baig

17

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

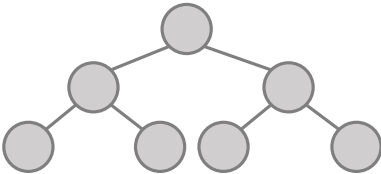
Heaps Properties

1. Completeness

18

Heap is constructed from left to right across each row

- The last row may not be fully completed



Hasan Baig

18

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Properties

1. Completeness

Heap is constructed from left to right across each row

- The last row may not be fully completed

Hasan Baig

19

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Properties

1. Completeness

Heap is constructed from left to right across each row

- The last row may not be fully completed

Becomes invalid Heap because the property is violated

Hasan Baig

20

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Properties

2. Heap Property

Every node can have 2 children, so heaps are almost-complete binary trees.

min heap: the parent node is always smaller than the child nodes (left and right nodes)

max heap: the parent node is always greater than the child nodes (left and right nodes)

Hasan Baig

21

21

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Properties

2. Heap Property

Max Heap: The root node of a max heap is the **largest item** in the data structure.

Min Heap: The root node of a min heap is the **smallest item** in the data structure.

```

graph TD
    subgraph Max_Heap [Max Heap]
        45((45)) --- 34((34))
        45 --- 12((12))
        34 --- 18((18))
        34 --- 9((9))
        18 --- 11((11))
        12 --- 1((1))
        12 --- 2((2))
    end
    subgraph Min_Heap [Min Heap]
        5((5)) --- 12_2((12))
        5 --- 8((8))
        12_2 --- 18_2((18))
        12_2 --- 34_2((34))
        18_2 --- 98((98))
        8 --- 9_2((9))
        8 --- 22((22))
    end
  
```

Hasan Baig

22

22

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Representation

23

Heaps can be represented in 1-D form

Consider an example of a Max Heap

Hasan Baig

23

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Representation

24

Heaps can be represented in 1-D form

- Each element can be given an index value

Any node placed at index i has:

- Left child placed at index $2i + 1$
- Right child placed at index $2i + 2$

0	
1	
2	
3	
4	
5	
6	
7	

Hasan Baig

24

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Representation

Heaps can be represented in 1-D form

- Each element can be given an index value

Any node placed at index i has:

- Left child placed at index $2i + 1$
- Right child placed at index $2i + 2$
- Parent of the node $\rightarrow (i - 1) // 2$

0	45
1	34
2	12
3	18
4	9
5	1
6	2
7	11

Hasan Baig

25

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

Insert(23)

0	23
1	
2	
3	
4	
5	
6	
7	

Hasan Baig

26

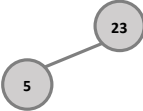
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

27

Insert(5)



```

graph TD
    23((23)) --- 5((5))
  
```

0	23
1	5
2	
3	
4	
5	
6	
7	

Hasan Baig

27

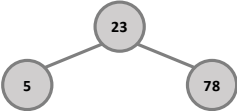
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

28

Insert(78)



```

graph TD
    23((23)) --- 5((5))
    23 --- 78((78))
  
```

0	23
1	5
2	78
3	
4	
5	
6	
7	

Hasan Baig

28

CSE-2050 – Data Structures and Object-Oriented Design

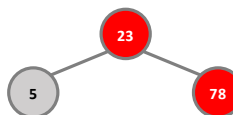
Priority Queues and Heaps

Heaps Max Heap Construction

29

Insert(78)

Violating max heap property



0	23
1	5
2	78
3	
4	
5	
6	
7	

Hasan Baig

29

CSE-2050 – Data Structures and Object-Oriented Design

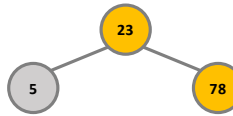
Priority Queues and Heaps

Heaps Max Heap Construction

30

Insert(78)

Violating max heap property



0	23
1	5
2	78
3	
4	
5	
6	
7	

Hasan Baig

30

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

31

```

graph TD
    78((78)) --- 5((5))
    78 --- 23((23))
  
```

0	78
1	5
2	23
3	
4	
5	
6	
7	

Hasan Baig

31

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

32

Insert(2)

```

graph TD
    78((78)) --- 5((5))
    78 --- 23((23))
  
```

0	78
1	5
2	23
3	
4	
5	
6	
7	

Hasan Baig

32

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

33

Insert(2)

