Efficiency and Computational Complexity (Part 2)
TYPES OF ORDERS OF GROWTH

- Constant
- Linear
- Quadratic
- Logarithmic
- \text{n log n}
- Exponential

BEST, AVERAGE, WORST CASES

- suppose you are given a list $L$ of some length $\text{len}(L)$

- **best case**: minimum running time over all possible inputs of a given size, $\text{len}(L)$
  - constant for `search_for_elmt`
  - first element in any list

- **average case**: average running time over all possible inputs of a given size, $\text{len}(L)$
  - practical measure

- **worst case**: maximum running time over all possible inputs of a given size, $\text{len}(L)$
  - linear in length of list for `search_for_elmt`
  - must search entire list and not find it

About Big O notation

- Big O notation measures an upper bound on the asymptotic growth, often called order of growth.

- Big O or $O()$ is used to describe worst case:
  - worst case occurs often and is the bottleneck when a program runs
  - express rate of growth of program relative to the input size
  - evaluate algorithm NOT machine or implementation
About Big O notation

- drop constants and multiplicative factors
- focus on **dominant terms**

\[
\begin{align*}
O(n^2) & : n^2 + 2n + 2 \\
O(n^2) & : n^2 + 100000n + 3^{1000} \\
O(n) & : \log(n) + n + 4 \\
O(n \log n) & : 0.0001 \times n \times \log(n) + 300n \\
O(3^n) & : 2n^{30} + 3^n
\end{align*}
\]
About Big O notation

Law of Addition for O():

- used with **sequential** statements
- $O(f(n)) + O(g(n))$ is $O(f(n) + g(n))$
- for example,
  ```python
  for i in range(n):
      print('a')
  for j in range(n*n):
      print('b')
  is $O(n) + O(n^2) = O(n+n^2) = O(n^2)$ because of dominant term
  ```
About Big O notation

**Law of Multiplication for O()**:
- used with **nested** statements/loops
- \( O(f(n)) \ast O(g(n)) \) is \( O(f(n) \ast g(n)) \)
- for example,
  ```python
  for i in range(n):
      for j in range(n):
          print('a')
  ```
  is \( O(n) \ast O(n) = O(n^2) \) because the outer loop goes \( n \) times and the inner loop goes \( n \) times for every outer loop iter.
About Big O notation

- $O(1)$ denotes constant running time
- $O(\log n)$ denotes logarithmic running time
- $O(n)$ denotes linear running time
- $O(n \log n)$ denotes log-linear running time
- $O(n^c)$ denotes polynomial running time ($c$ is a constant)
- $O(c^n)$ denotes exponential running time ($c$ is a constant being raised to a power based on size of input)

### About Big O notation

<table>
<thead>
<tr>
<th>CLASS</th>
<th>n=10</th>
<th>= 100</th>
<th>= 1000</th>
<th>= 100000</th>
</tr>
</thead>
<tbody>
<tr>
<td>O(1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>O(log n)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>O(n)</td>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>100000</td>
</tr>
<tr>
<td>O(n log n)</td>
<td>10</td>
<td>200</td>
<td>3000</td>
<td>600000</td>
</tr>
<tr>
<td>O(n^2)</td>
<td>100</td>
<td>10000</td>
<td>100000</td>
<td>10000000</td>
</tr>
<tr>
<td>O(2^n)</td>
<td>1024</td>
<td>12676506 00228229 40149670 3205376</td>
<td>1071508607186267320948425049060 00181056140481170555336074437503 8837035105112493612249319837881 5695858127594672917553146825187 1452856923140435984577574698574 8039345677748242309854210746050 6237114187795418215304647498358 1941267398767559165543946077062 9145711964776865421676604298316 52624386837205668069376</td>
<td><strong>Good luck!!</strong></td>
</tr>
</tbody>
</table>
Merging two sorted lists (arrays)

