Analysis of Algorithms
Outline

1 Performance

2 Time Complexity

3 Space Complexity
Performance

Algorithms are methods for solving computational problems.

Data structures are schemes for arranging data, amenable to efficient processing by algorithms.

The performance characteristics of a program is determined by:

- its time complexity, i.e., how long it takes;
- its space complexity, i.e., how much memory it needs.

The execution time of a program of size \( n \) is a function \( f(n) \) determined from the cost of executing each statement, and the frequency of execution of each statement.

The running time \( T(n) \) of the program is an approximation of \( f(n) \) obtained by ignoring any lower-order terms and constant coefficients.

For example, if \( f(n) = 31n^2 + 78n + 42 \), then \( T(n) = n^2 \).
Performance

Algorithms are methods for solving computational problems
Performance

Algorithms are methods for solving computational problems

Data structures are schemes for arranging data, amenable to efficient processing by algorithms
Algorithms are methods for solving computational problems.

Data structures are schemes for arranging data, amenable to efficient processing by algorithms.

The performance characteristics of a program is determined by:

- its time complexity, i.e., how long it takes; and
- its space complexity, i.e., how much memory it needs.
Performance

Algorithms are methods for solving computational problems

Data structures are schemes for arranging data, amenable to efficient processing by algorithms

The performance characteristics of a program is determined by

- its time complexity, ie, how long it takes; and
- its space complexity, ie, how much memory it needs

The execution time of a program of size $n$ is a function $f(n)$ determined from the cost of executing each statement, and the frequency of execution of each statement
Performance

Algorithms are methods for solving computational problems.

Data structures are schemes for arranging data, amenable to efficient processing by algorithms.

The performance characteristics of a program is determined by:
- its time complexity, i.e., how long it takes; and
- its space complexity, i.e., how much memory it needs.

The execution time of a program of size $n$ is a function $f(n)$ determined from the cost of executing each statement, and the frequency of execution of each statement.

The running time $T(n)$ of the program is an approximation of $f(n)$ obtained by ignoring any lower-order terms and constant coefficients.
Algorithms are methods for solving computational problems.

Data structures are schemes for arranging data, amenable to efficient processing by algorithms.

The performance characteristics of a program is determined by

- its time complexity, i.e., how long it takes; and
- its space complexity, i.e., how much memory it needs.

The execution time of a program of size $n$ is a function $f(n)$ determined from the cost of executing each statement, and the frequency of execution of each statement.

The running time $T(n)$ of the program is an approximation of $f(n)$ obtained by ignoring any lower-order terms and constant coefficients.

For example, if $f(n) = 31n^2 + 78n + 42$, then $T(n) = n^2$. 

Time Complexity

Program: threesum.py

- Command-line input: filename (String)
- Standard output: the number of unordered triples \((x, y, z)\) from the file such that \(x + y + z = 0\)

```
$ cat ../data/1Kints.txt
324110
-442472
...
745942
$ /usr/bin/time --format='%e seconds' python3 threesum.py ../data/1Kints.txt
70
0.7 seconds

$ /usr/bin/time --format='%e seconds' python3 threesum.py ../data/2Kints.txt
528
5.9 seconds
```
Program: threesum.py

Time Complexity

- Command-line input: filename (String)
- Standard output: the number of unordered triples $(x, y, z)$ from the file such that $x + y + z = 0$

```bash
~/workspace/dsa/programs
$ cat ../data/1Kints.txt
... 
745942
$ /usr/bin/time --format='%e seconds' python3 threesum.py ../data/1Kints.txt
700.7 seconds
$ /usr/bin/time --format='%e seconds' python3 threesum.py ../data/2Kints.txt
5285.9 seconds
```
Program: threesum.py

- Command-line input: filename (String)
Program: threesum.py
- Command-line input: filename (String)
- Standard output: the number of unordered triples \((x, y, z)\) from the file such that \(x + y + z = 0\)
Program: threesum.py

- Command-line input: filename (String)
- Standard output: the number of unordered triples \((x, y, z)\) from the file such that \(x + y + z = 0\)

```
> ~/workspace/dsa/programs
$ cat ../data/1Kints.txt
 324110
-442472
...
745942
$ /usr/bin/time --format='%e seconds' python3 threesum.py ../data/1Kints.txt
70
0.7 seconds
$ /usr/bin/time --format='%e seconds' python3 threesum.py ../data/2Kints.txt
528
5.9 seconds
```
```python
from instream import InStream
import stdio
import sys

def main():
inStream = InStream(sys.argv[1])
a = inStream.readAllInts()
stdio.writeln(count(a))

def count(a):
n = len(a)
count = 0
for i in range(0, n):
    for j in range(i + 1, n):
        for k in range(j + 1, n):
            if a[i] + a[j] + a[k] == 0:
                count += 1

return count

if __name__ == '__main__':
    main()
```

