
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>B</td>
</tr>
</tbody>
</table>

Semester Regular and Supplementary Examinations, December - 2013

1 A) (a) What is a priority queue? What are the different ways of representing a priority queue?
   B) Show the result of inserting 10, 12, 1, 14, 6, 5, 7, 8, 15, 3, 7, 9, one at a time into an initially empty min heap with neat diagrams.

2 A) Explain how to implement a priority queue using heaps
   B) Write a short note on lazy binomial queues
   C) Briefly describe graph the process for creating heap

3 A) What are the applications of priority queues and binomial queues?
   B) Explain how to perform insertion and deletion operations in a priority queue

4 A) What is a Binary Heap? Explain how to insert and delete an element into a Binary heap.
   B) Briefly explain the cost amortization operation of a binomial heap with an example.

Semester Supplementary Examinations, May 2013

1 A) Discuss about the insert and delete operation of heap with illustrative example
   B) Explain Kruskal’s algorithm and illustrate with suitable example.

2 A) What is priority queue? How can it be represented by a Heap?
   B) What is Binomial queue? Discuss about binomial amortized analysis

3 A) What is Binomial queue? How it differs from normal queue? What is its use?
   B) Discuss the operations of Binomial queue

4 A) What is Binomial queue? Discuss its operations
   B) Define heap. Discuss a procedure for maintaining the heap property

Semester Regular Examinations, November/December - 2012

1 A) Explain about binomial amortized analysis
   B) Explain about the procedure for delete min from Binary Heap

2 A) Explain about Binomial queue operations
   B) What are the properties of Binary Heap
   C) Explain the procedure for inserting an element into the Binomial Queue

3 A) explain the procedure for deleting the min from binary heap
   B) Explain about lazy binomial Queues.

4 A) Construct the Binary Heap for the following data with neat diagrams
   4, 67, 23, 89, 12, 8, 7, 44, 78, 64, 70, 17
Priority Queue:
A priority queue is an ADT (abstract data type) for maintaining a set $S$ of elements, each with an associated value called priority.

Means priority queue is like a regular queue or stack data structure, but where additionally each element has a “priority” associated with it. That’s why this data structure has “priority” name.

Properties:
The assigned priority and such that order in which elements are deleted and processed comes from the following rules:
- An element of higher priority is processed before any element of lower priority.
- Two elements with the same priority are processed according to the order in which they were added to the queue.

A priority queue supports the following operations:
- Insert($x$) $\rightarrow$ insert element ‘$x$’ in set $S(S \leftarrow S \cup \{x\})$
- Minimum ( ) $\rightarrow$ return the element of $S$ with smallest priority
- Delete-min() $\rightarrow$ returns and removes the elements of $S$ with smallest priority.
- Maximum() $\rightarrow$ return the element of $S$ with highest priority
- Delete-Max() $\rightarrow$ return and removes the element of $S$ with highest priority

Applications of Priority Queue:
The priority queues are extensive use in
- Implementing schedulers in OS, and distributed systems.
- Representing event lists in discrete event simulation
- Implementation of numerous graph algorithms efficiently
- Selecting $K^{th}$ largest or $K^{th}$ smallest element in lists.
- Car rental service
- Sorting Applications

Scheduling the jobs in OS:
A typically application of the Priority Queue is scheduling the jobs in operating system (OS). The os allocates priority to the jobs. The jobs are placed in the queue & position of the job in the queue determines their priorities in OS. There are three kinds of jobs in OS.
1. Real time jobs
2. Foreground jobs
3. Background jobs

The OS always schedules the real time jobs first, if there are no real time jobs in pending then it schedules the foreground jobs. Finally it schedules the back ground jobs.

Car Rental service:
A car rental service is a kind of service
This allows the user to use the car on rental bases
User can user the car by paying the charge on hour bases. there is fixed amount of charge per hour, to implement this application a priority queue is used.
When the user requests for the car his request is kept in priority
If users request is same time then paying money (amount) is kept in priority.
Implementation of Priority Queue:
Priority queue can be implemented by using linked list (Sorted & unsorted), binary search tree, Binary Heap.

- By using linked list
  - **For unsorted**
    - Insertion O(1)
      - [The items are pairs(priority, element). We can implement insert () using insertLast() on the sequence. This takes O(1). However, because we always insert at the end, irrespective of the key value, our sequence is not ordered]
    - Delete-min(), minimum()→ O(n)
      - Thus, for methods such as minimum(), delete-min() we need to look at all the elements of S. The worst case time complexity for these methods is O(n).

