

CSCI 136:
Data Structures
and
Advanced Programming
Lecture 33
Heap implementation

Instructor: Kelly Shaw
Williams

Topics

Heapsort
Which data structure?

Announcements

1. **Final exam**: Saturday, Dec 17, 1:30pm.
Room TBD.
2. **Final exam review session**,
in class, last day of class, **Friday 12/9**.

Your to-dos

1. Lab 10 (partner lab), **due Tuesday 12/6 by 10pm**.
2. **Review readings** from *Bailey*.
3. **Study** for the final exam.
 - a. Pro tip: **review quizzes**.
 - b. **Do problems** in study guide/practice exam.
 - c. **Don't stress out!** Just be methodical and do your best.
4. **Work on resubmissions** you plan to submit.

Announcements



Amy Babay, University of Pittsburgh

Friday, Dec 9 @ 2:35pm
Computer Science Colloquium – Wege TCL 123
Threats to Critical Infrastructure Control Systems

Amy runs the Resilient Systems and Societies Lab at the University of Pittsburgh School of Computing and Information, focusing on dependable infrastructure. The lab's work aims to make the networked systems our society relies on resilient to failures and attacks, and to develop new network technologies that help bring people together.

Amy conducts research on distributed systems and computer networks, aiming to not only fundamentally advance our understanding of how to design and build robust, performant systems that can be relied on to meet their requirements (despite failures and/or attacks), but also develop practical solutions with positive societal impact.

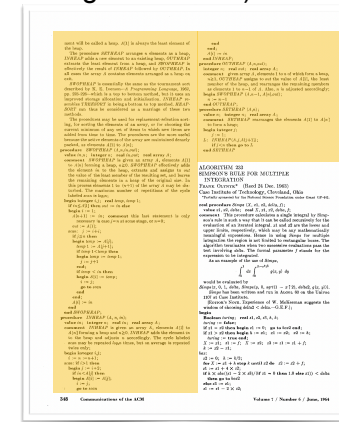
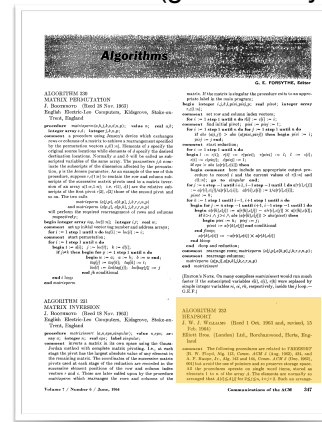
Heapsort

Heapsort

Heapsort is an **in-place, unstable sorting algorithm** that uses a binary min- or max-heap to order elements. The algorithm proceeds by first constructing a heap, then it removes elements one at a time to build a sorted array.

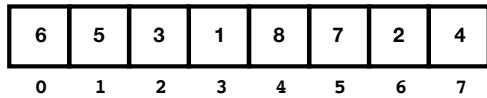
Like bubble sort, heapsort maintains the **invariant** that the rightmost **numSorted** elements are sorted. In other words, as elements are removed from the heap, they are inserted into the free space leftover after removal.

- Invented in 1963.
- Communications of the ACM generously gave the author two columns (glamorously titled “algorithm 232”).



Heapsort: example

Suppose we have the following array of values,



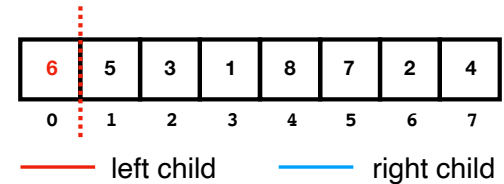
and that we want to produce an array in **ascending order** (i.e., from smallest to largest).

We first construct a **max heap** and then **remove** all the elements.

(If sorting in **descending order**, construct a **min heap**)

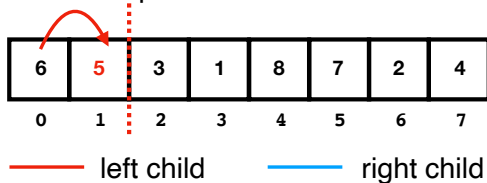
Heapsort: example

The first element is the new root.



