CSE 373 Analysis of Algorithms
Fall 2016
Instructor: Prof. Sael Lee

LEC05 SORTING I : HEAPSORT / PRIORITY QUEUES (PG. 103-120)

Lecture slide courtesy of Prof. Steven Skiena

PROBLEM OF THE DAY

Problem: You are given the task of reading n numbers and then printing them out in sorted order. Suppose you have access to a balanced dictionary data structure, which supports the operations search, insert, delete, minimum, maximum, successor, and predecessor each in $O(\log n)$ time.

1. How can you sort in $O(n \log n)$ time using only insert and in-order traversal?
2. How can you sort in $O(n \log n)$ time using only minimum, successor, and insert?
3. How can you sort in $O(n \log n)$ time using only minimum, insert, delete, search?

Solution:

```
Sort1()
        initialize-tree(t)
        While (not EOF)
                read(x);
                insert(x,t)
        Traverse(t)
```

1. We can build a search tree by inserting all *n* elements, then do a traversal to access the items in sorted order:

2. We can start from the minimum element, and then repeatedly find the successor to traverse the elements in sorted order.

3. We can repeatedly find and delete the minimum element to once again traverse all the elements in sorted order.

```
Sort2()
        initialize-tree(t)
        While (not EOF)
                read(x);
                insert(x,t);
        y = Minimum(t)
        While (y ≠ NULL) do
                print(y → item)
                y = Successor(y,t)
```

```
Sort3()
        initialize-tree(t)
        While (not EOF)
                read(x);
                insert(x,t);
        y = Minimum(t)
        While (y ≠ NULL) do
                print(y→item)
                Delete(y,t)
                y = Minimum(t)
```

Each of these algorithms does a linear number of logarithmic-time operations, and hence runs in $O(n \log n)$ time.

IMPORTANCE OF SORTING

- 1. Sorting is the basic building block that many other algorithms are built around.
- 2. Computers spend more time sorting than anything else, historically 25% on mainframes.
- 3. Sorting is the best studied problem in computer science, with a variety of different algorithms known.
- 4. Most of the interesting ideas we will encounter in the course can be taught in the context of sorting, such as divide-and-conquer, randomized algorithms, and lower bounds.

You should have seen most of the algorithms - we will concen trate on the analysis.

EFFICIENCY OF SORTING

- Sorting is important because that once a set of items is sorted, many other problems become easy.
- Further, using *O(n log n)* sorting algorithms leads naturally to sub-quadratic algorithms (o(n^2)) for these problems.

n	$n^2/4$	$n \lg n$
10	*25*	33
100	2,500	*664*
1,000	250,000	*9,965*
10,000	25,000,000	*132,877*
100,000	2,500,000,000	*1,660,960*

- Large-scale data processing would be impossible if sorting took $\Omega(n^2)$ time.

APPLICATION OF SORTING: SEARCHING

- Solution approach: After sorting, binary search lets you test whether an item is in a dictionary in *O(lg n)* time.

- **Search preprocessing** is perhaps the single most important application of sorting.

APPLICATION OF SORTING: 1-D CLOSEST PAIR

- Problem: Given n numbers, find the pair which are closest to each other.

- Solution approach: Once the numbers are sorted, the closest pair will be next to each other in sorted order.
- Analysis: An *O(n)* linear scan completes the job, for a total of $O(n \log n)$ time including the sorting

APPLICATION OF SORTING: 1-D ELEMENT UNIQUENESS

- Problem: Given a set of n items, are they all unique or are there any duplicates?

- Solution Approach: Sort them and do a linear scan to check all adjacent pairs.
- This is a special case of closest pair above.

APPLICATION OF SORTING: MODE

- Problem: Given a set of n items, which element occurs the largest number of times? More generally, compute the frequency distribution.
- Solution Approach 1: Sort them and do a linear scan to measure the length of all adjacent runs.
- Solution Approach 2 : The number of instances of k in a sorted array can be found in $O(\log n)$ by using binary search to look for the positions of both $k - \varepsilon$ and $k + \varepsilon$ and take the difference.

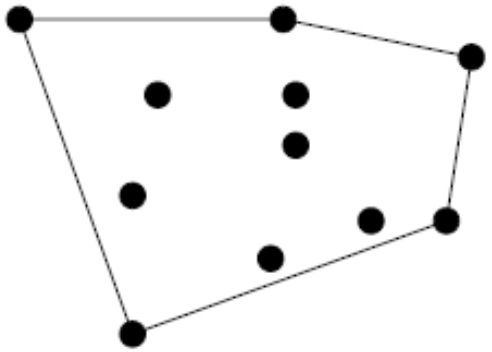
APPLICATION OF SORTING: MEDIAN AND SELECTION

- Problem: What is the kth largest item in the set?
- Solution Approach: Once the keys are placed in sorted order in an array, the kth largest can be found in **constant time** by simply looking in the kth position of the array.

- * There is a linear time algorithm for this problem, but the idea comes from partial sorting.

APPLICATION OF SORTING: CONVEX HULLS

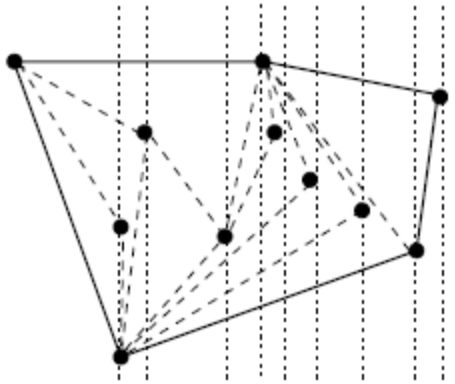
- Problem: Given n points in 2-D, find the smallest area polygon which contains them all.