0	78
1	5
2	23
3	2
4	
5	
6	
7	

Hasan Baig

33

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

34

0	78
1	5
2	23
3	2
4	
5	
6	
7	

Hasan Baig

34

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

35

Insert(92)

```

graph TD
    78((78)) --- 5((5))
    78 --- 23((23))
    5 --- 2((2))
    5 --- 92((92))
  
```

0	78
1	5
2	23
3	2
4	
5	
6	
7	

Hasan Baig

35

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

36

Insert(92)

```

graph TD
    78((78)) --- 5((5))
    78 --- 23((23))
    5 --- 2((2))
    5 --- 92((92))
  
```

0	78
1	5
2	23
3	2
4	92
5	
6	
7	

Hasan Baig

36

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

37

Insert(92)

Violating max heap property

0	78
1	5
2	23
3	2
4	92
5	
6	
7	

Hasan Baig

37

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

38

Insert(92)

Violating max heap property

0	78
1	5
2	23
3	2
4	92
5	
6	
7	

Hasan Baig

38

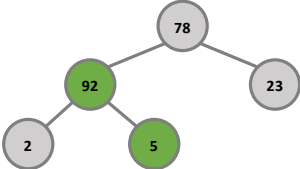
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

39

Insert(92)



0	78
1	92
2	23
3	2
4	5
5	
6	
7	

Hasan Baig

39

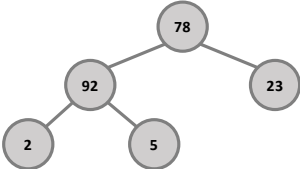
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

40

Insert(92)



0	78
1	92
2	23
3	2
4	5
5	
6	
7	

Hasan Baig

40

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

41

Insert(92)

Violating max heap property

0	78
1	92
2	23
3	2
4	5
5	
6	
7	

Hasan Baig

41

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

42

Insert(92)

Violating max heap property

0	78
1	92
2	23
3	2
4	5
5	
6	
7	

Hasan Baig

42

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

43

Insert(92)

0	92
1	78
2	23
3	2
4	5
5	
6	
7	

Hasan Baig

43

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

44

0	92
1	78
2	23
3	2
4	5
5	
6	
7	

Hasan Baig

44

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

45

Insert(12)

0	92
1	78
2	23
3	2
4	5
5	
6	
7	

Hasan Baig

45

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

46

Insert(12)

0	92
1	78
2	23
3	2
4	5
5	12
6	
7	

Hasan Baig

46

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

47

0	92
1	78
2	23
3	2
4	5
5	12
6	
7	

Hasan Baig

47

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

48

Insert(21)

0	92
1	78
2	23
3	2
4	5
5	12
6	
7	

Hasan Baig

48

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

49

Insert(21)

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

Hasan Baig

49

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

50

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

Hasan Baig

50

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

51

Insert(99)

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

Hasan Baig

51

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

52

Insert(99)

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	99

Hasan Baig

52

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

53

Insert(99)

Violating max heap property

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	99

Hasan Baig

53

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

54

Insert(99)

Violating max heap property

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	99

Hasan Baig

54

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

55

Insert(99)

Violating max heap property

0	92
1	78
2	23
3	99
4	5
5	12
6	21
7	2

Hasan Baig

55

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

56

Insert(99)

Violating max heap property

0	92
1	78
2	23
3	99
4	5
5	12
6	21
7	2

Hasan Baig

56

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

57

Insert(99)

Violating max heap property

0	92
1	99
2	23
3	78
4	5
5	12
6	21
7	2

Hasan Baig

57

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

58

Insert(99)

Violating max heap property

0	92
1	99
2	23
3	78
4	5
5	12
6	21
7	2

Hasan Baig

58

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

59

Insert(99)

Violating max heap property

0	99
1	92
2	23
3	78
4	5
5	12
6	21
7	2

Hasan Baig

59

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Max Heap Construction

60

0	99
1	92
2	23
3	78
4	5
5	12
6	21
7	2

Hasan Baig

60

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps


Activity # 1

Show how the following “Min” heap array will be filled up when the `insert` method is executed periodically on the given data. Also indicate where heap violation occurs (X) and where it is finally fixed (✓) as shown in the example below.

insert(A, 23)
insert(B, 5)
insert(C, 78)
insert(D, 2)
insert(E, 29)
insert(F, 12)
insert(G, 14)
insert(H, 1)