Heapsort: example

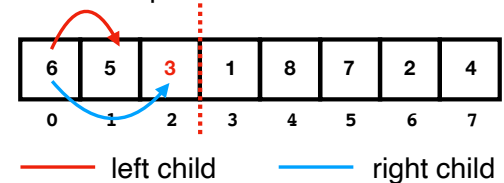
Add 5 to the max heap.



Since 5 is smaller than 6, it stays where it is (as the left child of 6).

Heapsort: example

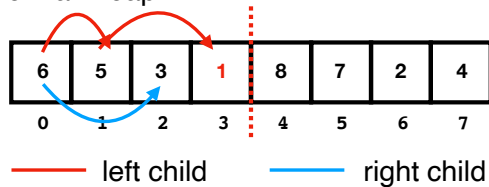
Add 3 to the max heap.



Since 3 is smaller than 6, it stays where it is (as the right child of 6).

Heapsort: example

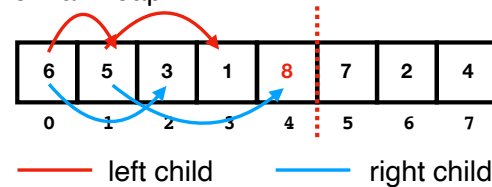
Add 1 to the max heap.



Since 1 is smaller than 5, it stays where it is (as the left child of 5).

Heapsort: example

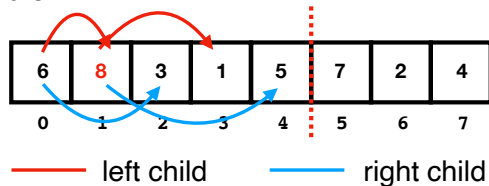
Add 8 to the max heap.



8 is not smaller than 5. So we run the “up heap” procedure.

Heapsort: example

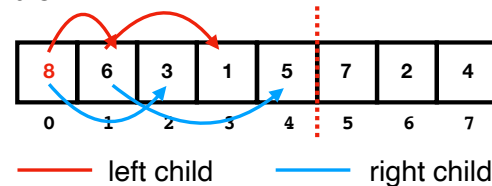
Swap 5 and 8.



8 is also not smaller than 6. So we run the “up heap” procedure.

Heapsort: example

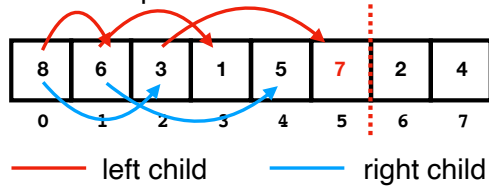
Swap 6 and 8.



Now 8 is in the correct location.

Heapsort: example

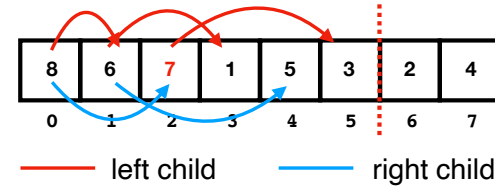
Add 7 to the max heap.



7 is not smaller than 3. So we run the “up heap” procedure.

Heapsort: example

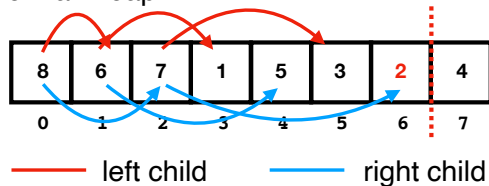
Swap 3 and 7.



7 is smaller than 8. So 7 is in the correct location.

Heapsort: example

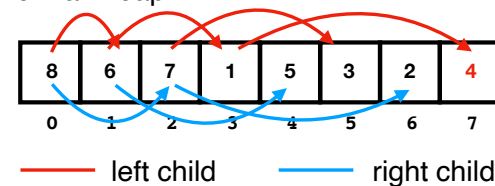
Add 2 to the max heap.



Since 2 is smaller than 7, it stays where it is (as the right child of 7).

Heapsort: example

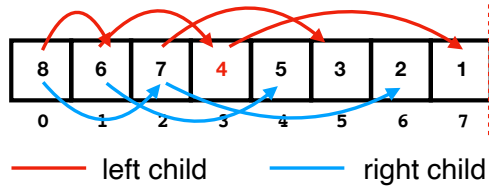
Add 4 to the max heap.



