Searching and Sorting

- Searching
  - Linear search
  - Binary search
- Sorting
  - Insertion sort
  - Merge sort

Philip Bille

Linear Search

- **Searching**
  - Linear search
  - Binary search
  - Time?
  - Challenge. Can we take advantage of the sorted order of the array?

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
1 4 7 12 16 18 25 28 31 33 36 42 45 47 50
```
**Binary Search**

- **Binary search.** Compare \( x \) to middle entry \( m \) in \( A \).
  - if \( A[m] = x \) return true and stop.
  - if \( A[m] < x \) continue recursively on the right half.
  - if \( A[m] > x \) continue recursively on the left half.
- If array size \( \leq 0 \) return false and stop.

\[
\begin{array}{cccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\
1 & 4 & 7 & 12 & 16 & 18 & 25 & 28 & 31 & 33 & 42 & 45 & 47 & 50
\end{array}
\]

**Binary Search**

- **Analysis 1.** Analogue of recursive peak algorithm.
  - A recursive call takes constant time.
  - Each recursive call halves the size of the array. We stop when the size is \( \leq 0 \).
  - Running time is \( \Theta(\log n) \)

\[
\text{BINARY SEARCH}(A,i,j,x)
\begin{align*}
&\text{if } j < i \text{ return false} \\
&m = \lceil i+j/2 \rceil \\
&\text{if } A[m] = x \text{ return true} \\
&\text{elseif } A[m] < x \text{ return BINARY SEARCH}(A,m+1,j,x) \\
&\text{else return BINARY SEARCH}(A,i, m-1, x) \quad // A[m] > x
\end{align*}
\]

\[
\begin{array}{cccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\
1 & 4 & 7 & 12 & 16 & 18 & 25 & 28 & 31 & 33 & 42 & 45 & 47 & 50
\end{array}
\]

**Binary Search**

- **Analysis 2.** Let \( T(n) \) be the running time for binary search.
  - Solve the recurrence relation for \( T(n) \).

\[
T(n) = \begin{cases} 
T(n/2) + c & \text{if } n > 1 \\
\text{d} & \text{otherwise}
\end{cases}
\]

\[
\begin{align*}
T(n) &= T\left(\frac{n}{2}\right) + c \\
&= T\left(\frac{n}{4}\right) + c + c \\
&= T\left(\frac{n}{8}\right) + c + c + c \\
&\vdots \\
&= T\left(\frac{n}{2^k}\right) + ck \\
&\vdots \\
&= T\left(\frac{n}{2^{\log_2 n}}\right) + c \log_2 n \\
&= T(1) + c \log_2 n \\
&= d + c \log_2 n \\
&= \Theta(\log n)
\end{align*}
\]

**Searching**

- We can search in
  - \( \Theta(n) \) time with linear search.
  - \( \Theta(\log n) \) time with binary search.
Searching and Sorting

- Searching
  - Linear search
  - Binary search
- Sorting
  - Insertion sort
  - Merge sort

Sorting

- Sorting. Given array A[0..n-1] return array B[0..n-1] with same values as A but in sorted order.

Application

- Obvious.
  - Sort list of names, show Google PageRank results, show social media feed in chronological order.
- Non obvious.
  - Data compression, computer graphics, bioinformatics, recommendations systems.
- Easy problem for sorted data.
  - Search, find median, find duplicates, find closest pair, find outliers.

Insertion Sort

- Insertion sort. Start with unsorted array A.
- Proceed left-to-right in n rounds.
- Round i:
  - Subarray A[0..i-1] is sorted.
Insertion Sort

\text{INSERTIONSORT}(A, n)
\begin{align*}
\text{for } i = 1 \text{ to } n-1 \\
\quad j = i \\
\quad \text{while } j > 0 \text{ and } A[j-1] > A[j] \\
\quad \quad \text{swap } A[j] \text{ og } A[j-1] \\
\quad \quad j = j - 1
\end{align*}

• Time?
  • To insert \(A[i]\) we use \(c \cdot i\) time for constant \(c\).
  • \(\Rightarrow\) total time \(T(n)\):
    \[ T(n) = \sum_{i=1}^{n-1} c = c \sum_{i=1}^{n-1} i = \frac{cn(n-1)}{2} = \Theta(n^2) \]
• Challenge. Can we sort faster?

Merge sort

• Merge sort.
  • Idea. Recursive sorting via merging sorted subarray.

Merge

• Goal. Combine two sorted array into a single sorted array.
  • Idea.
    • Scan both arrays left-to-right. In each step:
      • Insert smallest of the two entries in new array.
      • Move forward in array with smallest entry.
      • Repeat until input array exhausted.

Merge

• Time. Merging two arrays \(A_1\) og \(A_2\)?
  • Each step take \(\Theta(1)\) time.
  • Each step we move forward in one array.
  • \(\Rightarrow\) \(\Theta(|A_1| + |A_2|)\) time.
Merge sort

- Merge sort.
- If $|A| \leq 1$, return $A$.
- Otherwise:
  - Split $A$ into halves.
  - Sort each half recursively.
  - Merge the two halves.

\[
\begin{array}{cccccccccccc}
1 & 4 & 7 & 12 & 18 & 25 & 28 & 31 & 33 & 42 & 45 & 47 & 50 & 51 \\
4 & 7 & 12 & 18 & 25 & 28 & 31 & 33 & 42 & 45 & 47 & 50 & 51 & 16
\end{array}
\]

Time?
- Construct recursion tree.

\[
T(n) = cn \log_2 n + dn = \Theta(n \log_2 n)
\]
Sorting
• We can sort in
  • $\Theta(n^2)$ time with insertion sort.
  • $\Theta(n \log n)$ time with merge sort.

Divide and Conquer
• Merge sort is example of a divide and conquer algorithm.
  • Algorithmic design paradigm.
    • Divide. Split problem into subproblems.
    • Conquer. Solve subproblems recursively.
    • Combine. Combine solution for subproblem to a solution for problem.
• Merge sort.
  • Divide. Split array into halves.
  • Conquer. Sort each half.
  • Combine. Merge halves.