	(X)	(✓)														
0	A, 23	A, 23	B, 5													
1		B, 5	A, 23													
2																
3																
4																
5																
6																
7																

Hasan Baig



61

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Activity # 2

For the insert operation in “Min Heap” data structure exercised in Activity # 1, write the `insert` method, in the class `Heap`, to achieve the same functionality. You will be required to develop a helper function `upheap()` to check the nodes for heap order violation. Class, `Entry`, to create an object of item-priority pair is given below.

```
class Entry:
    def __init__(self, item, priority):
        self.item = item
        self.priority = priority


    def __lt__(self, other):
        return self.priority < other.priority
```

```
class Heap:
    def __init__(self):
        self._heap = []

    def insert(self, item, priority):
```

```
def upheap(self, i):
```

Hasan Baig



62

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Activity # 1 Solution

Show how the following “Min” heap array will be filled up when the `insert` method is executed periodically on the given data. Also indicate where heap violation occurs (X) and where it is finally fixed (✓) as shown in the example below.

insert(A, 23)
insert(B, 5)
insert(C, 78)
insert(D, 2)
insert(E, 29)
insert(F, 12)
insert(G, 14)
insert(H, 1)

	(X)	(✓)	(X)	(✓)	(X)	(✓)	(X)	(✓)
0	A, 23	A, 23	B, 5	B, 5	B, 5	B, 5	D, 2	D, 2
1		B, 5	A, 23	A, 23	A, 23	D, 2	B, 5	B, 5
2				C, 78	C, 78	C, 78	C, 78	F, 12
3					D, 2	A, 23	A, 23	A, 23
4							E, 29	E, 29
5								F, 12
6								G, 14
7								H, 1

Hasan Baig

63

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Activity # 2 Solution

For the insert operation in “Min Heap” data structure exercised in Activity # 1, write the `insert` method, in the class `Heap`, to achieve the same functionality. You will be required to develop a helper function `upheap()` to check the nodes for heap order violation. Class, `Entry`, to create an object of item-priority pair is given below.

```
class Entry:
    def __init__(self, item, priority):
        self.item = item
        self.priority = priority

    def __lt__(self, other):
        return self.priority < other.priority
```

```
class Heap:
    def __init__(self):
        self._heap = []

    def insert(self, item, priority):
        self._heap.append(Entry(item, priority))
        self.upheap(len(self._heap) - 1)

    def upheap(self, i):
        parent = (i - 1) // 2
        if i > 0 and self._heap[i] < self._heap[parent]:
            self._heap[i], self._heap[parent] = self._heap[parent], self._heap[i]
            self.upheap(parent)
```

Hasan Baig

64



Department of Computer Science and Engineering

Data Structures and Object-Oriented Design

(CSE – 2050)

Hasan Baig

Office: UConn (Stamford), 305C
email: hasan.baig@uconn.edu

65

CSE-2050 – Data Structures and Object-Oriented Design

Quick Recap

66

- A priority queue stores a collection of items in a (key, value) pair
 - Key \rightarrow defines the priority
 - Value \rightarrow the actual data

Unsorted List	Sorted List
What will be the time complexity of the following methods on above data?	
<code>insert(key, value)</code> $\rightarrow O(1)$	<code>insert(key, value)</code> $\rightarrow O(n)$
<code>findmin()</code> $\rightarrow O(n)$	<code>findmin()</code> $\rightarrow O(1)$
<code>removemin()</code> $\rightarrow O(n)$	<code>removemin()</code> $\rightarrow O(1)$

Hasan Baig

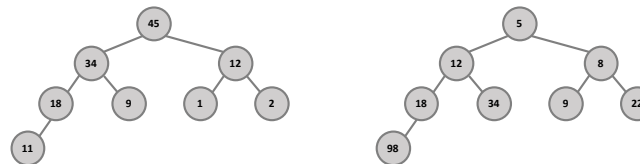


66

Quick Recap

67

- **Heaps** are data structures that are used to implement priority queues (ADT)
 - Implementation using Binary Tree
- Heaps are constructed from left to right
- Max Heap → Parent has the largest value
- Min Heap → Parent has the smallest value



