

**/\* 1) Given an infinite stream of integers, return the element representing the k th largest element in the stream. (Hint: Use Min-heap) \*/**

```
// Min heap
#include <bits/stdc++.h>
using namespace std;

void Adjust(vector<int>& A, int i, int n) {
    int j = 2 * i + 1; // Left child index , Adjusted for 0-based index
    int item = A[i]; // Store the current node value

    while (j < n) {
        // If the right child exists and is greater than the left child
        if ((j + 1 < n) && (A[j] > A[j + 1])) // Compare with right child
            j = j + 1;
        if (item <= A[j])
            break;
        A[(j - 1) / 2] = A[j];
        j = 2 * j + 1;
    }
    A[(j - 1) / 2] = item;
}

void Heapify(vector<int>& A, int n) {
    for (int i = (n - 1) / 2; i >= 0; i--) {
        Adjust(A, i, n);
    }
}

// Function to insert into a min-heap of fixed size k
void insertKthLargest(vector<int>& minHeap, int num, int k) {
    // If the heap has fewer than k elements, just add the new element
    if (minHeap.size() < k) {
        minHeap.push_back(num);
        Heapify(minHeap, minHeap.size());
    }
    // If the heap already has k elements, check if the new number should replace the
    root
    else if (num > minHeap[0]) {
        minHeap[0] = num; // Replace root with new element
        Adjust(minHeap, 0, k); // Re-adjust heap to maintain min-heap property
    }
}

int main() {
```

```

int k = 3; // Example: Finding the 3rd largest element
vector<int> minHeap; // Min-heap to store k largest elements

vector<int> stream = {10, 20, 11, 70, 50, 40, 90}; // Example stream

cout << "Processing stream:" << endl;
for (int num : stream) {
    insertKthLargest(minHeap, num, k);
    if (minHeap.size() == k) {
        cout << "After inserting " << num << ", " << k << "-th largest so far is "
<< minHeap[0] << endl;
    } else {
        cout << "After inserting " << num << ", less than " << k << " elements in
stream." << endl;
    }
}

return 0;
}

```

## Output:

The screenshot shows a C++ IDE with the following code and output:

```

19 }
20
21 void Heapify(vector<int>& A, int n) {
22     for (int i = (n - 1) / 2; i >= 0; i--) {
23         Adjust(A, i, n);
24     }
25 }
26
27 // Function to insert into a min-heap of fixed size k
28 void insertKthLargest(vector<int>& minHeap, int num, int k) {
29     // If the heap has fewer than k elements, just add the new element
30     if (minHeap.size() < k) {
31         minHeap.push_back(num);
32         Heapify(minHeap, minHeap.size());
33     }
34     // If the heap already has k elements, check if the new number is larger than the
35     // smallest element in the heap. If yes, replace the smallest element with the new
36     // number and re-heapify.
37     else if (num > minHeap[0]) {
38         minHeap[0] = num; // Replace root with new element
39         Adjust(minHeap, 0, k); // Re-adjust heap to maintain min-heap property
40     }
41 }
42
43 int main() {
44     int k = 3; // Example: Finding the 3rd largest element
45     vector<int> minHeap; // Min-heap to store k largest elements
46
47     vector<int> stream = {10, 20, 11, 70, 50, 40, 90}; // Example stream
48
49     cout << "Processing stream:" << endl;
50     for (int num : stream) {
51         insertKthLargest(minHeap, num, k);
52         if (minHeap.size() == k) {
53             cout << "After inserting " << num << ", " << k << "-th largest so far is " << minHeap[0] << endl;
54         } else {
55             cout << "After inserting " << num << ", less than " << k << " elements in stream." << endl;
56         }
57     }
58
59     return 0;
60 }

```

Output:

```

Processing stream:
After inserting 10, less than 3 elements in stream.
After inserting 20, less than 3 elements in stream.
After inserting 11, 3-th largest so far is 10
After inserting 70, 3-th largest so far is 11
After inserting 50, 3-th largest so far is 20
After inserting 40, 3-th largest so far is 40
After inserting 90, 3-th largest so far is 50
Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.

```

**/\* 2) Suppose a hospital's emergency room is filled with individuals of various ages. Sort the patients efficiently so that the oldest patients receive care first. (Hint: Use Max-heap) \*/**

```

// Max heap
#include <bits/stdc++.h>

```

```

#include <chrono>

using namespace std;

void Adjust(vector<int>& A, int i, int n) {
    int j = 2 * i + 1; // Left child index , Adjusted for 0-based index
    int item = A[i]; // Store the current node value

    while (j <= n) {
        // If the right child exists and is greater than the left child
        if (j < n && A[j] > A[j + 1]) {
            j++; // Move to right child
        }
        // If the item is greater than or equal to the largest child, we break
        if (item <= A[j]) {
            break;
        }
        // Move the child up to the parent
        A[i] = A[j];
        i = j; // Move down to the child
        j = 2 * i + 1; // Update the child index
    }
    A[i] = item; // Place the item at its correct position
}

void Heapify(vector<int>& A, int n) {
    for (int i = (n / 2) - 1; i >= 0; i--) {
        Adjust(A, i, n);
    }
}

void HeapSort(vector<int>& A, int n) {
    Heapify(A, n); // Build max-heap

    for (int i = n; i >= 1; i--) {
        // Swap the root of the heap (max element) with the last element
        swap(A[0], A[i]);
        Adjust(A, 0, i - 1); // Adjust the heap
    }
}

int main() {
    // Example patient ages
    vector<int> A = {2, 9, 7, 6, 5, 8, 5, 8, 10, 23};

    cout << "Original ages: ";
    for (int age : A) {
        cout << age << " "; // Display original ages
    }
}

