Principles of Computer Science II

Sorting Algorithms

Ioannis Chatzigiannakis

Sapienza University of Rome

Lecture 4

Simple Statistics

```
def main():
  sum = 0.0
  count = 0
  xStr = input("Enter a number (<Enter> to quit) >> ")
  while xStr != "":
    x = eval(xStr)
    sum = sum + x
    count = count + 1
    xStr = input("Enter a number (<Enter> to quit) >>")
  print( "\nThe average of the numbers is", sum / count)

main()
```

Simple Statistics: Observations

- The program itself does not keep track of the numbers that were entered – it only keeps a running total.
- We want to extend the program to compute not only the mean, but also the median and standard deviation.

Median

The median is the data value that splits the data into equal-sized parts.

Standard Deviation

$$s = \sqrt{\frac{\Sigma(\bar{x} - x_i)^2}{n-1}}$$

Simple Statistics: Extensions

- We need to keep track of all the values inserted by the user
- We do not know how many variables the user will provide.

Lists

- Python provides List to store sequences of values
- Lists in python are dynamic.
  - They grow/shrink on demand.
- Lists are mutable
  - Values can change on demand
  - Data type of individual items can change

Lists: Basic Examples

```
lst = [1,5,15,7]
print(lst)

lst[2] = 22
lst

lst[1] = "Hello"
lst

zeroes = [0] * 5
zerones = [0,1] * 3
zerones.append(2)
```

Lists: Operators

Operator	Meaning
seq + seq	Concatenation
seq * integer	Repetition
seq[]	Indexing
len(seq)	Length
sec[:]	Slicing
for var in sec:	Iteration
(expr) in sec	Membership (boolean)

Lists: Basic Examples

```
lst = lst + [22, 3]
len(lst)

15 in lst
3 in lst

sum = 0
for x in zerones:
        sum += x
print(sum)

X = zerones
zerones.append(2)

Y = lst[1:3]
Z = lst[3:-1]
K = lst[1:-3]
```

Lists: Methods

Method	Meaning
seq.append(x)	Add element x to end of list.
seq.sort()	Sort (order) the list. A comparison function may be passed as a parameter.
seq.reverse()	Reverse the list.
seq.index(x)	Returns index of first occurrence of x.
seq.insert(i, x)	Insert x into list at index i.
seq.count(x)	Returns the number of occurrences of x in list.
seq.remove(x)	Deletes the first occurrence of x in list.
seq.pop(i)	Deletes the ith element of the list and returns its value.

Lists: Basic Examples

```
lst = [3, 1, 4, 1, 5, 9]
lst.append(2)
lst

lst.sort()
lst

lst.reverse()

lst.index(4)

lst.insert(4, "Hello")

lst.count(1)

lst.remove(1)

lst.pop(3)
```

Simple Statistics: Modifications

- Collect input from user
- Store in a list

Simple Statistics: Modifications

- Collect input from user
- Store in a list

```
nums = []
x = input('Enter a number: ')
while x >= 0:
    nums.append(x)
    x = input('Enter a number: ')
```

Simple Statistics: Modifications

- Collect input from user
- Store in a list

```
nums = []
x = input('Enter a number: ')
while x >= 0:
    nums.append(x)
    x = input('Enter a number: ')
```

```
def mean(nums):
    sum = 0.0
    for num in nums:
        sum = sum + num
    return sum / len(nums)
```

Further Extensions

- How do we compute the standard deviation?
- Do we re-compute the mean?
  - Inefficient for large collections
- Do we pass the mean as a parameter?
  - Forced to invoke both functions sequentially

Further Extensions

- How do we compute the standard deviation?
- Do we re-compute the mean?
  - Inefficient for large collections
- Do we pass the mean as a parameter?
  - Forced to invoke both functions sequentially

```
def stdDev(nums, xbar):
    sumDevSq = 0.0
    for num in nums:
        dev = xbar - num
        sumDevSq = sumDevSq + dev * dev
    return sqrt(sumDevSq/(len(nums)-1))
```

Median

- How do we compute the median?
- Pseudocode

```
1. sort the numbers into ascending order
2. if the size of the data is odd:
3. median = the middle value
4. otherwise median = the average of the two middle values
5. return median
```

Median

- How do we compute the median?
- Pseudocode

```
1. sort the numbers into ascending order
2. if the size of the data is odd:
3. median = the middle value
4. otherwise median = the average of the two middle values
5. return median
```

```python
def median(nums):
    nums.sort()
    size = len(nums)
    midPos = size / 2
    if size % 2 == 0:
        median = (nums[midPos] + nums[midPos-1]) / 2.0
    else:
        median = nums[midPos]
    return median
```

Simple Statistics: New Version

```python
def main():
    print(\This program computes mean, median and standard deviation.")

    data = getNumbers()
    xbar = mean(data)
    std = stdDev(data, xbar)
    med = median(data)

    print('\nThe mean is', xbar)
    print('The standard deviation is', std)
    print('The median is', med)
```

Range

- range creates a list of numbers in a specified range
- range([start,] stop[, step])
- When step is given, it specifies the increment (or decrement).

```python
range(5)

range(5, 10)

range(0, 10, 2)

for i in range(0, len(lst), 2):
        print lst[i]
```

Zipping Lists

```python
lst = [1,5,15,7]
zerones = [0,1] * 3

k = zip(lst, zerones)

for (i,j) in k:
        print (i,j)
```

Tuples

```
data = [("julius", 3),
("maria", 2),
("alice", 4)]

for (n, a) in data:
        print("I met %s %s times" % (n, a))

data.sort()
```

Sorting in Python

```
list = [5, 6, 3, 7, 8, 11]
list.sort()

doubleList = [[8, 4, 5], [3, 8, 11], [4, 2, 19]]
doubleList.sort()


complexList = [('Alex', 5, M), ('Maria', 7, F),
               ('Katia', 1, F), ('Bruno', 2, M),
               ('Artemis', 1, F)]
complexList.sort()
```

- How do we sort with different criteria?