Hasan Baig

- High prioritized element in both cases is always at the top

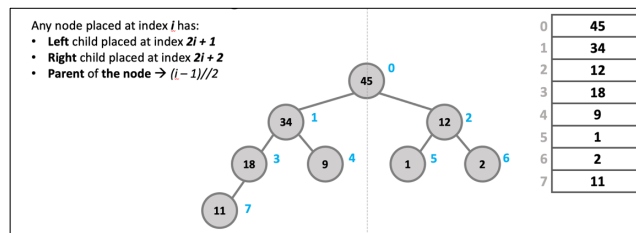


67

Quick Recap

68

- **Heaps** can be represented in 1D array (list)



- Insert operation
 - Fix heap structure if it violated the heap property
- Activity #1 → Insert element in Min Heap
- Activity #2 → Write a python code

Hasan Baig



68

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

69

```

graph TD
    99((99)) --> 92((92))
    99 --> 23((23))
    92 --> 78((78))
    92 --> 5((5))
    78 --> 2((2))
    23 --> 12((12))
    23 --> 21((21))
  
```

0	99
1	92
2	23
3	78
4	5
5	12
6	21
7	2

What should be the time complexity of removing element with highest priority?

Hasan Baig

69

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

70

RemoveMax()

```

graph TD
    99((99)) --> 92((92))
    99 --> 23((23))
    92 --> 78((78))
    92 --> 5((5))
    78 --> 2((2))
    23 --> 12((12))
    23 --> 21((21))
  
```

0	99
1	92
2	23
3	78
4	5
5	12
6	21
7	2

Hasan Baig

70

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

71

RemoveMax()

0	99
1	92
2	23
3	78
4	5
5	12
6	21
7	2

Hasan Baig

71

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

72

RemoveMax()

0	
1	92
2	23
3	78
4	5
5	12
6	21
7	2

Hasan Baig

72

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

73

RemoveMax()

0	2
1	92
2	23
3	78
4	5
5	12
6	21
7	

- Replaced the empty element with the last element

Hasan Baig

73

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

74

RemoveMax()

0	2
1	92
2	23
3	78
4	5
5	12
6	21
7	

- Now check for property violation starting from the root node to the leaf node
- **Heapify operation**

Hasan Baig

74

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

75

RemoveMax()

0	2
1	92
2	23
3	78
4	5
5	12
6	21
7	

- Now check for property violation starting from the root node to the leaf node
- **Heapify operation**
- Compare both children and swap with the greatest one

Hasan Baig

75

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

76

RemoveMax()

0	2
1	92
2	23
3	78
4	5
5	12
6	21
7	

- Now check for property violation starting from the root node to the leaf node
- **Heapify operation**
- Compare both children and swap with the greatest one

Hasan Baig

76

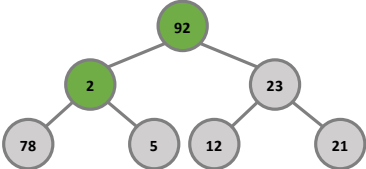
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

77

RemoveMax()



0	92
1	2
2	23
3	78
4	5
5	12
6	21
7	

- Now check for property violation starting from the root node to the lead node
- **Heapify operation**

Hasan Baig

77

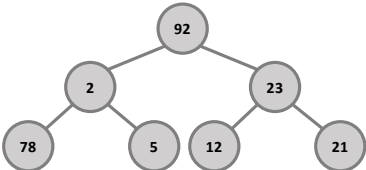
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

78

RemoveMax()



0	92
1	2
2	23
3	78
4	5
5	12
6	21
7	

- Now check for property violation starting from the root node to the lead node
- **Heapify operation**

Hasan Baig

78

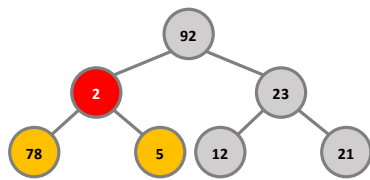
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

79

RemoveMax()



0	92
1	2
2	23
3	78
4	5
5	12
6	21
7	

- Now check for property violation starting from the root node to the lead node
- → **Heapify operation**
- Compare both children and swap with the greatest one

Hasan Baig

79

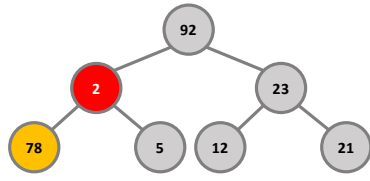
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

80

RemoveMax()



0	92
1	2
2	23
3	78
4	5
5	12
6	21
7	

- Now check for property violation starting from the root node to the lead node
- → **Heapify operation**
- Compare both children and swap with the greatest one