Given two sorted arrays, merge them to get a single ordered array

\[ a: \begin{array}{c} 15 \ 18 \ 42 \ 51 \end{array} \quad \text{elements} \]

\[ b: \begin{array}{c} 8 \ 11 \ 16 \ 17 \ 44 \ 58 \ 71 \ 74 \end{array} \quad \text{elements} \]

\[ c: \begin{array}{c} 8 \ 11 \ 15 \ 16 \ 17 \ 18 \ 42 \ 44 \ 51 \ 58 \ 71 \ 74 \end{array} \quad \text{elements} \]

Reference RG Dromey book
Merging two sorted lists (arrays)

\[ B[j] < A[i] \]
Merging two sorted lists (arrays)

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\hline
a & i & 15 & 18 & 42 & 51 & & & & & & \\
\hline
b & j & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
& & 8 & 11 & 16 & 17 & 44 & 58 & 71 & 74 & & & & \\
\hline
c & k & 8 & & & & & & & & & & & & \\
\end{array}
\]

- B[j] < A[i]
- C[k] = B[j]
Merging two sorted lists (arrays)

\[ \begin{align*}
 &\text{a} \quad 0 \quad 1 \quad 2 \quad 3 \\
 &\text{i} \quad 15 \quad 18 \quad 42 \quad 51 \\
 &\text{b} \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \\
 &\text{j} \quad 8 \quad 11 \quad 16 \quad 17 \quad 44 \quad 58 \quad 71 \quad 74 \\
 &\text{c} \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \\
 &\text{k} \quad 8 \\
\end{align*} \]

\[ \begin{align*}
 &j = j + 1 \\
 &k = k + 1
\end{align*} \]
Merging two sorted lists (arrays)

B[j] < A[i]
C[k] = B[j]
Merging two sorted lists (arrays)

\[
\begin{align*}
\text{a} & : i \\
\text{b} & : j = j + 1, k = k + 1 \\
\text{c} & : \\
\end{align*}
\]
Merging two sorted lists (arrays)

A[i] < B[j]
C[k] = A[i]
Merging two sorted lists (arrays)

\[ i = i + 1 \]
\[ k = k + 1 \]
Merging two sorted lists (arrays)

B\[j\] < A\[i\]  
C\[k\]=B\[j\]

a
\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
15 & 18 & 42 & 51 \\
i
\end{array}
\]

b
\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
8 & 11 & 16 & 17 & 44 & 58 & 71 & 74 \\
j
\end{array}
\]

c
\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
8 & 11 & 15 & 16 & & & & \\
k
\end{array}
\]
### Merging two sorted lists (arrays)

#### a

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>18</td>
<td>42</td>
<td>51</td>
</tr>
</tbody>
</table>

#### b

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>11</td>
<td>16</td>
<td>17</td>
<td>44</td>
<td>58</td>
<td>71</td>
<td>74</td>
</tr>
</tbody>
</table>

