Bài tập STL

**1. Maximize product of sum of speeds of K workers and their minimum efficiency** Given an integer N, representing the number of workers, an array speed[ ], where speed[i] represents the speed of the ith worker, and an array efficiency[ ], where efficiency[i] represents the efficiency of the ith worker, and an integer K, the task is to select K workers such that the ( Sum of speeds of all the workers ) \* ( Minimum efficiency of among K workers ) is maximum possible.

**Input: N = 6, speed[] = {2, 10, 3, 1, 5, 8}, efficiency[] = {5, 4, 3, 9, 7, 2}, K = 2 Output: 60**

Explanation:

Selecting 2nd worker (Speed = 10 and Efficiency = 4) and 5th worker ( Speed = 5 and Efficiency = 7). Therefore, the maximum sum possible = (10 + 5) \* min(4, 7) = 60

**Input: N = 6, speed[] = {2, 10, 3, 1, 5, 8}, efficiency[] = {5, 4, 3, 9, 7, 2}, K = 3 Output: 68**

**2. Length of the longest subsequence with negative sum of all prefixes**

Given an array arr[] consisting of N integers, the task is to find the length of the longest subsequence such that the prefix sum at each index of the subsequence is negative.

**Input: arr[] = {-1, -3, 3, -5, 8, 2}**

**Output: 5**

Explanation: Longest subsequence satisfying the condition is {-1, -3, 3, -5, 2}.

**Input: arr[] = {2, -5, 2, -1, 5, 1, -9, 10}**

**Output: 6**

Explanation: Longest subsequence satisfying the condition is {-1, -3, 3, -5, 2}.

**3. Maximum amount of capital required for selecting at most K projects**

Given an integer N, representing number of projects, two arrays P[] and C[], consisting of N integers, and two integers W and K where, W is the initial capital amount, P[i] and C[i] are the profits and capital required to choose the ith project. The task is to calculate the maximum amount of capital required to choose at most K projects, such that the profit of the chosen projects is added to W and choosing any project required at least C[i].

Input: N = 3, W = 0, K = 2, P[] = {1, 2, 3}, C[] = {0, 1, 1}

Output: 4

Explanation:

Project 1: With the given amount of W as 0, the 0th project can be chosen at a cost of C[0] i.e., 0, and after finishing the capital added to W i.e., W becomes W + P[0] = 1.

Project 2: With the given amount of W as 1, the 2nd project can be chosen at a cost of C[2] i.e., 1, and after finishing the capital added to W i.e., W becomes W + P[2] = 1 + 3 = 4.

Input: N = 3, W = 1, K = 1, P[] = {10000, 2000, 3000}, C[] = {1, 1, 1}

Output: 10001

**4. Efficient way to initialize a priority queue**

STL Priority Queue is the implementation of Heap Data Structure. By default, it’s a max heap, and can be easily for primitive data types. There are some important applications of it which can be found in this article.

Priority queue can be initialized in two ways either by pushing all elements one by one or by initializing using their constructor. In this article, we will discuss both methods and examine their time complexities. **Method 1:** The simplest approach is to traverse the given array and push each element one by one in the priority queue. In this method, the push method in the priority queue takes O(log N) time. Where N is the number of elements in the array.

**Method 2:** In this method, copy all the array elements into the priority queue while initializing it (this copying will be happened using the copy constructor of priority\_queue). In this method, the priority\_queue will use the build heap method internally. So the build heap method is taking O(N) time.

**5. Replace the middle element of the longest subarray of 0s from the right exactly K times** Given an array arr[] of size N, consisting of 0s initially, and a positive integer K, the task is to print the array elements by performing the following operations exactly K times.

For every ith operation select the rightmost longest subarray consisting of all 0s and replace the mid element of the subarray by i.

If two middle elements exist, then check if i is an even number or not. If found to be true, then replace the rightmost middle element with i.

Otherwise, replace the leftmost middle element with i.

Input: arr[] = { 0, 0, 0, 0, 0}, K = 3

Output: 3 0 1 0 2

Explanation:

In 1st operation selecting the subarray { arr[0], …, arr[4]} and replacing arr[2] by 1 modifies arr[] to { 0, 0, 1, 0, 0}

In 2nd operation selecting the subarray { arr[3], …, arr[4]} and replacing arr[4] by 2 modifies arr[] to { 0, 0, 1, 0, 2}

In 3rd operation selecting the subarray { arr[0], …, arr[1]} and replacing arr[1] by 3 modifies arr[] to { 0, 3, 1, 0, 2}

Therefore, the required output is 3 0 1 0 2.