Lambda Functions

- Lambda is a tool for building functions, or more precisely, for building function objects.
- Python has two tools for building functions: def and lambda.

Function declaration

```
def square_root(x): return math.sqrt(x)

def sum(x,y): return x + y
```

Lambda function

```
square_root = lambda x: math.sqrt(x)

sum = lambda x, y:   x + y
```

Lambda vs Functions

When using Lambda makes sense?

- the function is fairly simple, and
- it is going to be used only once.

When using Functions makes sense?

- to reduce code duplication, or
  If your application contains duplicate chunks of code in various places, then you can put one copy of that code into a function, and then call it from various places in your code.
- to modularize code.
  If you have a chunk of code that performs one well-defined operation — but is really long and interrupts the readable flow of your program.

What Can be expressed using Lambda

- If it does not return a value, it is not an expression and cannot be put into a lambda.
- If you can imagine it in an assignment statement, on the right-hand side of the equals sign, it is an expression and can be put into a lambda.

What Can be expressed using Lambda

1. Assignment statements cannot be used in lambda – do not return anything, not even None (null).
2. Simple things: mathematical operations, string operations, list comprehensions, etc. are OK in a lambda.
3. Function calls are expressions. It is OK to put a function call in a lambda, and to pass arguments to that function. Doing this wraps the function call (arguments and all) inside a new, anonymous function.
4. In Python 3, print became a function, so in Python 3+, print(...) can be used in a lambda.
5. Functions that return None: like the print function in Python 3, can be used in a lambda.
6. Conditional expressions, return a value, and can be used in a lambda.

Lambda Examples

```
lambda: a if some_condition() else b

lambda x: 'big' if x > 100 else 'small'

out=lambda *x:print(" ".join(map(str,x)))
```

Data Assignment For Lists

Set an item in a list using the member function __setitem__

```
list[4] = 42
list.__setitem__(4,42)
```

Example: function that swaps two elements in a given list

```
def swap(a,x,y):
    a[x] = (a[x], a[y])
    a[y] = a[x][0]
    a[x] = a[x][1]
```

Example: lambda expression that swaps two elements in a given list

```
swap = lambda a,x,y:(lambda f=a.__setitem__:
       (f(x,(a[x],a[y])), f(y,a[x][0]), f(x,a[x][1])))()
```

Sorting with Lambda functions

```
doubleList = [[8, 4, 5], [3, 8, 11],
              [4, 2, 19], [3, 2, 19]]
doubleList.sort()
doubleList.sort(key=lambda x: x[2])
doubleList.sort(key=lambda x: -x[0])
doubleList.sort(key=lambda x: (-x[0], x[1]))
```

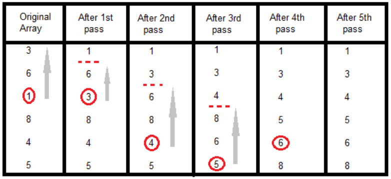
Sorting Algorithms

Can you design an algorithm that shorts the elements of a list?

Selection Sort Algorithm

This algorithm first finds the smallest element in the array and exchanges it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continues in this way until the entire array is sorted.

Selection Sort: Example



Original Array	After 1st pass	After 2nd pass	After 3rd pass	After 4th pass	After 5th pass
3	1	1	1	1	1
6	6	3	3	3	3
1	3	6	4	4	4
8	8	8	8	5	5
4	4	4	6	6	6
5	5	5	5	8	8

Selection Sort Code

```
a = [5, 1, 6, 2, 4, 3]
for i in range(0, len(a)):
    min = i
    for j in range(i + 1, len(a) - 1):
        if a[j] > a[min]:
            min = j

    temp = a[i]
    a[j] = a[min]
    a[min] = temp
```

How good is Selection Sort?

- How many comparisons are required until the list is sorted?

How good is Selection Sort?

- How many comparisons are required until the list is sorted?
  - 1st loop: n - 1
  - 2nd loop: n - 2
  - ...

How good is Selection Sort?

- How many comparisons are required until the list is sorted?
  - 1st loop: n - 1
  - 2nd loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required

How good is Selection Sort?

- How many comparisons are required until the list is sorted?
  - 1st loop: n - 1
  - 2nd loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\Sigma \frac{n(n-1)}{2}$ comparisons are required

How good is Selection Sort?

- How many comparisons are required until the list is sorted?
  - 1st loop: n - 1
  - 2nd loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\Sigma \frac{n(n-1)}{2}$ comparisons are required
- How much memory is needed ?

How good is Selection Sort?

- How many comparisons are required until the list is sorted?
  - 1st loop: n - 1
  - 2nd loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\Sigma \frac{n(n-1)}{2}$ comparisons are required
- How much memory is needed ?
  - 1 additional slot.