4 is not smaller than 1. So we run the “up heap” procedure.

Heapsort: example

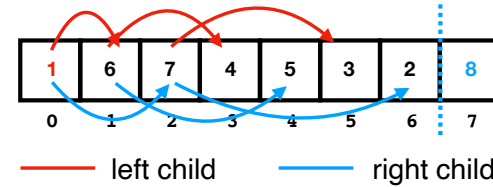
Swap 1 and 4.



4 is smaller than 6. So 4 is in the correct location.

Heapsort: example

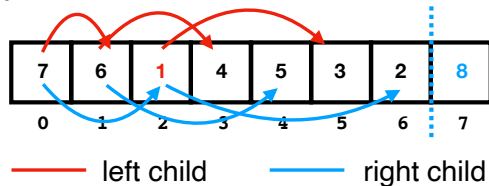
Extract the max by swapping it with the **size-1-numsortedth** element.



Since 1 is not bigger than 6 and 7, run the “down heap” procedure.

Heapsort: example

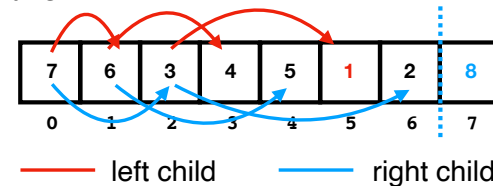
Swap 1 with 7.



Since 1 is not bigger than 3 and 2, run the “down heap” procedure.

Heapsort: example

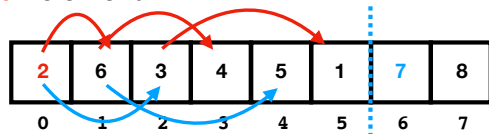
Swap 1 with 3.



Since 1 is now a leaf, we are done swapping.

Heapsort: example

Extract the max by swapping it with the **size-1-numsortedth** element.

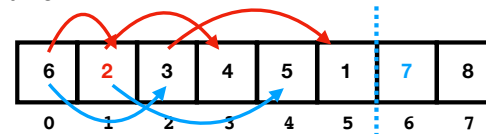


— left child — right child

Since 2 is not bigger than 6 and 3, run the “down heap” procedure.

Heapsort: example

Swap 2 with 6.

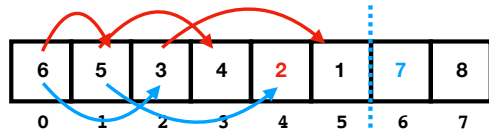


— left child — right child

Since 2 is not bigger than 4 and 5, run the “down heap” procedure.

Heapsort: example

Swap 2 with 5.

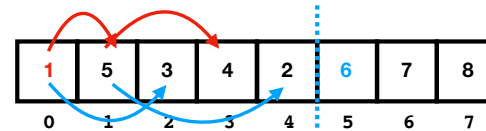


— left child — right child

Since 2 is now a leaf, we are done swapping.

Heapsort: example

Extract the max by swapping it with the **size-1-numsortedth** element.

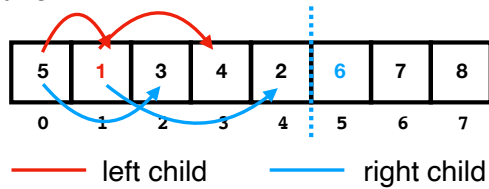


— left child — right child

Since 1 is not bigger than 5 and 3, run the “down heap” procedure.

Heapsort: example

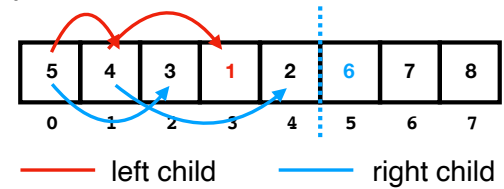
Swap 1 with 5.



Since 1 is not bigger than 4 and 2, run the “down heap” procedure.

Heapsort: example

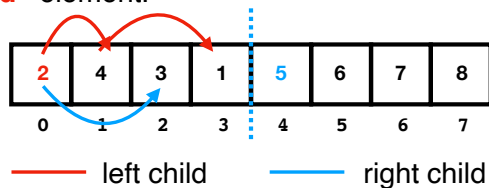
Swap 1 with 4.