Input: arr[] = { 0, 0, 0, 0, 0, 0, 0 }, K = 7

Output: 7 3 6 1 5 2 4

**6. Minimize remaining array element by removing pairs and replacing them with their average** Given an array arr[] of size N, the task is to find the smallest possible remaining array element by repeatedly removing a pair, say (arr[i], arr[j]) from the array and inserting the Ceil value of their average. Input: arr[] = { 1, 2, 3 }

Output:

Explanation:

Removing the pair (arr[1], arr[2]) from arr[] and inserting (arr[1] + arr[2] + 1) / 2 into arr[] modifies arr[] to { 1, 2 }.

Removing the pair (arr[0], arr[1]) from arr[] and inserting (arr[0] + arr[1] + 1) / 2 into arr[] modifies arr[] to { 2 }.

Therefore, the required output is 2.

Input: arr[] = { 30, 16, 40 }

Output: 26

Explanation:

Removing the pair (arr[0], arr[2]) from arr[] and inserting (arr[0] + arr[2] + 1) / 2 into arr[] modifies arr[] to { 16, 35 } .

Removing the pair (arr[0], arr[1]) from arr[] and inserting (arr[0] + arr[1] + 1) / 2 into arr[] modifies arr[] to { 26 } .

Therefore, the required output is 26.

**7. K-th Smallest Element in an Unsorted Array using Priority Queue**

Given an array arr[] consisting of N integers and an integer K, the task is to find the Kth smallest element in the array using Priority Queue.

Input: arr[] = {5, 20, 10, 7, 1}, N = 5, K = 2

Output: 5

Explanation: In the given array, the 2nd smallest element is 5. Therefore, the required output is 5.

Input: arr[] = {5, 20, 10, 7, 1}, N = 5, K = 5

Output: 20

Explanation: In the given array, the 5th smallest element is 20. Therefore, the required output is 20

**8. Priority Queue using Binary Heap**

Priority Queue is an extension of the queue with the following properties:

Every item has a priority associated with it.

An element with high priority is dequeued before an element with low priority. If two elements have the same priority, they are served according to their order in the queue. A Binary Heap is a Binary Tree with the following properties:

− It is a Complete Tree. This property of Binary Heap makes them suitable to be stored in an array. − A Binary Heap is either Min Heap or Max Heap.

− In a Min Binary Heap, the key at the root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree.

− Similarly, in a Max Binary Heap, the key at the root must be maximum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Operation on Binary Heap

− insert(p): Inserts a new element with priority p.

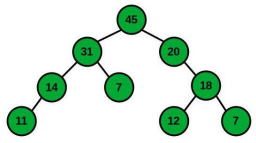
− extractMax(): Extracts an element with maximum priority.

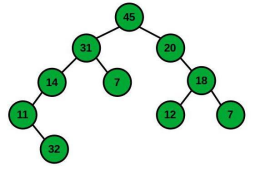
− remove(i): Removes an element pointed by an iterator i.

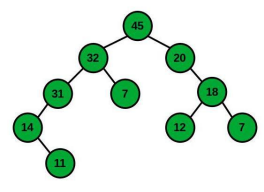
− getMax(): Returns an element with maximum priority.

− changePriority(i, p): Changes the priority of an element pointed by i to p.

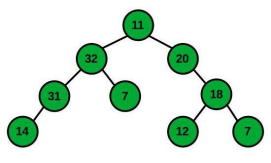
Example of A Binary Max Heap

Suppose below is the given Binary Heap that follows all the properties of Binary Max Heap. 

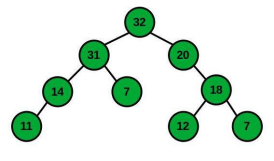
Now a node with value 32 need to be insert in the above heap: To insert an element, attach the new element to any leaf. For Example A node with priority 32 can be added to the leaf of the node 11. But this violates the heap property. To maintain the heap property, shift up the new node 32. 

Shift Up Operation get node with 32 at the correct position: Swap the incorrectly placed node with its parent until the heap property is satisfied. For Example: As node 11 is less than node 32 so, swap node 11 and node 32. Then, swap node 14 and node 32. And at last, swap node 31 and node 32. 

ExtractMax: The maximum value is stored at the root of the tree. But the root of the tree cannot be directly removed. First, it is replaced with any one of the leaves and then removed. For Example: To remove Node 45, it is first replaced with node 11. But this violates the heap property, so move the replaced node down. For that, use shift down operation.



ShiftDown operation: Swap the incorrectly placed node with a larger child until the heap property is satisfied. For Example: Node 11 is swapped with node 32 then, with node 31 and in last it is swapped with node 14.