  - **For Sorted**
    - Another implementation uses a sequence S, sorted by increasing priorities
      - Minimum() and delete-min() takes O(1)
        - Because the minimum value of given list is always placed in root in the sorted list.
      - Insert()→ O(1).
        - However, to implement insert(), we must now scan through the elements in the worst case. Thus insert() runs in O(n) time.

- **Binary search Tree**
<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>O(log N)</td>
<td>O(N)</td>
</tr>
<tr>
<td>Insert</td>
<td>O(log N)</td>
<td>O(N)</td>
</tr>
<tr>
<td>Delete</td>
<td>O(log N)</td>
<td>O(N)</td>
</tr>
</tbody>
</table>

- **Binary Heap:**
<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td>Insert</td>
<td>O(log N)</td>
<td>O(log N)</td>
</tr>
<tr>
<td>Delete</td>
<td>O(log N)</td>
<td>O(log N)</td>
</tr>
</tbody>
</table>

**Heap:** Heap is a specialized tree based data structure that satisfies the heap property.

**Heap Property:** all nodes are either “greater than or equal to” or “less than or Equal to” each of its children, according to a comparison “predicate” defined for the heap.

**Binary Heap:**
**Introduction:** Binary heaps are special form of binary trees.
  - Binary heap were first introduced by Williams in 1964
**Definition:**
  - A binary heap is a binary tree with two properties; those are structure property and Heap-order property.
Structure Property:
A binary heap is a complete binary tree that is all the levels of the tree, except possibly the last one (deepest) are fully filled, and if the last level of the tree is not complete, the nodes of that level are filled from left to right.

Height of a complete binary tree with N elements is \( \log_2 N \)

Example:

\[ N=10 \]

![Binary Heap Diagram]

Array representation:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Another example for Binary Heap:

![Binary Heap Diagram]

Array representation

<table>
<thead>
<tr>
<th>6</th>
<th>10</th>
<th>12</th>
<th>15</th>
<th>17</th>
<th>21</th>
<th>23</th>
<th>20</th>
<th>19</th>
<th>34</th>
</tr>
</thead>
</table>

Heap Order Property:
All nodes are either [greater than or equal to] or [less than or equal to] each of its children, according to a comparison predicate defined for the heap.

The heap ordering property can be one of two types: min-heap property & max-heap property

- The min-heap property: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.

It supports insert and delete-min operations

![Binary Heap Diagram]
- The *max-heap property*: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.
  It supports **insert** and **delete-max** operation

- Heaps could be binary or d-array.
- Heap order property follows any one property among two (*min-heap property* & *max-heap property*)
- Duplicates are allowed
- No order implied for elements, means which do not share ancestor and descendant relationship.

**Implementation of the heaps:**

It tree is complete we use the following implementation.

Consider an array A. Given element at position “i” in the array.
- Minimum element is root
- Left child(i)=at position 2i
- Right child(i)=at position 2i+1
- Parent (i)=at position \( \left \lfloor \frac{i}{2} \right \rfloor \)

The minimum element will always be present at the root of the heap. Thus the find min operation will have worst-case O (1) running time.

**Height of a heap:**

- Suppose a heap of n nodes has height h.
- Complete binary tree of height h has \( 2^{h+1} -1 \) nodes. (note: here h is consider as number of levels below the root node)
- Hence \( 2^h -1 < n <= 2^{h+1} -1 \)

\[
  n = \left \lfloor \log_2 N \right \rfloor
\]
Binary heap operations:

Create, insert, delete min and find minimum

Create:

```c
void buildHeap(int *data, int n) {
    int low;
    for (low = n/2 - 1; low >= 0; low--){
        insertHeap(data, low, n-1);
    }
    return;
}
```

**Step 1:** We will design a binary tree by storing the given elements in an array for concatenating heap.

**Step 2:** If we want to construct a min heap in a binary tree in which each parent node is less than its child node.

**Step 3:** If we want to construct a max heap in a binary tree in which each parent node is greater than its child node.

Consider the following example: Construct Binary heap by using following elements: 13, 21, 16, 24, 31, 19, 68, 65, 26, 32, 14

For constructing binary heap we must follow two properties. Structural property and heap order property.

First we follow structural proper, we must arrange the all the elements in complete binary tree form left to right.
Then we follow the heap order property.

![Heap Diagram]

**Insert:**

Insert new element into the heap at the next available slot, according to maintains a structure property.

Then “percolate” (filter) the element up the heap while heap-order property not satisfied.