Time Complexity
```python
from instream import InStream
import stdio
import sys

def main():
    inStream = InStream(sys.argv[1])
    a = inStream.readInts()
    stdio.writeln(count(a))

def count(a):
    n = len(a)
    count = 0
    for i in range(0, n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if a[i] + a[j] + a[k] == 0:
                    count += 1
    return count

if __name__ == '__main__':
    main()
```
Time Complexity

- $T(n) = n^3$
- $f(n) = 0.2273121n^3 + 0.007625303n^2 + 0.006868505n + 0.01817256$

<table>
<thead>
<tr>
<th>Size</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>0.28</td>
</tr>
<tr>
<td>2K</td>
<td>1.80</td>
</tr>
<tr>
<td>4K</td>
<td>14.06</td>
</tr>
<tr>
<td>8K</td>
<td>111.83</td>
</tr>
<tr>
<td>16K</td>
<td>892.19</td>
</tr>
</tbody>
</table>
## Time Complexity

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>0.28s</td>
</tr>
<tr>
<td>2K</td>
<td>1.8s</td>
</tr>
<tr>
<td>4K</td>
<td>14.06s</td>
</tr>
<tr>
<td>8K</td>
<td>111.83s</td>
</tr>
<tr>
<td>16K</td>
<td>892.19s</td>
</tr>
</tbody>
</table>
Time Complexity

<table>
<thead>
<tr>
<th>n</th>
<th>f(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>0.28s</td>
</tr>
<tr>
<td>2K</td>
<td>1.8s</td>
</tr>
<tr>
<td>4K</td>
<td>14.06s</td>
</tr>
<tr>
<td>8K</td>
<td>111.83s</td>
</tr>
<tr>
<td>16K</td>
<td>892.19s</td>
</tr>
</tbody>
</table>

\[ f(n) = 0.2273121n^3 + 0.007625303n^2 + 0.006868505n + 0.01817256 \]
Time Complexity

<table>
<thead>
<tr>
<th>n</th>
<th>f(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>0.28s</td>
</tr>
<tr>
<td>2K</td>
<td>1.8s</td>
</tr>
<tr>
<td>4K</td>
<td>14.06s</td>
</tr>
<tr>
<td>8K</td>
<td>111.83s</td>
</tr>
<tr>
<td>16K</td>
<td>892.19s</td>
</tr>
</tbody>
</table>

\[
f(n) = 0.2273121n^3 + 0.007625303n^2 + 0.006868505n + 0.01817256
\]

\[
T(n) = n^3
\]
Time Complexity

def count(a):
    n = len(a)
    count = 0
    for i in range(0, n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if a[i] + a[j] + a[k] == 0:
                    count += 1
    return count

Statement Block Time Frequency Total Time
A t 4 1
B t 3 n t 3 n
C t 2 (n^2) 1 = n^2 / 2 - n / 2 t 2 (n^2 / 2 - n / 2)
D t 1 (n^3) = n^3 / 6 - n^2 / 2 + n / 3 t 1 (n^3 / 6 - n^2 / 2 + n / 3)
E t 0 (depends on input) t 0 x

Grand total: f(n) = (t_1 / 6)n^3 + (t_2 / 2 - t_1 / 2)n^2 + (t_1 / 3 - t_2 / 2 + t_3)n + t_4 + t_0 x

Running time: T(n) = n^3 1 \binom{n}{k} = n! k! (n-k)!
def count(a):
    n = len(a)
    count = 0
    [A]
    for i in range(0, n):
        [B]
        for j in range(i + 1, n):
            [C]
            for k in range(j + 1, n):
                [D]
                if a[i] + a[j] + a[k] == 0:
                    count += 1
    [E]
    return count

---

**Time Complexity**

<table>
<thead>
<tr>
<th>Statement Block</th>
<th>Time Frequency</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>t</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>t</td>
<td>3n</td>
</tr>
<tr>
<td>C</td>
<td>t</td>
<td>(n^2)</td>
</tr>
<tr>
<td>D</td>
<td>t</td>
<td>(n^3)</td>
</tr>
<tr>
<td>E</td>
<td>t</td>
<td>(depends on input)</td>
</tr>
</tbody>
</table>

