Introduction to Programming in Python

Algorithms and Data Structures: Analysis of Algorithms

Outline

1. Performance
2. Time Complexity
3. Space Complexity

Performance

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

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

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, 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

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

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

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

Time Complexity

Program: `triplesum.py`

Time Complexity

Program: `triplesum.py`

- Command-line input: filename (String)

Time Complexity

Program: `triplesum.py`

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

Time Complexity

Program: `triplesum.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 triplesum.py ../data/1Kints.txt
70
0.7 seconds
$ /usr/bin/time --format='%e seconds' python3 triplesum.py ../data/2Kints.txt
528
5.9 seconds
```

Time Complexity

Time Complexity

</> triplesum.py

```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

Time Complexity

n	$f(n)$
1K	0.28s
2K	1.8s
4K	14.06s
8K	111.83s
16K	892.19s

Time Complexity

n	$f(n)$
1K	0.28s
2K	1.8s
4K	14.06s
8K	111.83s
16K	892.19s

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

Time Complexity

n	$f(n)$
1K	0.28s
2K	1.8s
4K	14.06s
8K	111.83s
16K	892.19s

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

$$T(n) = n^3$$

Time Complexity

1 $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Time Complexity

```
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
```

1 $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Time Complexity

```
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
```

Statement Block	Time	Frequency	Total Time
$[A]$	t_4	1	t_4
$[B]$	t_3	n	$t_3 n$
$[C]$	t_2	$\binom{n}{2}$[1] $= n^2/2 - n/2$	$t_2(n^2/2 - n/2)$
$[D]$	t_1	$\binom{n}{3} = n^3/6 - n^2/2 + n/3$	$t_1(n^3/6 - n^2/2 + n/3)$
$[E]$	t_0	x (depends on input)	$t_0 x$

[1] $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Time Complexity

```
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
```

Statement Block	Time	Frequency	Total Time
$[A]$	t_4	1	t_4
$[B]$	t_3	n	$t_3 n$
$[C]$	t_2	$\binom{n}{2}$[1] $= n^2/2 - n/2$	$t_2(n^2/2 - n/2)$
$[D]$	t_1	$\binom{n}{3} = n^3/6 - n^2/2 + n/3$	$t_1(n^3/6 - n^2/2 + n/3)$
$[E]$	t_0	x (depends on input)	$t_0 x$

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

[1] $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Time Complexity

```
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
```

Statement Block	Time	Frequency	Total Time
[A]	t_4	1	t_4
[B]	t_3	n	$t_3 n$
[C]	t_2	$\binom{n}{2}$[1] $= n^2/2 - n/2$	$t_2(n^2/2 - n/2)$
[D]	t_1	$\binom{n}{3} = n^3/6 - n^2/2 + n/3$	$t_1(n^3/6 - n^2/2 + n/3)$
[E]	t_0	x (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] $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Time Complexity

Time Complexity

Running time classifications

Name	$T(n)$	Code Description	Example
constant	1	statement	increment the ith element in an array
logarithmic	$\log n$	divide and discard	binary search
linear	n	loop	find the maximum
linearithmic	$n \log n$	divide and conquer	merge sort
quadratic	n^2	double loop	check all ordered pairs
cubic	n^3	triple loop	check all ordered triples
exponential	2^n	exhaustive search	check all subsets

Space Complexity

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

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

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

Object	Size in Bytes
integer	24
float	24
boolean	24
string of n characters	$40 + n$
list of n integers	$72 + 8n + 24n = 72 + 32n$
m-by-n list of integers	$72 + 8m + m(72 + 32n) = 72 + 80m + 32mn$
user-defined	hundreds of bytes, at least