Ask a Question

Introduction to Top K Elements

Let's go over the Top K Elements pattern, its real-world applications, and some problems we can solve with it.

We'll cover the following...

About the pattern

The **top k elements** pattern is an important technique in coding that helps us efficiently find a specific number of elements, known as k, from a set of data. This is particularly useful when we're tasked with identifying the largest, smallest, or most/least frequent elements within an unsorted collection.

To solve tasks like these, one might think to sort the entire collection first, which takes $O(n\log(n))$ time, and then select the top k elements, taking additional $O(k)$ time. However, the top k elements pattern bypasses the need for full sorting, reducing the time complexity to $O(n\log k)$ by managing which elements we compare and keep track of.

Which data structure can we use to solve such problems? A heap is the best data structure to keep track of the smallest or largest k elements. With this pattern, we either use a max heap or a min heap to find the smallest or largest k elements, respectively, because they allow us to efficiently maintain a collection of elements ordered in a way that gives u quick access to the smallest (min heap) or largest (max heap) element.

For example, let’s look at how this pattern operates to solve the problem of finding the top k largest elements (by using min heap) or top k smallest elements (by using max heap):

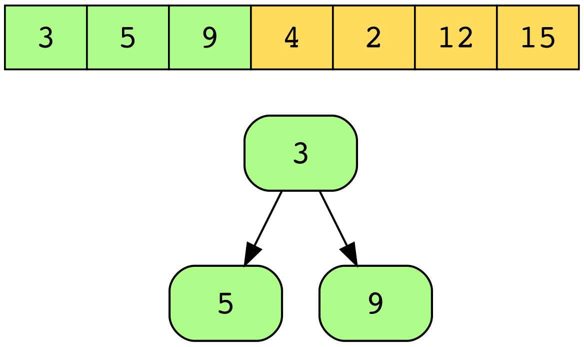
1. Insert the first k elements from the given set of elements into a heap. If we’re looking for the largest elements, use a min heap to keep the smallest of the large elements at the top. Conversely, for the smallest elements, use a max heap to keep the largest of the small elements at the top.
2. Iterate through the remaining elements of the given set.
   I. For a min heap, if we find an element larger than the top, remove the top element (the smallest of the large elements) and insert the new, larger element. This ensures the heap always contains the largest elements seen so far.
   II. For a max heap, if we find an element smaller than the top, remove the top element (the largest of the small elements) and insert the new, smaller element, keeping the heap filled with the smallest elements seen so far.

The efficiency of this pattern comes from the ability of the heap to insert and remove elements in $O(\log k)$ time. Because we only maintain k elements in the heap, these operations are quick, and we can process all n elements in the given set in $O(n \log k)$ time.

> It’s important to note that while accessing the top element of the heap can be done in $O(1)$ time, retrieving all k elements, if necessary, involves removing them one by one. This process takes

> $O(k \log k)$ time because each removal necessitates reorganizing the heap.

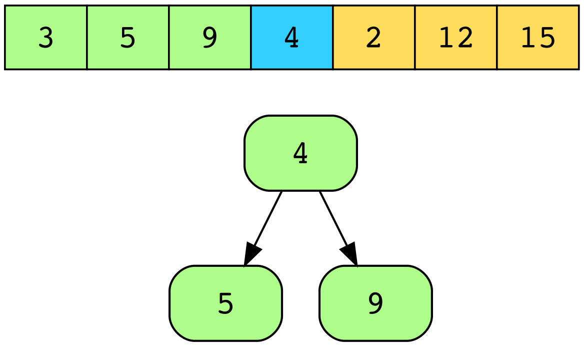
Let's look at the following illustration to understand how to use min heap to find the top three largest elements.



min heap

We add first 3 elements from the array into the heap.

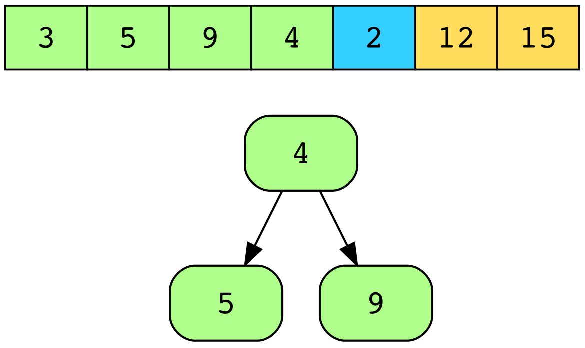
1 of 5



min heap

We pop the top element from the heap and push the current element since it is greater than the top.

