More on “Binary Heaps”

CSCI2100 Tutorial 8

Jianwen Zhao

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Introduction

In the previous lectures, we have implemented the priority queue (which supports \texttt{insert(e)} and \texttt{delete-min} operations) using a data structure called the \textit{binary heap} and achieved the following guarantees:

- $O(n)$ space consumption
- $O(\log n)$ insertion time
- $O(\log n)$ delete-min time

In this tutorial, we will try to enhance our understanding of the binary heap through some examples and exercises.
Example on Building a Binary Heap

Recap

Given an array $A$ that stores a set $S$ of $n$ integers, we can turn it into a binary heap on $S$ using the following simple algorithm, which views $A$ as complete binary tree $T$:

- For each $i = n$ downto 1
  - Perform root-fix on the subtree of $T$ rooted at $A[i]$

root-fix: Given a complete binary tree $T$ with root $r$. This operation guarantees that:

  - The left subtree of $r$ is a binary heap
  - The right subtree of $r$ is a binary heap
Example on Building a Binary Heap

Let $S = \{53, 27, 18, 91, 15, 3, 20, 37\}$, we store $S$ into an array $A$:

$$A: \begin{array}{cccccccc} 53 & 27 & 18 & 91 & 15 & 3 & 20 & 37 \end{array}$$

View $A$ as a complete binary tree $T$:

Now, we turn this complete binary tree into a binary heap.
Example on Building a Binary Heap

CSCI2100, The Chinese University of Hong Kong
Example on Insertion

Now we have built a binary heap. This building process just takes $O(n)$ time, which has been proved in our lecture.

Assume that we want to insert 12 into this binary heap. First, add 12 as a leaf, making sure that we still have a complete binary tree.

```
    3
   / \
  15 18
 / \ /  \
37 27 53 20
 / \     \
91 12
```
Then we fix the violations caused by this newly added element.

No more violations, insertion complete. An insertion can be processed in $O(\log n)$ time.
Example on Delete-min

Assume that we want to perform delete-min from this binary heap below:

First, find the rightmost leaf at the bottom level, namely, 37.
Example on Delete-min

Remove this leaf, but place the value 37 in the root.

```
         37
        /   \
       12    18
      /     /  \
    15     27   53
     |       |    |
    91      20
```
Example on Delete-min

Then we fix the violations caused by 37.

No more violations, delete-min complete. A delete-min can be processed in $O(\log n)$ time.
Problem

Suppose that we have \( k \) sorted arrays (in ascending order) \( A_1, A_2, \ldots, A_k \) of integers. Let \( n \) be the total number of integers in those arrays. Describe an algorithm to produce an array that sorts all the \( n \) integers in ascending order in \( O(n \log k) \) time.

Example

Suppose that \( k = 3 \), and the 3 arrays are as follows:

\[
\begin{align*}
A_1 & : 2 \ 23 \ 32 \ 35 \ 37 \\
A_2 & : 5 \ 10 \\
A_3 & : 33 \ 58 \ 82
\end{align*}
\]

Then you should produce an array \( B \) as below in \( O(n \log k) \) time.

\[
\begin{align*}
B & : 2 \ 5 \ 10 \ 23 \ 32 \ 33 \ 35 \ 37 \ 58 \ 82
\end{align*}
\]
Regular Exercise 8 Problem 5

Solution

Insert the smallest elements of each array into a binary heap $H$. This takes $O(k \log k)$ time. Then, repeat the following until $H$ is empty:

- Perform a `delete-min`. Let $e$ be the element fetched.
- Append $e$ to the output array.
- If $e$ comes from $A_i$ (for some $i$), obtain the next element from $A_i$, and insert it into $H$. If $A_i$ has been exhausted, then do nothing.
Example

Suppose that $k = 3$, and the 3 arrays are as follows:

$A_1$: 2  23  32  35  37  \quad A_2$: 5  10  \quad A_3$: 33  58  82

First, we insert the smallest elements of each array into a binary heap $H$:

2
   /
  5   33

Initially, the output array $B$ is empty.

