Donald E. Knuth (1938—)
The father of algorithm analysis
popularizing asymptotic notation.

The author of TAOCP -- The Art of Computer Programming

- Volume 1: Fundamental Algorithms
- Volume 2: Seminumerical Algorithms
- Volume 3: Sorting and Searching
- Volume 4: Combinatorial Algorithms
- Volume 5: Syntactical Algorithms
- Volume 6: Theory of Languages
- Volume 7: Compiler Programming Language Translation

“ I have been a happy man ever since January 1, 1990, when I no longer had an email address.”

Sorting: Arranging items into a certain order.

Items can be stored in an one dimensional array or linked list, or trees.

We prefer using one dimensional array

Why?

Insertion Sort:
Insert an item into the right place.

sorted

1 3 4 6 8 9 10 12

7
**Insertion Sort:** Insert an item into the right place.

<table>
<thead>
<tr>
<th>6</th>
<th>12</th>
<th>3</th>
<th>1</th>
<th>10</th>
<th>8</th>
<th>9</th>
<th>7</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>12</td>
<td>1</td>
<td>10</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>12</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

**Selection Sort:** Select the right item into the place.

<table>
<thead>
<tr>
<th>sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

*the minimum one*

**Analysis of Insertion/Selection Sorts:**

\[ T(n) = 1 + 2 + 3 + \ldots + n = O(n^2) \]

\[ O(n^{3/2}) \quad \text{Shell Sort in the worst case} \]

\[ O(n \log n) \quad \text{Lower bound of sorting algorithms} \]
Shell Sort: Incremental insertion

\[ h_1 < h_2 < \cdots < h_k \cdots < h_i \quad \text{with} \quad h_1 = 1 \]

Shell suggested: \( 1^2, 2^2, 3^2, \cdots 2^r \)

Not a good suggestion, worst case: \( O(n^2) \)

Shell Sort: using Shell’s suggestion \( 1, 2^2, 2^3, \cdots, 2^r \)

Worst case analysis: \( O(n^2) \)

Idea: Right before the final sort, let the smallest \( n/2 \) be distributed in the even position.

\[ \sum_{j=1}^{n/2} j = O(n^2) \]

Shell sort algorithm

void shell_sort(int a[], int size) {
    int h = size/2;
    while (h > 0) {
        for (int i = h; i < size; i++) {
            for (int j = i; j >= h && a[j] < a[j-h]; j -= h) {
                int t = a[j];
                a[j] = a[j-h];
                a[j-h] = t;
            }
        }
        h /= 2;
    }
}

If \( j \) is a linear combination of \( h_{j-2} \) and \( h_{j-1} \) with positive coefficients, then there is no need to check \( A[p-j] < A[p] \)

\[ xp + yq = \gcd(p, q) \]

if \( \gcd(p, q) = 1 \), then any number is a linear combination of \( p \) and \( q \)
If $\gcd(h_{k+1}, h_{k+2}) = 1$, then for sufficiently large $j$, we have that

\[ j = xh_{k+1} + yh_{k+2}, \quad x \geq 0, y \geq 0 \]


\[ (h_{k+1} - 1)(h_{k+2} - 1) = (2^{k+1} - 1)(2^{k+2} - 1) = (2 \times 2^{k+1} - 2) = (2(2^k - 1) + 2) = 8h_k^2 + 4h_k \]
/* Quick Sort */
void SwapTwo(int A[], int i, int j){
    int temp = A[i];
    A[i]=A[j];
    A[j]=temp;
}
void QuickSort(int A[], int h, int t){  // h: head  
    // t: tail
    if(h == t) return;
    int i=h+1, j=t;
    if (i==j) {
        if (A[h] > A[i]) SwapTwo(A,h,i);
        return;
    }
    while (i < j) {
        while (A[h] >= A[i] && i < t) i++;
        while (A[h] <= A[j] && j > h) j--;
        if (i < j) SwapTwo(A,i++,j--);
        if (i==j && A[h]>A[i]) SwapTwo(A,h,i);
    }
    QuickSort(A,h,i-1);
    QuickSort(A,i,t);
}