**Grand total:**

\[ T(n) = \frac{t_1}{6} n^3 + \frac{t_2}{2} n^2 - \frac{t_1}{2} n + t_3 n + t_4 + t_0 x \]

**Running time:**

\[ T(n) = n^3 - k \cdot n^2 + \cdots \]
Time Complexity

```python
def count(a):
    n = len(a)
    count = 0
    for i in range(0, n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if a[i] + a[j] + a[k] == 0:
                    count += 1
    return count
```

<table>
<thead>
<tr>
<th>Statement Block</th>
<th>Time</th>
<th>Frequency</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A]</td>
<td>$t_4$</td>
<td>1</td>
<td>$t_4$</td>
</tr>
<tr>
<td>[B]</td>
<td>$t_3$</td>
<td>$n$</td>
<td>$t_3n$</td>
</tr>
<tr>
<td>[C]</td>
<td>$t_2$</td>
<td>$\binom{n}{2}^1 = n^2/2 - n/2$</td>
<td>$t_2(n^2/2 - n/2)$</td>
</tr>
<tr>
<td>[D]</td>
<td>$t_1$</td>
<td>$\binom{n}{3} = n^3/6 - n^2/2 + n/3$</td>
<td>$t_1(n^3/6 - n^2/2 + n/3)$</td>
</tr>
<tr>
<td>[E]</td>
<td>$t_0$</td>
<td>$\times$ (depends on input)</td>
<td>$t_0\times$</td>
</tr>
</tbody>
</table>

1. $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
**Time Complexity**

```python
def count(a):
    n = len(a)
    count = 0

    for i in range(0, n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if a[i] + a[j] + a[k] == 0:
                    count += 1

    return count
```

<table>
<thead>
<tr>
<th>Statement Block</th>
<th>Time</th>
<th>Frequency</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A]</td>
<td>$t_4$</td>
<td>1</td>
<td>$t_4$</td>
</tr>
<tr>
<td>[B]</td>
<td>$t_3$</td>
<td>$n$</td>
<td>$t_3n$</td>
</tr>
<tr>
<td>[C]</td>
<td>$t_2$</td>
<td>$\binom{n}{2}^1 = n^2/2 - n/2$</td>
<td>$t_2(n^2/2 - n/2)$</td>
</tr>
<tr>
<td>[D]</td>
<td>$t_1$</td>
<td>$\binom{n}{3} = n^3/6 - n^2/2 + n/3$</td>
<td>$t_1(n^3/6 - n^2/2 + n/3)$</td>
</tr>
<tr>
<td>[E]</td>
<td>$t_0$</td>
<td>$\times$ (depends on input)</td>
<td>$t_0x$</td>
</tr>
</tbody>
</table>

Grand total: $f(n) = (t_1/6)n^3 + (t_2/2 - t_1/2)n^2 + (t_1/3 - t_2/2 + t_3)n + t_4 + t_0x$

\[ \binom{n}{k} = \frac{n!}{k!(n-k)!} \]
def count(a):
    n = len(a)
    count = 0
    for i in range(0, n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if a[i] + a[j] + a[k] == 0:
                    count += 1
    return count

<table>
<thead>
<tr>
<th>Statement Block</th>
<th>Time</th>
<th>Frequency</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A]</td>
<td>$t_4$</td>
<td>1</td>
<td>$t_4$</td>
</tr>
<tr>
<td>[B]</td>
<td>$t_3$</td>
<td>$n$</td>
<td>$t_3n$</td>
</tr>
<tr>
<td>[C]</td>
<td>$t_2$</td>
<td>$\binom{n}{2} = n^2/2 - n/2$</td>
<td>$t_2(n^2/2 - n/2)$</td>
</tr>
<tr>
<td>[D]</td>
<td>$t_1$</td>
<td>$\binom{n}{3} = n^3/6 - n^2/2 + n/3$</td>
<td>$t_1(n^3/6 - n^2/2 + n/3)$</td>
</tr>
<tr>
<td>[E]</td>
<td>$t_0$</td>
<td>$\times$ (depends on input)</td>
<td>$t_0x$</td>
</tr>
</tbody>
</table>

Grand total: $f(n) = (t_1/6)n^3 + (t_2/2 - t_1/2)n^2 + (t_1/3 - t_2/2 + t_3)n + t_4 + t_0x$