Hasan Baig

80

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

81

RemoveMax()

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

- Now check for property violation starting from the root node to the lead node
- **Heapify operation**
- Compare both children and swap with the greatest one

Hasan Baig

81

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing Max/Min Item

82

RemoveMax()

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

Hasan Baig

82

CSE-2050 – Data Structures and Object-Oriented Design


Priority Queues and Heaps

Heaps Removing arbitrary item

83

- Removing the root node (and usually this is the case) can be done in $O(\log n)$ running time
- What if we want to remove an arbitrary item?
 - Finding it in the array requires $O(n)$ and then we can remove it in $O(\log n)$
 - → Removing arbitrary item requires $O(n)$ time complexity
- This is the same if we want to find an item in a heap
- Heap data structure is recommended in application where we require finding a min/max element

Hasan Baig



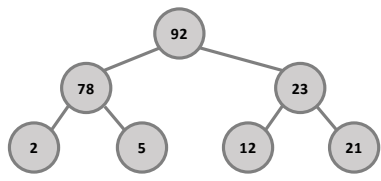
83

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps


Heaps Removing arbitrary item

84



0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

Hasan Baig



84

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing arbitrary item

85

REMOVE(12)

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

Hasan Baig

85

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing arbitrary item

86

REMOVE(12)

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

Hasan Baig

86

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing arbitrary item

87

REMOVE(12)

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

Hasan Baig

87

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing arbitrary item

88

REMOVE(12)

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

Hasan Baig

88

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing arbitrary item

89

REMOVE(12)

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

Hasan Baig

89

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing arbitrary item

90

REMOVE(12)

0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

Hasan Baig

90

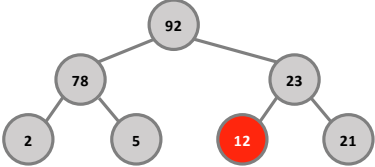
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing arbitrary item

91

REMOVE(12)



0	92
1	78
2	23
3	2
4	5
5	12
6	21
7	

Hasan Baig

91

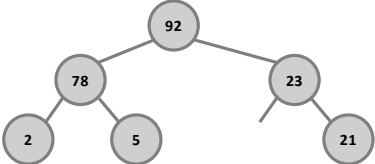
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing arbitrary item

92

REMOVE(12)



0	92
1	78
2	23
3	2
4	5
5	
6	21
7	

- There can not be a **hole** in the data structure
- Replace it with the last item in the heap

Hasan Baig

92

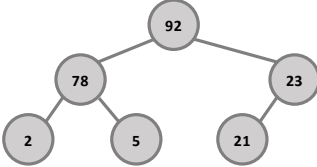
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing arbitrary item

93


REMOVE(12)



0	92
1	78
2	23
3	2
4	5
5	21
6	
7	

- There can not be a **hole** in the data structure
- Replace it with the last item in the heap

Hasan Baig



93

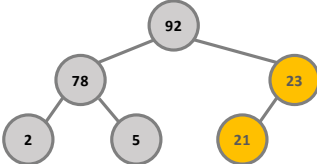
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing arbitrary item

94


REMOVE(12)



0	92
1	78
2	23
3	2
4	5
5	21
6	
7	

- After swapping, check for property violation all the way up till the root node

Hasan Baig



94

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heaps Removing arbitrary item

95

REMOVE(12)

0	92
1	78
2	23
3	2
4	5
5	21
6	
7	

Removing arbitrary item: $O(n) + O(\log n) = O(n)$

Hasan Baig

95

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort

96

We can use Heap data structure to get the data sorted

Heapsort is a comparison-based sorting algorithm that uses heap data structure rather than a linear-time search to find the maximum

- It is a bit slower in practice on most machines than a well implemented quicksort
 - Quicksort is in-place but has worst-case time complexity of $O(n^2)$
- Heapsort has a time complexity of $O(n \log n)$ for all worst, average, and best cases
 - It is also an in-place algorithm

Hasan Baig

96

CSE-2050 – Data Structures and Object-Oriented Design


Priority Queues and Heaps

Heapsort Sorting Procedure

97

1. Read the root node
2. Swap the root node with the last item
3. Heapify to abide by Heap properties
4. Repeat steps 1 and 2 for all nodes (except the last items which have already been read)