B:  \\
   \\
   \\
   \\
   \\
   \\
   \\

**Regular Exercise 8 Problem 5**

**Example**

\[ A_1: 2 \ 23 \ 32 \ 35 \ 37 \quad A_2: 5 \ 10 \quad A_3: 33 \ 58 \ 82 \]

Then, we perform a **delete-min** on \( H \), and fetch \( e = 2 \), then append 2 to the output array \( B \). Since \( e \) comes from \( A_1 \), we obtain the next element 23 from \( A_1 \), and insert it into \( H \).

**Diagram:**

```
  2
 / \
5 33
⇒ delete-min ⇒ 33 ⇒ insert(23) ⇒ 33 23
```

Output array \( B \): 2
Example

\[ A_1: \begin{array}{cccccc}
2 & 23 & 32 & 35 & 37 \\
\end{array} \quad A_2: \begin{array}{ccc}
5 & 10 \\
\end{array} \quad A_3: \begin{array}{cccc}
33 & 58 & 82 \\
\end{array} \]

Perform another delete-min on the new \( H \), and fetch \( e = 5 \), then append 5 to the output array \( B \). Since \( e \) comes from \( A_2 \), we obtain the next element 10 from \( A_2 \), and insert it into \( H \).

Output array \( B \): 2 5
$A_1$: 2 23 32 35 37  $A_2$: 5 10  $A_3$: 33 58 82

10
\[ \frac{}{} \]
33 23
$\Rightarrow$ delete-min $\Rightarrow$

23
\[ \frac{}{} \]
33
$\Rightarrow$ do nothing

Output array $B$: 2 5 10

$A_1$: 2 23 32 35 37  $A_2$: 5 10  $A_3$: 33 58 82

23
\[ \frac{}{} \]
33
$\Rightarrow$ delete-min $\Rightarrow$

Output array $B$: 2 5 10 23
More on “Binary Heaps”

Output array $B$: 2 5 10 23 32
$A_1$: 2 23 32 35 37  $A_2$: 5 10  $A_3$: 33 58 82

35
down
58  $\Rightarrow$ delete-min $\Rightarrow$ 58  $\Rightarrow$ insert(37) $\Rightarrow$ 58

Output array $B$: 2 5 10 23 32 33 35

___________________________________________________________________________

$A_1$: 2 23 32 35 37  $A_2$: 5 10  $A_3$: 33 58 82

37
down
58  $\Rightarrow$ delete-min $\Rightarrow$ 58  $\Rightarrow$ $A_1$ exhausted  $\Rightarrow$ do nothing

Output array $B$: 2 5 10 23 32 33 35 37
Finally, we produce the output array $B$ with all the $n = 10$ elements sorted in ascending order:
Cost Analysis

- Insert the smallest elements of each array into a binary Heap $H$ takes $O(k \log k)$ time.
- Each delete-min and insertion require $O(\log k)$ time.
  - Since $H$ has at most $k$ elements.
- At most $n$ delete-min and $n$ insertions.
  - Since those arrays contain $n$ elements in total.

Overall, our algorithms takes $O(k \log k) + n \cdot O(\log k) = O(n \log k)$ time.
Special Exercise 8 Problem 6

Problem

Let \( S \) be a dynamic set of integers. At the beginning, \( S \) is empty. Then, new integers are added to it one by one, but never deleted. Let \( k \) be a fixed integer. Describe an algorithm which achieves the following guarantees:

- **Space consumption** \( O(k) \)
- **Insert\((e)\)**: Insert a new element \( e \) into \( S \), which takes at most \( O(\log k) \) time.
- **Report-top-\( k \)**: Report the \( k \) largest integers in \( S \).

Example

Suppose that \( k = 3 \), and the sequence of integers inserted is 83, 21, 66, 5, 24, 76, 92, 33, 43, \cdots. Your algorithm must be keeping \{83, 66, 24\} after the insertion of 24, \{83, 66, 76\} after the insertion of 76, and \{83, 76, 92\} after the insertion of 43.
Solution 1

We maintain a binary heap with $k$ elements, which obviously consumes $O(k)$ space.