<table>
<thead>
<tr>
<th>Pseudo code for Insert operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>void insertHeap(int *data, int low, int count) {</td>
</tr>
<tr>
<td>int pos = (2 * low) + 1, current = data[low];</td>
</tr>
<tr>
<td>while (pos &lt;= count) {</td>
</tr>
<tr>
<td>if (pos &lt; count &amp;&amp; data[pos] &gt; data[pos + 1])</td>
</tr>
<tr>
<td>pos++;</td>
</tr>
<tr>
<td>if (current &lt;= data[pos])</td>
</tr>
<tr>
<td>break;</td>
</tr>
<tr>
<td>else {</td>
</tr>
<tr>
<td>data[low] = data[pos];</td>
</tr>
<tr>
<td>low = pos;</td>
</tr>
<tr>
<td>pos = (2 * low) + 1;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>data[low] = current;</td>
</tr>
<tr>
<td>return;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

| void addNode(int *data, int val, int n) { |
|     data[n] = val; |
|     buildUp(data, n); |
|     return; |
| } |

| void buildUp(int *data, int index) { |
|     int val = data[index]; |
|     while (data[(index - 1) / 2] >= val) |
|         { |
|             data[index] = data[(index - 1) / 2]; |
|             if (!index) |
|                 break; |
|             index = (index - 1) / 2; |
|         } |
|     data[index] = val; |
|     return; |
| } |
Step 1:- insert an element at the last portion of in a heap.

Step 2: compare with parent element & swap if it violates min-heap property (parent node value is always less than the child).

Step 3: continue comparing process and do swapping parent, newly inserted element up to fulfil the min-heap property to entire heap.

Example for Binary heap insertion:-

Finding the minimum element:

- The element with smallest priority always sits at the root of the heap.
- This because if it was elsewhere, it would have a parent with large priority and this would violate the heap property.
- Hence minimum() can be done in O(1).
Delete Operation:

Delete operation is performed to delete the minimum element, because the minimum element has high priority in min heap priority

- Minimum element is always at the root
- Heap decreases by one in size
- Move last element into hole at root
- **Percolate** (filter) **down** while heap-order property not satisfied
- It takes $O(\log N)$

### Delete operation Pseudo code

```c
void deleteNode(int *data, int n) {
    int val = data[0];
    data[0] = data[n];
    insertHeap(data, 0, n - 1);
    printf("%d is deleted from the heap\n", val);
    return;
}
```

### Heap DeleteMin

1. Copy 31 temporarily here and move it down
2. Is 31 > min(14, 16)?
3. Yes - swap 31 with min(14, 16)
4. Make this position empty
Binomial Queue:
Definition: Binomial queue is a priority queue that is implemented not as a single tree but as a collection of heap ordered trees.
- Each of the trees in the binomial queue has a very special shape called Binomial Tree.
- Binomial queue is a forest of binomial heap trees (Binomial trees).
- Binomial heap is a collection of binomial trees.

Binomial Tree:
Definition: A binomial tree is defined recursively
A binomial tree of order zero is a single node.
A binomial tree of order ‘k’ has a root node whose children are roots of binomial tree of order k-1,k-2---2,1,0. Means how much of height root contains those many sub trees are there and each sub tree height is k-1,k-2,---2,1,0.
<table>
<thead>
<tr>
<th>height ($h$)</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of elements</td>
<td>$2^3 = 8$</td>
<td>$2^2 = 4$</td>
<td>$2^1 = 2$</td>
<td>$2^0 = 1$</td>
</tr>
</tbody>
</table>
Binomial Tree Properties:

Binomial tree of height K has $2^K$ nodes.

This means, a binomial tree of n nodes has $k=\log_2 n$ height. The number of nodes at level ‘d’ of the tree with the height k is the binomial co-efficient.

$\binom{k}{d} = \frac{k!}{d!(k-d)!}$

For example in the above binomial tree at level 2 has

$\binom{4}{2} = \frac{4!}{2!2!} = 6$
**Binomial Heap:**

- Binomial heap is a collection of binomial trees. This is similar to binary heap but also supports quick merging of two heaps.

If satisfies heap properties:

- **Min-Heap:** the value of each node in the tree is ($\geq$) less than or equal to value of its parent.
- **Max-Heap:** The value of each node is ($\leq$) greater than or equal to value of its parent.

Min-heap property in binomial heap example:

![Min-heap example](image)

Max heap Property in binomial heap example:

![Max-heap example](image)

**Properties of binomial heap:**

- The binomial heap obeys MIN, MAX heap properties.
- For any non-negative integer $K$, there is at-most one binomial tree in $H$, whose root has degree.
- If the Binomial heap contains $n$ nodes then it consists at-most

$$\lceil \log_2 n \rceil + 1$$

trees.
Node representation of Binomial tree:-

Binomial Queue:-
Binomial queue Properties:
All properties of binomial heap and binomial tree must be satisfied in binomial queue.