Hasan Baig



97

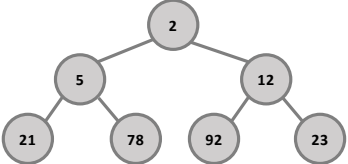
CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

98


Example: Min Heap



```
graph TD; 2((2)) --- 5((5)); 2 --- 12((12)); 5 --- 21((21)); 5 --- 78((78)); 12 --- 92((92)); 12 --- 23((23))
```

[]

Hasan Baig



98

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2]

Hasan Baig

99

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2]

Hasan Baig

100

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2]

Hasan Baig

101

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2]

Hasan Baig

102

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2]

Hasan Baig

103

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2]

Hasan Baig

104

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2]

Hasan Baig

105

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2]

Hasan Baig

106

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2]

Hasan Baig

107

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5]

Hasan Baig

108

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5]

Hasan Baig

109

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5]

Hasan Baig

110

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5]

Hasan Baig

111

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5]

Hasan Baig

112

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5]

Hasan Baig

113

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5]

Hasan Baig

114

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

115

Read
Swap
Heapify

[2, 5, 12]

Hasan Baig

115

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

116

Read
Swap
Heapify

[2, 5, 12]

Hasan Baig

116

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5, 12]

Hasan Baig

117

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5, 12]

Hasan Baig

118

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

119

Read

Swap

Heapify

[2, 5, 12]

Hasan Baig

119

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

120

Read

Swap

Heapify

[2, 5, 12, 21]

Hasan Baig

120

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5, 12, 21]

Hasan Baig

121

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5, 12, 21]

Hasan Baig

122

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5, 12, 21]

Hasan Baig

123

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps

Heapsort Sorting Procedure

Read
Swap
Heapify

[2, 5, 12, 21]

Hasan Baig

124

CSE-2050 – Data Structures and Object-Oriented Design


Priority Queues and Heaps

Heapsort Sorting Procedure

125

....

Hasan Baig



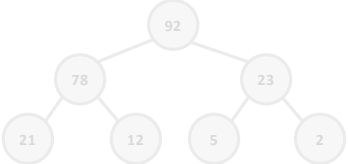
125

CSE-2050 – Data Structures and Object-Oriented Design

Priority Queues and Heaps


Heapsort Sorting Procedure

126



[2, 5, 12, 21, 23, 78, 92]

Hasan Baig

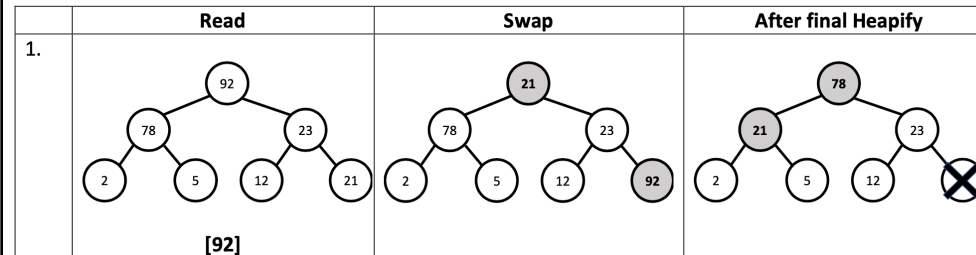


126

Activity # 3

127

Consider that the largest value defines the highest priority, sort the following heap data in reverse order. Highest priority element comes first.



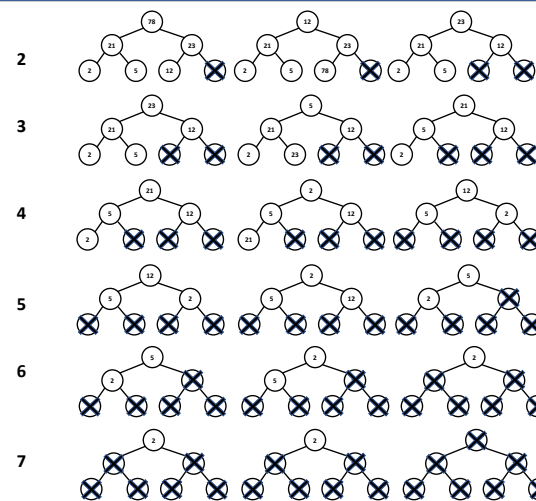
Hasan Baig



127

Activity # 3 Solution

128



Hasan Baig



128