- The convex hull is like a rubber band stretched over the points.
- Convex hulls are the most important building block for more sophisticated geometric algorithms.

FINDING CONVEX HULLS

- Once you have the points sorted by x-coordinate, they can be inserted from left to right into the hull, since the rightmost point is always on the boundary.



- Sorting eliminates the need check whether points are inside the current hull.
- Adding a new point might cause others to be deleted which can be quickly identify by checking whether they lie inside the polygon formed by adding the new point.
- The total time is linear after the sorting has been done.

PRAGMATICS OF SORTING: COMPARISON FUNCTIONS

- Alphabetizing is the sorting of text strings.
- Libraries have very complete and complicated rules concerning the relative *collating sequence* of characters and punctuation.
  - Is *Skiena* the same key as *skiena*?
  - Is *Brown-Williams* before or after *Brown America* before or after *Brown, John*?
- Explicitly controlling the order of keys is the job of the *comparison function* we apply to each pair of elements.
- This is how you resolve the question of increasing or decreasing order.

PRAGMATICS OF SORTING: EQUAL ELEMENTS

- Elements with equal key values will all bunch together in any total order, but sometimes the relative order among these keys matters.
- Sorting algorithms that always leave equal items in the same relative order as in the original permutation are called ***stable***.
- Unfortunately very few fast algorithms are stable, but *Stability* can be achieved by adding the initial position as a secondary key.

PRAGMATICS OF SORTING: LIBRARY FUNCTIONS

- Any reasonable programming language has a built-in sort routine as a library function.
- You are almost always better off using the system sort than writing your own routine.
- For example, the standard library for C contains the function `qsort` for sorting:

```
#include <stdlib.h>
void qsort(void *base, size t nel, size t width,
        int (*compare) (const void *, const void *));
```

sorts the first **nel** elements of an array (pointed to by **base**), where each element is **width**-bytes long using **compare** function.

SELECTION SORT

- Selection sort scans through the entire array, repeatedly finding the smallest remaining element.

```
SelectionSort(A)
    For i = 1 to n do
        T(A): Sort[i] = Find-Minimum from A
        T(B): Delete-Minimum from A
    Return(Sort)
```

- Selection sort takes $O(n(T(A) + T(B))$ time.

THE DATA STRUCTURE MATTERS

- Using arrays or unsorted linked lists as the data structure,
  - operation A takes O(n) time and
  - operation B takes O(1),

  for an $O(n^2)$ selection sort.

Key question: "Can we use a different data structure?"

- Using balanced search trees or heaps:
  - both of these operations can be done within O(lg n) time,

  for an O(n log n) selection sort, balancing the work and achieving a better tradeoff.
- Selection sort using a Heap is called a **Heapsort**.

PRIORITY QUEUES

Textbook Section 3.5

- Priority queues are data structures which provide extra flexibility over sorting.
- This is important because jobs often enter a system at arbitrary intervals.
- It is more cost-effective to insert a new job into a priority queue than to re-sort everything on each new arrival.

PRIORITY QUEUE OPERATIONS

Priority queue supports three primary operations:

- ***Insert(Q,x)***: Given an item x with key k, insert it into the priority queue Q.
- ***Find-Minimum(Q)*** or ***Find-Maximum(Q)***: Return a pointer to the item whose key value is smaller (larger) than any other key in the priority queue Q.
- ***Delete-Minimum(Q)*** or ***Delete-Maximum(Q):*** Remove the item from the priority queue Q whose key is minimum (maximum).

Each of these operations can be easily supported using heaps or balanced binary trees in O(log n).

BASIC PRIORITY QUEUE IMPLEMENTATIONS

	Unsorted Array	Sorted Array	Balanced Tree	Heap
Insert(*Q*, *x*)	O(1)	O(n)	O(lgn)	**O(lgn)**
Find-Min(*Q*)*	O(1)	O(1)	O(1)	**O(1)**
Delete-Min(*Q*)	O(n)	O(1)	O(lgn)	**O(lgn)**

* Making Find-Min constant:

Use extra variable to store a pointer/index to the minimum entry.

Updating this pointer on each insertion is easy

— we update it if and only if the newly inserted value is less than the current minimum.

* What happened for delete-minimum?

We can delete the minimum entry,

then do an honest find-minimum to restore our pointer/index to min value.

APPLICATIONS OF PRIORITY QUEUES: DATING

- What data structure should be used to suggest who to ask out next for a date?
- It needs to support retrieval by *desirability*, not name.
- Desirability changes (up or down), so you can re-insert the max with the new score after each date.
- New people you meet get inserted with your observed desirability level.
- There is never a reason to delete anyone until they arise to the top

APPLICATIONS OF PRIORITY QUEUES: DISCRETE EVENT SIMULATIONS

- In simulations of airports, and parking lots – priority queues can be used to maintain who goes next.
- The stack and queue orders are just special cases of orderings.
- In real life, certain people cut in line.

APPLICATIONS OF PRIORITY QUEUES: GREEDY ALGORITHMS

- In greedy algorithms, we always pick the next thing which locally maximizes our score. By placing all the things in a priority queue and pulling them off in order, we can improve performance over linear search or sorting, particularly if the weights change.
- War Story: sequential strips in triangulations