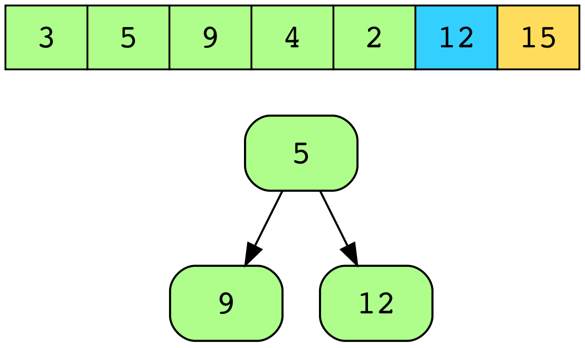
2 of 5



min heap

The heap will remain same since the top element is greater than the current.

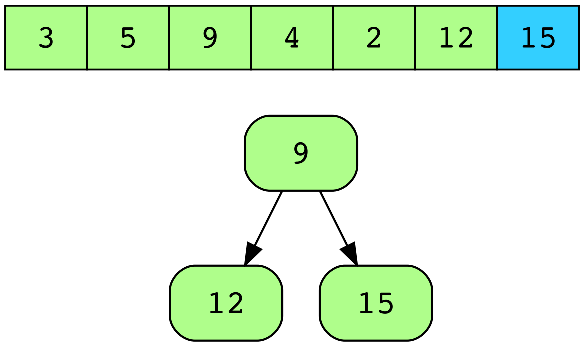
3 of 5



min heap

We pop the top element from the heap and push the current element since it is greater than the top.

4 of 5



min heap

We pop the top element from the heap and push the current element since it is greater than the top.

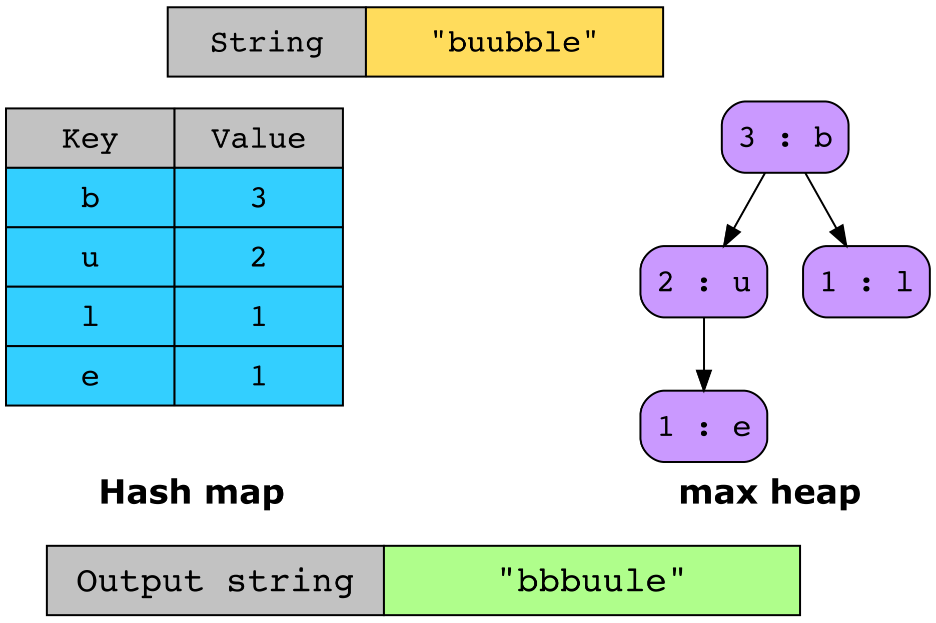
5 of 5

Examples

The following examples illustrate some problems that can be solved with this approach:

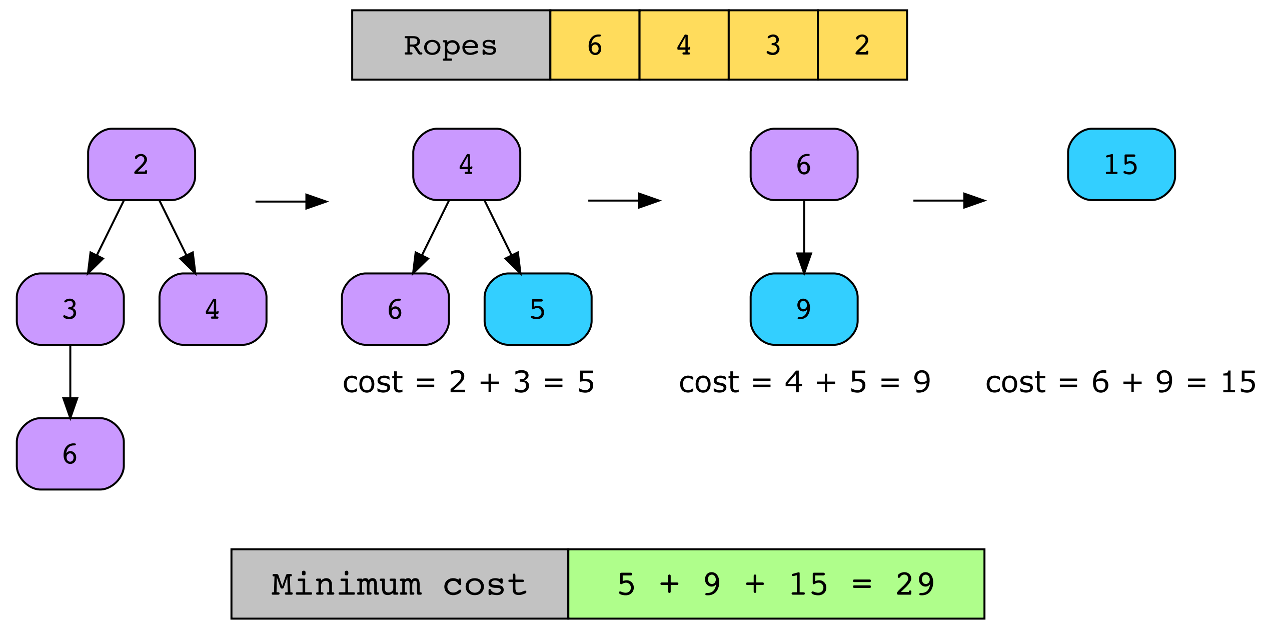
1. **Sort characters by frequency:** Sort a string in increasing order based on the frequency of its characters.

We'll calculate frequencies of each character and store them in the hash map. We'll use the frequencies to maintain the max-heap and sort characters by frequency.



2. **Connect n ropes with minimum cost:** Connect n ropes into one rope with minimum cost, such that the cost to connect two ropes is equal to the sum of their lengths.

First of all, insert all n ropes into the min-heap. Then, extract the minimum and the second minimum from the min-heap. Add the extracted values and insert the calculated value back to the min-heap. Maintain the total cost variable to keep track of minimum cost while calculating the cost.



Does your problem match this pattern?

- Yes, if both of these conditions are fulfilled:
    - **Unsorted list analysis:** We need to extract a specific subset of elements based on their size (largest or smallest), frequency (most or least frequent), or other similar criteria from an unsorted list. This may be the requirement of the final solution, or it may be necessary as an intermediate step toward the final solution.
    - **Identifying a specific subset:** The goal is to identify a subset rather than just a single extreme value. When phrases like top k kth largest/smallest, k most frequent, k closest, or k highest/lowest describe our task, it suggests the top k elements pattern is ideal for efficiently identifying a specific subset.
- No, if any of these conditions is fulfilled:

- **Presorted input:** The input data is already sorted according to the criteria relevant to solving the problem.
- **Single extreme value:** If only 1 extreme value (either the maximum or minimum) is required, that is, $k = 1$, as that problem can be solved in $O(n)$ with a simple linear scan through the input data.

Real-world problems

Many problems in the real world use the top K elements pattern. Let's look at some examples.

- **Location-based services in ride-sharing apps like Uber:** For a user requesting a ride, the app needs to find the nearest available drivers to ensure a quick pickup. However, it's not efficient or necessary to consider every driver in the city. Using the top k elements pattern, the app can efficiently determine the n closest drivers to the user's location. This involves sorting drivers based on their distance from the user but in a way that prioritizes computational efficiency, especially when n is much smaller than the total number of drivers.
- **Performance analysis in financial markets:** We can analyze broker performance by identifying those with the highest transaction volumes or other metrics of success. Given a dataset with broker IDs and their transaction volumes, the top K elements pattern can help sift through the data to highlight the brokers leading the market based on the frequency of their transactions or other relevant performance indicators.

- **Social media trend analysis:** Identifying the most popular or trending topics by analyzing hashtags or keywords. The top K elements pattern can help filter out the top topics based on their frequency over a certain time frame.

Strategy time!

Match the problems that can be solved using the top K elements pattern.

> **Note:** Select a problem in the left-hand column by clicking it, and then click one of the two options in the right-hand column.

Match The Answer

Select an option from the left-hand side

Rearrange the given string so that no two identical characters are adjacent to each other.	Top K Elements
Detect a cycle in a linked list.	Some other pattern
Find the five nearest points to the origin.	

Implement a stack in which the pop operation removes the most frequent element.

Reset Show Solution Submit