Name and ID:  **Solution**  Section: BAI-3A Marks: 15

**Question#01 [2+5 Marks]**

1. If we were to store a binary MinHeap in a linked tree, rather than an array, how will it affect the asymptotic running times/time complexity of the functions: getMin, deleteMin and insert? Based on your answer, explain why we choose to store heaps in arrays instead? You can assume that we know the size of the heap in advance.

Solution

For the heap, whether it is a linked or an array structure, the following always hold:

getMin: O(1)

deleteMin: O(log n)

insert: O(log n)

Normally we use the array implementation so that we don’t need to store extra pointers and it is easier to implement using arrays

1. Taking an initially empty HasHTable of size 10, insert the following keys using hash function h(key) = key % 7; (Keys appears in the following order) **150, 245,63, 421, 82, 107, 315** Provide the content of the HashTable when collision resolution strategies used are (i) Quadratic Probing and (ii) Separate Chaining, also state number of collisions in each case.

Solution

|  |  |
| --- | --- |
| Quadratic Probing | Separate Chaining |
| |  |  | | --- | --- | | Index | HashTable[Index] | | 0 | 245 | | 1 | 63 (actual index 0) | | 2 | 421 (actual index 1) | | 3 | 150 | | 4 | 315 (actual index 0) | | 5 | 82 | | 6 | 107 (actual index 2) | | 7 |  | | 8 |  | | 9 |  |   No. of collision: 6 | |  |  | | --- | --- | | Index | Linked List Chain | | 0 | 245 -> 63 -> 315 | | 1 | 421 | | 2 | 107 | | 3 | 150 | | 4 |  | | 5 | 82 | | 6 |  | | 7 |  | | 8 |  | | 9 |  |   No. of collision: 2 |

**Question#02 [4+4 Marks]**

1. Following is an array to be converted into a binary Max-Heap, using repeated MaxHeapify/InsertInHeap calls. Show the contents of this array after the Max-Heap has been constructed.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **…** |
| 11 | 17 | 8 | 21 | 5 | … |

* Delete key. Show the resulting heap and circle the changed positions.
* Insert the key "50". Show the updated heap and circle the changed positions.
* Delete the key again. Show the final heap and circle the changed positions.

Solution

Final Max Heap:

**21**

**/ \**

**17 8**

**/ \**

**11 5**

**Max Heap = [21, 17,8,11,5]**

After Deletion:

**17**

**/ \**

**11 8**

**/**

**5**

**Max Heap = [17, 11, 8, 5]**

Inserted 50:

**50**

**/ \**

**17 8**

**/ \**

**5 11**

**Max Heap = [50, 17, 8, 5, 11]**

After deletion:

**17**

**/ \**

**11 8**

**/**

**5**

**Max Heap = [17, 11, 8, 5]**

1. Write a function to decide whether a given integer array satisfy the condition of a Min Heap data structure or not.

Solution

bool isMinHeap(const vector<int>& arr) {

int n = arr.size();

// validing heap property on all the internal node

for (int i = 0; i <= (n - 2) / 2; ++i) {

int leftChild = 2 \* i + 1;

int rightChild = 2 \* i + 2;

// is parent > left child

if (leftChild < n && arr[i] > arr[leftChild]) {

return false;

}

// is parent > right child

if (rightChild < n && arr[i] > arr[rightChild]) {

return false;

}

}

return true;

}