- First, perform $k$ insertions to build a binary heap $H$ rooted at $r$ on the first inserted $k$ elements of $S$, and each insertion takes at most $O(\log k)$ time.

- For a newly inserted integer $e$, compare it with the root $r$ of $H$:
  - If $e > r$, replace $r$ with $e$, and perform root-fix on $H$.
    - This takes $O(\log k)$ time.
  - Otherwise, ignore $e$.

- Then, at any moment, $H$ contains the $k$ largest integers of $S$.
  - Report-top-$k = \text{Report}(H)$. 
Example

Suppose that the sequence of integers inserted is: 83, 21, 66, 5, 24, 76, 92, 33, 43, ⋯, and \( k = 3 \).

First of all, build a binary heap \( H \) on the first inserted 3 elements \{83, 21, 66\}: 

```
    21
   /  \
  83   66
```
Example

Suppose that the sequence of integers inserted is: 83, 21, 66, 5, 24, 76, 92, 33, 43, \ldots, and \( k = 3 \).

Next, we perform insertions one by one, and see what will happen on our binary heap \( H \):

\[
\begin{align*}
21 & \quad \Rightarrow \text{insert}(5) \Rightarrow \\
83 & \quad 66 \\
\Rightarrow \text{insert}(24) \Rightarrow \\
66 & \quad 24 \\
\Rightarrow \text{insert}(76) \Rightarrow \\
76 & \quad 92 \\
\Rightarrow \text{insert}(43) \Rightarrow
\end{align*}
\]
Solution 2

We maintain an array $A$ with length $2k$, which obviously consumes $O(k)$ space.

- First, append the first inserted $k$ elements of $S$ to $A$.
- Append the $i$-th ($i > k$) inserted integer of $S$ to $A$. Once $A$ is full, do the following:
  - Perform $k$-selection to find the $k$-th largest element of $A$, denoted by $v$.
  - Remove the elements which are smaller than $v$ from $A$.
  - Rearrange $A$ such that $A[1], A[2], \cdots, A[k]$ contains the $k$-largest element respectively.
  - **Report-top-$k$.**
Example

Suppose that the sequence of integers inserted is: 83, 21, 66, 5, 24, 76, 92, 33, 43, · · · , and $k = 3$.

First of all, creat an array $A$ with length $2k = 6$.

\[
A: \quad \square \quad \square \quad \square \quad \square \quad \square
\]

Then keep insertion one by one.
Special Exercise 8 Problem 6

Example

\[ S = \{83, 21, 66, 5, 24, 76, 92, 33, 43, \ldots \}, \ k = 3. \]

\[
\begin{array}{c}
83 \Rightarrow 83 \\
\Rightarrow 83 21 \Rightarrow 83 21 66 \\
\Rightarrow 83 21 66 5 \Rightarrow 83 21 66 5 24 \\
\Rightarrow 83 21 66 5 24 76
\end{array}
\]

A is full now. We perform \textit{k-selection} to find the 3rd-largest integer, which is 66. Then remove the elements which are smaller than 66 from \( A \):

\[
\begin{array}{c}
83 66 76 \Rightarrow \text{rearrange} \Rightarrow 83 66 76
\end{array}
\]

So the \textit{top-}k (top-3) elements are \( \{83, 66, 76\} \). We can continue insertion like this.
Cost Analysis

- Append the first inserted elements of \( S \) to \( A \) takes \( O(k) \) time.
- Keep insertion, once \( A \) is full, we perform \( k \)-selection to report top-\( k \), which takes \( O(k) \) time.
- Remove the elements and rearrange \( A \) takes \( O(k) \) time.

Overall, our algorithm takes \( O(k) \) time. Charge these costs to the \( k \) insertions indicated below, each insertion bears \( O(1) \) time, and each insertion is only charged once.