```

```

    }
    cout << endl;

    HeapSort(A, A.size() - 1); // Sort the ages

    cout << "Sorted by priority (oldest to youngest): ";
    for (int age : A) {
        cout << age << " "; // Display sorted ages
    }
    cout << endl;

    return 0;
}

```

## Output:

```

25     A[i] = item; // Place the item at its correct position
26 }
27
28 void Heapify(vector<int>& A, int n) {
29     for (int i = (n / 2) - 1; i >= 0; i--) {
30         Adjust(A, i, n);
31     }
32 }
33
34 void HeapSort(vector<int>& A, int n) {
35     Heapify(A, n); // Build max-heap
36
37     for (int i = n; i >= 1; i--) {
38         // Swap the root of the heap (max element) with the
39         swap(A[0], A[i]);
40         Adjust(A, 0, i - 1); // Adjust the heap
41     }
42 }
43
44 int main() {
45     // Example patient ages
46     vector<int> A = {2, 9, 7, 6, 5, 8, 5, 8, 10, 23};
47
48     cout << "Original ages: ";
49     for (int age : A) {
50         cout << age << " "; // Display original ages
51     }
52     cout << endl;
53
54     HeapSort(A, A.size() - 1); // Sort the ages
55
56     cout << "Sorted by priority (oldest to youngest): ";
57     for (int age : A) {
58         cout << age << " "; // Display sorted ages
59     }
60     cout << endl;
61
62     return 0;
}

```

"D:\Education\DUET\3rd year 1st semester\Sessional\Algorithm Design and Analysis Sessional\2nd 21-10-24\Assignment\pblm 2.exe"  
 Original ages: 2 9 7 6 5 8 5 8 10 23  
 Sorted by priority (oldest to youngest): 23 10 9 8 8 7 6 5 5 2  
 Process returned 0 (0x0) execution time : 0.108 s  
 Press any key to continue.

**/\* 3) You are given k sorted arrays, each containing n integers. Write a function that efficiently merges these k sorted arrays into a single sorted array. (Hint: Use Min-heap) \*/**

```

// Min-heap implementation to merge k sorted arrays
#include <bits/stdc++.h>
#include <chrono>

using namespace std;

```

```

// Adjust the heap to maintain the min-heap property
void Adjust(vector<int>& A, int i, int n) {
    int j = 2 * i + 1; // Start with the left child
    int item = A[i];

    while (j <= n) {
        if (j + 1 <= n && A[j] > A[j + 1]) // Compare with right child
            j = j + 1;
        if (item <= A[j])
            break;
        A[i] = A[j]; // Move the smaller child up
        i = j; // Move down the tree
        j = 2 * i + 1; // Update the child index
    }
    A[i] = item; // Place the item in its correct position
}

// Function to heapify the array
void Heapify(vector<int>& A, int n) {
    for (int i = (n / 2) - 1; i >= 0; i--) {
        Adjust(A, i, n);
    }
}

// Function to merge k sorted arrays
vector<int> mergeKSortedArrays(vector<vector<int>>& arrays) {
    vector<int> heap;

    // Push the first element of each array into the heap
    for (const auto& array : arrays) {
        for (int num : array) {
            heap.push_back(num);
        }
    }

    int n = heap.size();
    Heapify(heap, n - 1); // Create a min-heap from the elements

    vector<int> result;

    // Extract elements from the min-heap
    for (int i = n - 1; i >= 0; i--) {
        result.push_back(heap[0]); // Get the minimum element
        heap[0] = heap[i]; // Move the last element to the root
        Adjust(heap, 0, i - 1); // Adjust the heap
    }

    return result; // Return the merged array
}

```

```

}

int main() {
    vector<vector<int>> arrays = {
        {1, 4, 17, 10},
        {2, 5, 8},
        {7, 3, 6, 9 }
    };

    vector<int> mergedArray = mergeKSortedArrays(arrays);

    cout << "Merged sorted array: ";
    for (int num : mergedArray) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}

```

## Output:

The screenshot shows a C++ IDE with two windows. The left window displays the source code for a program that merges k sorted arrays. The right window shows the program's output.

**Source Code (pbim 3.cpp):**

```

7 // Adjust the heap to maintain the min-heap property
8 void Adjust(vector<int>& A, int i, int n) {
9     int j = 2 * i + 1; // Start with the left child
10    int item = A[i];
11
12    while (j <= n) {
13        if (j + 1 <= n && A[j] > A[j + 1]) // Compare with right child
14            j = j + 1;
15        if (item <= A[j])
16            break;
17        A[i] = A[j]; // Move the smaller child up
18        i = j; // Move down the tree
19        j = 2 * i + 1; // Update the child index
20    }
21    A[i] = item; // Place the item in its correct position
22 }
23
24 // Function to heapify the array
25 void Heapify(vector<int>& A, int n) {
26     for (int i = (n / 2) - 1; i >= 0; i--) {
27         Adjust(A, i, n);
28     }
29 }
30
31 // Function to merge k sorted arrays
32 vector<int> mergeKSortedArrays(vector<vector<int>>& arrays) {
33     vector<int> heap;
34
35     // Push the first element of each array into the heap
36     for (const auto& array : arrays) {
37         for (int num : array) {
38             heap.push_back(num);
39         }
40     }
41
42     int n = heap.size();
43     Heapify(heap, n - 1); // Create a min-heap from the elements
44 }

```

**Output Window:**

```

"D:\Education\DUET\3rd year 1st semester\Sessional\Algorithm Design and Analysis Sessional\2nd 21-10-24\Assignment\pbim 3.exe
Merged sorted array: 1 2 3 4 5 6 7 8 9 10 17
Process returned 0 (0x0)   execution time : 0.087 s
Press any key to continue.

```