Since 1 is now a leaf, we are done swapping.

Heapsort: example

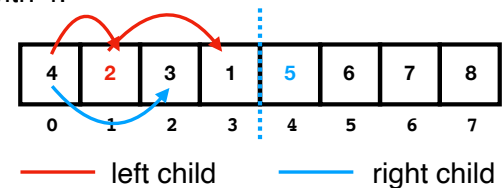
Extract the max by swapping it with the **size-1-numsortedth** element.



Since 2 is not bigger than 4 and 3, run the “down heap” procedure.

Heapsort: example

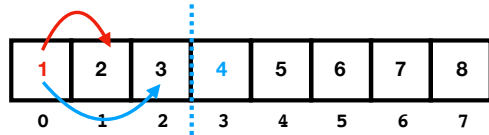
Swap 2 with 4.



Since 2 is bigger than 1, 2 is in the correct location.

Heapsort: example

Extract the max by swapping it with the **size-1-numsortedth** element.

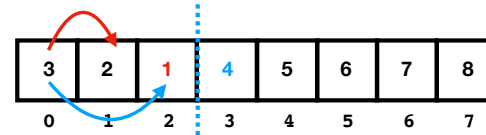


— left child — right child

Since 1 is not bigger than 2 and 3, run the “down heap” procedure.

Heapsort: example

Swap 1 with 3.

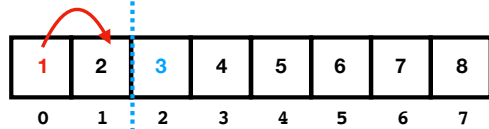


— left child — right child

Since 1 is now a leaf, we are done swapping.

Heapsort: example

Extract the max by swapping it with the **size-1-numsortedth** element.

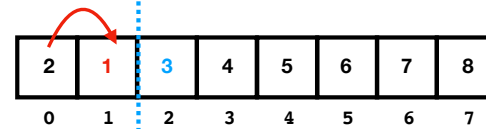


— left child — right child

Since 1 is not bigger than 2, run the “down heap” procedure.

Heapsort: example

Swap 1 with 2.

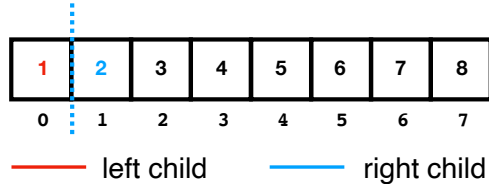


— left child — right child

Since 1 is now a leaf, we are done swapping.

Heapsort: example

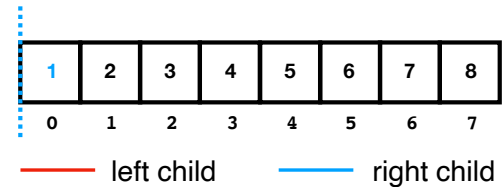
Extract the max by swapping it with the **size-1-numsortedth** element.



But since the heap only contains one element, it must be the max, so...

Heapsort: example

Done.



Activity:

Which data structure should I choose?

For each of the following **scenarios**, **any** of the data structures we've discussed this semester are possible choices.

However, not every data structure is a **good** choice.

Spend some time working with a partner and choose what you think is the **best data structure** for that **scenario**.

Justify your choice by **writing down the reasons** you chose it (hint: asymptotic arguments about space and time are good justifications!)

Activity:

Which data structure should I choose?

- You want to count occurrences of each word in a document then print an alphabetical list of word frequencies.
- You are writing code that will be used to store thousands of records (e.g. each record contains all the academic information for a specific student) and to retrieve them using a key (e.g. student name). The data rarely changes.
- Assume you are given a collection of pairs as input. Each pair contains the names of two people. Taken together, the pairs describe a social network. Over time, you will add pairs when friendships begin and remove them when friendships end. Data changes frequently. At any point in time, you want to be able to find the k most "popular" people.
- You want to count the occurrences of each letter in a document, then print an alphabetical list of letter frequencies.

Recap & Next Class

Today:

Heapsort

Which data structure should I choose?

Next class:

Dijkstra's algorithm