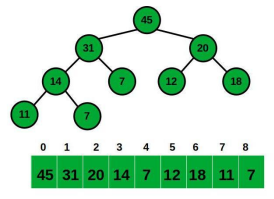


− ChangePriority: Let the changed element shift up or down depending on whether its priority decreased or increased. For Example: Change the priority of nodes 11 to 35, due to this change the node has to shift up the node in order to maintain the heap property.

− Remove: To remove an element, change its priority to a value larger than the current maximum, then shift it up, and then extract it using extract max. Find the current maximum using getMax. − GetMax: The max value is stored at the root of the tree. To getmax, just return the value at the root of the tree.

**Array Representation of Binary Heap**

Since the heap is maintained in form of a complete binary tree, because of this fact the heap can be represented in the form of an array. To keep the tree complete and shallow, while inserting a new element insert it in the leftmost vacant position in the last level i.e., at the end of our array. Similarly, while extracting maximum replace the root with the last leaf at the last level i.e., the last element of the array. Below is the illustration of the same:



**9. Maximum number of unique Triplets such that each element is selected only once** Given an array arr[] of size, N. Find the maximum number of triplets that can be made using array elements such that all elements in each triplet are different. Print the maximum number of possible triplets along with a list of the triplets.

Note: Each element of the array can belong to only 1 triplet.

nput: arr[] = {2, 2, 3, 3, 4, 4, 4, 4, 5}

Output:

Maximum number of possible triples : 2

2 3 4

3 4 5

Explanation:

We can form at most 2 triples using the given array such that each triple contains different elements. Input: arr[] = {1, 2, 3, 4, 5, 6, 7 }

Output:

Maximum number of possible triples : 2

5 6 7

2 3 4

Explanation:

We can form at most 2 triples using the given array such that each triple contains different elements. Naive Approach: The idea is to run three nested loops to generate all triplets and for every triplet, check if they are pairwise distinct and also check if each element of the array belongs to exactly 1 triplet. Time Complexity: O(N3)

Auxiliary Space: O(1)

Efficient Approach: The problem can be solved Greedy Approach and keep taking triplets having a maximum frequency. Below are the steps:

− Store the frequency of all the numbers in a Map.

− Make a max priority queue ans to store pairs in it where the first element in pair is the frequency of some element and the second element in pair is the element itself.

− Now repeatedly extract the top 3 elements from the priority queue, make triplets using those 3 elements, decrease their frequency by 1 and again insert elements in priority queue using frequency is greater than 0.

**10.Maximize jobs that can be completed under given constraint**

Given an integer N denoting number of jobs and a matrix ranges[] consisting of a range [start day, end day] for each job within which it needs to be completed, the task is to find the maximum possible jobs that can be completed.

Input: N = 5, Ranges = {{1, 5}, {1, 5}, {1, 5}, {2, 3}, {2, 3}}

Output: 5

Explanation: Job 1 on day 1, Job 4 on day 2, Job 5 on day 3, Job 2 on day 4, Job 3 on day 5 Input: N=6, Ranges = {{1, 3}, {1, 3}, {2, 3}, {2, 3}, {1, 4}, {2, 5}}

Output: 5

Approach: The above problem can be solved using a Priority Queue. Follow the steps below to solve the problems:

Find the minimum and maximum day in the range of jobs.

Sort all jobs in increasing order of start day.

Iterate from the minimum to maximum day, and for every ith day, select the job having least end day which can be completed on that day.

In order to perform the above step, maintain a Min Heap, and on every ith day, insert the jobs that can be completed on that day, into the Min Heap sorted by end day. If any job can completed on the ith day, consider the one with the lowest end day and increase the count of jobs completed. Repeat this process for all the days and finally print the count of jobs completed.

**11.Convert Infix To Prefix Notation**

While we use infix expressions in our day to day lives. Computers have trouble understanding this format because they need to keep in mind rules of operator precedence and also brackets. Prefix and Postfix expressions are easier for a computer to understand and evaluate.

Given two operands a and b and an operator 🖸, the infix notation implies that O will be placed in between a and b i.e a🖸b. When the operator is placed after both operands i.e ab🖸, it is called postfix notation. And when the operator is placed before the operands i.e 🖸ab, the expression in prefix notation. Given any infix expression, we can obtain the equivalent prefix and postfix format. Input : A \* B + C / D

Output : + \* A B/ C D

Input : (A - B/C) \* (A/K-L)

Output : \*-A/BC-/AKL

Step 1: Reverse the infix expression i.e A+B\*C will become C\*B+A. Note while reversing each ‘(‘ will become ‘)’ and each ‘)’ becomes ‘(‘.

Step 2: Obtain the “nearly” postfix expression of the modified expression i.e CB\*A+. Step 3: Reverse the postfix expression. Hence in our example prefix is +A\*BC.