#### c

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>11</td>
<td>15</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
j = j + 1
k = k + 1
```
Merging two sorted lists (arrays)

B[j] < A[i]
C[k]=B[j]
Merging two sorted lists (arrays)

a

b

j=j+1
k=k+1
Merging two sorted lists (arrays)

A[i] < B[j]
C[k]=A[i]
Merging two sorted lists (arrays)

\[ a = \begin{array}{cccc}
0 & 1 & 2 & 3 \\
15 & 18 & 42 & 51 \\
\end{array} \]

\[ b = \begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
8 & 11 & 16 & 17 & 44 & 58 & 71 & 74 \\
\end{array} \]

\[ c = \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
8 & 11 & 15 & 16 & 17 & 18 & \text{ } & \text{ } & \text{ } & \text{ } & \text{ } & \text{ } \\
\end{array} \]

\[ i = i + 1 \]
\[ k = k + 1 \]
Merging two sorted lists (arrays)

\[
A[i] < B[j]
\]
\[
C[k] = A[i]
\]
Merging two sorted lists (arrays)

i = i + 1
k = k + 1

i

j

k
Merging two sorted lists (arrays)

B[j] < A[i]
C[k] = B[j]
Merging two sorted lists (arrays)

Merging two lists:

\[ a = [15, 18, 42, 51] \]
\[ b = [8, 11, 16, 17, 44, 58, 71, 74] \]

Initialize:

\[ i = 0 \]
\[ j = 0 \]
\[ k = 0 \]

Comparison:

While \( i \lt 4 \) and \( j \lt 8 \):

If \( a[i] \lt b[j] \):

\[ c[k] = a[i] \]
\[ i = i + 1 \]
\[ k = k + 1 \]

Else:

\[ c[k] = b[j] \]
\[ j = j + 1 \]
\[ k = k + 1 \]

End while

Final result:

\[ c = [15, 18, 42, 51, 8, 11, 16, 17, 44, 58, 71, 74] \]
Merging two sorted lists (arrays)

A[i] < B[j]
C[k] = A[i]
Merging two sorted lists (arrays)

Merging two sorted lists (arrays)

\[ \begin{array}{cccc}
0 & 1 & 2 & 3 \\
\color{orange}{15} & \color{brown}{18} & \color{teal}{42} & \color{orange}{51} \\
\end{array} \]

\[ \begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\color{green}{8} & \color{olive}{11} & \color{cyan}{16} & \color{olive}{17} & \color{teal}{44} & \color{green}{58} & \color{teal}{71} & \color{olive}{74} \\
\end{array} \]

\[ k = k + 1 \]

\[ \begin{array}{ccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\color{orange}{8} & \color{brown}{11} & \color{teal}{15} & \color{brown}{16} & \color{teal}{17} & \color{orange}{18} & \color{olive}{42} & \color{teal}{44} & \color{orange}{51} & \color{olive}{10} & \color{olive}{11} & \\
\end{array} \]
Merging two sorted lists (arrays)

a

0 1 2 3
15 18 42 51

i

b

0 1 2 3 4 5 6 7
8 11 16 17 44 58 71 74

j

c

0 1 2 3 4 5 6 7 8 9 10 11
8 11 15 16 17 18 42 44 51 58 71 74

k

copy the remaining B[j] to C[k]
Merging two sorted lists (arrays)

```python
# Algorithm Merge
def merge(A, B):
    C = []
    i, j = 0, 0
    while (i < len(A)) and (j < len(B)):
        if A[i] <= B[j]:
            C.append(A[i])
            i += 1
        else:
            C.append(B[j])
            j += 1
    C += A[i:]
    C += B[j:]
    return C
```

Order of time complexity

$O(p+q)$

$p = \text{len}(A)$
$q = \text{len}(B)$

$O(n)$
Given a randomly ordered array of $n$ elements, partition the elements into two subsets such that elements $\leq x$ are in one subset and elements $> x$ are in the other set.

$\text{Random set}$

$\text{A partitioned solution}$

$x = 17$

Reference RG Dromey book
Partitioning

Note: In place partitioning another array/list not allowed

Approach:

- Random set
- Left partition (growing to the right)
- Right partition (growing to the left)
def partition(arr,x):
    i = 0   # leftcounter
    j = len(arr)-1   # rightcounter
    while i < j and arr[i] <= x:    # position for the first left counter
        i += 1
    while i < j and arr[j] > x:     # position for the first right counter
        j -= 1
    if arr[j]>x:    # special case when x is less than all elements
        j -= 1
    while i<j:
        temp = arr[i]
        arr[i] = arr[j]
        arr[j] = temp
        i+=1
        j-=1
    while arr[i] <= x:   i += 1
    while arr[j] > x:    j -= 1
return j
def partition(arr, x):
    i = 0  # leftcounter
    j = len(arr) - 1  # rightcounter
    while i < j and arr[i] <= x:  # position for the first left counter
        i += 1
    while i < j and arr[j] > x:  # position for the first right counter
        j -= 1
    if arr[j] > x:  # special case when x is less than all elements
        j -= 1
    while i < j:
        temp = arr[i]
        arr[i] = arr[j]
        arr[j] = temp
        i += 1
        j -= 1
    while arr[i] <= x: i += 1
    while arr[j] > x: j -= 1
    return j

Order of time complexity
O(n)