CSCI 2100 Tutorial 9

WU Hao
Outline

• A review on the binary heap

• Regular exercise 8 problem 4

• Special exercise 8 problem 4
Binary Heap (Review)

Let $S$ be a set of $n$ integers. A binary heap on $S$ is a binary tree $T$ satisfying:

1. $T$ is a complete binary tree.
2. Every node $u$ in $T$ stores a distinct integer in $S$, called the key of $u$.
3. If $u$ is an internal node, the key of $u$ is smaller than those of its child nodes.

The third property may be violated after insertion and delete-min.
Heap Property Violation

Original:

```
3
/   \
15   20
|    /
37  27  53  25
   |    \
  91   
```

After insertion:

```
3
/   \
15   20
|    /
37  27  53  25
   |    
  91   12
```

After delete-min:

```
91
/   \
15   20
|    /
37  27  53  25
   |    
```

Original:

```
3
/   \
15   20
|    /
37  27  53  25
   |    
  91   
```

After insertion:

```
3
/   \
15   20
|    /
37  27  53  25
   |    
  91   12
```

After delete-min:

```
91
/   \
15   20
|    /
37  27  53  25
   |    
```
Restoring the Heap Property
After Insertion

Swap up:
If node $u$ has a smaller key than its parent $p$, swap the keys of $u$ and $p$. Set $u$ to $p$, and repeat until there is no violation.
Swap Up

Swap up at most $O(\log n)$ times to restore the heap property.
Restoring the Heap Property
After Delete-min

Swap down:
Let \( v \) be the child of node \( u \) with a smaller key. If the key of \( u \) is larger than the key of \( v \), swap the keys of \( u \) and \( v \). Set \( u \) to \( v \), and repeat until there is no violation.
Swap Down

Swap down at most $O(\log n)$ times to restore the heap property.
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.
Solution 1: Merge Operation

• Input
  \( k = 8, \ n = 20 \)

\[
\begin{array}{cccc}
2 & 12 & 17 & 8 & 11 \\
1 & 25 & 23 & 28 & 6 & 9 & 10 & 3 & 18 & 19 \\
\end{array}
\]

Merge

\[
\begin{array}{cccc}
2 & 8 & 11 & 12 & 17 & 3 & 6 & 9 & 10 & 18 & 19 \\
1 & 23 & 25 & 28 & 5 & 7 & 15 & 30 & 40 \\
\end{array}
\]

Merge

\[
\begin{array}{cccc}
2 & 3 & 6 & 8 & 9 & 10 & 11 & 12 & 17 & 18 & 19 \\
1 & 5 & 7 & 15 & 23 & 25 & 28 & 30 & 40 \\
\end{array}
\]

8 arrays

4 arrays

2 arrays
Solution 1: Merge Operation

Need $O(\log k)$ passes. Each pass takes $O(n)$ time on $n$ integers (the cost of merging is proportional to the number of elements involved).

Therefore, the total time complexity is $O(n \log k)$. 
Solution 2: Binary Heap

• Input:
  k = 3, n = 15

[2 15 30 40 47]  [5 8 11 12]
[9 14 21 26 27 37]

• Output

[2 5 8 9 11 12 14 15 21 26 27 30 37 40 47]
Solution 2: Binary Heap

Ideas:

• A binary heap of size $k$ can perform delete-min and insertion in $O(\log k)$ time.

• Perform a delete-min to obtain the smallest integer that has not been output.

• After delete-min, insert a new integer into the heap from the integer’s origin array.
Solution 2: Binary Heap

```
2  15  30  40  47
9  14  21  26  27  37
```

```
2  15  30  40  47
9  14  21  26  27  37
```

```
2  15  30  40  47
9  14  21  26  27  37
```
Solution: Binary Heap

Initialization cost:
  creating the output array: $O(n)$

Processing cost:
  $n$ insertions: $O(n \log k)$  
  delete-min: $O(n \log k)$

Total time complexity:
$O(n \log k)$
Special Exercise 8 Problem 4

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 $S$ in $O(\log k)$ time.
- Report-top-$k$: Report the $k$ largest integers in $S$ in $O(k)$ time.
Example:

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

The 3 largest integers are 83, 66, 24 after the insertion of 24, they become 83, 66, 76 after the insertion of 76, and so on.
Solution

Intuition:

• A heap $H$ of size $k$ takes $O(k)$ space.
• $H$ performs insertion and delete-min in $O(\log k)$ time.
• The root $r$ of $H$ stores the minimal integer in $H$.
• Make sure that $H$ always contains the $k$ largest integers. If the incoming integer $m$ is larger than the minimal integer stored in $H$, we perform delete-min and insert($m$). Otherwise, we do nothing.
Solution

• Input:
83, 21, 66, 5, 24, 76, 92, 33, 43, ..., and k=3
Solution

Maintain a binary heap $H$ with $k$ integers.

1. Insert first $k$ integers into $H$. Each insertion takes $O(\log k)$ time.

2. For a newly added integer $e$ from the sequence, compare it with the integer $e_r$ stored at the root $r$ of $H$:

   (1) If $e > e_r$, perform delete-min and insert$(e)$, which take $O(\log k)$ time in total.

   (2) Otherwise, ignore $e$. 
Solution

Report-top-$k$: Report all integers in $H$ by traversing the heap.
A challenging problem for you

• For this problem, we can actually achieve
  • $O(k)$ space
  • $O(1)$ amortized insertion time
  • $O(k)$ top-k report time.

• Hint: $k$-selection.