Sorting and Searching

Sequential Search

```
int sequentialSearch( int[] a, int item, int n){
   for (int i = 0; i < n && a[i]!= item; i++);
   if (i == n)
       return -1;
   return i;
}
```

Unsuccessful Search: ➔ O(n)

Successful Search:

Best-Case: *item* is in the first location of the array ➔ O(1)
Worst-Case: *item* is in the last location of the array ➔ O(n)
Average-Case: The number of key comparisons 1, 2, ..., n

$$\frac{\sum_{i=1}^{n} i}{n} = \frac{(n^2+n)/2}{n}$$

➔ O(n)

Binary Search

```
int binarySearch( int[] a, int size, int x) {
   int low =0;
   int high = size -1;
   int mid;          // mid will be the index of
                     // target when it's found.
   while (low <= high) {
     mid = (low + high)/2;
     if (a[mid] < x)
        low = mid + 1;
     else if (a[mid] > x)
         high  = mid - 1;
     else
         return mid;
   }
   return -1;
}
```


Binary Search – Analysis

- For an unsuccessful search:
  - The number of iterations in the loop is $\lfloor \log_2 n \rfloor + 1$ ➔ $O(\log_2 n)$
- For a successful search:
  - ***Best-Case:*** The number of iterations is 1. ➔ $O(1)$
  - ***Worst-Case:*** The number of iterations is $\lfloor \log_2 n \rfloor + 1$ ➔ $O(\log_2 n)$
  - ***Average-Case:*** The avg. # of iterations < $\log_2 n$ ➔ $O(\log_2 n)$

0	1	2	3	4	5	6	7	← an array with size 8
3	2	3	1	3	2	3	4	← # of iterations

The average # of iterations = 21/8 < $\log_2 8$

How much better is $O(log_2 n)$?

n	$O(log_2 n)$
16	4
64	6
256	8
1024 (1KB)	10
16,384	14
131,072	17
262,144	18
524,288	19
1,048,576	20
1,073,741,824	30

Sorting

Importance of Sorting

Why don't CS profs ever stop talking about sorting?

1. Computers spend more time sorting than anything else, historically 25% on mainframes.

2. Sorting is the best studied problem in computer science, with a variety of different algorithms known.

3. 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.

(slide by Steven Skiena)

Sorting

- Organize data into ascending / descending order.
  - Useful in many applications
  - Any examples can you think of?

- Internal sort vs. external sort
  - We will analyze only internal sorting algorithms.

- Sorting also has other uses. It can make an algorithm faster.
  - e.g. Find the intersection of two sets.

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 for these problems.
- Large-scale data processing would be impossible if sorting took $O(n^2)$ time.

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*

(slide by Steven Skiena)

Applications of Sorting

- **Closest Pair:** Given n numbers, find the pair which are closest to each other.
  - Once the numbers are sorted, the closest pair will be next to each other in sorted order, so an O(n) linear scan completes the job. – Complexity of this process: O(??)
- **Element Uniqueness:** Given a set of n items, are they all unique or are there any duplicates?
  - Sort them and do a linear scan to check all adjacent pairs.
  - This is a special case of closest pair above.
  - Complexity?
- **Mode:** Given a set of n items, which element occurs the largest number of times? More generally, compute the frequency distribution.
  - How would you solve it?

Sorting Algorithms

- There are many sorting algorithms, such as:
  - Selection Sort
  - Insertion Sort
  - Bubble Sort
  - Merge Sort
  - Quick Sort

- First three sorting algorithms are not so efficient, but last two are efficient sorting algorithms.