Running time: $T(n) = n^3$
<table>
<thead>
<tr>
<th>Time Complexity</th>
<th>Code</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>1</td>
<td>increment the ( i )th element in an array</td>
<td></td>
</tr>
<tr>
<td>Logarithmic</td>
<td>( \log n )</td>
<td>divide and discard</td>
<td>binary search</td>
</tr>
<tr>
<td>Linear</td>
<td>( n )</td>
<td>loop</td>
<td>find the maximum</td>
</tr>
<tr>
<td>Linearithmic</td>
<td>( n \log n )</td>
<td>divide and conquer</td>
<td>merge sort</td>
</tr>
<tr>
<td>Quadratic</td>
<td>( n^2 )</td>
<td>double loop</td>
<td>check all ordered pairs</td>
</tr>
<tr>
<td>Cubic</td>
<td>( n^3 )</td>
<td>triple loop</td>
<td>check all ordered triples</td>
</tr>
<tr>
<td>Exponential</td>
<td>( 2^n )</td>
<td>exhaustive search</td>
<td>check all subsets</td>
</tr>
</tbody>
</table>
## Running time classifications

<table>
<thead>
<tr>
<th>Name</th>
<th>$T(n)$</th>
<th>Code Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>1</td>
<td>statement</td>
<td>increment the $i$th element in an array</td>
</tr>
<tr>
<td>logarithmic</td>
<td>$\log n$</td>
<td>divide and discard</td>
<td>binary search</td>
</tr>
<tr>
<td>linear</td>
<td>$n$</td>
<td>loop</td>
<td>find the maximum</td>
</tr>
<tr>
<td>linearithmic</td>
<td>$n \log n$</td>
<td>divide and conquer</td>
<td>merge sort</td>
</tr>
<tr>
<td>quadratic</td>
<td>$n^2$</td>
<td>double loop</td>
<td>check all ordered pairs</td>
</tr>
<tr>
<td>cubic</td>
<td>$n^3$</td>
<td>triple loop</td>
<td>check all ordered triples</td>
</tr>
<tr>
<td>exponential</td>
<td>$2^n$</td>
<td>exhaustive search</td>
<td>check all subsets</td>
</tr>
</tbody>
</table>
Space Complexity

The sizes of objects of built-in types differ from system to system, so the sizes of data types that we create also differ accordingly.

The function call `sys.getsizeof(x)` returns the number of bytes that a built-in object `x` consumes on a particular system.

Sizes of built-in objects on a typical system:

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Size (in bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>24</td>
</tr>
<tr>
<td>float</td>
<td>24</td>
</tr>
<tr>
<td>boolean</td>
<td>24</td>
</tr>
<tr>
<td>string of <code>n</code> characters</td>
<td>40 + <code>n</code></td>
</tr>
<tr>
<td>list of <code>n</code> integers</td>
<td>72 + 8 + <code>n</code></td>
</tr>
<tr>
<td><code>m</code>-by-<code>n</code> list of integers</td>
<td>72 + 8 + <code>m</code> + (72 + 32 <code>n</code>) = 72 + 8 + 32 <code>mn</code></td>
</tr>
</tbody>
</table>

User-defined objects are hundreds of bytes, at least.
The sizes of objects of built-in types differ from system to system, so the sizes of data types that we create also differ accordingly.
Space Complexity

The sizes of objects of built-in types differ from system to system, so the sizes of data types that we create also differ accordingly.

The function call \texttt{sys.getsizeof(x)} returns the number of bytes that a built-in object \(x\) consumes on a particular system.
Space Complexity

The sizes of objects of built-in types differ from system to system, so the sizes of data types that we create also differ accordingly.

The function call `sys.getsizeof(x)` returns the number of bytes that a built-in object `x` consumes on a particular system.

Sizes of built-in objects on a typical system

<table>
<thead>
<tr>
<th>Object</th>
<th>Size in Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>24</td>
</tr>
<tr>
<td>float</td>
<td>24</td>
</tr>
<tr>
<td>boolean</td>
<td>24</td>
</tr>
<tr>
<td>string of $n$ characters</td>
<td>$40 + n$</td>
</tr>
<tr>
<td>list of $n$ integers</td>
<td>$72 + 8n + 24n = 72 + 32n$</td>
</tr>
<tr>
<td>$m$-by-$n$ list of integers</td>
<td>$72 + 8m + m(72 + 32n) = 72 + 80m + 32mn$</td>
</tr>
<tr>
<td>user-defined</td>
<td>hundreds of bytes, at least</td>
</tr>
</tbody>
</table>