Binomial Queue operations:
Insert
Delete
Find-min
Merge
Merge operation implementation in binomial queue:-

Add corresponding trees from the two forests.

For k, 0 to max-height
If neither queue has a Bk (binary queue of height k), then skip
If only one, leave it.
If two, attach the large priority root as a child of other, producing a tree of height K+1.
End for loop.
Return all trees after merging.
Insertion operation in Binomial Queue:

Insertion in binomial queue is a special case of merging since only create one node tree and perform merging. i.e., to insert a node $x$, into a binomial queue $H$, we know that a single node is a binomial tree of height zero.
Delete Operation in Binomial tree:
Step 1:- First find the binomial tree with small root in the priority queue.
Step 2:- Let $B_k$ be the binomial tree of binomial queue in priority queue remove $B_k$ from $H$ and forming another binomial queue $H'$.
Step 3:- Now remove the root of $B_k$, then creating binomial trees $B_0, B_1, B_2, \ldots, B_{k-1}$ which collectively form a binomial queue $H''$. 
Step 4:- Now merge $H'$ and $H''$. 

Diagram:

- Step 1: $12$ is the smallest root.
- Step 2: $H' = \begin{array}{c}
  12 \\
  23 \\
  51 \\
  24 \\
\end{array}$
- Step 3: $H'' = \begin{array}{c}
  21 \\
  24 \\
  65 \\
  26 \\
\end{array}$
Find-MIN operation in Binomial Queue:-

Find MIN operation implementation by scanning the roots of all the trees. Since there are atmost “logn” different trees this leads to worst case complexity of \( O(\log(n)) \).

Binomial Amortized Analysis:-

Amortized analysis is a method of analysing algorithms that consider the entire sequence of operations of the program. It allows for the establishment of worst case bound for the performance of the algorithm irrespective of input.

This analysis commonly discussed by using Big O notation.

Or

Simple the amortized analysis finds the worst case running time of sequence of operations. Binomial amortized analysis is finding worst case running time of binomial queue sequence of operations.

The amortized analysis running time operation of binomial queue is:
Merge:

1. For merge, assume the two trees have \( n_1 \) and \( n_2 \) nodes with \( T_1 \) and \( T_2 \) trees respectively.
2. Let \( n = n_1 + n_2 \).
3. The actual time for performing the merge is \( O(\log n_1 + \log n_2) = O(\log n) \).
4. After the merge, there can be at most \( \log n \) trees i.e., \( O(\max(\log n_1, \log n_2)) \) so the potential can increase by at most \( O(\log n) \).
5. This is an amortized bound of \( O(\log n) \).

**Insert:** \( O(1) \)

Let \( c_i \) is the cost of \( i \)th insertion.

- \( T_i \) is the no. of trees after \( i \)th insertion.
- \( T_0 = 0 \) is the no. of trees initially.

Then we have following constants \( c_i + (T_i - T_{i-1}) = 2 \)

We have
\[
\begin{align*}
c_1 + (T_1 - T_0) &= 2 \\
c_2 + (T_2 - T_1) &= 2 \\
&\vdots \\
c_{n-1} + (T_{n-1} - T_{n-2}) &= 2 \\
c_n + (T_n - T_{n-1}) &= 2
\end{align*}
\]

If we add all these equations most of the \( T_i \) terms cancel except \( T_n \) and \( T_0 \).
Then remove Bk from heap/queue, then forming another binomial queue H’. now remove the root of Bk then create binomial trees, B_0, B_{k+1}, B_2, …… B_{k-1} which collectively form binomial queue H’’. Since each tree has atmost log n Children.

Creating new heap trees O(log n)
Merging heap is O(log n)

By considering all these operations the entire delete min operations is O(log n).
Lazy Binomial Queue:
- Lazy binomial queue is a binomial queue in which merging is done quickly.
- Here to merge two binomial queues, we simply concatenate two lists of binomial trees in the resulting forest, they may have several trees of same size.
- Because of this lazy merging worst case time for merge, insert is O(1).

Delete-Min:-
Convert the lazy binomial queue a standard binomial queue so delete min, time complexity is same as standard binomial queue time complexity.

Fibonacci Heap’s:-
In binomial queue, the amortized time is more, so to solve this we go for Fibonacci heaps. It supports all basic operations in O(1) amortized time except del-min & delete which takes O(log (n)) worst/amortized time.
Fibonacci heap’s generalize binomial queue by adding two new concepts.
1. A different implementation of decrease key.
2. Lazy merging: two heaps are merged only when it is required.